大数据分析中的算法

文再文

北京大学北京国际数学研究中心

http://bicmr.pku.edu.cn/~wenzw/bigdata2024.html

提纲

1. Course information
2. Machine learning
3. Sparse and low-rank optimization
4. AI For Science
5. Railway Timetabling

课程信息

- 大数据分析中的算法
  侧重数值代数和最优化算法
- 课程代码：00136720 (本科生），00100863 （本研合）
- 教师信息：文再文，wenzw@pku.edu.cn, 微信：wendoublewen
- 助教信息：李天佑，陈铖
- 上课地点：二教401
- 上课时间：每周周一3-4节(10:10am - 12:00am)，双周周四1-2节(8:00am - 9:50am)
- 课程主页:
  `http://bicmr.pku.edu.cn/~wenzw/bigdata2024.html`

参考资料

class notes, and reference books or papers

- “Convex optimization”, Stephen Boyd and Lieven Vandenberghe
- “Numerical Optimization”, Jorge Nocedal and Stephen Wright, Springer
- “Optimization Theory and Methods”, Wenyu Sun, Ya-Xiang Yuan
- 刘浩洋, 户将, 李勇锋，文再文，最优化：建模、算法与理论, 高教出版社，ISBN：9787040550351
- 文再文课题组，大数据分析中的算法

课程计划

侧重数值代数和最优化的模型与算法

- 线性规划，半定规划
- 压缩感知和稀疏优化基本理论和算法
- 低秩矩阵恢复的基本理论和算法
- Optimal transport
- 整数规划
- 随机优化算法
- 随机特征值算法
- 相位恢复
- 强化学习(reinforcement learning)

课程信息

- 教学方式：课堂讲授
- 成绩评定办法：
  - 迟交一天(24小时）打折10%，不接受晚交4天的作业和项目（任何时候理由都不接受）
  - 大作业，包括习题和程序：40%
  - 期中考试：30%
  - 课程项目：30%
  - 作业要求：i) 计算题要求写出必要的推算步骤，证明题要写出关键推理和论证。数值试验题应该同时提交书面报告和程序，其中书面报告有详细的推导和数值结果及分析。ii) 可以同学间讨论或者找助教答疑，但不允许在讨论中直接抄袭，应该过后自己独立完成。iii) 严禁从其他学生，从互联网，从往年的答案，其它课程等等任何途径直接抄袭。iv) 如果有讨论或从其它任何途径取得帮助，请列出来源。
- 请谨慎选课

提纲

1. Course information
2. Machine learning
3. Sparse and low-rank optimization
4. AI For Science
5. Railway Timetabling

机器学习：监督学习简介

- 机器学习，人工智能
  - 计算机视觉，自然语言处理(ChatGPT)
  - 围棋：AlphaGo, AlphaGo Zero
- 科学发现经常围绕建立函数关系：$f : X \to Y$
  - 蛋白质结构预测(Alphafold2): X: 蛋白质序列; Y: 三维结构
  - 深度势能: X: 分子结构; Y: 原子势能
- 挑战: f 非常高维、高度非线性，只有它的很少量已知的知识或者计算非常昂贵
- 机遇: 我们有很多的数据:

$$\{(x_i, y_i) \mid x_i \in X, y_i \in Y, 1 \le i \le N\}.$$

- 机器学习是构造 $\hat{f} \approx f$ 的强大工具

监督学习中典型问题形式

机器学习构造$\hat{f}$ 的典型方式：$\hat{f} = h(x, \theta)$

$$\min_{\theta \in \mathcal{W}} \quad \frac{1}{N} \sum_{i=1}^{N} \|x_i^\top \theta - y_i\|_2^2 + \mu \varphi(\theta) \quad \text{线性回归}$$

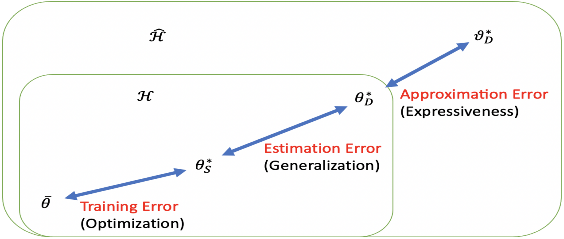
$$\min_{\theta \in \mathcal{W}} \quad \frac{1}{N} \sum_{i=1}^{N} \log(1 + \exp(-y_i x_i^\top \theta)) + \mu \varphi(\theta) \quad \text{逻辑回归}$$

$$\min_{\theta \in \mathcal{W}} \quad \frac{1}{N} \sum_{i=1}^{N} \ell(h(x_i, \theta), y_i) + \mu \varphi(\theta) \quad \text{一般形式}$$

- (x_i, y_i) 是给定的数据对，y_i 是数据x_i对应的标签
- $\ell_i(\cdot)$: 度量模型拟合数据点i的程度(避免拟合不足)
- $\varphi(\theta)$: 避免过拟合的正则项: $\|\theta\|_2^2$ 或者$\|\theta\|_1$等等
- $h(x, \theta)$: 线性函数、决策树/森林或者由深度神经网络构造的模型

机器学习：表达能力，泛化，优化

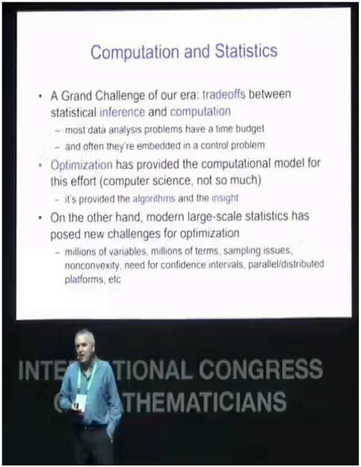
- ground truth: $\vartheta_D^* = \underset{\vartheta \in \widehat{\mathcal{H}}}{\operatorname{argmin}} \mathbf{E}[\ell(h(x,\theta), y)]$
- optimal hypothesis: $\theta_D^* = \arg\min_{\theta \in \mathcal{H}} \mathbf{E}[\ell(h(x,\theta), y)]$
- empirical optimal hypothesis: $\theta_S^* = \underset{\theta \in \mathcal{H}}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^N \ell(h(x_i,\theta), y_i)$
- returned hypothesis: $\bar{\theta}$



机器学习与优化

美国三院院士Michael Jordan教授国际数学家大会一小时报告

- 我们时代的一个很大挑战是统计推理和计算的平衡。大部分数据分析有时间限制，他们经常被嵌入到某个控制问题里。
- 最优化为这个努力提供了计算模型，给出了算法和深刻的理解。
- 现代大规模统计给优化带来了新的挑战：百万量级变量/函数项，抽样问题，非凸，置信区间，并行/分布式平台等等



Computation and Statistics

- A Grand Challenge of our era: tradeoffs between statistical inference and computation
  - most data analysis problems have a time budget
  - and often they're embedded in a control problem
- Optimization has provided the computational model for this effort (computer science, not so much)
  - it's provided the algorithms and the insight
- On the other hand, modern large-scale statistics has posed new challenges for optimization
  - millions of variables, millions of terms, sampling issues, nonconvexity, need for confidence intervals, parallel/distributed platforms, etc

机器学习与优化

“Statistics and the Oncoming AI Revolution”

- What has made ML so successful? What are the disciplines supporting ML and providing a good basis to understand the challenges, open problems, and limitations of the current techniques?
  1) **basic statistical tools**: linear models, generalized linear models, logistic regression, cross validation, overfitting . . .
  2) **probability theory and probabilistic modeling.**
- How about engineering disciplines? **Clearly, progress in optimization—particularly in convex optimization—has fueled ML algorithms for the last two decades.**

美国科学院院士Emmanuel Candès

通用最优化模型和算法

最优化问题一般可以描述为

$$\begin{aligned} \min \quad & f(x), \\ \text{s.t.} \quad & x \in \mathcal{X} \end{aligned}$$

- 按照目标函数和约束函数的形式来分：
  线性规划/非线性规划、凸优化/非凸优化、非光滑优化、半定规划、锥规划、整数规划、无导数优化、几何优化、稀疏优化、低秩矩阵优化、张量优化、鲁棒优化、全局优化、组合优化、网络规划、随机优化、动态规划、带微分方程约束优化、微分流形约束优化、分布式优化等

- 具体应用涵盖：
  运筹学、供应链管理、物流管理、资产管理、统计学习、压缩感知、最优运输、信号处理、图像处理、机器学习、强化学习、模式识别、金融工程、电力系统等领域

提纲

1. Course information
2. Machine learning
3. Sparse and low-rank optimization
4. AI For Science
5. Railway Timetabling

稀疏优化

$$(\ell_0)\begin{cases}\displaystyle\min_{x\in\mathbb{R}^n} & \|x\|_0,\\ \text{s.t.} & Ax=b.\end{cases}\qquad(\ell_2)\begin{cases}\displaystyle\min_{x\in\mathbb{R}^n} & \|x\|_2,\\ \text{s.t.} & Ax=b.\end{cases}\qquad(\ell_1)\begin{cases}\displaystyle\min_{x\in\mathbb{R}^n} & \|x\|_1,\\ \text{s.t.} & Ax=b.\end{cases}$$

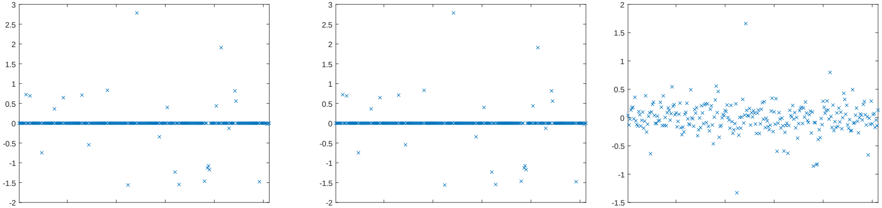
- 其中$\|x\|_0$是指x中非零元素的个数．由于$\|x\|_0$是不连续的函数，且取值只可能是整数，ℓ_0问题实际上是NP难的，求解起来非常困难．
- 若定义ℓ_1范数：$\|x\|_1=\sum_{i=1}^n|x_i|$，我们得到了另一个形式上非常相似的问题，又称$\ell_1$范数优化问题，基追踪问题
- ℓ_2范数：$\|x\|_2=\left(\sum_{i=1}^n x_i^2\right)^{1/2}$

稀疏优化

在MATLAB环境里构造A, u 和b：

```
m = 128; n = 256;
A = randn(m, n);
u = sprandn(n, 1, 0.1);
b = A * u;
```

构造一个128×256矩阵A，它的每个元素都服从高斯（Gauss）随机分布．精确解u只有10%的元素非零，每一个非零元素也服从高斯分布



(a) 精确解u　(b) ℓ_1问题的解　(c) ℓ_2问题的解

Figure: 稀疏优化的例子

低秩矩阵恢复

- 某视频网站提供了约48万用户对1万7千多部电影的上亿条评级数据，希望对用户的电影评级进行预测，从而改进用户电影推荐系统，为每个用户更有针对性地推荐影片．
- 显然每一个用户不可能看过所有的电影，每一部电影也不可能收集到全部用户的评级．电影评级由用户打分1 星到5 星表示，记为取值1~5 的整数．我们将电影评级放在一个矩阵M中，矩阵M的每一行表示不同用户，每一列表示不同电影．由于用户只对看过的电影给出自己的评价，矩阵M中很多元素是未知的

$$\begin{array}{c|cccccc|}
 & \text{电影1} & \text{电影2} & \text{电影3} & \text{电影4} & \cdots & \text{电影n} \\
\hline
\text{用户1} & 4 & ? & ? & 3 & \cdots & ? \\
\text{用户2} & ? & 2 & 4 & ? & \cdots & ? \\
\text{用户3} & 3 & ? & ? & ? & \cdots & ? \\
\text{用户4} & 2 & ? & 5 & ? & \cdots & ? \\
\vdots & \vdots & \vdots & \vdots & \vdots & & \vdots \\
\text{用户m} & ? & 3 & ? & 4 & \cdots & ?
\end{array}$$

低秩矩阵恢复

- 令Ω 是矩阵M 中所有已知评级元素的下标的集合，则该问题可以初步描述为构造一个矩阵X，使得在给定位置的元素等于已知评级元素，即满足$X_{ij} = M_{ij},\ (i,j) \in \Omega$.
- 低秩矩阵恢复（low rank matrix completion）

$$\begin{aligned} \min_{X \in \mathbb{R}^{m\times n}} \quad & \operatorname{rank}(X), \\ \text{s.t.} \quad & X_{ij} = M_{ij},\ (i,j) \in \Omega. \end{aligned}$$

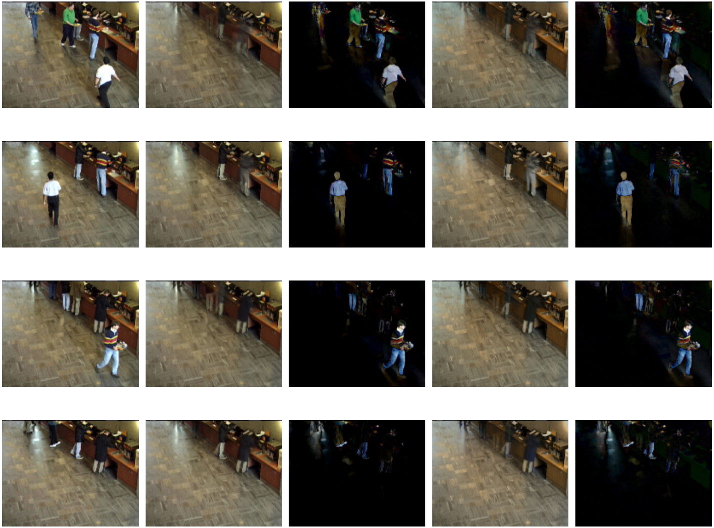
rank(X)正好是矩阵X所有非零奇异值的个数

- 矩阵X的核范数（nuclear norm）为矩阵所有奇异值的和，即：$\|X\|_* = \sum_i \sigma_i(X)$:

$$\begin{aligned} \min_{X \in \mathbb{R}^{m\times n}} \quad & \|X\|_*, \\ \text{s.t.} \quad & X_{ij} = M_{ij},\ (i,j) \in \Omega. \end{aligned}$$

视频分离

- 将视频分成移动和静态的部分



稀疏和低秩矩阵分离

- 给定矩阵M，我们想找到一个低秩矩阵W和稀疏矩阵E，使得

$$W + E = M.$$

- 非凸模型：

$$\begin{aligned} \min_{W,E\in\mathbb{R}^{m\times n}} \quad & \mathrm{rank}(W) + \mu\|E\|_0, \\ \text{s.t.} \quad & W + E = M. \end{aligned}$$

- 凸松弛：

$$\min_{W,E} \ \|W\|_* + \mu\|E\|_1, \ \text{s.t.} \ W + E = M$$

- 鲁棒主成分分析

提纲

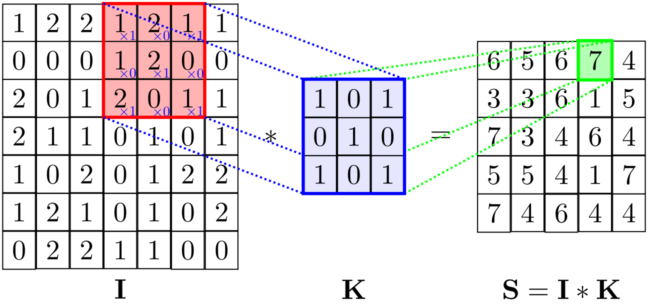
1. Course information
2. Machine learning
3. Sparse and low-rank optimization
4. **AI For Science**
5. Railway Timetabling

卷积神经网络Convolutional neural network (CNN)

- 给定二维图像$\boldsymbol{I} \in \mathbb{R}^{n\times n}$和卷积核$\boldsymbol{K} \in \mathbb{R}^{k\times k}$，定义卷积操作$\boldsymbol{S} = \boldsymbol{I} * \boldsymbol{K}$，它的元素是

$$S_{i,j} = \langle \boldsymbol{I}(i : i+k-1, j : j+k-1), \boldsymbol{K} \rangle ,$$

  其中两个矩阵X, Y的内积是它们相应元素乘积之和
- 生成的结果$\boldsymbol{S}$可以根据卷积核的维数、$\boldsymbol{I}$的边界是否填充、卷积操作时滑动的大小等相应变化.



1	2	2	1	2	1	1
0	0	0	1	2	0	0
2	0	1	2	0	1	1
2	1	1	0	1	0	1
1	0	2	0	1	2	2
1	2	1	0	1	0	2
0	2	2	1	1	0	0

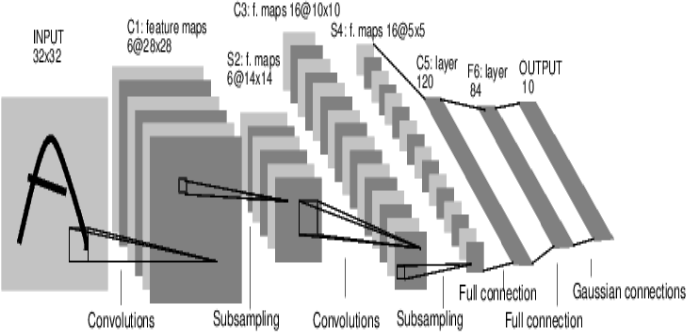
1	0	1
0	1	0
1	0	1

6	5	6	7	4
3	3	6	1	5
7	3	4	6	4
5	5	4	1	7
7	4	6	4	4

Figure: 卷积操作

卷积神经网络Convolutional neural network (CNN)

LeCun等人开创性的建立了数字分类的神经网络。几家银行使用它来识别支票上的手写数字。



Schrödinger equation

- The N-electron Schrödinger equation

$$H\Psi = E\Psi,$$

  where H is the molecular Hamiltonian operator, Ψ is a N-electron antisymmetric wave function.

- Curse of dimensionality: computational work goes as 10^{3N}.

- Kohn-Sham total energy minimization (Manifold Optimization):

$$\min_{X^*X=I} E_{KS}(X) := E_{kinetic}(X) + E_{ion}(X) + E_{Hartree}(X) + E_{xc}(X),$$

  where $\nabla E_{KS}(X) = H(X)X$.

- Kohn-Sham equation:

$$H(X)X = X\Lambda, \quad X^*X = I$$

FermiNet: 特征值优化问题+深度神经网络+SGD

- Minimizing the following Rayleigh quotient:

$$E_0 = \min_{\Psi} E[\Psi] = \frac{\int \Psi^*(\mathbf{r})\hat{H}\Psi(\mathbf{r})\mathrm{d}\mathbf{r}}{\int \Psi^*(\mathbf{r})\Psi(\mathbf{r})\mathrm{d}\mathbf{r}}$$

- Variational Quantum Monte Carlo:

$$\begin{aligned}\min_{\theta} \mathcal{L}(\theta) &= \frac{\int \Psi_\theta^*(\mathbf{r})\hat{H}\Psi_\theta(\mathbf{r})\mathrm{d}\mathbf{r}}{\int \Psi_\theta^*(\mathbf{r})\Psi_\theta(\mathbf{r})\mathrm{d}\mathbf{r}} \\ &= \int \frac{|\Psi_\theta(\mathbf{r})|^2}{\int |\Psi_\theta(\mathbf{r})|^2\mathrm{d}\mathbf{r}} \left(\Psi_\theta^{-1}(\mathbf{r})\hat{H}\Psi_\theta(\mathbf{r})\right) \mathrm{d}\mathbf{r} \\ &= \mathbb{E}_{P_\theta(\mathbf{r})}\left[E_L(\mathbf{r})\right] \Longrightarrow \text{solved by SGD}\end{aligned}$$

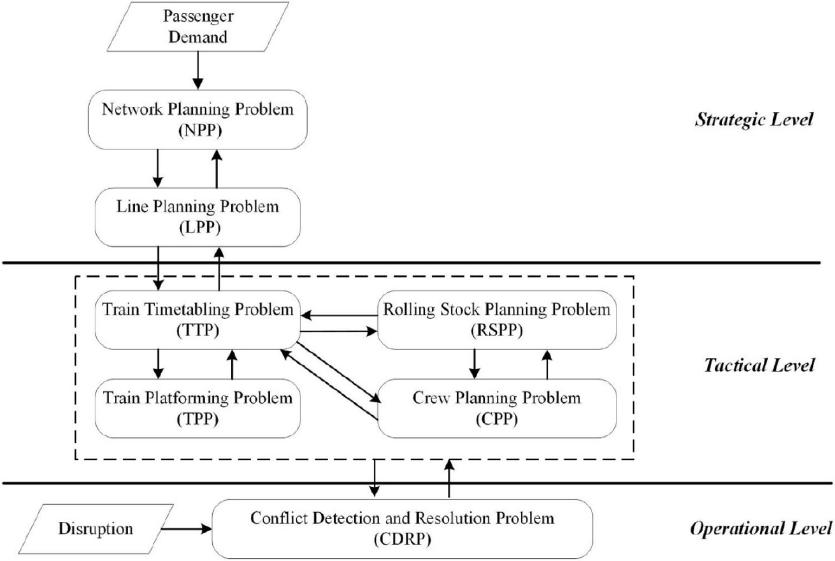
- Local Energy: $E_L(\mathbf{r}) = \Psi_\theta^{-1}(\mathbf{r})\hat{H}\Psi_\theta(\mathbf{r})$,
  Probability distribution: $P_\theta(\mathbf{r}) = \frac{|\Psi_\theta(\mathbf{r})|^2}{\int |\Psi_\theta(\mathbf{r})|^2\mathrm{d}\mathbf{r}}$

提纲

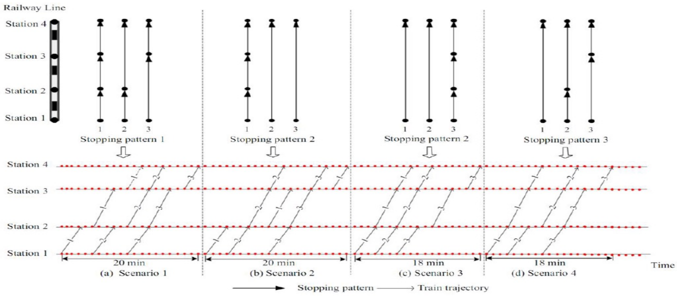
1. Course information
2. Machine learning
3. Sparse and low-rank optimization
4. AI For Science
5. Railway Timetabling

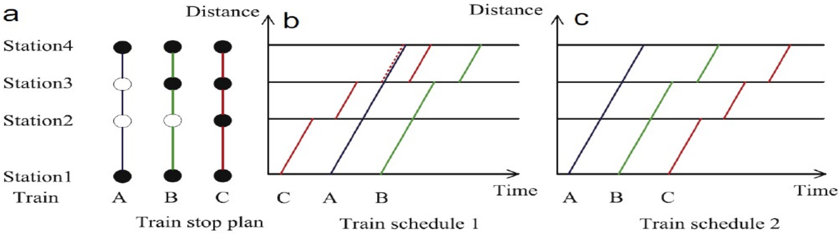
铁路运行图的地位

致谢：北京交通大学乐逸祥教授团队

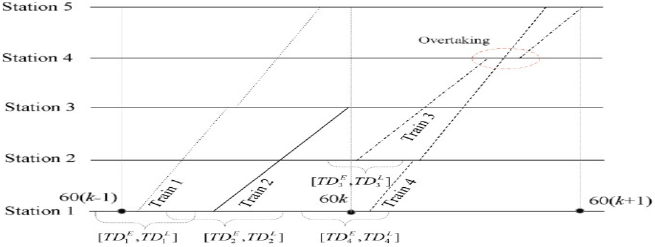

Passenger Demand
Network Planning Problem (NPP)
Line Planning Problem (LPP)
Strategic Level
Train Timetabling Problem (TTP)
Rolling Stock Planning Problem (RSPP)
Train Platforming Problem (TPP)
Crew Planning Problem (CPP)
Tactical Level
Disruption
Conflict Detection and Resolution Problem (CDRP)
Operational Level

开行方案

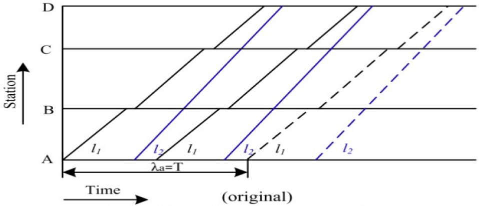

Railway Line
Station 4
Station 3
Station 2
Station 1
1 2 3
Stopping pattern 1
Stopping pattern 2
Stopping pattern 2
Stopping pattern 3
Station 4
Station 3
Station 2
Station 1
20 min
(a) Scenario 1
20 min
(b) Scenario 2
18 min
(c) Scenario 3
18 min
(d) Scenario 4
Time
Stopping pattern
Train trajectory


a
Station4
Station3
Station2
Station1
Train
A B C
Train stop plan
Distance
b
C A B
Time
Train schedule 1
Distance
c
A B C
Time
Train schedule 2

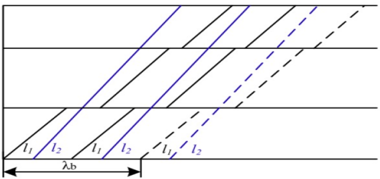
铁路运行图难点：速差、列车越行



(i) Traditional train departure time window

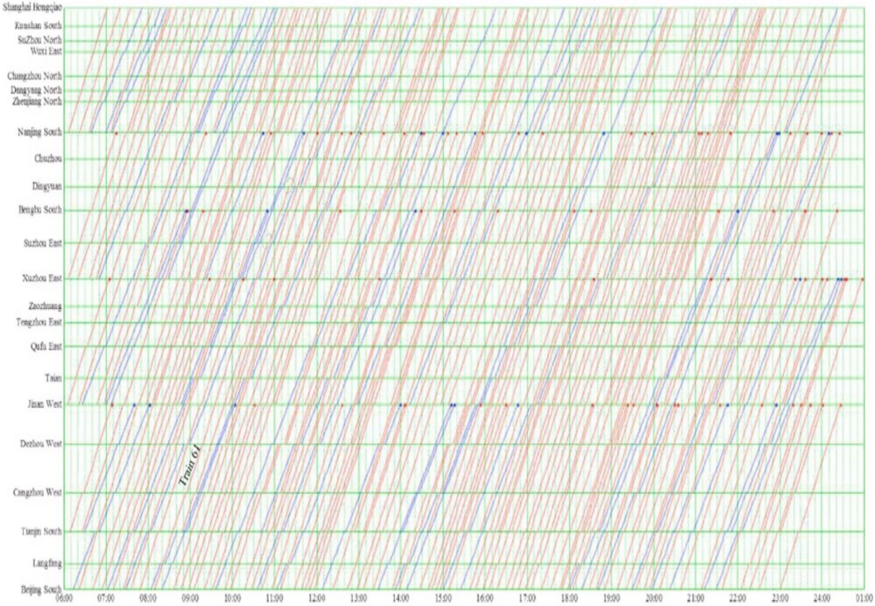


(original)
(a) Timetable without overtaking allowance



(capacity)
(b) Timetable with overtaking allowance

铁路运行图难点：高密度

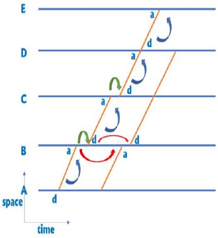

Shanghai Hongqiao
Kunshan South
SuZhou North
Wuxi East
Changzhou North
Danyang North
Zhenjiang North
Nanjing South
Chuzhou
Dingyuan
Bengbu South
Suzhou East
Xuzhou East
Zaozhuang
Tengzhou East
Qufu East
Taian
Jinan West
Dezhou West
Cangzhou West
Tianjin South
Langfang
Beijing South
Train 61
06:00
07:00
08:00
09:00
10:00
11:00
12:00
13:00
14:00
15:00
16:00
17:00
18:00
19:00
20:00
21:00
22:00
23:00
24:00
01:00

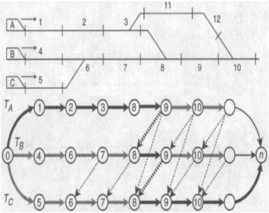
铁路运行图建模

难点：列车前后关系的确定，MILP

基于列车事件（到达、出发事件）进行建模

1. **Job-shop like**



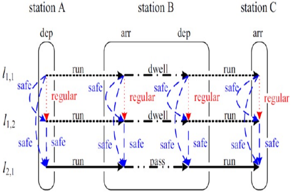


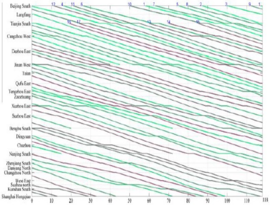
Commercial Solvers

B&B (Overall, Inserting)

Heuristics (GA,SA,TS,ACO)

2. **PESP**



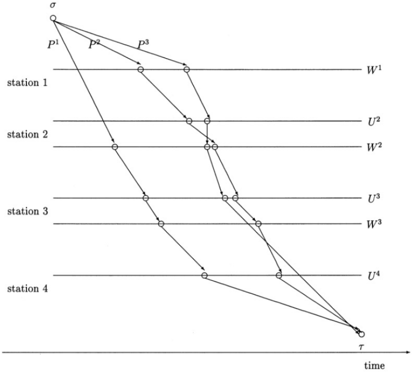


Commercial Solvers

Modulo Simplex Heuristic

铁路运行图建模

时空网络模型：



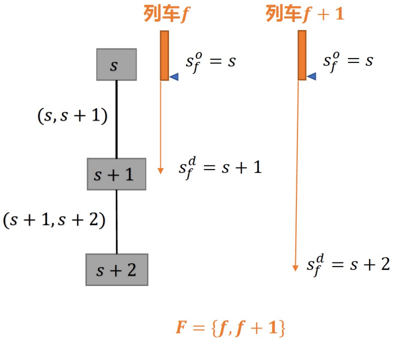
Lagrangian Relaxation

ADMM

Branch and Price

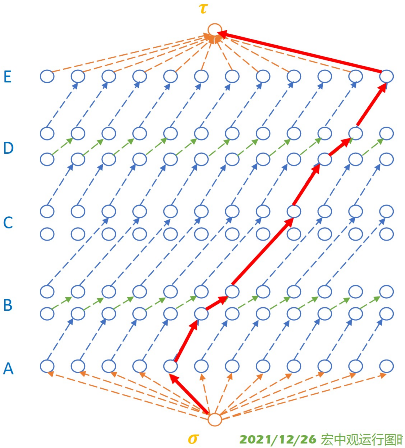
宏观运行图建模：时空网络模型

符号	解释
s	车站索引
SG	区间集合
(s_1, s_2)	区间索引
F	列车集合
f	列车索引
s_f^o	列车f的始发站索引
s_f^d	列车f的终到站索引
t	时间索引
$u, u_{s,t}$	通用时空节点
V_f	列车f可能经过的时空节点集合
σ	虚拟源节点
τ	虚拟汇节点



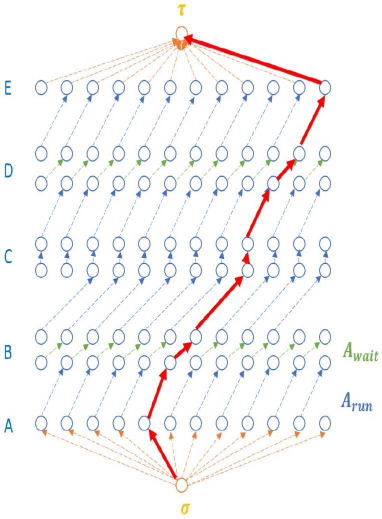
宏观运行图建模：时空网络模型

符号	解释
s	车站索引
SG	区间集合
(s_1, s_2)	区间索引
F	列车集合
f	列车索引
s_f^o	列车f的始发站索引
s_f^d	列车f的终到站索引
t	时间索引
$u, u_{s,t}$	通用时空节点
V_f	列车f可能经过的时空节点集合
σ	虚拟源节点
τ	虚拟汇节点



宏观运行图建模：时空网络模型

符号	解释
A	时空弧段集合
$A^{+}(u)$	流入时空节点u的弧段集合
$A^{-}(u)$	流出时空节点u的弧段集合
A_f	属于列车f的弧段集合
A_f^{wait}	属于列车f的停站弧段集合
A_f^{run}	属于列车f的区间运行弧段集合
A_f^{vt}	属于列车f的虚拟弧段集合 用于连接源节点与汇节点
a_f	属于列车f的弧段索引

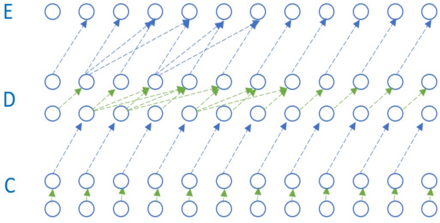


宏观运行图建模：目标函数

目标函数：列车总旅行时分最短

$$\min \sum_{f\in F}\sum_{a_f\in A_f} w_{a_f}x_{a_f}$$

$$w_{a_f}=\begin{cases} p_f(s_1,s_2) & \forall a_f\in A_f^{run}\cap(s_1,s_2)\\ \delta^{a_f} & \forall a_f\in A_f^{wait}\cap s\\ 0 & \forall a_f\in A_f^{vt}\end{cases}$$



$$\delta_{min}^{f,s}\le\delta^{a_f}\le\delta_{max}^{f,s}$$

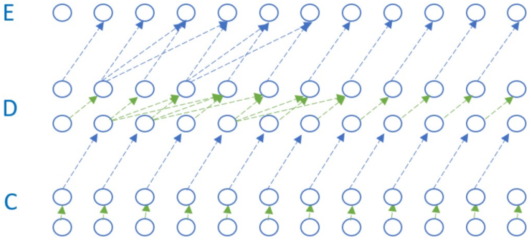
变量x_{af} 表示列车f是否在弧段a上运行:

$$x_{af}=\begin{cases}1 & \text{if 列车}f\text{在弧段}a\text{上运行}\\ 0 & \text{o.w.}\end{cases}$$

宏观运行图建模：流平衡约束

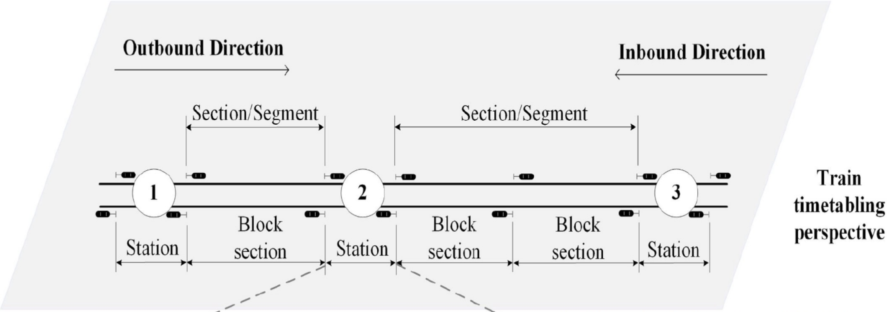
流平衡约束

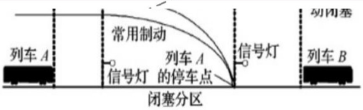
$$\sum_{a_f \in A^+(u_f)} x_{a_f} - \sum_{a_f \in A^-(u_f)} x_{a_f} = \begin{cases} -1 & if\ u_f = \sigma \\ 1 & if\ u_f = \tau \\ 0 & otherwise \end{cases} \qquad \forall u_f \in V_f, \forall f \in F$$



宏观运行图建模：轨道占用约束

轨道占用约束





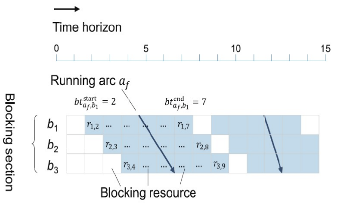
$$R(a_f) = \{r_{b,t} | bt_{a_f,b} \le t \le bt_{a_f,b}\}, \quad \forall b \in B(a_f)$$

$$\sum_{f\in F}\sum_{a\in A_f^{run}:r\in R(a_f)} x_{a_f} \le 1, \quad \forall r \in R$$

宏观运行图建模：轨道占用约束

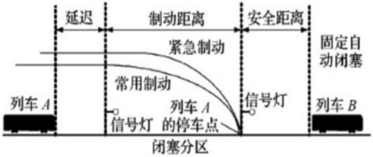
轨道占用约束

b	资源分区索引（实际闭塞分区）
$B_{(a)}$	弧段a占用的资源分区集合（实际闭塞分区）
$r, r_{b,t}$	时空资源，资源分区在各时段的资源体现
R	时空资源集合
$R_{(a)}$	弧段a占用的时空资源集合


Time horizon
0
5
10
15
Running arc a_f
$bt^{start}_{a_f,b_1} = 2$
$bt^{end}_{a_f,b_1} = 7$
Blocking section
b_1
b_2
b_3
$r_{1,2}$ ... ... $r_{1,7}$
$r_{2,3}$ ... ... $r_{2,8}$
$r_{3,4}$ ... ... $r_{3,9}$
Blocking resource

$$R(a_f) = \{r_{1,2}, \ldots r_{1,7}, r_{2,3}, \ldots, r_{2,8}, r_{3,4}, \ldots, r_{3,9}\}$$

🟦 Occupied blocking resource



$$R(a_f) = \left\{ r_{b,t} \middle| bt^{start}_{a_f,b} \le t \le bt^{end}_{a_f,b} \right\}, \qquad \forall b \in B(a_f)$$

$$\sum_{f \in F} \sum_{a_f \in A_f^{run}: r \in R(a_f)} x_{a_f} \le 1, \qquad \forall r \in R$$

宏观运行图建模：整体模型

总体模型

$$\min \sum_{f\in F}\sum_{a_f\in A_f} w_{a_f} x_{a_f}$$

$$w_{a_f} = \begin{cases} p_f(s_1,s_2) & \forall a_f \in A_f^{run} \cap (s_1,s_2) \\ \delta^{a_f} & \forall a_f \in A_f^{wait} \cap s \\ 0 & \forall a_f \in A_f^{vt} \end{cases}$$

$$\sum_{a_f\in A^{+}(u_f)} x_{a_f} - \sum_{a_f\in A^{-}(u_f)} x_{a_f} = \begin{cases} -1 & if\ u_f = \sigma \\ 1 & if\ u_f = \tau \\ 0 & otherwise \end{cases} \qquad \forall u_f \in V_f, \forall f \in F$$

$$\sum_{f\in F}\sum_{a\in A_f^{run}: r\in R(a_f)} x_{a_f} \le 1, \qquad \forall r \in R$$

$$x_{a_f} \in \{0,1\}$$

References: Simultaneously re-optimizing timetables and platform schedules under planned track maintenance for a high-speed railway network, `https://doi.org/10.1016/j.trc.2020.102823`

课程项目

课程项目文件描述（持续更新）：
`http://faculty.bicmr.pku.edu.cn/~wenzw/bigdata/trainsch.pdf`

- （混合）整数规划建模: 从自然语言描述用大语言模型自动建模
- 调用Gurobi, Mosek或者SCIP等算法软件直接求解
- 基于ALM等连续优化算法+贪婪算法求解
- 实现AM系列算法求解
- 强化学习方法求解

Lecture: Introduction to LP, SDP and SOCP

Zaiwen Wen

Beijing International Center For Mathematical Research
Peking University

`http://bicmr.pku.edu.cn/~wenzw/bigdata2023.html`
wenzw@pku.edu.cn

Acknowledgement: this slides is based on Prof. Farid Alizadeh lecture notes

Outline

1. Linear Programming (LP)
2. Semidefinite Programming (SDP)
3. Second Order Cone Programming (SOCP)

Linear Programming (LP)

Primal

$$\begin{aligned} \min_{x_i} \quad & c_1x_1 + \ldots + c_nx_n \\ \text{s.t.} \quad & a_{11}x_1 + \ldots + a_{1n}x_n = b_1 \\ & \ldots \\ & a_{m1}x_1 + \ldots + a_{mn}x_n = b_m \\ & x_i \geq 0 \end{aligned}$$

Dual

$$\begin{aligned} \max_{y_i} \quad & b_1y_1 + \ldots + b_my_m \\ \text{s.t.} \quad & a_{11}y_1 + \ldots + a_{m1}y_m \leq c_1 \\ & \ldots \\ & a_{1n}y_1 + \ldots + a_{mn}y_m \leq c_n \end{aligned}$$

Linear Programming (LP)

more succinctly

Primal (P)

$$\begin{aligned} \min_{x} \quad & c^\top x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

Dual (D)

$$\begin{aligned} \max_{y,s} \quad & b^\top y \\ \text{s.t.} \quad & A^\top y + s = c \\ & s \geq 0 \end{aligned}$$

Problems with absolute values

$$\begin{array}{ll} \min & \sum_i c_i|x_i|, \qquad \text{assume } c \geq 0 \\ \text{s.t.} & Ax \geq b \end{array}$$

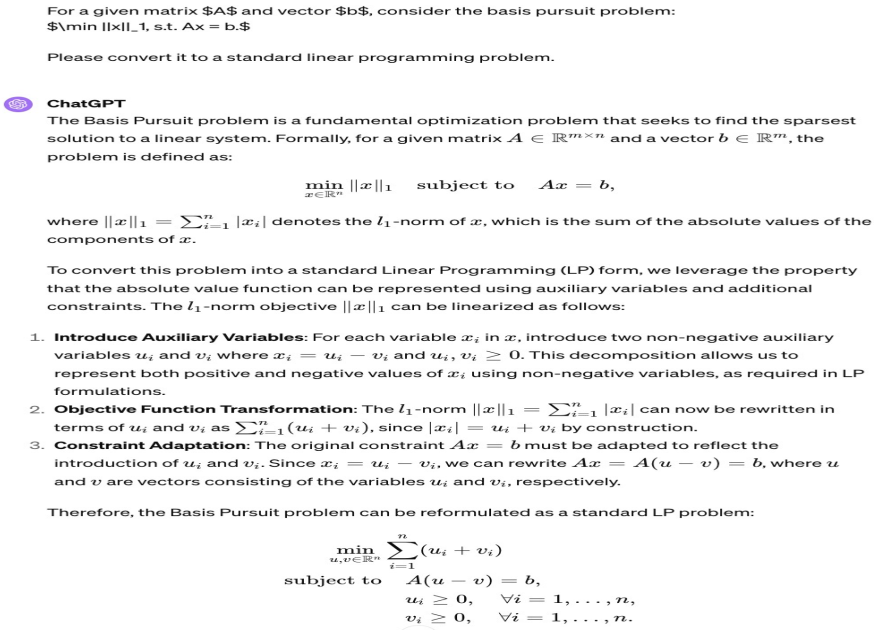
- Reformulation 1:

$$\begin{array}{ll} \min & \sum_i c_i z_i \\ \text{s.t.} & Ax \geq b \\ & |x_i| \leq z_i \end{array} \iff \begin{array}{ll} \min & \sum_i c_i z_i \\ \text{s.t.} & Ax \geq b \\ & -z_i \leq x_i \leq z_i \end{array}$$

- Reformulation 2: $x_i = x_i^+ - x_i^-$, $x_i^+, x_i^- \geq 0$. Then $|x_i| = x_i^+ + x_i^-$

$$\begin{array}{ll} \min & \sum_i c_i(x_i^+ + x_i^-) \\ \text{s.t.} & Ax^+ - Ax^- \geq b, x^+, x^- \geq 0 \end{array}$$

LLM: GPT4



For a given matrix A and vector b, consider the basis pursuit problem:
$\min ||x||_1, s.t. Ax = b.$

Please convert it to a standard linear programming problem.

ChatGPT

The Basis Pursuit problem is a fundamental optimization problem that seeks to find the sparsest solution to a linear system. Formally, for a given matrix $A \in \mathbb{R}^{m \times n}$ and a vector $b \in \mathbb{R}^m$, the problem is defined as:

$$\min_{x \in \mathbb{R}^n} ||x||_1 \quad \text{subject to} \quad Ax = b,$$

where $||x||_1 = \sum_{i=1}^n |x_i|$ denotes the l_1-norm of x, which is the sum of the absolute values of the components of x.

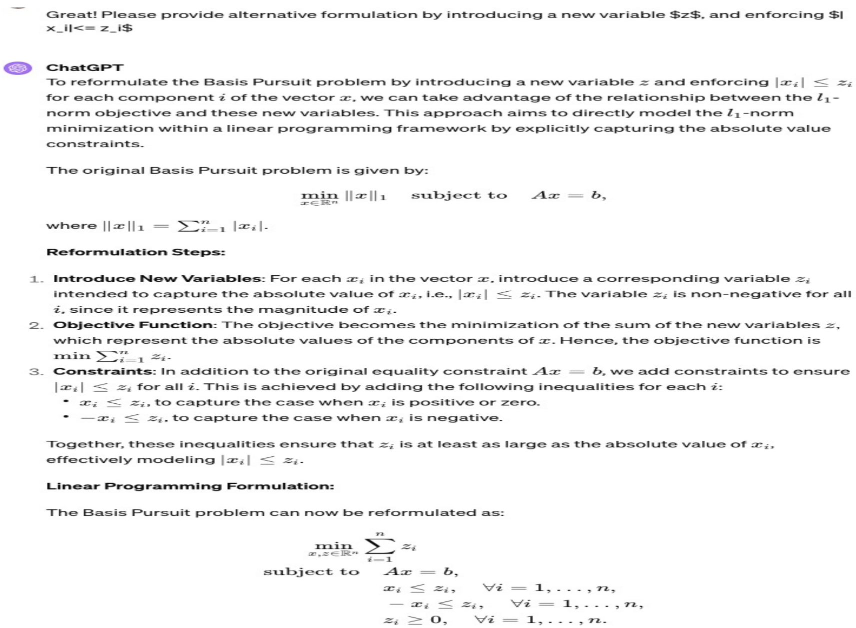
To convert this problem into a standard Linear Programming (LP) form, we leverage the property that the absolute value function can be represented using auxiliary variables and additional constraints. The l_1-norm objective $||x||_1$ can be linearized as follows:

1. **Introduce Auxiliary Variables**: For each variable x_i in x, introduce two non-negative auxiliary variables u_i and v_i where $x_i = u_i - v_i$ and $u_i, v_i \geq 0$. This decomposition allows us to represent both positive and negative values of x_i using non-negative variables, as required in LP formulations.
2. **Objective Function Transformation**: The l_1-norm $||x||_1 = \sum_{i=1}^n |x_i|$ can now be rewritten in terms of u_i and v_i as $\sum_{i=1}^n (u_i + v_i)$, since $|x_i| = u_i + v_i$ by construction.
3. **Constraint Adaptation**: The original constraint $Ax = b$ must be adapted to reflect the introduction of u_i and v_i. Since $x_i = u_i - v_i$, we can rewrite $Ax = A(u - v) = b$, where u and v are vectors consisting of the variables u_i and v_i, respectively.

Therefore, the Basis Pursuit problem can be reformulated as a standard LP problem:

$$\begin{aligned} \min_{u,v \in \mathbb{R}^n} \quad & \sum_{i=1}^n (u_i + v_i) \\ \text{subject to} \quad & A(u - v) = b, \\ & u_i \geq 0, \quad \forall i = 1, \ldots, n, \\ & v_i \geq 0, \quad \forall i = 1, \ldots, n. \end{aligned}$$

LLM: GPT4



Great! Please provide alternative formulation by introducing a new variable z, and enforcing $|x_i|<= z_i$

ChatGPT

To reformulate the Basis Pursuit problem by introducing a new variable z and enforcing $|x_i| \leq z_i$ for each component i of the vector x, we can take advantage of the relationship between the l_1-norm objective and these new variables. This approach aims to directly model the l_1-norm minimization within a linear programming framework by explicitly capturing the absolute value constraints.

The original Basis Pursuit problem is given by:

$$\min_{x \in \mathbb{R}^n} \|x\|_1 \quad \text{subject to} \quad Ax = b,$$

where $\|x\|_1 = \sum_{i=1}^n |x_i|$.

Reformulation Steps:

1. **Introduce New Variables**: For each x_i in the vector x, introduce a corresponding variable z_i intended to capture the absolute value of x_i, i.e., $|x_i| \leq z_i$. The variable z_i is non-negative for all i, since it represents the magnitude of x_i.
2. **Objective Function**: The objective becomes the minimization of the sum of the new variables z, which represent the absolute values of the components of x. Hence, the objective function is $\min \sum_{i=1}^n z_i$.
3. **Constraints**: In addition to the original equality constraint $Ax = b$, we add constraints to ensure $|x_i| \leq z_i$ for all i. This is achieved by adding the following inequalities for each i:
   - $x_i \leq z_i$, to capture the case when x_i is positive or zero.
   - $-x_i \leq z_i$, to capture the case when x_i is negative.

Together, these inequalities ensure that z_i is at least as large as the absolute value of x_i, effectively modeling $|x_i| \leq z_i$.

Linear Programming Formulation:

The Basis Pursuit problem can now be reformulated as:

$$\begin{aligned} \min_{x, z \in \mathbb{R}^n} \quad & \sum_{i=1}^n z_i \\ \text{subject to} \quad & Ax = b, \\ & x_i \leq z_i, \quad \forall i = 1, \ldots, n, \\ & -x_i \leq z_i, \quad \forall i = 1, \ldots, n, \\ & z_i \geq 0, \quad \forall i = 1, \ldots, n. \end{aligned}$$

Problems with absolute values

- data fitting:

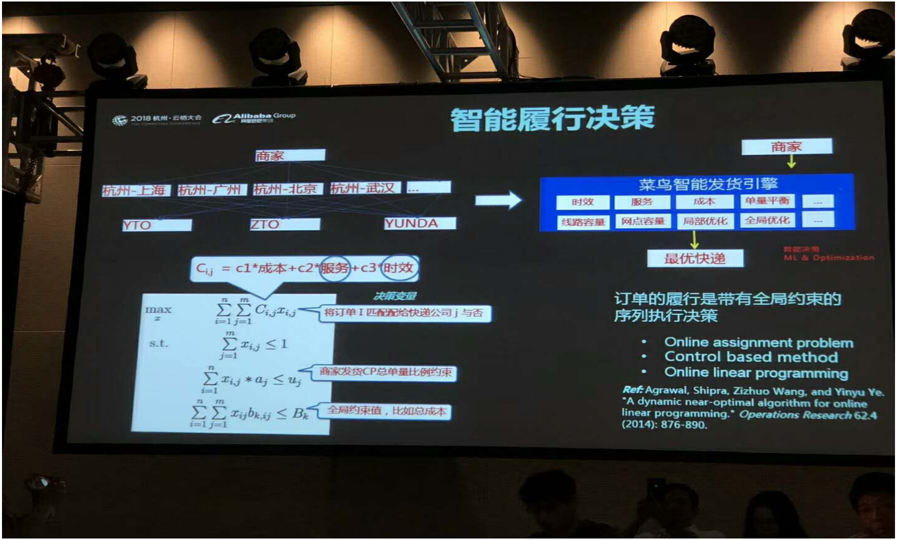
$$\min_x \quad \|Ax - b\|_\infty$$

$$\min_x \quad \|Ax - b\|_1$$

- Compressive sensing

$$\begin{array}{lcl}
\min & \|x\|_1, \text{ s.t. } \quad Ax = b & (LP) \\
\min & \mu\|x\|_1 + \frac{1}{2}\|Ax - b\|^2 & (QP, SOCP) \\
\min & \|Ax - b\|, \text{ s.t. } \quad \|x\|_1 \leq 1 &
\end{array}$$

An example of linear programming: 菜鸟



2018 杭州·云栖大会 THE COMPUTING CONFERENCE

Alibaba Group 阿里巴巴集团

智能履行决策

商家

杭州-上海 杭州-广州 杭州-北京 杭州-武汉 ...

YTO ZTO YUNDA

$C_{i,j}$ = c1*成本+c2*服务+c3*时效

决策变量

$$\max_{x} \sum_{i=1}^{n}\sum_{j=1}^{m} C_{i,j}x_{i,j}$$

将订单 I 匹配配给快递公司 j 与否

$$\text{s.t.} \quad \sum_{j=1}^{m} x_{i,j} \le 1$$

$$\sum_{i=1}^{n} x_{i,j} * a_j \le u_j$$

商家发货CP总单量比例约束

$$\sum_{i=1}^{n}\sum_{j=1}^{m} x_{ij}b_{k,ij} \le B_k$$

全局约束值，比如总成本

商家

菜鸟智能发货引擎

时效 服务 成本 单量平衡 ...

线路容量 网点容量 局部优化 全局优化 ...

最优快递

智能决策
ML & Optimization

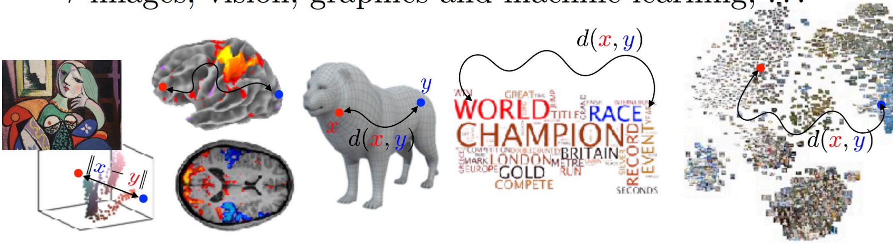
订单的履行是带有全局约束的序列执行决策

- Online assignment problem
- Control based method
- Online linear programming

Ref: Agrawal, Shipra, Zizhuo Wang, and Yinyu Ye. "A dynamic near-optimal algorithm for online linear programming." *Operations Research* 62.4 (2014): 876-890.

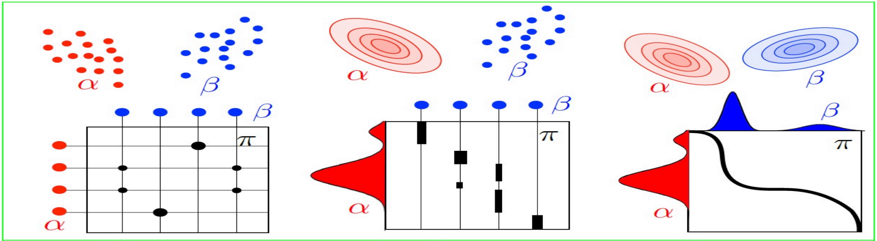
Optimal transport

→ images, vision, graphics and machine learning, … .



Optimal transport: LP

$$\begin{aligned}
\min_{\pi \in \mathbb{R}^{m\times n}} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}\pi_{ij} \\
\text{s.t.} \quad & \sum_{j=1}^{n} \pi_{ij} = \mu_i, \quad \forall i = 1,\dots,m, \\
& \sum_{i=1}^{m} \pi_{ij} = \nu_i, \quad \forall j = 1,\dots,n \\
& \pi \geq 0
\end{aligned}$$



Discrete Semi-discrete Continuous

Weak duality

Suppose

- x is feasible to (P)
- (y, s) is feasible to (D)

Then

$$\begin{aligned} 0 &\leq x^\top s \quad \text{because } x_i s_i \geq 0 \\ &= x^\top (c - A^\top y) \\ &= c^\top x - (Ax)^\top y \\ &= c^\top x - b^\top y \\ &= \text{duality gap} \end{aligned}$$

Key Properties of LP

- Strong duality: If both Primal and Dual are feasible then at the optimum

$$c^\top x = b^\top y \iff x^\top s = 0$$

- complementary slackness: This implies

$$\begin{aligned} x^\top s &= x_1 s_1 + \ldots + x_n s_n = 0 \quad \text{and therefore} \\ x_i s_i &= 0 \end{aligned}$$

complementarity

- Putting together primal feasibility, dual feasibility and complementarity together we get a square system of equations

$$\begin{array}{rcll} Ax & = & b, & x \geq 0, \\ A^\top y + s & = & c, & s \geq 0, \\ x_i s_i & = & 0 & \text{for } i = 1, \ldots, n \end{array}$$

- At least in principle this system determines the primal and dual optimal values

Outline

1. Linear Programming (LP)

2. Semidefinite Programming (SDP)

3. Second Order Cone Programming (SOCP)

Semidefinite Programming (SDP)

- $X \succeq Y$ means that the the symmetric matrix $X - Y$ is positive semidefinite

- X is positive semidefinite

$$a^\top X a \geq 0 \text{ for all vector } a \iff X = B^\top B \iff$$

$$\text{all eigenvalues of } X \text{ is nonnegative}$$

SDP

- For simplicity we deal with single variable SDP:

Primal (P) **Dual (D)**

$$\begin{aligned} \min_{X} \quad & \langle C, X\rangle \\ \text{s.t.} \quad & \langle A_1, X\rangle = b_1 \\ & \dots \\ & \langle A_m, X\rangle = b_m \\ & X \succeq 0 \end{aligned}$$

$$\begin{aligned} \max_{y,S} \quad & b^\top y \\ \text{s.t.} \quad & \sum_i y_i A_i + S = C \\ & S \succeq 0 \end{aligned}$$

- A single variable LP is trivial
- But a single matrix SDP is as general as a multiple matrix

Facts on matrix calcuation

- If $A, B \in \mathbb{R}^{m\times n}$, then $\mathrm{Tr}(AB^\top) = \mathrm{Tr}(B^\top A)$
- If $U, V \in \mathcal{S}^n$ and Q is orthogonal, then $\langle U, V\rangle = \langle Q^\top UQ, Q^\top UQ\rangle$
- If $X \in \mathcal{S}^n$, then $U = Q^\top \Lambda Q$, where $Q^\top Q = I$ and Λ is diagonal.
- Matrix norms: $\|X\|_F = \|\lambda(X)\|_2$, $\|X\|_2 = \|\lambda(X)\|_\infty$, $\lambda(X) = \mathrm{diag}(\Lambda)$
- $X \succeq 0 \iff v^\top Xv \geq$ for all $v \in \mathbb{R}^n \iff \lambda(X) \geq 0 \iff X = B^\top B$
- The dual cone of $\mathcal{S}^n_+$ is $\mathcal{S}^n_+$
- If $X \succeq 0$, then $X_{ii} \geq 0$. If $X_{ii} = 0$, then $X_{ik} = X_{ki} = 0$ for all k.
- If $X \succeq 0$, then $PXP^\top \succeq 0$ for any P of approriate dimensions
- If $X = \begin{pmatrix} X_{11} & X_{12} \\ X_{12}^\top & X_{22} \end{pmatrix} \succeq 0$, then $X_{11} \succeq 0$.
- $X \succeq 0$ iff every principal submatrix is positive semidefinite (psd).

Facts on matrix calcuation

- Let $U = \begin{pmatrix} A & B \\ B^\top & C \end{pmatrix}$ with A and C symmetric and $A \succ 0$. Then

  $$U \succeq 0 \text{ (or } \succ 0) \iff C - B^\top A^{-1} B \succeq 0 \text{ (or } \succ 0).$$

  The matrix $C - B^\top A^{-1} B$ is the **Schur complement** of A in U:

  $$\begin{pmatrix} A & B \\ B^\top & C \end{pmatrix} = \begin{pmatrix} I & 0 \\ B^\top A^{-1} & I \end{pmatrix} \begin{pmatrix} A & 0 \\ 0 & C - B^\top A^{-1} B \end{pmatrix} \begin{pmatrix} I & A^{-1} B \\ 0 & I \end{pmatrix}$$

- If $A \in \mathcal{S}^n$, then $x^\top A x = \langle A, xx^\top \rangle$

- If $A \succ 0$, then $\langle A, B \rangle > 0$ for every nonzero $B \succeq 0$ and $\{B \succeq 0 \mid \langle A, B \rangle \leq \beta\}$ is bounded for $\beta > 0$

- If $A, B \succeq 0$, then $\langle A, B \rangle = 0$ iff $AB = 0$

- $A, B \in \mathcal{S}^n$, then A and B are commute iff AB is symmetric, iff A and B can be simultaneously diagonalized

Eigenvalue optimization

- minimizing the largest eigenvalue $\lambda_{\max}(A_0 + \sum_i x_i A_i)$:

$$\min \quad \lambda_{\max}(A_0 + \sum_i x_i A_i)$$

can be expressed as an SDP

$$\begin{aligned} \min \quad & z \\ \text{s.t.} \quad & zI - \sum_i x_i A_i \succeq A_0 \end{aligned}$$

and its dual is

$$\begin{aligned} \max \quad & \langle A_0, Y \rangle \\ \text{s.t.} \quad & \langle A_i, Y \rangle = k \\ & \langle I, Y \rangle = 1 \\ & Y \succeq 0 \end{aligned}$$

- follows from

$$\lambda_{\max}(A) \le t \Longleftrightarrow A \preceq tI$$

Eigenvalue optimization

- Let $A_i \in \mathbb{R}^{m\times n}$. Minimizing the 2-norm of $A(x) = A_0 + \sum_i x_i A_i$:

$$\min_{x} \quad \|A(x)\|_2$$

can be expressed as an SDP

$$\begin{aligned} \min_{x,t} \quad & t \\ \text{s.t.} \quad & \begin{pmatrix} tI & A(x) \\ A(x)^\top & tI \end{pmatrix} \succeq 0 \end{aligned}$$

- Constraint follows from

$$\begin{aligned} \|A\|_2 \le t \quad & \iff \quad A^\top A \preceq t^2 I, \quad t \ge 0 \\ & \iff \quad \begin{pmatrix} tI & A(x) \\ A(x)^\top & tI \end{pmatrix} \succeq 0 \end{aligned}$$

Quadratically Constrained Quadratic Programming

Consider QCQP

$$\begin{array}{ll} \min & x^\top A_0 x + 2b_0^\top x + c_0 \qquad \text{assume } A_i \in \mathcal{S}^n \\ \text{s.t.} & x^\top A_i x + 2b_i^\top x + c_i \leq 0, \quad i = 1, \ldots, m \end{array}$$

- If $A_0 \succ 0$ and $A_i = B_i^\top B_i$, $i = 1, \ldots, m$, then it is a SOCP
- If $A_i \in \mathcal{S}^n$ but may be indefinite

$$x^\top A_i x + 2b_i^\top x + c_i \quad = \quad \left\langle A_i, xx^\top \right\rangle + 2b_i^\top x + c_i$$

- The original problem is equivalent to

$$\begin{array}{ll} \min & \mathrm{Tr} A_0 X + 2b_0^\top x + c_0 \\ \text{s.t.} & \mathrm{Tr} A_i X + 2b_i^\top x + c_i \leq 0, \quad i = 1, \ldots, m \\ & X = xx^\top \end{array}$$

QCQP

- If $A_i \in \mathcal{S}^n$ but may be indefinite

$$x^\top A_i x + 2b_i^\top x + c_i \quad = \quad \left\langle \begin{pmatrix} A_i & b_i \\ b_i^\top & c_i \end{pmatrix}, \begin{pmatrix} X & x \\ x^\top & 1 \end{pmatrix} \right\rangle := \langle \bar{A}_i, \bar{X} \rangle$$

  $\bar{X} \succeq 0$ is equivalent to $X \succeq xx^\top$

- The SDP relaxation is

$$\begin{array}{ll} \min & \mathrm{Tr}\bar{A}_0\bar{X} \\ \text{s.t.} & \mathrm{Tr}\bar{A}_i\bar{X} \le 0, \quad i = 1, \ldots, m \\ & \bar{X} \succeq 0 \end{array}$$

- Maxcut: $\max\ x^\top W x, \quad$ s.t. $\quad x_i^2 = 1$
- Phase retrieval: $|a_i^\top x| = b_i$, the value of $a_i^\top x$ is complex

Max cut

- For graph (V, E) and weights $w_{ij} = w_{ji} \geq 0$, the maxcut problem is

$$(Q) \quad \max_{x} \ \frac{1}{2} \sum_{i<j} w_{ij}(1 - x_i x_j), \ \text{ s.t. } \ x_i \in \{-1, 1\}$$

- Relaxation:

$$(P) \quad \max_{v_i \in \mathbb{R}^n} \ \frac{1}{2} \sum_{i<j} w_{ij}(1 - v_i^\top v_j), \ \text{ s.t. } \ \|v_i\|_2 = 1$$

- Equivalent SDP of (P):

$$(SDP) \quad \max_{X \in S^n} \ \frac{1}{2} \sum_{i<j} w_{ij}(1 - X_{ij}), \ \text{ s.t. } \ X_{ii} = 1, X \succeq 0$$

Max cut: rounding procedure

Goemans and Williamson's randomized approach

- Solve (SDP) to obtain an optimal solution X. Compute the decomposition $X = V^\top V$, where

$$V = [v_1, v_2, \ldots, v_n]$$

- Generate a vector r uniformly distributed on the unit sphere, i.e., $\|r\|_2 = 1$
- Set

$$x_i = \begin{cases} 1 & v_i^\top r \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Max cut: theoretical results

- Let W be the objective function value of x and $E(W)$ be the expected value. Then

$$E(W) = \frac{1}{\pi} \sum_{i<j} w_{ij} \arccos(v_i^\top v_j)$$

- Goemans and Williamson showed:

$$E(W) \geq \alpha \frac{1}{2} \sum_{i<j} w_{ij} (1 - v_i^\top v_j)$$

where

$$\alpha = \min_{0 \leq \theta \pi} \frac{2}{\pi} \frac{\theta}{1 - \cos\theta} > 0.878$$

- Let $Z^*_{(SDP)}$ and $Z^*_{(Q)}$ be the optimal values of (SDP) and (Q)

$$E(W) \geq \alpha Z^*_{(SDP)} \geq \alpha Z^*_{(Q)}$$

Weak duality in SDP

- Just as in LP
$$\langle X, S\rangle = \langle C, X\rangle - b^\top y$$
- Also if both $X \succeq 0$ and $S \succeq 0$ then
$$\langle X, S\rangle = \mathrm{Tr}(XS^{1/2}S^{1/2}) = \mathrm{Tr}(S^{1/2}XS^{1/2}) \geq 0$$
because $S^{1/2}XS^{1/2} \succeq 0$
- Thus
$$\langle X, S\rangle = \langle C, X\rangle - b^\top y \geq 0$$

Complementarity Slackness Theorem

- $X \succeq 0$ and $S \succeq 0$ and $\langle X, S \rangle = 0$ implies

$$XS = 0$$

- Proof:

$$\langle X, S \rangle = \mathrm{Tr}(XS^{1/2}S^{1/2}) = \mathrm{Tr}(S^{1/2}XS^{1/2})$$

  Thus $\mathrm{Tr}(S^{1/2}XS^{1/2}) = 0$. Since $S^{1/2}XS^{1/2} \succeq 0$, then

$$\begin{aligned} S^{1/2}XS^{1/2} = 0 &\Longrightarrow S^{1/2}X^{1/2}X^{1/2}S^{1/2} = 0 \\ X^{1/2}S^{1/2} = 0 &\Longrightarrow XS = 0 \end{aligned}$$

Equivalent complementarity slackness

- For reasons to become clear later it is better to write complementary slackness conditions as

$$\frac{XS + SX}{2} = 0$$

- It can be shown that if $X \succeq 0$ and $S \succeq 0$, then $XS = 0$ iff

$$XS + SX = 0$$

Constraint Qualifications

- Unlike LP we need some conditions for the optimal values of Primal and Dual SDP to coincide
- Here are two:
  - If there is primal-feasible $X \succ 0$ (i.e. X is positive definite)
  - If there is dual-feasible $S \succ 0$
- When strong duality holds $\langle X, S \rangle = 0$

KKT Condition

- Thus just like LP, the system of equations are

$$\begin{array}{rcll} \langle A_i, X\rangle &=& b_i, & X \succeq 0, \\ \sum_i y_i A_i + S &=& C, & S \succeq 0, \\ X \circ S &=& 0. & \end{array}$$

It gives us a square system.

Outline

1. Linear Programming (LP)

2. Semidefinite Programming (SDP)

3. Second Order Cone Programming (SOCP)

Second Order Cone Programming (SOCP)

- For simplicity we deal with single variable SOCP:

Primal (P)

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & Ax = b \\ & x_{\mathcal{Q}} \succeq 0 \end{aligned}$$

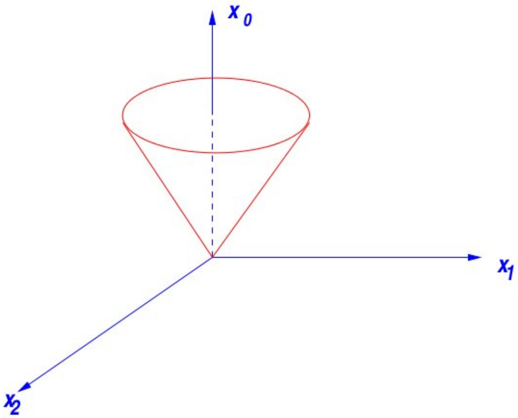
Dual (D)

$$\begin{aligned} \max \quad & b^\top y \\ \text{s.t.} \quad & A^\top y + s = c \\ & s_{\mathcal{Q}} \succeq 0 \end{aligned}$$

- the vectors x, s, c are indexed from zero
- If $z = (z_0, z_1, \ldots, z_n)^\top$ and $\bar{z} = (z_1, \ldots, z_n)^\top$

$$z_{\mathcal{Q}} \geq 0 \Longleftrightarrow z_0 \geq \|\bar{z}\|$$

Illustration of SOC



$$\mathcal{Q} = \{z \mid z_0 \geq \|\bar{z}\|\}$$

Quadratic Programming (QP)

$$\begin{aligned} \min \quad & q(x) = x^\top Q x + a^\top x + \beta \qquad \text{assume} \quad Q \succ 0, Q = Q^\top \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

- $q(x) = \|\bar{u}\|^2 + \beta - \frac{1}{4} a^\top Q^{-1} a$, where $\bar{u} = Q^{1/2} x + \frac{1}{2} Q^{-1/2} a$.
- equivalent SOCP

$$\begin{aligned} \min \quad & u_0 \\ \text{s.t.} \quad & \bar{u} = Q^{1/2} x + \frac{1}{2} Q^{-1/2} a \\ & Ax = b \\ & x \geq 0, \quad (u_0, \bar{u}) \succeq_{\mathcal{Q}} 0 \end{aligned}$$

Robust linear programming

the parameters in LP are often uncertain

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & a_i^\top x \le b_i \end{aligned}$$

There can be uncertainty in c, a_i, b.

two common approaches to handling uncertainty (in a_i, for simplicity)

- deterministic model: constraints must hold for all $a_i \in \mathcal{E}_i$

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & a_i^\top x \le b_i, \ \text{for all } a_i \in \mathcal{E}_i \end{aligned}$$

- stochastic model: a_i is random variable; constraints must hold with probability η

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & \text{prob}(a_i^\top x \le b_i) \ge \eta \end{aligned}$$

deterministic approach via SOCP

- Choose an ellipsoid as $\mathcal{E}_i$:

$$\mathcal{E}_i = \{\bar{a}_i + P_i u \mid \|u\|_2 \le 1\}, \quad \bar{a}_i \in \mathbb{R}^n, P_i \in \mathbb{R}^{n\times n}$$

- Robust LP

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & a_i^\top x \le b_i, \text{ for all } a_i \in \mathcal{E}_i \end{aligned}$$

is equivalent to the SOCP

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & \bar{a}_i^\top x + \|P_i^\top x\|_2 \le b_i \end{aligned}$$

since

$$\sup_{\|u\|_2 \le 1} (\bar{a}_i + P_i u)^\top x = \bar{a}_i^\top x + \|P_i^\top x\|_2$$

stochastic approach via SOCP

- a_i is Gaussian with mean $\bar{a}_i$, covariance Σ_i ($a_i \sim \mathcal{N}(\bar{a}_i, \Sigma_i)$
- $a_i^\top x$ is Gaussian r.v. with mean $\bar{a}_i^\top x$, variance $x^\top \Sigma_i x$; hence

$$\operatorname{prob}(a_i^\top x \le b_i) = \Phi\left(\frac{b_i - \bar{a}_i^\top x}{\|\Sigma^{1/2} x\|_2}\right)$$

where $\Phi(x) = (1/\sqrt{2\pi}) \int_{-\infty}^{x} e^{-t^2/2} dt$ is CDF of $\mathcal{N}(0,1)$

- robust LP

$$\begin{array}{ll} \min & c^\top x \\ \text{s.t.} & \operatorname{prob}(a_i^\top x \le b_i) \ge \eta \end{array}$$

is equivalent to the SOCP

$$\begin{array}{ll} \min & c^\top x \\ \text{s.t.} & \bar{a}_i^\top x + \Phi^{-1}(\eta) \|\Sigma^{1/2} x\|_2 \le b_i \end{array}$$

Weak Duality in SOCP

- The single block SOCP is not as trivial as LP but it still can be solved analytically
- weak duality: Again as in LP and SDP

$$x^\top s = c^\top x - b^\top y = \text{ duality gap}$$

If $x, s \succeq_{\mathcal{Q}} 0$, then

$$\begin{aligned} x^\top s &= x_0 s_0 + \bar{x}^\top \bar{s} \\ &\geq \|\bar{x}\| \cdot \|\bar{s}\| + \bar{x}^\top \bar{s} \quad \text{since } x, s \succeq_{\mathcal{Q}} 0 \\ &\geq |\bar{x}^\top \bar{s}| + \bar{x}^\top \bar{s} \quad \text{Cauchy-Schwartz inequality} \\ &\geq 0 \end{aligned}$$

Complementary Slackness for SOCP

- Given $x \succeq_{\mathcal{Q}} 0$, $s \succeq_{\mathcal{Q}} 0$ and $x^\top s = 0$. Assume $x_0 > 0$ and $s_0 > 0$
- We have

$$(*) \quad x_0^2 \geq \sum_{i=1}^{n} x_i^2$$

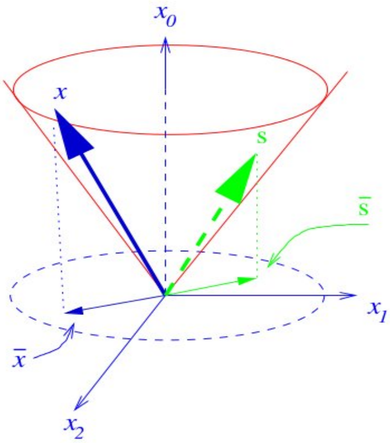
$$(**) \quad s_0^2 \geq \sum_{i=1}^{n} s_i^2 \iff x_0^2 \geq \sum_{i=1}^{n} \frac{s_i^2 x_0^2}{s_0^2}$$

$$(***) \quad x^\top s = 0 \iff -x_0 s_0 = \sum_{i} x_i s_i \iff -2x_0^2 = \sum_{i=1}^{n} \frac{2x_i s_i x_0}{s_0}$$

- Adding (*), (**), (***), we get $0 \geq \sum_{i=1}^{n} \left(x_i + \frac{s_i x_0}{s_0} \right)^2$
- This implies

$$x_i s_0 + x_0 s_i = 0, \text{ for } i = 1, \ldots, n$$

Illustration of SOC



When $x \succeq_Q 0$, $s \succeq_Q 0$ are orthogonal both must be on the boundary in such a way that their projection on the $x_1, \ldots, x_n$ plane is collinear

Strong Duality

- at the optimum

$$c^\top x = b^\top y \Longleftrightarrow x^\top s = 0$$

- Like SDP constraint qualifications are required
- If there is primal-feasible $x \succ_{\mathcal{Q}} 0$
- If there is dual-feasible $s \succ_{\mathcal{Q}} 0$

Complementary Slackness for SOCP

- Thus again we have a square system

$$\begin{aligned} Ax &= b, \quad x \succeq_{\mathcal{Q}} 0, \\ A^{\top}y + s &= c, \quad s \succeq_{\mathcal{Q}} 0, \\ x^{\top}s &= 0 \\ x_0 s_i + s_0 x_i &= 0 \end{aligned}$$

Lecture: Duality of LP, SOCP and SDP

Zaiwen Wen

Beijing International Center For Mathematical Research
Peking University

`http://bicmr.pku.edu.cn/~wenzw/bigdata2019.html`
wenzw@pku.edu.cn

Acknowledgement: this slides is based on Prof. Lieven Vandenberghe's lecture notes

Lagrangian

standard form problem (not necessarily convex)

$$\begin{array}{ll} \min & f_0(x) \\ \text{s.t.} & f_i(x) \leq 0, \quad i = 1, \ldots, m \\ & h_i(x) = 0, \quad i = 1, \ldots, p \end{array}$$

variable $x \in \mathbb{R}^n$, domain $\mathcal{D}$, optimal value p^*.
Lagrangian: $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$, with $\mathbf{dom}\mathcal{L} = \mathcal{D} \times \mathbb{R}^m \times \mathbb{R}^p$

$$\mathcal{L}(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x)$$

- weighted sum of objective and constraint functions
- λ_i is the Lagrange multiplier associated with $f_i(x) \leq 0$
- ν_i is the Lagrange multiplier associated with $h_i(x) = 0$

Lagrange dual function

Lagrange dual function: $g : \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$

$$\begin{aligned} g(\lambda, \nu) &= \inf_{x \in \mathcal{D}} \quad \mathcal{L}(x, \lambda, \nu) \\ &= \inf_{x \in \mathcal{D}} \quad \left(f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x) \right) \end{aligned}$$

g is concave, can be $-\infty$ for some λ, ν

- lower bound property: if $\lambda \geq 0$, then $g(\lambda, \nu) \leq p^*$
- Proof: If $\tilde{x}$ is feasible and $\lambda \geq 0$, then

$$f_0(\tilde{x}) \geq \mathcal{L}(\tilde{x}, \lambda, \nu) \geq \inf_{x \in \mathcal{D}} \mathcal{L}(x, \lambda, \nu) = g(\lambda, \nu).$$

minimizing over all feasible $\tilde{x}$ gives $p^* \geq g(\lambda, \nu)$.

The dual problem

Lagrange dual problem

$$\begin{array}{ll} \max & g(\lambda, \nu) \\ \text{s.t.} & \lambda \geq 0 \end{array}$$

- finds best lower bound on p^*, obtained from Lagrange dual function
- a convex optimization problem; optimal value denoted d^*
- λ, ν are dual feasible if $\lambda \geq 0$, $(\lambda, \nu) \in \mathbf{dom} g$
- often simplified by making implicit constraint $(\lambda, \nu) \in \mathbf{dom} g$ explicit

Standard form LP

$$\textbf{(P)}\quad \begin{array}{ll} \min & c^\top x \\ \text{s.t.} & Ax = b \\ & x \geq 0 \end{array} \qquad \textbf{(D)}\quad \begin{array}{ll} \max & b^\top y \\ \text{s.t.} & A^\top y + s = c \\ & s \geq 0 \end{array}$$

- Lagrangian is

$$\mathcal{L}(x, \lambda, \nu) \quad = \quad c^\top x + \nu^\top (Ax - b) - \lambda^\top x = -b^\top \nu + (c + A^\top \nu - \lambda)^\top x$$

- $\mathcal{L}$ is affine in x, hence

$$g(\lambda, \nu) = \inf_x \mathcal{L}(x, \lambda, \nu) = \begin{cases} -b^\top \nu, & A^\top \nu - \lambda + c = 0 \\ -\infty & \text{otherwise} \end{cases}$$

- lower bound property: $p^* \geq -b^\top \nu$ if $A^\top \nu + c \geq 0$
- Check the dual of **(D)**

Inequality form LP

$$\begin{aligned} \min \quad & c^\top x \\ \text{s.t.} \quad & Ax \le b \end{aligned}$$

- Lagrangian is

$$\mathcal{L}(x,\lambda,\nu) \quad = \quad c^\top x + \nu^\top (Ax - b) = -b^\top \nu + (c + A^\top v)^\top x$$

- $\mathcal{L}$ is affine in x, hence

$$g(\lambda,\nu) = \inf_x \mathcal{L}(x,\lambda,\nu) = \begin{cases} -b^\top \nu, & A^\top \nu + c = 0 \\ -\infty & \text{otherwise} \end{cases}$$

- Dual problem

$$\begin{aligned} \max \quad & b^\top v \\ \text{s.t.} \quad & A^\top v = c, \quad v \le 0 \end{aligned}$$

Equality constrained norm minimization

$$\begin{array}{ll} \min & \|x\| \\ \text{s.t.} & Ax = b \end{array} \qquad \begin{array}{ll} \max & b^\top v \\ \text{s.t.} & \|A^\top v\|_* \leq 1 \end{array}$$

Dual function

$$g(\nu) = \inf_x (\|x\| - \nu^\top Ax + b^\top \nu) = \begin{cases} b^\top \nu & \|A^\top \nu\|_* \leq 1 \\ -\infty & \text{otherwise,} \end{cases}$$

where $\|v\|_* = \sup_{\|u\| \leq 1} u^\top v$ is the dual norm of $\| \cdot \|$

Proof: follows from $\inf_x(\|x\| - y^\top x) = 0$ if $\|y\|_* \leq 1$, $-\infty$ otherwise

- if $\|y\|_* \leq 1$, then $\|x\| - y^\top x \geq 0$ for all x, with equality if $x = 0$
- if $\|y\|_* > 1$, then choose $x = tu$, where $\|u\| \leq 1$, $u^\top y = \|y\|_* > 1$:

$$\|x\| - y^\top x = t(\|u\| - \|y\|_*) \to -\infty \quad \text{as } t \to \infty$$

for all x, with equality if $x = 0$

LP with box constraints

$$\begin{array}{llcll}
\min & c^\top x & \qquad & \max & -b^\top \nu - 1^\top \lambda_1 - 1^\top \lambda_2 \\
\text{s.t.} & Ax = b & & \text{s.t.} & c + A^\top \nu + \lambda_1 - \lambda_2 = 0 \\
 & -1 \le x \le 1 & & & \lambda_1 \ge 0, \lambda_2 \ge 0
\end{array}$$

reformulation with box constraints made implicit

$$\begin{array}{ll}
\min & f_0(x) = \begin{cases} c^\top x & -1 \le x \le 1 \\ \infty & \text{otherwise} \end{cases} \\
\text{s.t.} & Ax = b
\end{array}$$

Dual function

$$g(\nu) = \inf_{-1 \le x \le 1} (c^\top x + \nu^\top (Ax - b)) = -b^\top \nu - \|A^\top \nu + c\|_1$$

Dual problem:

$$\max_{\nu} \quad -b^\top \nu - \|A^\top \nu + c\|_1$$

Lagrange dual and conjugate function

$$\begin{array}{ll} \min & f_0(x) \\ \text{s.t.} & Ax \leq b \\ & Cx = d \end{array}$$

- dual function

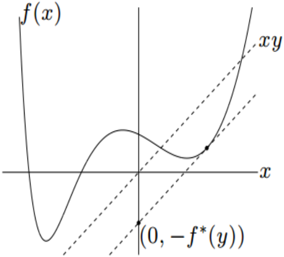
$$\begin{aligned} g(\lambda, \nu) &= \inf_{x \in \mathbf{dom} f_0} \left(f_0(x) + (A^\top \lambda + C^\top \nu)^\top x - b^\top \lambda - d^\top \nu \right) \\ &= -f^*(-A^\top \lambda - C^\top \nu) - b^\top \lambda - d^\top \nu \end{aligned}$$

- recall definition of conjugate $f^*(y) = \sup_{x \in \mathbf{dom} f}(y^\top x - f(x))$
- simplifies derivation of dual if conjugate of f_0 is known

Conjugate function

the **conjugate** of a function f is

$$f^*(y) = \sup_{x \in \mathbf{dom}\, f} (y^T x - f(x))$$



f^* is closed and

convex even if f is not **Fenchel's inequality**

$$f(x) + f(y) \geq x^T y \quad \forall x, y$$

(extends inequality $x^T x/2 + y^T y/2 \geq x^T y$ to non-quadratic convex f)

Quadratic function

$$f(x) = \frac{1}{2}x^T A x + b^T x + c$$

strictly convex case ($A \succ 0$)

$$f^*(y) = \frac{1}{2}(y - b)^T A^{-1}(y - b) - c$$

general convex case ($A \succeq 0$)

$$f^*(y) = \frac{1}{2}(y - b)^T A^{\dagger}(y - b) - c, \quad \mathbf{dom}\, f^* = range(A) + b$$

Negative entropy and negative logarithm

negative entropy

$$f(x) = \sum_{i=1}^{n} x_i \log x_i \qquad f^*(y) = \sum_{i=1}^{n} e^{y_i - 1}$$

negative logarithm

$$f(x) = -\sum_{i=1}^{n} \log x_i \qquad f^*(y) = -\sum_{i=1}^{n} \log(-y_i) - n$$

matrix logarithm

$$f(x) = -\log\det X \quad (\mathbf{dom}\, f = \mathbf{S}^n_{++}) \quad f^*(Y) = -\log\det(-Y) - n$$

Indicator function and norm

indicator of convex set C: **conjugate** is support function of C

$$f(x) = \begin{cases} 0, & x \in C \\ +\infty, & x \notin C \end{cases} \qquad f^*(y) = \sup_{x \in C} y^T x$$

norm: conjugate is indicator of unit dual norm ball

$$f(x) = ||x|| \qquad f^*(y) = \begin{cases} 0, & ||y||_* \leq 1 \\ +\infty, & ||y||_* > 1 \end{cases}$$

(see next page)

proof: recall the definition of dual norm:

$$||y||_* = \sup_{||x|| \leq 1} x^T y$$

to evaluate $f^*(y) = \sup_x(y^T x - ||x||)$ we distinguish two cases

- if $||y||_* \leq 1$, then (by definition of dual norm)

  $$y^T x \leq ||x|| \quad \forall x$$

  and equality holds if $x = 0$; therefore $\sup_x(y^T x - ||x||) = 0$
- if $||y||_* > 1$, there exists an x with $||x|| \leq 1, x^T y > 1$; then

  $$f^*(y) \geq y^T(tx) - ||tx|| = t(y^T x - ||x||)$$

  and $r.h.s.$ goes to infinity if $t \to \infty$

The second conjugate

$$f^{**}(x) = \sup_{y \in \mathbf{dom}\, f^*} (x^T y - f^*(y))$$

- $f^{**}(x)$ is closed and convex
- from Fenchel's inequality $x^T y - f^*(y) \leq f(x)$ for all y and x):

  $$f^{**} \leq f(x) \quad \forall x$$

  equivalently, epi $f \subseteq$ epi f^{**} (for any f)
- if f is closed and convex, then

  $$f^{**}(x) = f(x) \quad \forall x$$

  equivalently, epi $f =$ epi f^{**} (if f is closed convex); proof on next page

Conjugates and subgradients

if f is closed and convex, then

$$y \in \partial f(x) \quad \Leftrightarrow \quad x \in \partial f^*(x) \quad \Leftrightarrow \quad x^T y = f(x) + f^*(y)$$

proof: if $y \in \partial f(x)$, then $f^*(y) = \sup_u (y^T u - f(u)) = y^T x - f(x)$

$$\begin{aligned} f^*(v) &= \sup_u (v^T u - f(u)) \\ &\geq v^T x - f(x) \\ &= x^T (v - y) - f(x) + y^T x \\ &= f^*(y) + x^T (v - y) \end{aligned} \qquad (1)$$

for all v; therefore, x is a subgradient of f^* at y $(x \in \partial f^*(y))$
reverse implication $x \in \partial f^*(y) \Rightarrow y \in \partial f(x)$ follows from $f^{**} = f$

Some calculus rules

separable sum

$$f(x_1, x_2) = g(x_1) + h(x_2) \qquad f^*(y_1, y_2) = g^*(y_1) + h^*(y_2)$$

scalar multiplication: (for $\alpha > 0$)

$$f(x) = \alpha g(x) \qquad f^*(y) = \alpha g^*(y/\alpha)$$

addition to affine function

$$f(x) = g(x) + a^T x + b \qquad f^*(y) = g^*(y - a) - b$$

infimal convolution

$$f(x) = \inf_{u+v=x} (g(u) + h(v)) \qquad f^*(y) = g^*(y) + h^*(y)$$

Duality and problem reformulations

- equivalent formulations of a problem can lead to very different duals
- reformulating the primal problem can be useful when the dual is difficult to derive, or uninteresting

Common reformulations

- introduce new variables and equality constraints
- make explicit constraints implicit or vice-versa
- transform objective or constraint functions
  e.g., replace $f_0(x)$ by $\phi(f_0(x))$ with ϕ convex, increasing

Introducing new variables and equality constraints

$$\min \quad f_0(Ax+b)$$

- dual function is constant: $g = \inf_x \mathcal{L}(x) = \inf_x f_0(Ax+b) = p^*$
- we have strong duality, but dual is quite useless

reformulated problem and its dual

$$\textbf{(P)} \quad \begin{array}{ll} \min & f_0(y) \\ \text{s.t.} & Ax+b=y \end{array} \qquad \textbf{(D)} \quad \begin{array}{ll} \max & b^\top y - f_0^*(\nu) \\ \text{s.t.} & A^\top \nu = 0 \end{array}$$

dual functions follows from

$$\begin{aligned} g(\nu) &= \inf_{x,y} \quad f_0(y) - \nu^\top y + \nu^\top Ax + b^\top \nu \\ &= \begin{cases} -f_0^*(\nu) + b^\top \nu, & A^\top \nu = 0 \\ -\infty & \text{otherwise} \end{cases} \end{aligned}$$

Norm approximation problem

$$\min \quad \|Ax-b\| \Longleftrightarrow \begin{array}{ll} \min & \|y\| \\ \text{s.t.} & Ax-b=y \end{array}$$

can look up conjugate of $\|\cdot\|$, or derive dual directly

$$\begin{aligned} g(\nu) &= \inf_{x,y} \quad \|y\| + \nu^\top y - \nu^\top Ax + b^\top \nu \\ &= \begin{cases} b^\top \nu + \inf_y \quad \|y\| + \nu^\top y, & A^\top \nu = 0 \\ -\infty, & \text{otherwise} \end{cases} \\ &= \begin{cases} b^\top \nu, \quad A^\top \nu = 0, \quad \|v\|_* \leq 1 \\ -\infty, \quad \text{otherwise} \end{cases} \end{aligned}$$

Dual problem is

$$\begin{array}{ll} \max & b^\top \nu \\ \text{s.t.} & A^\top \nu = 0, \quad \|v\|_* \leq 1 \end{array}$$

Weak and strong duality

weak duality: $d^* \leq p^*$

- always holds (for convex and nonconvex problems)
- can be used to find nontrivial lower bounds for difficult problems
  for example, solving the maxcut SDP

strong duality: $d^* = p^*$

- does not hold in general
- (usually) holds for convex problems
- conditions that guarantee strong duality in convex problems are called **constraint qualifications**

Slater's constraint qualification

strong duality holds for a convex problem

$$\begin{aligned} \min \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \le 0, \quad i = 1, \ldots, m \\ & Ax = b \end{aligned}$$

If it is strictly feasible, i.e.,

$$\exists x \in \mathrm{int}\mathcal{D} : \quad f_i(x) < 0, \quad i = 1, \ldots, m, \quad Ax = b$$

- also guarantees that the dual optimum is attained (if $p^* > -\infty$)
- can be sharpened: e.g., can replace $\mathrm{int}\mathcal{D}$ with $\mathrm{relint}\mathcal{D}$ (interior relative to affine hull); linear inequalities do not need to hold with strict inequality, . . .
- there exist many other types of constraint qualifications

Complementary slackness

assume strong duality holds, x^* is primal optimal, (λ^*, ν^*) is dual feasible

$$\begin{aligned} f_0(x^*) = g(\lambda^*, \nu^*) &= \inf_{x \in \mathcal{D}} \left(f_0(x) + \sum_{i=1}^{m} \lambda_i^* f_i(x) + \sum_{i=1}^{p} \nu_i^* h_i(x) \right) \\ &\leq f_0(x^*) + \sum_{i=1}^{m} \lambda_i^* f_i(x^*) + \sum_{i=1}^{p} \nu_i^* h_i(x^*) \\ &\leq f_0(x^*) \end{aligned}$$

hence, the two inequalities hold with equality

- x^* minimizes $\mathcal{L}(x, \lambda^*, \nu^*)$
- $\lambda_i^* f_i(x^*) = 0$ for $i = 1, \ldots, m$ (known as complementary slackness):

$$\lambda_i^* > 0 \Longrightarrow f_i(x^*) = 0, \quad f_i(x^*) < 0 \Longrightarrow \lambda_i^* = 0$$

Karush-Kuhn-Tucker (KKT) conditions

the following four conditions are called KKT conditions (for a problem with differentiable f_i, h_i):

1. primal constraints: $f_i(x) \leq 0, i = 1, \ldots, m, \quad h_i(x) = 0, i = 1, \ldots, p$
2. dual constraints: $\lambda \geq 0$
3. complementary slackness: $\lambda_i f_i(x) = 0,\ i = 1, \ldots, m$
4. gradient of Lagrangian with respect to x vanishes:

$$\nabla f_0(x) + \sum_{i=1}^{m} \lambda_i \nabla f_i(x) + \sum_{i=1}^{p} \nu_i \nabla h_i(x) = 0$$

If strong duality holds and x, λ, ν are optimal, then they must satisfy the KKT conditions

KKT conditions for convex problem

If $\tilde{x}, \tilde{\lambda}, \tilde{\nu}$ satisfy KKT for a convex problem, then they are optimal

- from complementary slackness: $f_0(\tilde{x}) = \mathcal{L}(\tilde{x}, \tilde{\lambda}, \tilde{\nu})$
- from 4th condition (and convexity): $g(\tilde{\lambda}, \tilde{\nu}) = \mathcal{L}(\tilde{x}, \tilde{\lambda}, \tilde{\nu})$

Hence, $f_0(\tilde{x}) = g(\tilde{\lambda}, \tilde{\nu})$

if **Slater's condition** is satisfied:
x is optimal if and only if there exist λ, ν that satisfy KKT conditions

- recall that Slater implies strong duality, and dual optimum is attained
- generalizes optimality condition $\nabla f_0(x) = 0$ for unconstrained problem

LP Duality

Strong duality: If a LP has an optimal solution, so does its dual, and their objective fun. are equal.

primal dual	finite	unbounded	infeasible
finite	√	×	×
unbounded	×	×	√
infeasible	×	√	√

- If $p^* = -\infty$, then $d^* \leq p^* = -\infty$, hence dual is infeasible
- If $d^* = +\infty$, then $+\infty = d^* \leq p^*$, hence primal is infeasible
-

$$\begin{array}{lllll} \min & x_1 + 2x_2 & \qquad & \max & p_1 + 3p_2 \\ \text{s.t.} & x_1 + x_2 = 1 & & \text{s.t.} & p_1 + 2p_2 = 1 \\ & 2x_1 + 2x_2 = 3 & & & p_1 + 2p_2 = 2 \end{array}$$

Problems with generalized inequalities

$$\begin{array}{ll} \min & f_0(x) \\ \text{s.t.} & f_i(x) \preceq_{\mathsf{K}_i} 0, \quad i = 1, \dots, m \\ & h_i(x) = 0, \quad i = 1, \dots, p \end{array}$$

- $f_i(x) \preceq \mathsf{K}_i$ means $-f_i(x) \in \mathsf{K}_i$.
- Lagrangian: $\langle \cdot, \cdot \rangle_{\mathsf{K}_i}$ inner product in K_i

$$\mathcal{L}(x, \lambda_1, \dots, \lambda_m, \nu) = f_0(x) + \sum_{i=1}^{m} \langle \lambda_i, f_i(x) \rangle_{\mathsf{K}_i} + \sum_{i=1}^{p} \nu_i h_i(x)$$

- dual function

$$g(\lambda_1, \dots, \lambda_m, \nu) = \inf_{x \in \mathcal{D}} \quad \mathcal{L}(x, \lambda_1, \dots, \lambda_m, \nu)$$

lower bound property: if $\lambda_i \in \mathsf{K}_i^*$, then $g(\lambda_1, \dots, \lambda_m, \nu) \le p^*$
proof: If $\tilde{x}$ is feasible and $\lambda \succeq_{\mathsf{K}_i^*} 0$, then

$$\begin{aligned} f_0(\tilde{x}) &\ge f_0(\tilde{x}) + \sum_{i=1}^{m} \langle \lambda_i, f_i(\tilde{x}) \rangle_{\mathsf{K}_i} + \sum_{i=1}^{p} \nu_i h_i(\tilde{x}) \\ &\ge \inf_{x \in \mathcal{D}} \quad \mathcal{L}(x, \lambda_1, \dots, \lambda_m, \nu) \\ &= g(\lambda_1, \dots, \lambda_m, \nu) \end{aligned}$$

minimizing over all feasible $\tilde{x}$ gives $g(\lambda_1, \dots, \lambda_m, \nu) \le p^*$.
Dual problem

$$\begin{aligned} \max \quad & g(\lambda_1, \dots, \lambda_m, \nu) \\ \text{s.t.} \quad & \lambda_i \succeq_{\mathsf{K}_i^*} 0, \quad i = 1, \dots, m \end{aligned}$$

- weak duality: $p^* \ge d^*$
- strong duality: $p^* = d^*$ for convex problem with constraint qualification (for example, Slater's: primal problem is strictly feasible)

Semidefinite program

primal SDP ($A_i, C \in \mathbb{S}^n$)

$$\begin{array}{ll} \min & b^T y \\ \text{s.t.} & y_1 A_1 + \cdots + y_m A_m \preceq C \end{array}$$

- Lagrange multiplier is matrix $Z \in \mathbb{S}^n$
- Lagrangian $L(y, Z) = b^T y + \operatorname{tr}(Z(y_1 A_1 + \cdots + y_m A_m - C))$
- dual function

$$g(Z) = \inf_y L(y, Z) = \begin{cases} -\operatorname{tr}(CZ) & \operatorname{tr}(A_i Z) + b_i = 0, \quad i = 1, \ldots, m \\ -\infty & \text{otherwise} \end{cases}$$

dual SDP

$$\begin{array}{ll} \max & -\operatorname{tr}(CZ) \\ \text{s.t.} & Z \succeq 0, \operatorname{tr}(A_i Z) + b_i = 0, \quad i = 1, \ldots, m \end{array}$$

$p^* = d^*$ if primal SDP is strictly feasible ($\exists y$ with $y_1 A_1 + \cdots + y_m A_m \prec C$)

SDP Relaxtion of Maxcut

$$\begin{array}{ll} \min & x^\top W x \\ \text{s.t.} & x_i^2 = 1 \end{array} \Longrightarrow \begin{array}{ll} \max & -1^\top \nu \\ \text{s.t.} & W + \mathrm{diag}(\nu) \succeq 0 \end{array} \Longleftrightarrow \begin{array}{ll} \min & \mathrm{Tr}(WX) \\ \text{s.t.} & X_{ii} = 1 \\ & X \succeq 0 \end{array}$$

- a nonconvex problem; feasible set contains 2n discrete points
- interpretation: partition $\{1, \ldots, n\}$ in two sets; W_{ij} is cost of assigning i, j to the same set;; $-W_{ij}$ is cost of assigning to different sets

dual function

$$g(\nu) = \inf_x \left(x^\top W x + \sum_i \nu_i (x_i^2 - 1) \right) \quad = \quad \inf_x \; x^\top (W + \mathrm{diag}(\nu)) x - 1^\top \nu$$

$$= \begin{cases} -1^\top \nu & W + \mathrm{diag}(\nu) \succeq 0 \\ -\infty & \text{otherwise} \end{cases}$$

SOCP/SDP Duality

(P) $\min \quad c^\top x$

s.t. $Ax = b, x_Q \succeq 0$

(D) $\max \quad b^\top y$

s.t. $A^\top y + s = c, s_Q \succeq 0$

(P) $\min \quad \langle C, X \rangle$

s.t. $\langle A_1, X \rangle = b_1$

$\dots$

$\langle A_m, X \rangle = b_m$

$X \succeq 0$

(D) $\max \quad b^\top y$

s.t. $\sum_i y_i A_i + S = C$

$S \succeq 0$

Strong duality

- If $p^* > -\infty$, (P) is **strictly** feasible, then (D) is feasible and $p^* = d^*$
- If $d^* < +\infty$, (D) is **strictly** feasible, then (P) is feasible and $p^* = d^*$
- If (P) and (D) has **strictly** feasible solutions, then both have optimal solutions.

Failure of SOCP Duality

$$
\begin{array}{llcll}
\inf & (1,-1,0)x & \qquad & \sup & y \\
\text{s.t.} & (0,0,1)x = 1 & & \text{s.t.} & (0,0,1)^\top y + z = (1,-1,0)^\top \\
 & x_{\mathcal{Q}} \succeq 0 & & & z_{\mathcal{Q}} \succeq 0
\end{array}
$$

- primal: $\min\ x_0 - x_1,$ s.t. $x_0 \geq \sqrt{x_1^2 + 1}$; It holds $x_0 - x_1 > 0$ and $x_0 - x_1 \to 0$ if $x_0 = \sqrt{x_1^2 + 1} \to \infty$. Hence, $p^* = 0$, no finite solution
- dual: $\sup\ y$ s.t. $1 \geq \sqrt{1 + y^2}$. Hence, $y = 0$

$p^* = d^*$ but primal is not attainable.

Failure of SDP Duality

Consider

$$\begin{aligned} \min \quad & \left\langle \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, X \right\rangle \\ \text{s.t.} \quad & \left\langle \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, X \right\rangle = 0 \\ & \left\langle \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 2 \end{pmatrix}, X \right\rangle = 2 \\ & X \succeq 0 \end{aligned}$$

$$\begin{aligned} \max \quad & 2y_2 \\ \text{s.t.} \quad & \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} y_1 + \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 2 \end{pmatrix} y_2 \preceq \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{aligned}$$

- primal: $X^* = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$, $p^* = 1$
- dual: $y^* = (0, 0)$. Hence, $d^* = 0$

Both problems have finite optimal values, but $p^* \neq d^*$

Lecture: Algorithms for LP, SOCP and SDP

Zaiwen Wen

Beijing International Center For Mathematical Research
Peking University

`http://bicmr.pku.edu.cn/~wenzw/bigdata2023.html`
wenzw@pku.edu.cn

Acknowledgement: this slides is based on chapters 13 and 14 of “Numerical Optimization”,
Jorge Nocedal and Stephen Wright, Springer
some parts are based on Prof. Farid Alizadeh lecture notes

Outline

1. Properties of LP
2. Primal Simplex method
3. Dual Simplex method
4. Interior Point method

Standard form LP

$$\textbf{(P)}\quad \begin{array}{ll} \min & c^\top x \\ \text{s.t.} & Ax = b \\ & x \geq 0 \end{array} \qquad \textbf{(D)}\quad \begin{array}{ll} \max & b^\top y \\ \text{s.t.} & A^\top y + s = c \\ & s \geq 0 \end{array}$$

- KKT condition

$$\begin{aligned} Ax &= b, & x &\geq 0 \\ A^\top y + s &= c, & s &\geq 0 \\ x_i s_i &= 0 & &\text{for } i = 1, \ldots, n \end{aligned}$$

- **Strong duality**: If a LP has an optimal solution, so does its dual, and their objective fun. are equal.

primal dual	finite	unbounded	infeasible
finite	$\surd$	$\times$	$\times$
unbounded	$\times$	$\times$	$\surd$
infeasible	$\times$	$\surd$	$\surd$

Geometry of the feasible set

- Assume that $A \in \mathbb{R}^{m\times n}$ has **full row rank.** Let A_i be the ith column of A:
$$A = \begin{pmatrix} A_1 & A_2 & \dots & A_n \end{pmatrix}$$
- A vector x is a **basic feasible solution (BFS)** if x is feasible and there exists a subset $\mathcal{B} \subset \{1, 2, \dots, n\}$ such that
  - $\mathcal{B}$ contains exactly m indices
  - $i \notin \mathcal{B} \Longrightarrow x_i = 0$
  - The $m \times m$ submatrix $B = [A_i]_{i\in\mathcal{B}}$ is nonsingular

  $\mathcal{B}$ is called a basis and B is called the basis matrix

Properties:

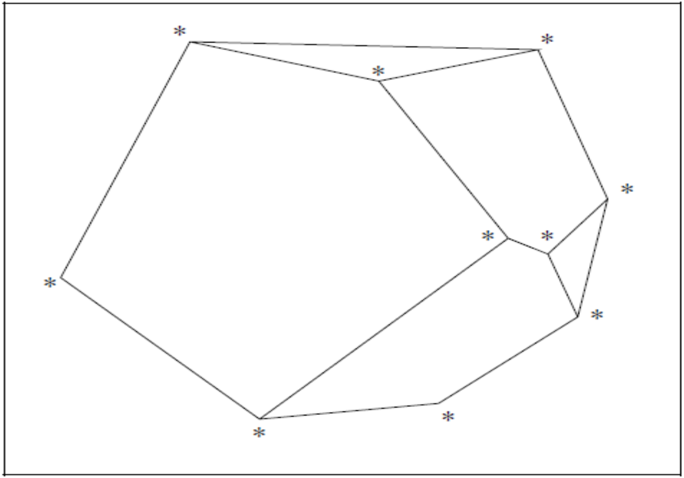
- If (P) has a nonempty feasible region, then there is at least one basic feasible point;
- If (P) has solutions, then at least one such solution is a basic optimal point.
- If (P) is feasible and bounded, then it has an optimal solution.

If (P) has a nonempty feasible region, then there is at least one BFS;

- Choose a feasible x with the minimal number (p) of nonzero x_i: $\sum_{i=1}^{p} A_i x_i = b$
- Suppose that $A_1, \ldots, A_p$ are linearly dependent $A_p = \sum_{i=1}^{p-1} z_i A_i$. Let $x(\epsilon) = x + \epsilon(z_1, \ldots, z_{p-1}, -1, 0, \ldots, 0)^\top = x + \epsilon z$. Then $Ax(\epsilon) = b$, $x_i(\epsilon) > 0$, $i = 1, \ldots, p$, for ϵ sufficiently small. There exists $\bar{\epsilon}$ such that $x_i(\bar{\epsilon}) = 0$ for some $i = 1, \ldots, p$. Contradiction to the choice of x.
- If $p = m$, done. Otherwise, choose $m - p$ columns from among $A_{p+1}, \ldots, A_n$ to build up a set set of m linearly independent vectors.

Polyhedra, extreme points, vertex, BFS

- A **Polyhedra** is a set that can be described in the form $\{x \in \mathbb{R}^n \mid Ax \geq b\}$
- Let P be a polyhedra. A vector $x \in P$ is an **extreme point** if we cannot find two vectors $y, z \in P$ (both different from x) such that $x = \lambda y + (1 - \lambda)z$ for $\lambda \in [0, 1]$
- Let P be a polyhedra. A vector $x \in P$ is a **vertex** if there exists some c such that $c^\top x < c^\top y$ for all $y \in P$ and $y \neq x$
- Let P be a nonempty polyhedra. Let $x \in P$. The following statements are equivalent: (i) x is vertex; (ii) x is an extreme point; (iii) x is a BFS
- A basis $\mathcal{B}$ is said to be **degenerate** if $x_i = 0$ for some $i \in \mathcal{B}$, where x is the BFS corresponding to $\mathcal{B}$. A linear program (P) is said to be degenerate if it has at least one degenerate basis.



Vertices of a three-dimensional polyhedron (indicated by $*$)

Outline

1. Properties of LP
2. Primal Simplex method
3. Dual Simplex method
4. Interior Point method

The Simplex Method For LP

Basic Principle

Move from a BFS to its adjacent BFS unitil convergence (either optimal or unbounded)

- Let x be a BFS and $\mathcal{B}$ be the corresponding basis
- Let $\mathcal{N} = \{1, 2, \ldots, n\} \backslash \mathcal{B}$, $N = [A_i]_{i \in \mathcal{N}}$, $x_B = [x_i]_{i \in \mathcal{B}}$ and $x_N = [x_i]_{i \in \mathcal{N}}$
- Since x is a BFS, then $x_N = 0$ and $Ax = Bx_B + Nx_N = b$:

$$x_B = B^{-1}b$$

- Find exactly one $q \in \mathcal{N}$ and exactly one $p \in \mathcal{B}$ such that

$$\mathcal{B}^+ = \{q\} \cup (\mathcal{B} \backslash \{p\})$$

Finding $q \in \mathcal{N}$ to enter the basis

Let x^+ be the new BFS:

$$x^+ = \begin{pmatrix} x^+_{\mathcal{B}} \\ x^+_{\mathcal{N}} \end{pmatrix}, \quad Ax^+ = b \Longrightarrow x^+_{\mathcal{B}} = B^{-1}b - B^{-1}Nx^+_{\mathcal{N}}$$

The cost at x^+ is

$$\begin{aligned} c^\top x^+ &= c_B^\top x^+_{\mathcal{B}} + c_N^\top x^+_{\mathcal{N}} \\ &= c_B^\top B^{-1}b - c_B^\top B^{-1}Nx^+_{\mathcal{N}} + c_N^\top x^+_{\mathcal{N}} \\ &= c^\top x + (c_N^\top - c_B^\top B^{-1}N)x^+_{\mathcal{N}} \\ &= c^\top x + \sum_{j\in\mathcal{N}} (\underbrace{c_j - c_B^\top B^{-1}A_j}_{s_j})x^+_j \end{aligned}$$

- s_j is also called **reduced cost.** It is actually the dual slackness
- If $s_j \geq 0$, $\forall j \in \mathcal{N}$, then x is optimal as $c^\top x^+ \geq c^\top x$
- Otherwise, find q such that $s_q < 0$. Then $c^\top x^+ = c^\top x + s_q x^+_q \leq c^\top x$

Finding $p \in \mathcal{B}$ to exit the basis

What is x^+: select $q \in \mathcal{N}$ and $p \in \mathcal{B}$ such that

$$x_{\mathcal{B}}^+ = B^{-1}b - B^{-1}A_q x_q^+, \quad x_q^+ \geq 0, x_p^+ = 0, x_j^+ = 0, j \in \mathcal{N} \setminus \{q\}$$

Let $u = B^{-1}A_q$. Then $x_{\mathcal{B}}^+ = x_{\mathcal{B}} - u x_q^+$

- If $u \leq 0$, then $c^\top x^+ = c^\top x + s_q x_q^+ \to -\infty$ as $x_q^+ \to +\infty$ and x^+ is feasible. **(P) is unbounded**
- If $\exists u_k > 0$, then find x_q^+ and p such that

  $$x_{\mathcal{B}}^+ = x_{\mathcal{B}} - u x_q^+ \geq 0, \quad x_p^+ = 0$$

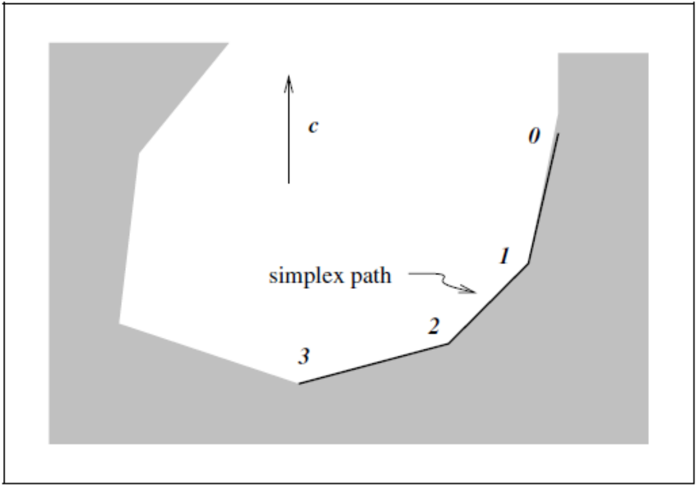
  Let p be the index corresponding to

  $$x_q^+ = \min_{i=1,\dots,m | u_i > 0} \frac{x_{\mathcal{B}(i)}}{u_i}$$

An iteration of the simplex method

Typically, we start from a BFS x and its associate basis $\mathcal{B}$ such that $x_B = B^{-1}b$ and $x_N = 0$.

- Solve $y^\top = c_B^\top B^{-1}$ and then the reduced costs $s_N = c_N - N^\top y$
- If $s_N \geq 0$, x is optimal and stop; Else, choose $q \in \mathcal{N}$ with $s_q < 0$.
- Compute $u = B^{-1}A_q$. If $u \leq 0$, then (P) is unbounded and stop.
- If $\exists u_k > 0$, then find $x_q^+ = \min\limits_{i=1,\ldots,m|u_i>0} \quad \frac{x_{\mathcal{B}(i)}}{u_i}$ and use p to denote the minimizing i. Set $x_{\mathcal{B}}^+ = x_{\mathcal{B}} - ux_q^+$.
- Change $\mathcal{B}$ by adding q and removing the basic variable corresponding to column p of B.



Simplex iterates for a two-dimensional problem

Finite Termination of the simplex method

Theorem

Suppose that the LP (P) is nondegenerate and bounded, the simplex method terminates at a basic optimal point.

- nondegenerate: $x_{\mathcal{B}} > 0$ and $c^\top x$ is bounded
- A strict reduction of $c^\top x$ at each iteration
- There are only a finite number of BFS since the number of possible bases $\mathcal{B}$ is finite (there are only a finite number of ways to choose a subset of m indices from $\{1, 2, \ldots, n\}$), and since each basis defines a single basic feasible point

Finite termination does not mean a polynomial-time algorithm

Linear algebra in the simplex method

- Given B^{-1}, we need to compute $\bar{B}^{-1}$, where

$$B = [A_1, \dots, A_m], \quad \bar{B} := B^+ = [A_1, \dots, A_{p-1}, A_q, A_{p+1}, \dots, A_m]$$

- the cost of inversion $\bar{B}^{-1}$ from scratch is $O(m^3)$
- Since $BB^{-1} = I$, we have

$$\begin{aligned} B^{-1}\bar{B} &= [e_1, \dots e_{p-1}, u, e_{p+1}, \dots, e_m] \\ &= \begin{pmatrix} 1 & & u_1 & & \\ & \ddots & \vdots & & \\ & & u_p & & \\ & & \vdots & \ddots & \\ & & u_m & & 1 \end{pmatrix}, \end{aligned}$$

where e_i is the ith column of I and $u = B^{-1}A_q$

Linear algebra in the simplex method

- Apply a sequence of “elementary row operation”
  - For each $j \neq p$, we add the p-th row times $-\frac{u_j}{u_p}$ to the jth row. This replaces u_j by zero.
  - We divide the pth row by u_p. This replaces u_p by one.

$$Q_{ip} = I + D_{ip}, \quad (D_{ip})_{jl} = \begin{cases} -\frac{u_j}{u_p}, & (j,l) = (i,p) \\ 0, & \text{otherwise} \end{cases}, \text{ for } i \neq p$$

- Find Q such that $QB^{-1}\bar{B} = I$. Computing $\bar{B}^{-1}$ needs only $O(m^2)$
- What if B^{-1} is computed by the LU factorization, i.e., $B = LU$? L is is unit lower triangular, U is upper triangular. Read section 13.4 in “Numerical Optimization”, Jorge Nocedal and Stephen Wright,

An iteration of the revised simplex method

Typically, we start from a BFS x and its associate basis $\mathcal{B}$ such that $x_B = B^{-1}b$ and $x_N = 0$.

- Solve $y^\top = c_B^\top B^{-1}$ and then the reduced costs $s_N = c_N - N^\top y$
- If $s_N \geq 0$, x is optimal and stop; Else, choose $q \in \mathcal{N}$ with $s_q < 0$.
- Compute $u = B^{-1}A_q$. If $u \leq 0$, then (P) is unbounded and stop.
- If $\exists u_k > 0$, then find $x_q^+ = \min_{i=1,\ldots,m | u_i > 0} \frac{x_{\mathcal{B}(i)}}{u_i}$ and use p to denote the minimizing i. Set $x_{\mathcal{B}}^+ = x_{\mathcal{B}} - u x_q^+$.
- Form the $m \times (m+1)$ matrix $[B^{-1} \mid u]$. Add to each one of its rows a multiple of the pth row to make the last column equal to the unit vector e_p. The first m columns of the result is the matrix $\bar{B}^{-1}$.

Selection of the entering index (pivoting rule)

Reduced costs $s_N = c_N - N^\top y$, $c^\top x^+ = c^\top x + s_q x_q^+$

- Dantzig: chooses $q \in \mathcal{N}$ such that s_q is the most negative component

- Bland's rule: choose the smallest $j \in \mathcal{N}$ such that $s_j < 0$; out of all variables x_i that are tied in the test for choosing an exiting variable, select the one with with the smallest value i.

- Steepest-edge: choose $q \in \mathcal{N}$ such that $\frac{c^\top \eta_q}{\|\eta_q\|}$ is minimized, where

$$x^+ = \begin{pmatrix} x_B^+ \\ x_N^+ \end{pmatrix} = \begin{pmatrix} x_B \\ x_N \end{pmatrix} + \begin{pmatrix} -B^{-1}A_q \\ e_q \end{pmatrix} x_q = x + \eta_q x_q^+$$

  efficient computation of this rule is available

Degenerate steps and cycling

Let q be the entering variable:

$$x_{\mathcal{B}}^{+} = B^{-1}b - B^{-1}A_q x_q^{+} = x_B - x_q^{+}u, \text{ where } u = B^{-1}A_q$$

- Degenerate step: there exists $i \in \mathcal{B}$ such that $x_i = 0$ and $u_i > 0$. Then $x_i^{+} < 0$ if $x_q^{+} > 0$. Hence, $x_q^{+} = 0$ and do the pivoting
- Degenerate step may still be useful because they change the basis $\mathcal{B}$, and the updated $\mathcal{B}$ may be closer to the optimal basis.
- cycling: after a number of successive degenerate steps, we may return to an earlier basis $\mathcal{B}$
- Cycling has been observed frequently in the large LPs that arise as relaxations of integer programming problems
- Avoid cycling: Bland's rule and Lexicographically pivoting rule

Finding an initial BFS

The two-phase simplex method

$$\textbf{(P)} \quad \begin{array}{ll} \min & c^\top x \\ \text{s.t.} & Ax = b \\ & x \geq 0 \end{array} \qquad \textbf{(P0)} \quad \begin{array}{ll} \tilde{f} = \min & z_1 + z_2 + \ldots + z_m \\ \text{s.t.} & Ax + z = b \\ & x \geq 0, \quad z \geq 0 \end{array}$$

- A BFS to (P0): $x = 0$ and $z = b$
- If x is feasible to (P), then $(x, 0)$ is feasible to (P0)
- If the optimal cost $\tilde{f}$ of (P0) is nonzero, then (P) is infeasible
- If $\tilde{f} = 0$, then its optimal solution must satisfies: $z = 0$ and x is feasible to (P)
- An optimal basis $\mathcal{B}$ to (P0) may contain some components of z

Finding an initial BFS

(x, z) is optimal to (P0) with some components of z in the basis

- Assume $A_1, \ldots, A_k$ are in the basis matrix with $k < m$. Then

$$B = [A_1, \ldots, A_k \mid \text{ some columns of } I]$$

$$B^{-1}A = [e_1, \ldots, e_k, B^{-1}A_{k+1}, \ldots, B^{-1}A_n]$$

- Suppose that ℓth basic variable is an artificial variable
- If the ℓth row of $B^{-1}A$ is zero, then $g^\top A = 0^\top$, where $g^\top$ is the ℓth row of B^{-1}. If $g^\top b \neq 0$, (P) is infeasible. Otherwise, A has linearly dependent rows. Remove the ℓth row.
- There exists j such that the ℓth entry of $B^{-1}A_j$ is nonzero. Then A_j is linearly independent to $A_1, \ldots, A_k$. Perform elementary row operation to replace $B^{-1}A_j$ to be the ℓth unit vector. Driving one of z out of the basis

The primal simplex method for LP

$$\textbf{(P)}\quad \begin{array}{ll} \min & c^\top x \\ \text{s.t.} & Ax = b \\ & x \geq 0 \end{array} \qquad \textbf{(D)}\quad \begin{array}{ll} \max & b^\top y \\ \text{s.t.} & A^\top y + s = c \\ & s \geq 0 \end{array}$$

- KKT condition

$$\begin{array}{rcll} Ax & = & b, & x \geq 0 \\ A^\top y + s & = & c, & s \geq 0 \\ x_i s_i & = & 0 & \text{for } i = 1, \ldots, n \end{array}$$

- The primal simplex method generates

$$\begin{array}{rcl} x_B & = & B^{-1}b \geq 0, \quad x_N = 0, \\ y & = & B^{-T}c_B, \\ s_B & = & c_B - B^\top y = 0, s_N = c_N - N^\top y ? 0 \end{array}$$

Outline

1. Properties of LP
2. Primal Simplex method
3. Dual Simplex method
4. Interior Point method

The dual simplex method for LP

- The dual simplex method generates

$$\begin{aligned} x_B &= B^{-1}b?0, \quad x_N = 0, \\ y &= B^{-T}c_B, \\ s_B &= c_B - B^\top y = 0, s_N = c_N - N^\top y \geq 0 \end{aligned}$$

- If $x_B \geq 0$, then (x, y, s) is optimal
- Otherwise, select $q \in \mathcal{B}$ such that $x_q < 0$ to exit the basis, select $r \in \mathcal{N}$ to enter the basis, i.e., $s_r^+ = 0$
- The update is of the form

$$\begin{aligned} s_B^+ &= s_B + \alpha e_q \quad \text{obvious} \\ y^+ &= y + \alpha v \quad \text{requirement} \end{aligned}$$

The dual simplex method for LP

- What is v? Since $A^\top y^+ + s^+ = c$, it holds

$$\begin{aligned} & s_B^+ = c_B - B^\top y^+ \\ \Longrightarrow \quad & s_B + \alpha e_q = c_B - B^\top (y + \alpha v) \Longrightarrow e_q = -B^\top v \end{aligned}$$

- The update of the dual objective function

$$\begin{aligned} b^\top y^+ &= b^\top y + \alpha b^\top v \\ &= b^\top y - \alpha b^\top B^{-T} e_q \\ &= b^\top y - \alpha x_B^\top e_q \\ &= b^\top y - \alpha x_q \end{aligned}$$

- Since $x_q < 0$ and we maximize $b^\top y^+$, we choose α as large as possible, but require $s_N^+ \geq 0$

The dual simplex method for LP

- Let $w = N^\top v = -N^\top B^{-T} e_q$. Since $Ay + s = c$ and $A^\top y^+ + s^+ = c$, it holds

$$s_N^+ = c_N - N^\top y^+ = s_N - \alpha N^\top v = s_N - \alpha w \geq 0$$

- The largest α is

$$\alpha = \min_{j \in \mathcal{N}, w_j > 0} \quad \frac{s_j}{w_j}.$$

  Let r be the index at which the minimum is achieved.

$$s_r^+ = 0, \quad w_r = A_r^\top v > 0$$

- (D) is unbounded if $w \leq 0$

The dual simplex method for LP: update of x^+

We have: $Bx_B = b$, $x_q^+ = 0$, $x_r^+ = \gamma$ and $Ax^+ = b$, i.e.,

$$Bx_{\mathcal{B}}^+ + \gamma A_r = b \Longrightarrow x_{\mathcal{B}}^+ = B^{-1}b - \gamma B^{-1}A_r,$$

where $Bd = A_r$. Then $Ax^+ = b$ gives

$$B(x_{\mathcal{B}} - \gamma d) + \gamma A_r = b \text{ for any } \gamma.$$

Since it is required $x_q^+ = 0$, we set

$$\gamma = \frac{x_q}{d_q}, \text{ where } d_q = d^\top e_q = A_r^\top B^{-T} e_q = -A_r^\top v = -w_r < 0.$$

Therefore

$$x_i^+ = \begin{cases} x_i - \gamma d_i, & \text{for } i \in \mathcal{B} \text{ with } i \neq q, \\ 0, & i = q, \\ 0, & i \in \mathcal{N} \text{ with } i \neq r, \\ \gamma, & i = r \end{cases}$$

An iteration of the dual simplex method

Typically, we start from a dual feasible (y, s) and its associate basis $\mathcal{B}$ such that $x_B = B^{-1}b$ and $x_N = 0$.

- If $x_B \geq 0$, then x is optimal and stop. Else, choose q such that $x_q < 0$.
- Compute $v = -B^{-T}e_q$ and $w = N^\top v$. If $w \leq 0$, then (D) is unbounded and stop.
- If $\exists w_k > 0$, then find $\alpha = \min\limits_{j \in \mathcal{N}, w_j > 0} \frac{s_j}{w_j}$ and use r to denote the minimizing j. Set $s_B^+ = s_B + \alpha e_q$, $s_N^+ = s_N - \alpha w$ and $y^+ = y + \alpha v$.
- Change $\mathcal{B}$ by adding r and removing the basic variable corresponding to column q of B.

Outline

1. Properties of LP
2. Primal Simplex method
3. Dual Simplex method
4. Interior Point method

Primal-Dual Methods for LP

(P) $\min \; c^\top x$ s.t. $Ax = b$, $x \geq 0$ **(D)** $\max \; b^\top y$ s.t. $A^\top y + s = c$, $s \geq 0$

$$
\begin{array}{llll}
\textbf{(P)} & \min & c^\top x & \\
 & \text{s.t.} & Ax = b & \\
 & & x \geq 0 &
\end{array}
\qquad
\begin{array}{llll}
\textbf{(D)} & \max & b^\top y & \\
 & \text{s.t.} & A^\top y + s = c & \\
 & & s \geq 0 &
\end{array}
$$

- KKT condition

$$
\begin{aligned}
Ax &= b, & x &\geq 0 \\
A^\top y + s &= c, & s &\geq 0 \\
x_i s_i &= 0 & \text{for } i &= 1, \ldots, n
\end{aligned}
$$

- Perturbed system

$$
\begin{aligned}
Ax &= b, & x &\geq 0 \\
A^\top y + s &= c, & s &\geq 0 \\
x_i s_i &= \sigma\mu & \text{for } i &= 1, \ldots, n
\end{aligned}
$$

Newton's method

- Let (x, y, s) be the current estimate with $(x, s) > 0$
- Let $(\Delta x, \Delta y, \Delta s)$ be the search direction
- Let $\mu = \frac{1}{n}x^\top s$ and $\sigma \in (0, 1)$. Hope to find

$$\begin{aligned} A(x + \Delta x) &= b \\ A^\top (y + \Delta y) + s + \Delta s &= c \\ (x_i + \Delta x_i)(s_i + \Delta s_i) &= \sigma\mu \end{aligned}$$

- dropping the nonlinaer term $\Delta x_i \Delta s_i$ gives

$$\begin{aligned} A\Delta x &= r_p := b - Ax \\ A^\top \Delta y + \Delta s &= r_d := c - A^\top y - s \\ x_i \Delta s_i + \Delta x_i s_i &= (r_c)_i := \sigma\mu - x_i s_i \end{aligned}$$

Newton's method

- Let $\mathsf{L}_x = \mathrm{Diag}(x)$ and $\mathsf{L}_s = \mathrm{Diag}(s)$. The matrix form is:

$$\begin{pmatrix} A & 0 & 0 \\ 0 & A^\top & I \\ \mathsf{L}_s & 0 & \mathsf{L}_x \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta s \end{pmatrix} = \begin{pmatrix} r_p \\ r_d \\ r_c \end{pmatrix}$$

- Solving this system we get

$$\begin{aligned} \Delta y &= (A\mathsf{L}_s^{-1}\mathsf{L}_x A^\top)^{-1}(r_p + A\mathsf{L}_s^{-1}(\mathsf{L}_x r_d - r_c)) \\ \Delta s &= r_d - A^\top \Delta y \\ \Delta x &= -\mathsf{L}_s^{-1}(\mathsf{L}_x \Delta s - r_c) \end{aligned}$$

- The matrix $A\mathsf{L}_s^{-1}\mathsf{L}_x A^\top$ is symmetric and positive definite if A is full rank

The Primal-Dual Path-following Method

Given (x^0, y^0, s^0) with $(x^0, s^0) \geq 0$. A typical iteration is

- Choose $\mu = (x^k)^\top s^k / n$, $\sigma \in (0, 1)$ and solve

$$\begin{pmatrix} A & 0 & 0 \\ 0 & A^\top & I \\ \mathsf{L}_{s^k} & 0 & \mathsf{L}_{x^k} \end{pmatrix} \begin{pmatrix} \Delta x^k \\ \Delta y^k \\ \Delta s^k \end{pmatrix} = \begin{pmatrix} r_p^k \\ r_d^k \\ r_c^k \end{pmatrix}$$

- Set

$$(x^{k+1}, y^{k+1}, s^{k+1}) = (x^k, y^k, s^k) + \alpha_k (\Delta x^k, \Delta y^k, \Delta s^k),$$

  choosing α_k such that $(x^{k+1}, s^{k+1}) > 0$

The choices of centering parameter σ and step length α_k are crucial to the performance of the method.

The Central Path

- The primal-dual feasible and strictly feasible sets:

$$\begin{array}{rcl}\mathcal{F} & = & \{(x,y,s) \mid Ax=b, A^\top y+s=c, (x,s)\geq 0\} \\ \mathcal{F}^o & = & \{(x,y,s) \mid Ax=b, A^\top y+s=c, (x,s)> 0\}\end{array}$$

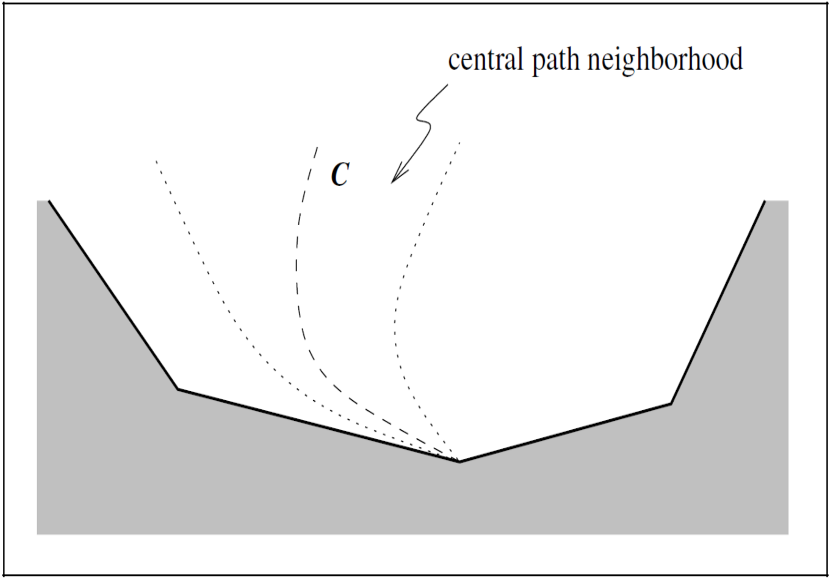
- The central path is $\mathcal{C}=\{(x_\tau,y_\tau,s_\tau) \mid \tau>0\}$, where

$$\begin{array}{rcll}Ax_\tau & = & b, & x_\tau>0 \\ A^\top y_\tau+s_\tau & = & c, & s_\tau>0 \\ (x_\tau)_i(s_\tau)_i & = & \tau & \text{for } i=1,\ldots,n\end{array}$$

- Central path neighborhoods, for $\theta,\gamma\in[0,1)$:

$$\begin{array}{rcl}\mathcal{N}_2(\theta) & = & \{(x,y,s)\in\mathcal{F}^o \mid \|\mathsf{L}_x\mathsf{L}_s e-\mu e\|_2\leq\theta\mu\} \\ \mathcal{N}_{-\infty}(\gamma) & = & \{(x,y,s)\in\mathcal{F}^o \mid x_is_i\geq\gamma\mu\}\end{array}$$

  Tyically, $\theta=0.5$ and $\gamma=10^{-3}$



Central path, projected into space of primal variables x, showing a typical neighborhood $\mathcal{N}$

The Long-Step Path-following Method

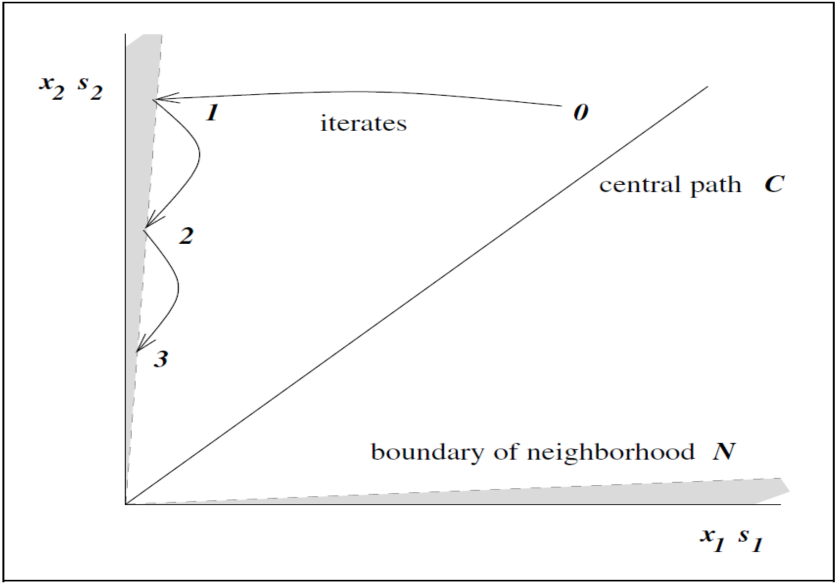
Given $(x^0, y^0, s^0) \in \mathcal{N}_{-\infty}(\gamma)$. A typical iteration is

- Choose $\mu = (x^k)^\top s^k/n$, $\sigma \in (0,1)$ and solve

$$\begin{pmatrix} A & 0 & 0 \\ 0 & A^\top & I \\ \mathsf{L}_{s^k} & 0 & \mathsf{L}_{x^k} \end{pmatrix} \begin{pmatrix} \Delta x^k \\ \Delta y^k \\ \Delta s^k \end{pmatrix} = \begin{pmatrix} r_p^k \\ r_d^k \\ r_c^k \end{pmatrix}$$

- Set α_k be the largest value of $\alpha \in [0,1]$ such that $(x^{k+1}, y^{k+1}, s^{k+1}) \in \mathcal{N}_{-\infty}(\gamma)$ where

$$(x^{k+1}, y^{k+1}, s^{k+1}) = (x^k, y^k, s^k) + \alpha_k(\Delta x^k, \Delta y^k, \Delta s^k),$$


$x_2 s_2$
0
iterates
1
2
3
central path C
boundary of neighborhood N
$x_1 s_1$

Analysis of Primal-Dual Path-Following

1. If $(x, y, s) \in \mathcal{N}_{-\infty}(\gamma)$, then $\|\Delta x \circ \Delta s\| \leq 2^{-3/2}(1 + 1/\gamma)n\mu$
2. The long-step path-following method yields
$$\mu_{k+1} \leq \left(1 - \frac{\delta}{n}\right)\mu_k,$$
where $\delta = 2^{3/2}\gamma\frac{1-\gamma}{1+\gamma}\sigma(1-\sigma)$
3. Given $\epsilon, \gamma \in (0, 1)$, suppose that the starting point $(x^0, y^0, s^0) \in \mathcal{N}_{-\infty}(\gamma)$. Then there exists $K = O(nlog(1/\epsilon))$ such that
$$\mu_k \leq \epsilon\mu_0, \qquad \text{for all } k \geq K$$

Proof of 3:

$$\begin{aligned} \log(\mu_{k+1}) &\leq \log\left(1 - \frac{\delta}{n}\right) + \log(\mu_k) \\ \log(1+\beta) &\leq \beta, \quad \forall\beta > -1 \end{aligned}$$

Primal-Dual Hybrid Gradient Algorithm for Linear Programming

Zaiwen Wen

Beijing International Center For Mathematical Research
Peking University

`http://bicmr.pku.edu.cn/~wenzw/bigdata2025.html`

Acknowledgement: this slides is Prepared by Zhonglin Xie

Mathematical Formulation: LP Problem

General LP Problem:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & c^\top x \\ \text{s.t.} \quad & \ell_c \leq Ax \leq u_c \\ & \ell_v \leq x \leq u_v \end{aligned}$$

Deriving the Lagrangian:

- Introduce dual variables for constraints to form unconstrained problem
- Rewrite constraints: $Ax - u_c \leq 0$, $\ell_c - Ax \leq 0$, $x - u_v \leq 0$, $\ell_v - x \leq 0$
- Associate non-negative multipliers y^-, y^+, r^-, r^+ with each inequality

Lagrangian Function:

$$\begin{aligned} \mathcal{L}(x, y^-, y^+, r^-, r^+) = c^\top x + (y^-)^\top (Ax - u_c) + (y^+)^\top (\ell_c - Ax) \quad (1) \\ + (r^-)^\top (x - u_v) + (r^+)^\top (\ell_v - x) \quad (2) \end{aligned}$$

where all multipliers are non-negative

Deriving the Dual Problem

Derivation: Group x terms and form the dual function

$$\begin{aligned}\mathcal{L}(x, y^-, y^+, r^-, r^+) =& x^\top(c + A^\top(y^- - y^+) + (r^- - r^+)) && (3)\\ & - (y^-)^\top u_c + (y^+)^\top \ell_c - (r^-)^\top u_v + (r^+)^\top \ell_v && (4)\end{aligned}$$

The minimum over x is $-\infty$ unless $c + A^\top(y^- - y^+) + (r^- - r^+) = 0$

Dual Problem: Using substitutions $y = y^+ - y^-$ and $r = r^+ - r^-$

$$\begin{aligned}\max_{y\in\mathbb{R}^m, r\in\mathbb{R}^n} \quad & -(y^-)^\top u_c + (y^+)^\top \ell_c - (r^-)^\top u_v + (r^+)^\top \ell_v\\ \text{s.t.} \quad & c - A^\top y = r\\ & y^-, y^+, r^-, r^+ \geq 0\end{aligned}$$

Dual Problem Formulation

Simplified notation: Define $p(y;\ell,u) := u^\top y^+ - \ell^\top y^-$ where $y^+ = \max(y,0)$ and $y^- = \max(-y,0)$

Rewriting the dual:

$$\begin{aligned} \max_{y\in\mathbb{R}^m, r\in\mathbb{R}^n} \quad & -p(-y;\ell_c,u_c) - p(-r;\ell_v,u_v) \\ \text{s.t.} \quad & c - A^\top y = r \\ & y \in \mathcal{Y},\ r \in \mathcal{R} \end{aligned}$$

Dual feasible sets $\mathcal{Y}$ and $\mathcal{R}$: Based on constraint types

$$\mathcal{Y}_i := \begin{cases} \{0\} & (\ell_c)_i = -\infty, (u_c)_i = \infty \text{ (unconstrained)} \\ \mathbb{R}^- & (\ell_c)_i = -\infty, (u_c)_i \in \mathbb{R} \text{ (upper bound)} \\ \mathbb{R}^+ & (\ell_c)_i \in \mathbb{R}, (u_c)_i = \infty \text{ (lower bound)} \\ \mathbb{R} & \text{otherwise (both upper and lower bounds)} \end{cases} \quad \mathcal{R}_i := \begin{cases} \{0\} & (\ell_v)_i = -\infty, (u_v)_i = \infty \\ \mathbb{R}^- & (\ell_v)_i = -\infty, (u_v)_i \in \mathbb{R} \\ \mathbb{R}^+ & (\ell_v)_i \in \mathbb{R}, (u_v)_i = \infty \\ \mathbb{R} & \text{otherwise} \end{cases}$$

Saddle Point Formulation of LP

- Keeping the bounds on x, we obtain the Lagrangian function:

$$\mathcal{L}(x, y^-, y^+) = c^\top x + (y^-)^\top (Ax - u_c) + (y^+)^\top (\ell_c - Ax)$$

- Using the notation $y = y^+ - y^-$ and the function $p(y; \ell, u) = u^\top y^+ - \ell^\top y^-$:

$$\begin{aligned}(y^-)^\top (Ax - u_c) + (y^+)^\top (\ell_c - Ax) &= -((y^-)^\top u_c - (y^+)^\top \ell_c) + (y^- - y^+)^\top (Ax) \\ &= -p(-y; \ell_c, u_c) - y^\top (Ax)\end{aligned}$$

- Then the saddle point problem is:

$$\min_x \max_y \{c^\top x + y^\top (Ax) - p(-y; \ell_c, u_c)\} \quad \text{s.t. } \ell_v \leq x \leq u_v$$

Final saddle point formulation:

$$\min_{x \in \mathcal{X}} \max_{y \in \mathcal{Y}} \mathcal{L}(x, y) := c^\top x + y^\top Ax - p\,(y; -u_c, -\ell_c)$$

where $\mathcal{X} := \{x \in \mathbb{R}^n : \ell_v \leq x \leq u_v\}$

Problems with Classical Solvers for Large-Scale LP

- **Simplex Method:**
    - Iterations potentially exponential in problem size
    - Poor parallelization on modern hardware
- **Interior Point Methods (IPMs):**
    - Memory requirements: $O(nnz(A))$ for matrix factorization
    - Often exceeds 1TB for problems with billions of nonzeros
- **First-Order Methods:**
    - Low memory requirements
    - Highly parallelizable matrix-vector operations
    - But historically struggle with achieving high accuracy
    - Small constraint violations can lead to significant errors

First-Order Methods for LP

- **State-of-the-art First-Order Method Solvers:**
  - SCS: ADMM-based solver with homogeneous self-dual embedding
  - OSQP: ADMM-based for convex quadratic programming
  - ECLIPSE: Gradient descent on smoothed dual formulation
  - ABIP/ABIP+: Interior-point solvers using ADMM
- **PDHG (Primal-Dual Hybrid Gradient) Advantages:**
  - Requires only matrix-vector products: Ax and $A^\top y$
  - No matrix factorization or systems of equations
  - Form of operator splitting (related to ADMM)
  - Linear convergence for LP established in theory

Generic Convex-Concave Saddle Point Problems

General Form:

$$\min_{x \in \mathcal{X}} \max_{y \in \mathcal{Y}} \mathcal{L}(x, y) = \langle Kx, y \rangle + g(x) - f^*(y)$$

Key components:

- K: Linear operator (matrix) mapping primal to dual space
- $g(x)$: Convex function (often includes constraints on x)
- $f^*(y)$: Convex conjugate of function f (handles dual constraints)

Convex Conjugate Definition: For any convex function f

$$f^*(y) = \sup_x \{\langle x, y \rangle - f(x)\}$$

This transforms constraints into penalties in the optimization

PDHG: Abstract Form

Primal-Dual Hybrid Gradient Algorithm:

$$x^{k+1} = \text{prox}_{\tau g}(x^k - \tau K^* y^k) \tag{5}$$

$$y^{k+1} = \text{prox}_{\sigma f^*}(y^k + \sigma K(2x^{k+1} - x^k)) \tag{6}$$

Proximal Operator: A generalization of projection

$$\text{prox}_{\tau g}(z) = \arg\min_x \left\{ g(x) + \frac{1}{2\tau}\|x - z\|^2 \right\}$$

Moreau Decomposition: Allows computing proximal operator of f^* using f

$$\text{prox}_{\sigma f^*}(y) = y - \sigma \cdot \text{prox}_{f/\sigma}(y/\sigma)$$

This is crucial for implementing PDHG efficiently without explicitly forming the conjugate function!

Applying PDHG to LP Problems

For LP saddle point problem:

$$\min_{x \in \mathcal{X}} \max_{y \in \mathcal{Y}} \mathcal{L}(x, y) := c^\top x + y^\top Ax - p\,(y; -u_c, -\ell_c)$$

We identify:

- $K = A$ (linear constraint matrix)
- $g(x) = c^\top x + \delta_{\mathcal{X}}(x)$ (objective + variable bounds)
- $f^*(y) = p(y; -u_c, -\ell_c)$ (constraint bounds)

Computing proximal operators:

$$\text{prox}_{\tau g}(z) = \text{proj}_{\mathcal{X}}(z - \tau c) \tag{7}$$

$$\text{prox}_{\sigma f^*}(y) = y - \sigma \cdot \text{proj}_{[-u_c, -\ell_c]}(y/\sigma) \tag{8}$$

where projections enforce the constraints efficiently

PDHG Algorithm for LP

PDHG iterations for LP:

$$x^{k+1} = \mathrm{proj}_{[\ell_v, u_v]}(x^k - \tau(c + A^\top y^k)) \tag{9}$$

$$\tilde{y}^{k+1} = y^k + \sigma A(2x^{k+1} - x^k) \tag{10}$$

$$y^{k+1} = \tilde{y}^{k+1} - \sigma \mathrm{proj}_{[-u_c, -\ell_c]}(\tilde{y}^{k+1}/\sigma) \tag{11}$$

Key benefits:

- Only requires matrix-vector products (Ax and $A^\top y$)
- Projections computed element-wise (highly parallelizable)
- No matrix factorization or linear systems to solve
- Memory-efficient for very large-scale problems

Convergence Theory for PDHG on LP

Step Size Parameterization:

- $\tau = \eta/\omega$ and $\sigma = \omega\eta$ with $\eta \in (0, \infty), \omega \in (0, \infty)$
- Convergence guaranteed when $\eta < 1/\|A\|_2$
- ω: primal weight, controls scaling between primal and dual iterates

Special Norm for Convergence Analysis:

$$\|z\|_\omega := \sqrt{\omega\|x\|_2^2 + \frac{\|y\|_2^2}{\omega}}$$

for $z = (x, y)$ - Used in convergence theory, restart criteria, and primal-dual balance

Linear convergence: Under certain conditions, PDHG converges linearly for LP:

$$\|z^k - z^*\|_\omega \leq C(1-\gamma)^k$$

where $\gamma \in (0, 1)$ depends on problem structure

Main Algorithm

Algorithm PDLP (Main Structure)

```
**Input:** Initial solution $z^{0,0}$
Initialize outer loop counter $n \leftarrow 0$, total iterations $k \leftarrow 0$
Initialize step size $\hat{\eta}^{0,0} \leftarrow 1/\|A\|_\infty$, primal weight $\omega^0 \leftarrow$ InitializePrimalWeight
**repeat**
    $t \leftarrow 0$ {Inner restart loop counter}
    **repeat**
        $z^{n,t+1}, \eta^{n,t+1}, \hat{\eta}^{n,t+1} \leftarrow$ AdaptiveStepOfPDHG($z^{n,t}, \omega^n, \hat{\eta}^{n,t}, k$)
        $\bar{z}^{n,t+1} \leftarrow \frac{1}{\sum_{i=1}^{t+1} \eta^{n,i}} \sum_{i=1}^{t+1} \eta^{n,i} z^{n,i}$ {Step-size weighted average}
        $z_c^{n,t+1} \leftarrow$ GetRestartCandidate($z^{n,t+1}, \bar{z}^{n,t+1}, z^{n,0}$)
        $t \leftarrow t+1,\ k \leftarrow k+1$
    **until** restart or termination criteria holds
    **restart the outer loop:** $z^{n+1,0} \leftarrow z_c^{n,t}$, $n \leftarrow n+1$
    $\omega^n \leftarrow$ PrimalWeightUpdate($z^{n,0}, z^{n-1,0}, \omega^{n-1}$)
**until** termination criteria holds
**Output:** $z^{n,0}$
```

PDLP Key Improvements Overview

- **Adaptive step sizes:** Dynamic adjustment based on convergence conditions
- **Restart strategies:** Reset algorithm when progress slows
- **Primal weight updates:** Balance progress in primal and dual spaces
- **Special Norm for Convergence Analysis:**

$$\|z\|_\omega := \sqrt{\omega\|x\|_2^2 + \frac{\|y\|_2^2}{\omega}}$$

for $z = (x, y)$ - Used in convergence theory, restart criteria, and primal-dual balance

Adaptive Step Size

Traditional PDHG: Fixed step size $\eta = \frac{1}{\|A\|_2}$

- Overly pessimistic
- Requires estimation of $\|A\|_2$

PDLP Approach: Adaptive step size based on convergence condition

- Calculate maximum allowable step size:

$$\bar{\eta} = \frac{\|z^{k+1} - z^k\|_\omega^2}{2(y^{k+1} - y^k)^\top A(x^{k+1} - x^k)}$$

- This ensures the iteration remains convergent

Adaptive Step Size Algorithm

Algorithm One step of PDHG with adaptive step size

```
function AdaptiveStepOfPDHG($z^{n,t}, \omega^n, \hat{\eta}^{n,t}, k$)
$(x, y) \leftarrow z^{n,t}$, $\eta \leftarrow \hat{\eta}^{n,t}$
for $i = 1, \ldots, \infty$ do
    $x' \leftarrow \mathrm{proj}_{\mathcal{X}}(x - \frac{\eta}{\omega^n}(c - A^\top y))$
    $y' \leftarrow y - A(2x' - x) - \eta\omega^n \mathrm{proj}_{[\ell_c, u_c]}\left((\eta\omega^n)^{-1} y - A(2x' - x)\right)$
    $\bar{\eta} \leftarrow \frac{\|(x'-x, y'-y)\|_{\omega^n}^2}{2(y'-y)^\top A(x'-x)}$
    $\eta' \leftarrow \min\left((1 - (k+1)^{-0.3})\bar{\eta}, (1 + (k+1)^{-0.6})\eta\right)$
    if $\eta \leq \bar{\eta}$ then
        return $(x', y'), \eta, \eta'$
    end if
    $\eta \leftarrow \eta'$
end for
```

Key Properties:

- Guarantee convergence: Only accept step if $\eta \leq \bar{\eta}$
- Aggressive adaptation: Next step size η' can grow up to factor of $(1 + (k+1)^{-0.6})$

Normalized Duality Gap and Adaptive Restarts

Normalized Duality Gap Definition:

$$\rho_r^n(z) := \frac{1}{r} \max_{\|\hat{z}-z\|_\omega \leq r} \{\mathcal{L}(x, \hat{y}) - \mathcal{L}(\hat{x}, y)\}$$

where $\mathcal{L}(x, y) = c^\top x + y^\top Ax - p(y; -u_c, -\ell_c)$ is the Lagrangian.

Key Properties:

- Always finite (bounded by search radius)
- Zero if and only if solution is optimal
- Provides a meaningful measure of progress toward optimality

Notation: $\mu_n(z, z_{\text{ref}}) := \rho^n_{\|z-z_{\text{ref}}\|_{\omega^n}}(z)$, where z_{ref} is a user-chosen reference point (typically start of current restart)

Adaptive Restart Criteria

PDLP Parameters:

$$\beta_{\text{necessary}} = 0.9, \quad \beta_{\text{sufficient}} = 0.1, \quad \beta_{\text{artificial}} = 0.5$$

Restart triggered when any of these criteria hold:

- **Sufficient decay:**

  $$\mu_n(z_c^{n,t+1}, z^{n,0}) \leq \beta_{\text{sufficient}} \mu_n(z^{n,0}, z^{n-1,0})$$

  Guarantees linear convergence on LP problems

- **Necessary decay + no progress:**

  $$\begin{aligned} &\mu_n(z_c^{n,t+1}, z^{n,0}) \leq \beta_{\text{necessary}} \mu_n(z^{n,0}, z^{n-1,0}) \\ \text{and} \quad &\mu_n(z_c^{n,t+1}, z^{n,0}) > \mu_n(z_c^{n,t}, z^{n,0}) \end{aligned}$$

  Detects when progress begins to stall

- **Long inner loop:** $t \geq \beta_{\text{artificial}} k$

Restart Mechanism - Implementation Details

Restart Candidate Selection:

$$\mathrm{GetRestartCandidate}(z^{n,t+1}, \bar{z}^{n,t+1}, z^{n,0}) = \begin{cases} z^{n,t+1} & \text{if } \mu_n(z^{n,t+1}, z^{n,0}) < \mu_n(\bar{z}^{n,t+1}, z^{n,0}) \\ \bar{z}^{n,t+1} & \text{otherwise} \end{cases}$$

Implementation Note:

- Restart criteria evaluated every 64 iterations to reduce overhead
- Makes minimal impact on total iteration count
- Running averages $\bar{z}^{n,t+1}$ weighted by step sizes

Primal Weight Updates

Motivation: Balance progress in primal and dual spaces

- For optimal convergence, we want equal progress in both spaces:

$$\|(x^{n,t} - x^*, \mathbf{0})\|_{\omega^n} = \|(\mathbf{0}, y^{n,t} - y^*)\|_{\omega^n}$$

- This yields the ideal primal weight:

$$\omega^n = \frac{\|y^{n,t} - y^*\|_2}{\|x^{n,t} - x^*\|_2}$$

Implementation:

- Estimate using consecutive iterates:

$$\Delta_x^n = \|x^{n,0} - x^{n-1,0}\|_2, \quad \Delta_y^n = \|y^{n,0} - y^{n-1,0}\|_2$$

- Apply log-scale exponential smoothing:

$$\omega^n = \exp\left(\theta \log\left(\frac{\Delta_y^n}{\Delta_x^n}\right) + (1-\theta)\log(\omega^{n-1})\right)$$

Primal Weight Update Algorithm

Algorithm Primal weight update

```
function PrimalWeightUpdate($z^{n,0}, z^{n-1,0}, \omega^{n-1}$)
$\Delta_x^n = \|x^{n,0} - x^{n-1,0}\|_2$, $\Delta_y^n = \|y^{n,0} - y^{n-1,0}\|_2$
if $\Delta_x^n > \varepsilon_0$ and $\Delta_y^n > \varepsilon_0$ then
    return $\exp\left(\theta \log\left(\frac{\Delta_y^n}{\Delta_x^n}\right) + (1-\theta)\log(\omega^{n-1})\right)$
else
    return $\omega^{n-1}$
end if
```

Key innovation: Updates only occur after restarts

- Allows larger weight changes without causing instability
- Focuses on balancing distance traveled rather than residuals
- Significantly improves performance compared to per-iteration updates

GPU vs. CPU Architecture

CPU Design:

- Few cores (16-64) with deep pipelines
- Optimized for sequential processing
- Sophisticated branch prediction
- Limited memory bandwidth (100-200 GB/sec)

GPU Design:

- Thousands of cores (7296 on NVIDIA H100)
- Single Instruction Multiple Data (SIMD)
- Optimized for parallel computation
- Very high memory bandwidth (2 TB/sec)

Why GPUs for LP?

- Previous attempts failed with simplex/IPM methods
- First-order methods like PDHG rely on matrix-vector operations
- PDLP's core operations highly parallelizable

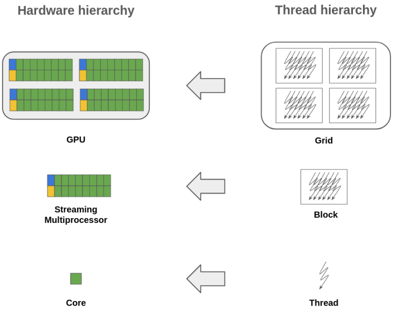
GPU Thread Hierarchy and Execution Model

Thread Hierarchy:

- **Thread:** Basic execution unit
- **Warp:** 32 threads executing in lockstep
- **Block:** Group of threads with shared memory
- **Grid:** Collection of blocks executing same kernel

Implications for PDLP:

- Matrix-vector operations highly parallelizable
- Each thread can process individual vector elements
- Challenge: Reducing CPU-GPU communication overhead



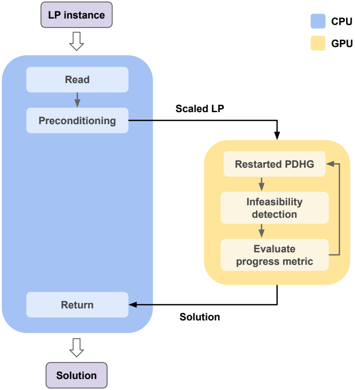
cuPDLP.jl: Design Principles

Minimizing CPU-GPU Communication:

- Initial transfer: Problem instance from CPU to GPU
- Final transfer: Solution from GPU to CPU
- All iterations computed entirely on GPU

Implementation Framework:

- Implemented in Julia using CUDA.jl
- Custom CUDA kernels for PDHG updates
- cuSPARSE library for sparse matrix operations



Key Acceleration Points

Matrix and Vector Operations:

- Sparse matrix stored in Compressed Sparse Row (CSR) format
- Matrix-vector multiplication via cuSPARSE library
- Custom CUDA kernels for vector operations and projections
- One thread per vector element for maximum throughput

KKT-Based Restart Scheme:

- Original PDLP: Trust-region algorithm for normalized duality gap
  - Sequential nature - poor fit for GPU architecture
- cuPDLP.jl innovation: KKT error-based restart
  - Highly parallelizable computation
  - Maintains convergence properties

KKT-Based Restart Details

KKT Error Definition:

$$\mathrm{KKT}_{\omega}(z) = \sqrt{\omega^2 \left\| \begin{pmatrix} Ax - b \\ [h - Gx]^{+} \end{pmatrix} \right\|_2^2 + \frac{1}{\omega^2} \| c - K^{\top} y - \lambda \|_2^2 + (q^{\top} y + l^{\top} \lambda^{+} - u^{\top} \lambda^{-} - c^{\top} x)^2}$$

Restart Candidate Selection:

$$z_c^{n,t+1} = \begin{cases} z^{n,t+1} & \text{if } \mathrm{KKT}_{\omega^n}(z^{n,t+1}) < \mathrm{KKT}_{\omega^n}(\bar{z}^{n,t+1}) \\ \bar{z}^{n,t+1} & \text{otherwise} \end{cases}$$

Restart Conditions: Algorithm restarts if any of these holds:

- **Sufficient decay:** $\mathrm{KKT}_{\omega^n}(z_c^{n,t+1}) \leq 0.2 \cdot \mathrm{KKT}_{\omega^n}(z^{n,0})$
- **Necessary decay + no progress:** $\mathrm{KKT}_{\omega^n}(z_c^{n,t+1}) \leq 0.8 \cdot \mathrm{KKT}_{\omega^n}(z^{n,0})$ and no improvement
- **Long inner loop:** Iteration count exceeds threshold

Primal and Dual Updates on GPU

Primal Update CUDA Kernel:

- **Input:** x^k, y^k, c, A, τ, lower/upper bounds
- **Parallel operations:**
    - Matrix-vector product: $A^\top y^k$ (via cuSPARSE)
    - Vector addition: $c - A^\top y^k$
    - Projection onto bounds: $\text{proj}_{\mathcal{X}}(x^k - \tau(c - A^\top y^k))$

Dual Update CUDA Kernel:

- **Input:** x^{k+1}, x^k, y^k, A, σ, constraint bounds
- **Parallel operations:**
    - Extrapolation: $2x^{k+1} - x^k$
    - Matrix-vector product: $A(2x^{k+1} - x^k)$ (via cuSPARSE)
    - Projection onto bounds for constraint relaxation
    - Final dual update computation

Performance vs. Gurobi: Moderate Accuracy (10^{-4})

	Small (269)		**Medium (94)**		**Large (20)**		**Total (383)**	
	Count	**Time**	**Count**	**Time**	**Count**	**Time**	**Count**	**Time**
cuPDLP.jl	266	8.61	92	14.80	19	111.19	377	12.02
Primal simplex	268	12.56	69	188.81	11	3145.49	348	39.81
Dual simplex	268	8.75	84	66.67	15	591.63	367	21.75
Barrier	268	5.30	88	45.01	18	415.78	374	14.92

Key Observations:

- cuPDLP.jl solves 377/383 instances (98.4%)
- Clear advantage on medium and large instances:
  - 3x faster than simplex on medium instances
  - 3.7x faster than barrier on large instances
- Especially strong for problems with complex structures

Performance vs. CPU PDLP

	Small (269)		Medium (94)		Large (20)		Total (383)	
	Count	**Time**	**Count**	**Time**	**Count**	**Time**	**Count**	**Time**
cuPDLP.jl	266	8.61	92	14.80	19	111.19	377	12.02
FirstOrderLp.jl	253	35.94	82	155.67	12	2002.21	347	66.67
PDLP (1 thread)	256	22.69	85	98.38	15	1622.91	356	43.81
PDLP (4 threads)	260	24.03	91	42.94	15	736.20	366	34.57
PDLP (16 threads)	238	104.72	84	142.79	15	946.24	337	127.49

GPU Speedup vs. CPU:

- vs. FirstOrderLp.jl: 4x on small, 10x on medium, 18x on large instances
- vs. PDLP with 4 threads: 2.9x overall speedup
- Solved 30 more instances than FirstOrderLp.jl at tolerance 10^{-4}
- Speedup increases with problem size

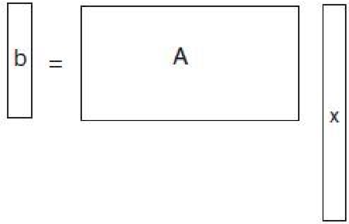
Introduction to Compressed Sensing
Sparse Recovery Guarantees

`http://bicmr.pku.edu.cn/~wenzw/bigdata2023.html`

Acknowledgement: this slides is based on Prof. Emmanuel Candes' and Prof. Wotao Yin's lecture notes

Underdetermined systems of linear equations

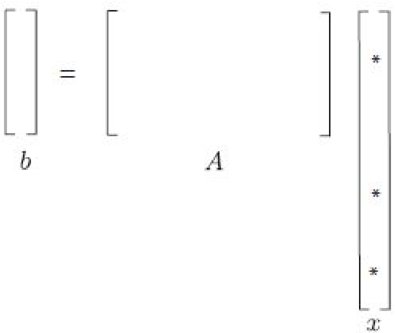
- $x \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$



When fewer equations than unknowns

- Fundamental theorem of algebra says that we cannot find x
- In general, this is absolutely correct

Special structure



If unknown is assumed to be

- sparse
- low-rank

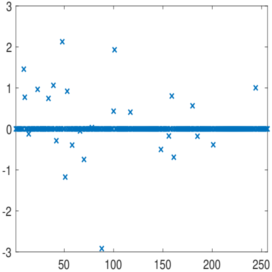
then one can *often* find solutions to these problems by convex optimization

Compressive Sensing

`http://bicmr.pku.edu.cn/~wenzw/courses/sparse_l1_example.m`

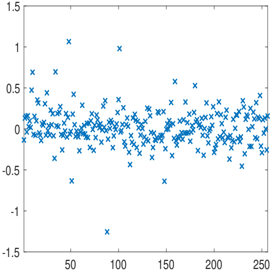
Find the sparest solution

- Given n=256, m=128.
- A = randn(m,n); u = sprandn(n, 1, 0.1); b = A*u;



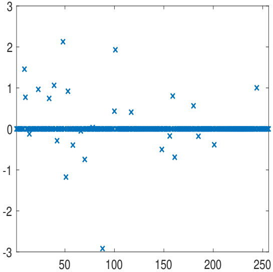
$$\begin{cases} \min\limits_{x} \|x\|_0 \\ \text{s.t. } Ax = b \end{cases}$$

(a) ℓ_0-minimization



$$\begin{cases} \min\limits_{x} \|x\|_2 \\ \text{s.t. } Ax = b \end{cases}$$

(b) ℓ_2-minimization



$$\begin{cases} \min\limits_{x} \|x\|_1 \\ \text{s.t. } Ax = b \end{cases}$$

(c) ℓ_1-minimization

Linear programming formulation

ℓ_0 minimization

$$\begin{array}{ll} \min & \|x\|_0 \\ s.t. & Ax = b \end{array}$$

Combinatorially hard

ℓ_1 minimization

$$\begin{array}{ll} \min & \|x\|_1 \\ s.t. & Ax = b \end{array}$$

Linear program

$$\begin{array}{ll} \text{minimize} & \sum_i |x_i| \\ \text{subject to} & Ax = b \end{array}$$

is equivalent to

$$\begin{array}{ll} \text{minimize} & \sum_i t_i \\ \text{subject to} & Ax = b \\ & -t_i \leq x_i \leq t_i \end{array}$$

with variables $x, t \in \mathbb{R}^n$

$x^\star$is a solution $\quad\Longleftrightarrow\quad$ $(x^\star, t^\star = |x^\star|)$ is a solution

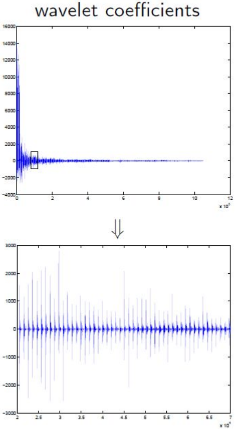
Compressed sensing

- Name coined by David Donoho
- Has become a label for sparse signal recovery
- But really one instance of underdetermined problems

- Informs analysis of underdetermined problems
- Changes viewpoint about underdetermined problems
- Starting point of a general burst of activity in
    - information theory
    - signal processing
    - statistics
    - some areas of computer science
    - ...
- Inspired new areas of research, e. g. low-rank matrix recovery

Sparsity in signal processing

1 megapixel image



zoom in

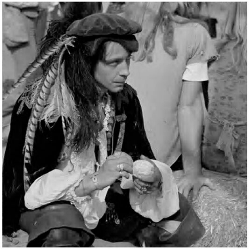
Implication: can discard small coefficients without much perceptual loss

Sparsity and wavelet "compression"

Take a mega-pixel image

- Compute $1,000,000$ wavelet coefficients
- Set to zero all but the $25,000$ largest coefficients
- Invert the wavelet transform

1 megapixel image



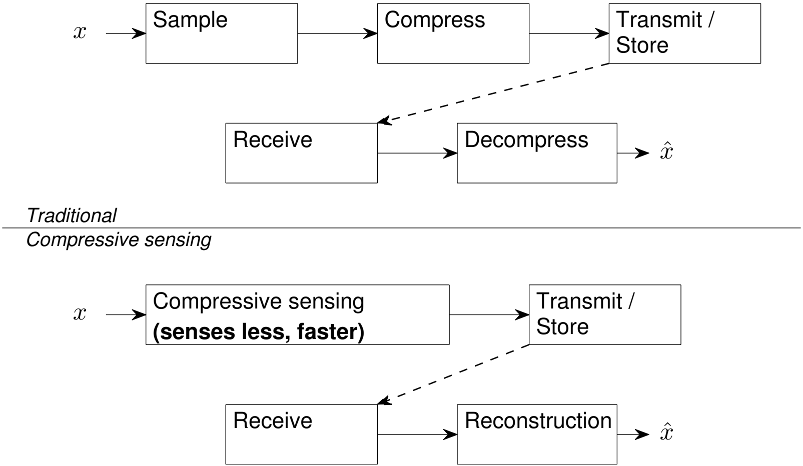
25k term approximation

This principle underlies modern lossy coders

Comparison

Sparse representation = good compression

Why? Because there are fewer things to send/store



x

Sample

Compress

Transmit / Store

Receive

Decompress

$\hat{x}$

Traditional

Compressive sensing

x

Compressive sensing **(senses less, faster)**

Transmit / Store

Receive

Reconstruction

$\hat{x}$

Restricted isometries: C. and Tao (04)

Definition (Restricted isometry constants)

For each $k = 1, 2, \ldots, \delta_k$ is the smallest scalar such that

$$(1 - \delta_k)\|x\|_2^2 \leq \|Ax\|_2^2 \leq (1 + \delta_k)\|x\|_2^2$$

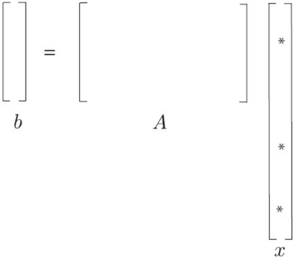
for all k-sparse x

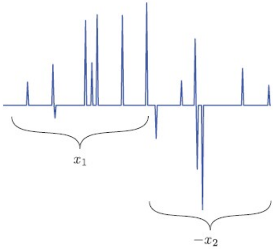
- Note slight change of normalization
- When δ_k is not too large, condition says that all $m \times k$ submatrices are well conditioned (sparse subsets of columns are not too far from orthonormal)

Interlude: when does sparse recovery make sense?

- x is s-sparse: $\|x\|_0 \leq s$
- **can we recover x from $Ax = b$?**

Perhaps possible if sparse vectors lie away from null space of A





Yes if any $2s$ columns of A are linearly independent

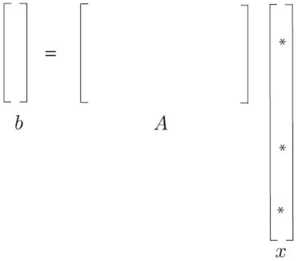
If x_1, x_2 s-sparse such that $Ax_1 = Ax_2 = b$

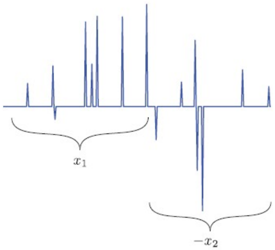
$A(x_1 - x_2) = 0 \Rightarrow x_1 - x_2 = 0 \Leftrightarrow x_1 = x_2$

Interlude: when does sparse recovery make sense?

- x is s-sparse: $\|x\|_0 \leq s$
- **can we recover x from $Ax = b$?**

Perhaps possible if sparse vectors lie away from null space of A





In general, **No** if A has $2s$ linearly dependent columns

$h \neq 0$ is $2s$-sparse with $Ah = 0$

$h = x_1 - x_2 \quad x_1, x_2$ both s-sparse

$Ah = 0 \Leftrightarrow Ax_1 = Ax_2$ and $x_1 \neq x_2$

Equivalent view of restricted isometry property

δ_{2k} is the smallest scalar such that

$$(1-\delta_{2k})\|x_1-x_2\|_2^2 \leq \|Ax_1-Ax_2\|_2^2 \leq (1+\delta_{2k})\|x_1-x_2\|_2^2$$

for all k-sparse vectors x_1, x_2.

The positive lower bounds is that which really matters

- If lower bound does not hold, then we may have x_1 and x_2 both sparse and with disjoint supports, obeying

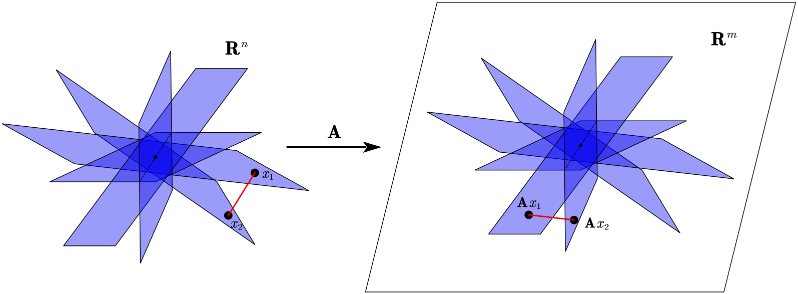
$$Ax_1 = Ax_2$$

- Lower bound guarantees that distinct sparse signals cannot be mapped too closely (analogy with codes)

With a picture

For all k-sparse x_1 and x_2

$$1 - \delta_{2k} \leq \frac{\|Ax_1 - Ax_2\|_2^2}{\|x_1 - x_2\|_2^2} \leq 1 + \delta_{2k}$$



Characterization of ℓ_1 solutions

Underdetermined system: $A \in \mathbb{R}^{m\times n}, m < n$

$$\min_{x\in\mathbb{R}^n} \|x\|_1 \quad \text{s.t. } Ax = b$$

x is solution iff

$$\|x+h\|_1 \geq \|x\|_1 \quad \forall h \in \mathbb{R}^n \ \text{ s.t. } Ah = 0$$

Notations: x supported on $T = \{i : x_i \neq 0\}$

$$\begin{aligned} \|x+h\|_1 &= \sum_{i\in T} |x_i + h_i| + \sum_{i\in T^c} |h_i| \\ &\geq \sum_{i\in T} |x_i| + \sum_{i\in T} \operatorname{sgn}(x_i) h_i + \sum_{i\in T^c} |h_i| \end{aligned}$$

because $|x_i + h_i| \geq |x_i| + \operatorname{sgn}(x_i) h_i$

Necessay and sufficient condition for ℓ_1 recovery

For all $h \in$ null(A)

$$\sum_{i\in T} \operatorname{sgn}(x_i) h_i \leq \sum_{i\in T^c} |h_i|$$

Why is this necessary? If there is $h \in \text{null}(A)$ with

$$\sum_{i \in T} \text{sgn}(x_i) h_i > \sum_{i \in T^c} |h_i|$$

then

$$\|x - h\|_1 < \|x\|_1.$$

Proof: There exists a small enough t such that

$$|x_i - th_i| = \begin{cases} x_i - th_i = x_i - t\text{sgn}(x_i)h_i & \text{if } x_i > 0 \\ -(x_i - th_i) = -x_i - t\text{sgn}(x_i)h_i & \text{if } x_i < 0 \\ t|h_i| & \text{otherwise} \end{cases}$$

Then

$$\|x - th\|_1 = \|x\|_1 - t \sum_{i \in T} \text{sgn}(x_i) h_i + t \sum_{i \in T^c} |h_i| < \|x\|_1$$

Characterization via KKT conditions

$$\min_{x\in\mathbb{R}^n} f(x) \quad \text{s.t.} \quad Ax = b$$

- f convex and differentiable Lagrangian
- $\mathcal{L}(x,\lambda) = f(x) + \langle \lambda, b - Ax\rangle$

$Ax = 0$ if and only if x is orthogonal to each of the row vectors of A.

KKT condition

x is solution iff x is feasible and $\exists\lambda \in \mathbb{R}^m$ s.t.

$$\nabla_x\mathcal{L}(x,\lambda) = 0 = \nabla f(x) - A^\top\lambda$$

Geometric interpretation: $\nabla f(x) \perp \mathrm{null}(A)$.
When f is not differentiable, condition becomes: x feasible and $\exists\lambda \in \mathbb{R}^m$ s.t.

$$A^\top\lambda \text{ is a subgradient of } f \text{ at } x$$

Subgradient

Definition

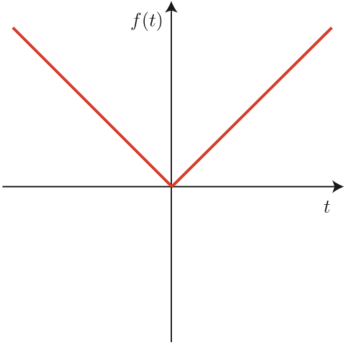
u is a subgradient of convex f at x_0 if for all x

$$f(x) \geq f(x_0) + u \cdot (x - x_0)$$

if f is differentiable at x_0, the only subgradient is $\nabla f(x_0)$

Subgradients of $f(t) = |t|, t \in \mathbb{R}$

$$\begin{cases} \{\text{subgradients}\} = \{\text{sgn(t)}\} & t \neq 0 \\ \{\text{subgradients}\} = [-1, 1] & t = 0 \end{cases}$$



Subgradients of $f(x) = \|x\|_1, x \in \mathbb{R}^n$:
$u \in \partial\|x\|_1$ (u is a subgradient) iff

$$\begin{cases} u_i = \operatorname{sgn}(x_i) & x_i \neq 0 \\ |u_i| \leq 1 & x_i = 0 \end{cases}$$

Optimality conditions II

Given $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$.

$$(P) \qquad \min_{x\in\mathbb{R}^n} \|x\|_1 \quad \text{s.t.} \quad Ax = b$$

The dual problem is

$$\max_{\lambda} \quad \lambda^\top b, \quad \text{s.t. } \|A^\top \lambda\|_\infty \le 1$$

x optimal solution iff x is feasible and there exists $u = A^\top \lambda (u \perp \text{null}(A))$ with

$$\begin{cases} u_i = \operatorname{sgn}(x_i) & x_i \neq 0 \ (i \in T) \\ |u_i| \le 1 & x_i = 0 \ (i \in T^c) \end{cases}$$

If in addition

- $|u_i| < 1$ when $x_i = 0$
- A_T has full column rank (implies by RIP)

Then x is the unique solution. We will call such a u or λ a dual certificate.

Uniqueness

Notation

- x_T: restriction of x to indices in T
- A_T: submatrix with column indices in T

If $\operatorname{supp}(x) \subseteq T$,

$$Ax = A_T x_T.$$

Let $h \in \operatorname{null}(A)$. Since $u \perp \operatorname{null}(A)$, we have

$$\begin{aligned}\sum_{i \in T} \operatorname{sgn}(x_i) h_i = \sum_{i \in T} u_i h_i = \langle u, h \rangle - \sum_{i \in T^c} u_i h_i \\ = -\sum_{i \in T^c} u_i h_i < \sum_{i \in T^c} |h_i|\end{aligned}$$

unless $h_{T^c} \neq 0$. Now if $h_{T^c} = 0$, then since A_T has full column rank,

$$Ah = A_T h_T = 0 \Rightarrow h_T = 0 \Rightarrow h = 0$$

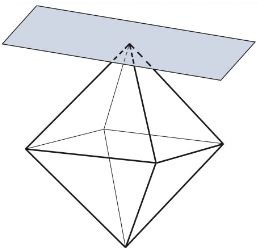
In conclusion, for any $h \in \operatorname{null}(A)$, $\|x + h\|_1 > \|x\|_1$ unless $h \neq 0$

Sufficient conditions

- $T = \text{supp}(x)$ and A_T has full column rank ($A_T^\top A_T$ invertible)
- $\text{sgn}(x_T)$ is the sign sequence of x on T and set

$$\lambda = A_T(A_T^\top A_T)^{-1}\text{sgn}(x_T) \text{ and } u := A^\top \lambda$$

- if $|u_i| \leq 1$ for all $i \in T^c$, then x is solution
- if $|u_i| < 1$ for all $i \in T^c$, then x is the unique solution



Why?

- $u_i = \text{sgn}(x_i)$ if $i \in T$, since

$$u_T = A_T^\top A_T(A_T^\top A_T)^{-1}\text{sgn}(x_T) = \text{sgn}(x_T)$$

- $u_i = A_i^\top \lambda$ if $i \notin T$.

So u is a valid dual certificate

RIP

- RIP: For each $k = 1, 2, \ldots, \delta_k$ is the smallest scalar such that

  $$(1-\delta_k)\|x\|_2^2 \leq \|Ax\|_2^2 \leq (1+\delta_k)\|x\|_2^2$$

  for all k-sparse x
- Define the constant $\theta_{S,S'}$ such that :

  $$\left\langle A_T c, A_{T'} c' \right\rangle \leq \theta_{S,S'} \|c\| \|c'\|$$

  holds for all disjoint sets T, T' of cardinality $|T| \leq S$ and $|T'| \leq S'$,
- For all S and S', we have

  $$\theta_{S,S'} \leq \delta_{S+S'} \leq \delta_{S,S'} + \max\{\delta_S, \delta_{S'}\}$$

Why this dual certificate? Why $|u_i| < 1$ for all $i \in T^c$?

- Let $S \geq 1$ be such that $\delta_S + \theta_{S,S'} + \theta_{S,2S} < 1$. Then there exits a vector λ such that $\lambda^\top A_j = \mathrm{sgn}(x_j)$ for all $j \in T$ and for all $j \notin T$:

$$|u_j| = |\lambda^\top A_j| \leq \frac{\theta_{S,S'}}{(1 - \delta_S - \theta_{S,2S})\sqrt{S}} \|\mathrm{sgn}(x)\| \leq \frac{\theta_{S,S'}}{(1 - \delta_S - \theta_{S,2S})} < 1$$

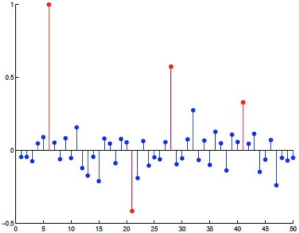
- Assume $S \geq 1$ such that $\delta_S + \theta_{S,S'} + \theta_{S,2S} < 1$. Let x be a real vector supported on T such that $|T| \leq S$. Let $b = Ax$. Then x is a unique minimizer to (P).

- Read Lemma 2.1 and Lemma 2.2 in “E. Candes and T. Tao. Decoding by linear programming. IEEE Transactions on Information Theory, 51:4203–4215, 2005”.

General setup

- x not necessarily sparse
- observe $b = Ax$
- recover by ℓ_1 minimization

$$\min \|\hat{x}\|_1 \text{ s. t. } A\hat{x} = b$$

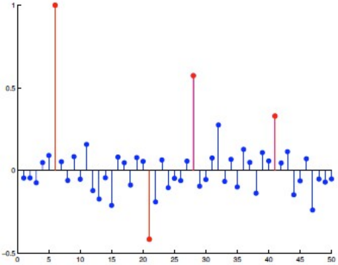


Interested in comparing performance with sparsest approximation x_s:

$$x_s = \arg \min_{\|z\|_0 \leq s} \|x - z\|$$

- x_s: s-sparse
- s-largest entries of x are the nonzero entries of x_s

General signal recovery



Theorem (Noiseless recovery (C., Romberg and Tao[a]))

If $\delta_{2s} < \sqrt{2} - 1 = 0.414\ldots$, ℓ_1 recovery obeys

$$\begin{aligned} \|\hat{x} - x\|_2 &\lesssim \|x - x_s\|_1 / \sqrt{s} \\ \|\hat{x} - x\|_1 &\lesssim \|x - x_s\|_1 \end{aligned}$$

- Deterministic (nothing is random)
- Universal (applies to all x)

- Exact if x is s-sparse
- Otherwise, essentially reconstructs the s largest entries of x
- Powerful if s is close to m

General signal recovery from noisy data

Inaccurate measurements: z error term (stochastic or deterministic)

$$b = Ax + z, \text{ with } \|z\|_2 \leq \epsilon$$

Recovery via the LASSO: ℓ_1 minimization with relaxed constraints

$$\min \|\hat{x}\|_1 \text{ s. t. } \|A\hat{x} - b\|_2 \leq \epsilon$$

Theorem (C., Romberg and Tao)

Assume $\delta_{2s} < \sqrt{2} - 1$, then

$$\|\hat{x} - x\|_2 \lesssim \frac{\|x - x_s\|_1}{\sqrt{s}} + \epsilon = approx.error + measurement\ error$$

(numerical constants hidden in $\lesssim$ are explicit)

- When $\epsilon = 0$ (no noise), earlier result
- Says when we can solve underdetermined systems of equations accurately

Proof of noisy recovery result

Let $h = \hat{x} - x$. Since $\hat{x}$ and x are feasible, we obtain

$$\|Ah\|_2 \leq \|A\hat{x} - b\|_2 + \|b - Ax\|_2 \leq 2\epsilon$$

The RIP gives

$$| \langle Ah_T, Ah \rangle | \leq \|Ah_T\|_2 \|Ah\|_2 \leq 2\epsilon\sqrt{1+\delta_{2s}}\|h_T\|_2.$$

Hence,

$$\begin{aligned} \|h\|_2 &\leq C_0 \frac{\|x - x_s\|_1}{\sqrt{s}} + C_1 \frac{| \langle Ah_T, Ah \rangle |}{\|h_T\|_2} \quad \text{lemma 4} \\ &\leq C_0 \frac{\|x - x_s\|_1}{\sqrt{s}} + C_1 2\epsilon\sqrt{1+\delta_{2s}} \end{aligned}$$

Preliminaries: Lemma 1

Let $\Sigma_k = \{x \in \mathbb{R}^n \mid x$ has k nonzero components$\}$

1. If $u \in \Sigma_k$, then $\|u\|_1/\sqrt{k} \leq \|u\|_2 \leq \sqrt{k}\|u\|_\infty$.
   Proof: $\|u\|_1 = |\langle u, \mathrm{sgn}(u)\rangle| \leq \|u\|_2\|\mathrm{sgn}(u)\|_2$.

2. Let u, v be orthogonal vectors. Then $\|u\|_2 + \|v\|_2 \leq \sqrt{2}\|u+v\|_2$.
   Proof: Apply the first statement with $w = (\|u\|_2, \|v\|_2)^\top$

3. Let A satisfies RIP of order $2k$. then for any $x, x' \in \Sigma_k$ with disjoint supports

$$|<Ax, Ax'>| \leq \delta_{s+s'}\|x\|_2\|x'\|_2$$

Proof: Suppose x and x' are unit vectors as above. Then $\|x+x'\|_2^2 = 2$, $\|x-x'\|_2^2 = 2$ due to the disjoint supports. The RIP gives

$$2(1-\delta_{s+s'}) \leq \|Ax \pm Ax'\|_2^2 \leq 2(1+\delta_{s+s'})$$

Parallelogram identity

$$|<Ax, Ax'>| = \frac{1}{4}\left|\|Ax+Ax'\|_2^2 - \|Ax-Ax'\|_2^2\right| \leq \delta_{s+s'}$$

Preliminaries: Lemma 2

1. Let T_0 be any subset $\{1, 2, \ldots, n\}$ such that $|T_0| \leq s$. For any $u \in \mathbb{R}^n$, define T_1 as the index set corresponding to the s entries of $u_{T_0^c}$ with largest magnitude, T_2 as indices of the next s largest coefficients, and so on. Then

$$\sum_{j \geq 2} \|u_{T_j}\|_2 \leq \frac{\|u_{T_0^c}\|_1}{\sqrt{s}}$$

Proof: We begin by observing that for $j \geq 2$,

$$\|u_{T_j}\|_\infty \leq \frac{\|u_{T_{j-1}}\|_1}{s}$$

since the T_j sort u to have decreasing magnitude. Using Lemma 1.1, we have

$$\sum_{j \geq 2} \|u_{T_j}\|_2 \leq \sqrt{s} \sum_{j \geq 2} \|T_j\|_\infty \leq \sum_{j \geq 1} \frac{\|u_{T_j}\|_1}{\sqrt{s}} = \frac{\|u_{T_0^c}\|_1}{\sqrt{s}}$$

Preliminaries: Lemma 3

- Let A satisfies the RIP with order $2s$. Let T_0 be any subset $\{1, 2, \ldots, n\}$ such that $|T_0| \leq s$ and $h \in \mathbb{R}^n$ be given. Define T_1 as the index set corresponding to the s entries of $h_{T_0^c}$ with largest magnitude, and set $T = T_0 \cup T_1$.Then

$$\|h_T\|_2 \leq \alpha \frac{\|h_{T_0^c}\|_1}{\sqrt{s}} + \beta \frac{|\langle Ah_T, Ah\rangle|}{\|h_T\|_2}$$

  where $\alpha = \frac{\sqrt{2}\delta_{2s}}{1-\delta_{2s}}$ and $\beta = \frac{1}{1-\delta_{2s}}$

Proof: Since $h_T \in \Sigma_{2s}$, the RIP gives

$$(1 - \delta_{2s})\|h_T\|_2^2 \leq \|Ah_T\|_2^2.$$

Continue: Proof Lemma 3

Define T_j as Lemma 2. Since $Ah_T = Ah - \sum_{j\geq 2} Ah_{T_j}$, we have

$$(1-\delta_{2s})\|h_T\|_2^2 \leq \|Ah_T\|_2^2 =< Ah_T, Ah > - < Ah_T, \sum_{j\geq 2} Ah_{T_j} >$$

Lemma 1.3 gives

$$| < Ah_{T_i}, Ah_{T_j} > | \leq \delta_{2s}\|Ah_T\|_2\|Ah\|_2$$

Note that $\|h_{T_0}\|_2 + \|h_{T_1}\|_2 \leq \sqrt{2}\|h_T\|_2$, we have

$$\begin{aligned} & | < Ah_T, \sum_{j\geq 2} Ah_{T_j} > | = | \sum_{j\geq 2} < Ah_{T_0}, Ah_{T_j} > + \sum_{j\geq 2} < Ah_{T_1}, Ah_{T_j} > | \\ \leq \quad & \delta_{2s}(\|h_{T_0}\|_2 + \|h_{T_1}\|_2) \sum_{j\geq 2} \|h_{T_j}\|_2 \leq \sqrt{2}\delta_{2s}\|h_T\|_2 \sum_{j\geq 2} \|h_{T_j}\|_2 \\ \leq \quad & \sqrt{2}\delta_{2s}\|h_T\|_2 \frac{\|u_{T_0^c}\|_1}{\sqrt{s}} \end{aligned}$$

Preliminaries: Lemma 4

- Let A satisfies the RIP with order $2s$ with $\delta_{2s} < \sqrt{2}-1$. Let $x, \hat{x}$ be given and define $h = \hat{x} - x$. Let T_0 denote the index set corresponding to the s entries of x with largest magnitude. Define T_1 be the index set corresponding to the s entries of $h_{T_0^c}$. Set $T = T_0 \cup T_1$. If $\|\hat{x}\|_1 \leq \|x\|_1$. Then

$$\|h\|_2 \leq C_0 \frac{\|x - x_s\|_1}{\sqrt{s}} + C_1 \frac{|\langle Ah_T, Ah\rangle|}{\|h_T\|_2}$$

  where $C_0 = 2\frac{1-(1-\sqrt{2})\delta_{2s}}{1-(1+\sqrt{2})\delta_{2s}}$ and $C_1 = \frac{2}{1-(1+\sqrt{2})\delta_{2s}}$

Proof: Note that $h = h_T + h_{T^c}$, then $\|h\|_2 \leq \|h_T\|_2 + \|h_{T^c}\|_2$. Let T_j be defined similarly as Lemma 2, then we have

$$\|h_{T^c}\|_2 = \|\sum_{j\geq 2} h_{T_j}\|_2 \leq \sum_{j\geq 2} \|h_{T_j}\|_2 \leq \frac{\|h_{T_0^c}\|_1}{\sqrt{s}}$$

Continue: Proof Lemma 4

Since $\|\hat{x}\|_1 \leq \|x\|_1$, we obtain

$$\|x\|_1 \geq \|x_{T_0} + h_{T_0}\|_1 + \|x_{T_0^c} + h_{T_0^c}\|_1 \geq \|x_{T_0}\|_1 - \|h_{T_0}\|_1 + \|h_{T_0^c}\|_1 - \|x_{T_0^c}\|_1.$$

Rearranging and again applying the triangle inequality

$$\|h_{T_0^c}\|_1 \leq \|x\|_1 - \|x_{T_0}\|_1 + \|h_{T_0}\|_1 + \|x_{T_0^c}\|_1 \leq \|x - x_{T_0}\|_1 + \|h_{T_0}\|_1 + \|x_{T_0^c}\|_1.$$

Hence, we have $\|h_{T_0^c}\|_1 \leq \|h_{T_0}\|_1 + 2\|x - x_s\|_1$. Therefore,

$$\|h_{T^c}\|_2 \leq \frac{\|h_{T_0}\|_1 + 2\|x - x_s\|_1}{\sqrt{s}} \leq \|h_{T_0}\|_2 + \frac{2\|x - x_s\|_1}{\sqrt{s}}.$$

Since $\|h_{T_0}\|_2 \leq \|h_T\|_2$, we have

$$\|h\|_2 \leq 2\|h_T\|_2 + \frac{2\|x - x_s\|_1}{\sqrt{s}}$$

Continue: Proof Lemma 4

Lemma 3 gives

$$\begin{aligned}
\|h_T\|_2 &\leq \alpha\frac{\|h_{T_0^c}\|_1}{\sqrt{s}} + \beta\frac{|\langle Ah_T, Ah\rangle|}{\|h_T\|_2} \\
&\leq \alpha\frac{\|h_{T_0}\|_1 + 2\|x - x_s\|_1}{\sqrt{s}} + \beta\frac{|\langle Ah_T, Ah\rangle|}{\|h_T\|_2} \\
&\leq \alpha\|h_{T_0}\|_2 + 2\alpha\frac{\|x - x_s\|_1}{\sqrt{s}} + \beta\frac{|\langle Ah_T, Ah\rangle|}{\|h_T\|_2}
\end{aligned}$$

Using $\|h_{T_0}\|_2 \leq \|h_T\|_2$ gives

$$(1-\alpha)\|h_T\|_2 \leq 2\alpha\frac{\|x - x_s\|_1}{\sqrt{s}} + \beta\frac{|\langle Ah_T, Ah\rangle|}{\|h_T\|_2}.$$

Dividing by $1-\alpha$ gives

$$\|h\|_2 \leq \left(\frac{4\alpha}{1-\alpha} + 2\right)\frac{\|x - x_s\|_1}{\sqrt{s}} + \frac{2\beta}{1-\alpha}\frac{|\langle Ah_T, Ah\rangle|}{\|h_T\|_2}$$

Spark

First questions for finding the sparsest solution to $Ax = b$

- Can sparsest solution be unique? Under what conditions?
- Given a sparse x, how to verify whether it is actually the sparsest one?

Definition (Donoho and Elad 2003)

The spark of a given matrix A is the smallest number of columns from A that are linearly dependent, written as spark(A).

rank(A) is the largest number of columns from A that are linearly independent. In general, spark(A) $\neq$ rank(A) + 1; except for many randomly generated matrices.

Rank is easy to compute, but spark needs a combinatorial search.

Spark

Theorem (Gorodnitsky and Rao 1997)

If $Ax = b$ has a solution x obeying $\|x\|_0 < spark(A)/2$, then x is the sparsest solution.

- **Proof idea**: if there is a solution y to $Ax = b$ and $x - y \neq 0$, then $A(x - y) = 0$ and thus
$$\|x\|_0 + \|y\|_0 \geq \|x - y\|_0 \geq spark(A),$$
or $\|y\|_0 \geq spark(A) - \|x\|_0 > spark(A)/2 > \|x\|_0$
- The result does not mean this x can be efficiently found numerically.
- For many random matrices $A \in \mathbb{R}^{m \times n}$, the result means that if an algorithm returns x satisfying $\|x\|_0 < (m + 1)/2$, then x is optimal with probability 1.
- What to do when spark(A) is difficult to obtain?

General Recovery - Spark

- Rank is easy to compute, but spark needs a combinatorial search.
- However, for matrix with entries in general positions, spark(A) = rank(A)+1.
- For example, if matrix $A \in \mathbb{R}^{m \times n}$ $(m < n)$ has entries $A_{ij} \sim \mathcal{N}(0, 1)$, then $rank(A) = m = spark(A) - 1$ with probability 1.
- In general, any full rank matrix $A \in \mathbb{R}^{m \times n}$ $(m < n)$, any $m + 1$ columns of A is linearly dependent, so

$$spark(A) \leq m + 1 = rank(A) + 1$$

Coherence

Definition (Mallat and Zhang 1993)

The (mutual) coherence of a given matrix A is the largest absolute normalized inner product between different columns from A. Suppose $A = [a_1, a_2, \ldots, a_n]$. The mutual coherence of A is given by

$$\mu(A) = \max_{k,j,k\neq j} \frac{|a_k^\top a_j|}{\|a_k\|_2 \cdot \|a_j\|_2}$$

- It characterizes the dependence between columns of A
- For unitary matrices, $\mu(A) = 0$
- For recovery problems, we desire a small $\mu(A)$ as it is similar to unitary matrices.
- For $A = [\Psi, \Phi]$ where Φ and Ψ are $n \times n$ unitary, it holds $n^{-1/2} \leq \mu(A) \leq 1$. Note $\mu(A) = n^{-1/2}$ is achieved with [I, Fourier], [I, Hadamard]. ($|a_k^\top a_j| = 1, \|a_j\| = \sqrt{n}$)
- if $A \in \mathbb{R}^{m\times n}$ where n > m, then $\mu(A) \geq m^{-1/2}$

Coherence

Theorem (Donoho and Elad 2003)

$$spark(A) \geq 1 + \mu^{-1}(A)$$

Proof Sketch

- $\bar{A} \leftarrow A$ with columns normalized to unit 2-norm
- $p \leftarrow spark(A)$
- $B \leftarrow$ a $p \times p$ minor of $\bar{A}^\top \bar{A}$
- $|B_{ii}| = 1$ and $\sum_{j \neq i} |B_{ij}| \leq (p-1)\mu(A)$
- Suppose $p < 1 + \mu^{-1}(A) \Rightarrow 1 > (p-1)\mu(A) \Rightarrow |B_{ii}| > \sum_{j \neq i} |B_{ij}|, \forall i$
- Then $B \succ 0$ (Gershgorin circle theorem) $\Rightarrow spark(A) > p$. Contradiction.

Coherence-base guarantee

Corollary

If $Ax = b$ has a solution x obeying $\|x\|_0 < (1 + \mu^{-1}(A))/2$, then x is the unique sparsest solution.

Compare with the previous

Theorem (Gorodnitsky and Rao 1997)

If $Ax = b$ has a solution x obeying $\|x\|_0 < spark(A)/2$, then x is the sparsest solution.

- For $A \in \mathbb{R}^{m \times n}$ where m < n, $(1 + \mu^{-1}(A))$ is at most $1 + \sqrt{m}$ but spark can be $1 + m$. spark is more useful.
- Assume $Ax = b$ has a solution with $\|x\|_0 = k < spark(A)/2$. It will be the unique ℓ_0 minimizer. Will it be the ℓ_1 minimizer as well? Not necessarily. However, $\|x\|_0 < (1 + \mu^{-1}(A))/2$ is a sufficient condition.

Coherence-based $\ell_0 = \ell_1$

Theorem (Donoho and Elad 2003, Gribonval and Nielsen 2003)

If A has normalized columns and $Ax = b$ has a solution x satisfying $\|x\|_0 \leq (1 + \mu^{-1}(A))/2$, then x is the unique minimizer with respect to both ℓ_0 and ℓ_1.

Proof Sketch

- Previously we know x is the unique ℓ_0 minimizer; let $S := supp(x)$
- Suppose y is the ℓ_1 minimizer but not x; we study $h := y - x$
- h must satisfy $Ah = 0$ and $\|h\|_1 < 2\|h_S\|_1$ since $0 > \|y\|_1 - \|x\|_1 = \sum_{i \in S^c} |y_i| + \sum_{i \in S}(|y_i| - |x_i|) \geq \|h_{S^c}\|_1 - \sum_{i \in S} |y_i - x_i| = \|h_{S^c}\|_1 - \|h_S\|_1$
- $A^\top Ah = 0 \Rightarrow |h_j| \leq (1 + \mu(A))^{-1}\mu(A)\|h\|_1, \forall j$. (Expand $A^\top A$ and use $\|h\|_1 = \sum_{k \neq j} |h_k| + |h_j|$)
- the last two points together contradict the assumption

Result bottom line: allow $\|x\|_0$ up to $O(\sqrt{m})$ for exact recovery

The null space of A

- Definition: $\|x\|_p = (\sum_i |x_i|^p)^{1/p}$
- Lemma: Let $0 < p \leq 1$. If $\|(y-x)_{S^c}\|_p > \|(y-x)_S\|_p$, then $\|x\|_p < \|y\|_p$.
  Proof: Let $h = y - x$.
  $\|y\|_p^p = \|x+h\|_p^p = \|x_S + h_S\|_p^p + \|h_{S^c}\|_p^p =$
  $\|x\|_p^p + (\|h_{S^c}\|_p^p - \|h_S\|_p^p) + (\|x_S + h_S\|_p^p - \|x_S\|_p^p + \|h_S\|_p^p))$
  The last term is nonnegative for $0 < p \leq 1$. Hence, a sufficient condition is $\|h_{S^c}\|_p^p > \|h_S\|_p^p$.
- If the condition holds for $0 < p \leq 1$, it also holds for $q \in (0, p]$.
- **Definition** (null space property $NSP(k, \gamma)$). Every nonzero $h \in \mathcal{N}(A)$ satisfies $\|h_S\|_1 < \gamma \|h_{S^c}\|_1$ for all index sets S with $|S| \leq k$.

The null space of A

Theorem (Donoho and Huo 2001, Gribonval and Nielsen 2003)

$\min\ \|x\|_1,$ *s.t.* $Ax = b$ *uniquely recovers all k-sparse vectors* x^o *from measurements* $b = Ax^o$ *if and only if* A *satisfies* $NSP(k, 1)$.

Proof:

- Sufficiency: Pick any k-sparse vector x^o. Let $S := supp(x^o)$. For any non-zero $h \in \mathcal{N}(A)$, we have $A(x^o + h) = Ax^o = b$ and

$$\begin{aligned} \|x^0 + h\|_1 &= \|x^o_S + h_S\|_1 + \|h_{S^c}\|_1 \\ &\geq \|x^o_S\|_1 - \|h_S\|_1 + \|h_{S^c}\|_1 \\ &= \|x^o_S\|_1 - (\|h_S\|_1 - \|h_{S^c}\|_1) \end{aligned}$$

- Necessity. The inequality holds with equality if $\operatorname{sgn}(x^o_S) = -\operatorname{sgn}(h_S)$ and h_S has a sufficiently small scale. Therefore, basis pursuit to uniquely recovers all k-sparse vectors x^o, $NSP(k, 1)$ is also necessary.

The null space of A

- Another sufficient condition (Zhang [2008]) for $\|x\|_1 < \|y\|_1$ is

$$\|x\|_0 < \frac{1}{4}\left(\frac{\|y-x\|_1}{\|y-x\|_2}\right)^2$$

- Proof:

$$\|h_S\|_1 \le \sqrt{|S|}\|h_S\|_2 \le \sqrt{|S|}\|h\|_2 = \sqrt{\|x\|_0}\|h\|_2.$$

Then, the above inequality and the sufficient condition gives $\|y-x\|_1 > 2\|(y-x)_S\|_1$ which is $\|(y-x)_{S^c}\|_1 > \|(y-x)_S\|_1$.

Theorem (Zhang, 2008)

$\min \ \|x\|_1$, *s.t.* $Ax = b$ *recovers* x *uniquely if*

$$\|x\|_0 < \min\left\{\frac{1}{4}\left(\frac{\|h\|_1}{\|h\|_2}\right)^2, \quad h \in \mathcal{N}(A)\backslash\{0\}\right\}$$

The null space of A

- $1 \leq \frac{\|v\|_1}{\|v\|_2} \leq \sqrt{n}, \quad \forall v \in \mathbb{R}^n \setminus \{0\}$
- Garnaev and Gluskin established that for any natural number $p < n$, there exist p-dimensional subspaces $V_p \subset \mathbb{R}^n$ in which

$$\frac{\|v\|_1}{\|v\|_2} \geq \frac{C\sqrt{n-p}}{\sqrt{\log(n/(n-p))}}, \forall v \in V_p \setminus \{0\},$$

- vectors in the null space of A will satisfy, with high probability, the Garnaev and Gluskin inequality for $V_p = \mathrm{Null}(A)$ and $p = n - m$.
- for a random Gaussian matrix A, $\bar{x}$ will uniquely solve ℓ_1-min with high probability whenever

$$\|\bar{x}\|_0 < \frac{C^2}{4} \frac{m}{\log(n/m)}.$$

Formal equivalence

Suppose there is an s-sparse solution to $Ax = b$

- $\delta_{2s} < 1$ solution to combinatorial optimization (min ℓ_0) is unique
- $\delta_{2s} < 0.414$ solution to LP relaxation is unique **and the same**

Comments:

- RIP needs a matrix to be properly scaled
- the tight RIP constant of a given matrix A is difficult to compute
- the result is universal for all s-sparse
- $\exists$ tighter conditions (see next slide)
- all methods (including ℓ_0) require $\delta_{2s} < 1$ for universal recovery; every s-sparse x is unique if $\delta_{2s} < 1$
- the requirement can be satisfied by certain A (e.g., whose entries are i.i.d samples following a subgaussian distribution) and lead to exact recovery for $\|x\|_0 = O(m/log(m/k))$.

More Comments

- (Foucart-Lai) If $\delta_{2s+2} < 1$, then $\exists$ a sufficiently small p so that ℓ_p minimization is guaranteed to recovery any s-sparse x
- (Candes) $\delta_{2s} < \sqrt{2} - 1$ is sufficient
- (Foucart-Lai) $\delta_{2s} < 2(3 - \sqrt{2})/7 \sim 0.4531$ is sufficient
- RIP gives $\kappa(A_S) \leq \sqrt{(1+\delta_s)/(1-\delta_s)}$, $\forall |S| \leq k$. so $\delta_{2s} < 2(3 - \sqrt{2})/7$ gives $\kappa(A_S) \leq 1.7$, $\forall |S| \leq 2m$, very well-conditioned.
- (Mo-Li) $\delta_{2s} < 0.493$ is sufficient
- (Cai-Wang-Xu) $\delta_{2s} < 0.307$ is sufficient
- (Cai-Zhang) $\delta_{2s} < 1/3$ is sufficient and necessary for universal ℓ_1 recovery

Lecture: Algorithms for Compressed Sensing

Zaiwen Wen

Beijing International Center For Mathematical Research
Peking University

`http://bicmr.pku.edu.cn/~wenzw/bigdata2024.html`
wenzw@pku.edu.cn

Outline

- Proximal gradient method
- Accelerated gradient method
- Alternating direction methods of Multipliers (ADMM)
- Linearized Alternating direction methods of Multipliers
- Greedy methods
- Algorithm unrolling

ℓ_1-regularized least square problem

Consider

$$\min \ \psi_\mu(x) := \mu\|x\|_1 + \frac{1}{2}\|Ax - b\|_2^2$$

Approaches:

- Interior point method: l1_ls
- Spectral gradient method: GPSR
- Fixed-point continuation method: FPC
- Active set method: FPC_AS
- Alternating direction augmented Lagrangian method
- Nesterov's optimal first-order method
- many others

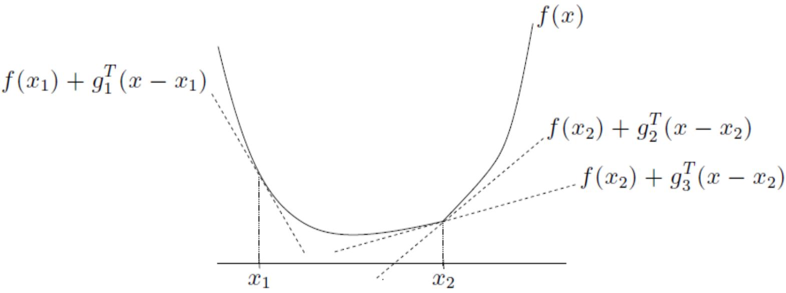
Subgradient

recall basic inequality for convex differentiable f:

$$f(y) \geq f(x) + \nabla f(x)^\top (y - x).$$

g is a subgradient of a convex function f at $x \in \mathrm{dom} f$ if

$$f(y) \geq f(x) + g^\top (y - x), \forall y \in \mathrm{dom} f.$$

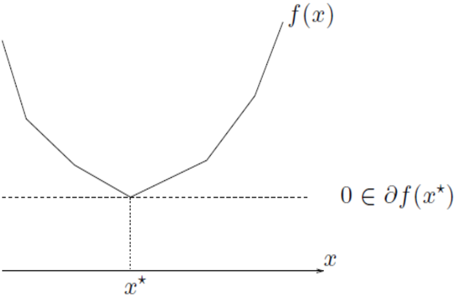

$f(x)$
$f(x_1) + g_1^T(x - x_1)$
$f(x_2) + g_2^T(x - x_2)$
$f(x_2) + g_3^T(x - x_2)$
x_1
x_2

g_2, g_3 are subgradients at x_2, g_1 is a subgradient at x_1.

Optimality conditions — unconstrained

x^* minimizes $f(x)$ if and only

$$0 \in \partial f(x^*)$$



Proof: by definition

$$f(y) \geq f(x^*) + 0^\top (y - x^*) \text{ for all } y \quad \Longleftrightarrow 0 \in \partial f(x^*).$$

Optimality conditions — constrained

$$\begin{aligned} \min \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \leq 0, i = 1, \ldots, m. \end{aligned}$$

From **Lagrange duality**: if strong duality holds, then x^*, λ^* are primal, dual optimal if and only if

- x^* is primal feasible
- $\lambda^* \geq 0$
- complementary: $\lambda_i^* f_i(x^*) = 0$ for $i = 1, \ldots, m$
- x^* is a minimizer of $\min \ \mathcal{L}(x, \lambda^*) = f_0(x) + \sum_i \lambda_i^* f_i(x)$, i.e.,

$$0 \in \partial_x \mathcal{L}(x, \lambda^*) = \partial f_0(x^*) + \sum_i \lambda_i^* \partial f_i(x^*)$$

Proximal Gradient Method

Let $f(x) = \frac{1}{2}\|Ax - b\|_2^2$. The gradient $\nabla f(x) = A^\top(Ax - b)$. Consider

$$\min \ \psi_\mu(x) := \mu\|x\|_1 + f(x).$$

- First-order approximation + proximal term:

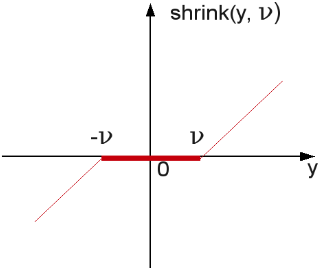
$$\begin{aligned} x^{k+1} \quad &:= \quad \arg\min_{x\in\mathbb{R}^n} \ \mu\|x\|_1 + (\nabla f(x^k))^\top(x - x^k) + \frac{1}{2\tau}\|x - x^k\|_2^2 \\ &= \quad \arg\min_{x\in\mathbb{R}^n} \ \mu\|x\|_1 + \frac{1}{2\tau}\|x - (x^k - \tau\nabla f(x^k))\|_2^2 \\ &= \quad \text{shrink}(x^k - \tau\nabla f(x^k), \mu\tau) \end{aligned}$$

- gradient step: bring in candidates for nonzero components
- shrinkage step: eliminate some of them by “soft” thresholding

Shrinkage (soft thresholding)

$$\begin{aligned}\text{shrink}(y,\nu): &= \arg\min_{x\in\mathbb{R}} \nu\|x\|_1 + \frac{1}{2}\|x-y\|_2^2 \\ &= \text{sgn}(y)\max(|y|-\nu,0) \\ &= \begin{cases} y-\nu\text{sgn}(y), & \text{if } |y|>\nu \\ 0, & \text{otherwise} \end{cases}\end{aligned}$$

- Chambolle, Devore, Lee and Lucier
- Figueirdo, Nowak and Wright
- Elad, Matalon and Zibulevsky
- Hales, Yin and Zhang
- Darbon, Osher
- Many others



Proximal gradient method For General Problems

Consider the model

$$\min \ F(x) := f(x) + h(x)$$

- $f(x)$ is convex, differentiable
- $h(x)$ is convex but may be nondifferentiable

General scheme: linearize $f(x)$ and add a proximal term:

$$\begin{aligned} x^{k+1} &:= \arg\min_{x\in\mathbb{R}^n} \ h(x) + (\nabla f(x^k))^\top (x - x^k) + \frac{1}{2\tau}\|x - x^k\|_2^2 \\ &= \arg\min_{x\in\mathbb{R}^n} \ \tau h(x) + \frac{1}{2}\|x - (x^k - \tau\nabla f(x^k))\|_2^2 \\ &= \mathrm{prox}_{\tau h}(x^k - \tau\nabla f(x^k)) \end{aligned}$$

Proximal Operator

$$\mathrm{prox}_h(y) := \arg\min_x \ h(x) + \frac{1}{2}\|x - y\|_2^2.$$

Convergence of proximal gradient method

to minimize $f + h$, choose x^0 and repeat

$$x^k = \mathrm{prox}_{t_k h}\left(x^{k-1} - t\nabla f(x^{k-1})\right), \quad k \geq 1$$

assumptions

- f convex with $\mathbf{dom}\, g = \mathbf{R}^n$; ∇f Lipschitz continuous with constant L:

$$\|\nabla f(x) - \nabla f(y)\|_2 \leq L\|x - y\|_2 \quad \forall x, y$$

- h is closed and convex (so that prox_{th} is well defined)
- optimal value F^* is finite and attained at x^* (not necessarily unique)

convergence result: $1/k$ rate convergence with fixed step size $t_k = 1/L$

Gradient map

$$G_t(x) = \frac{1}{t}(x - \mathrm{prox}_{th}(x - t\nabla f(x)))$$

$G_t(x)$ is the negative 'step' in the proximal gradient update

$$\begin{aligned} x^+ &= \mathrm{prox}_{th}(x - t\nabla f(x)) \\ &= x - tG_t(x) \end{aligned}$$

- $G_t(x)$ is not a gradient or subgradient of $F = g + h$
- from subgradient definition of prox-operator

$$G_t(x) \in \partial f(x) + \partial h(x - tG_t(x))$$

- $G_t(x) = 0$ if and only if x minimizes $F(x) = f(x) + h(x)$

Consequences of Lipschitz assumption

recall upper bound (lecture on "gradient method") for convex f with Lipschitz continuous gradient

$$f(y) \leq f(x) + \nabla f(x)^\top (y - x) + \frac{L}{2}\|y - x\|_2^2 \quad \forall x, y$$

- substitute $y = x - tG_t(x)$:

$$f(x - tG_t(x)) \leq f(x) - t\nabla f(x)^\top G_t(x) + \frac{t^2 L}{2}\|G_t(x)\|_2^2$$

- if $0 < t \leq 1/L$, then

$$f(x - tG_t(x)) \leq f(x) - t\nabla f(x)^\top G_t(x) + \frac{t}{2}\|G_t(x)\|_2^2 \qquad (1)$$

A global inequality

if the inequality (1) holds, then for all z,

$$F(x - tG_t(x)) \leq F(x) + G_t(x)^\top (x - z) - \frac{t}{2}\|G_t(x)\|_2^2 \qquad (2)$$

proof: (define $v = G_t(x) - \nabla f(x)$)

$$\begin{aligned} F(x - tG_t(x)) &\leq f(x) - t\nabla f(x)^\top G_t(x) + \frac{t}{2}\|G_t(x)\|_2^2 + h(x - tG_t(x)) \\ &\leq f(z) + \nabla f(x)^\top (x - z) - t\nabla f(x)^\top G_t(x) + \frac{t}{2}\|G_t(x)\|_2^2 \\ &+ h(z) + v^\top (x - z - tG_t(x)) \\ &= f(z) + h(z) + G_t(x)^\top (x - z) - \frac{t}{2}\|G_t(x)\|_2^2 \end{aligned}$$

line 2 follows from convexity of f and h, and $v \in \partial h(x - tG_t(x))$

Progress in one iteration

$$x^+ = x - tG_t(x)$$

- inequality (2) with $z = x$ shows the algorithm is a descent method:

$$F(x^+) \leq F(x) - \frac{t}{2}\|G_t(x)\|_2^2$$

- inequality (2) with $z = x^*$

$$\begin{aligned} F(x^+) - F^* &\leq G_t(x)^\top (x - x^*) - \frac{t}{2}\|G_t(x)\|_2^2 \\ &= \frac{1}{2t}\left(\|x - x^*\|_2^2 - \|x - x^* - tG_t(x)\|_2^2\right) \qquad (3) \\ &= \frac{1}{2t}\left(\|x - x^*\|_2^2 - \|x^+ - x^*\|_2^2\right) \end{aligned}$$

(hence, $\|x^+ - x^*\|_2^2 \leq \|x - x^*\|_2^2$, *i.e.*, distance to optimal set decreases)

Analysis for fixed step size

add inequalities (3) for $x = x^{i-1}, x^+ = x^i, t = t_i = 1/L$

$$
\begin{aligned}
\sum_{i=1}^{k}(F(x^i) - F^*) &\le \frac{1}{2t}\sum_{i=1}^{k}\left(\|x^{i-1} - x^*\|_2^2 - \|x^i - x^*\|_2^2\right) \\
&= \frac{1}{2t}\left(\|x^0 - x^*\|_2^2 - \|x^k - x^*\|_2^2\right) \\
&\le \frac{1}{2t}\|x^0 - x^*\|_2^2
\end{aligned}
$$

since $f(x^i)$ is nonincreasing,

$$
F(x^k) - F^* \le \frac{1}{k}\sum_{i=1}^{k}(F(x^i) - F^*) \le \frac{1}{2kt}\|x^0 - x^*\|_2^2
$$

conclusion: reaches $F(x^k) - F^* \le \epsilon$ after $O(1/\epsilon)$ iterations

Outline: Accelerated Gradient Method

- Amir Beck and Marc Teboulle, *A fast iterative shrinkage thresholding algorithm for linear inverse problems*
- Paul Tseng, *On accelerated proximal gradient methods for convex-concave optimization*
- Paul Tseng, *Approximation accuracy, gradient methods and error bound for structured convex optimization*

FISTA: Accelerated proximal gradient

Consider the model

$$\min \ F(x) := f(x) + h(x).$$

Given $t = 1/L$, $y^1 = x_0$ and $\gamma^1 = 1$, compute:

$$\begin{aligned} x^k &= \mathrm{prox}_{th}(y^k - t\nabla f(y^k)) \\ \gamma_{k+1} &= \frac{1+\sqrt{1+4\gamma_k^2}}{2} \\ y^{k+1} &= x^k + \frac{\gamma_k - 1}{\gamma_{k+1}}(x^k - x^{k-1}) \end{aligned}$$

Complexity results:

$$F(x^k) - F(x^*) \le \frac{2L\|x^0 - x^*\|_2^2}{(k+1)^2}$$

APG Variant 1

Acclerated proximal gradient (APG):
Set $x^{-1} = x^0$ and $\theta_{-1} = \theta_0 = 1$:

$$\begin{aligned} y^k &= x^k + \theta_k(\theta_{k-1}^{-1} - 1)(x^k - x^{k-1}) \\ x^{k+1} &= \mathrm{prox}_{th}(y^k - t\nabla f(y^k)) \\ \theta_{k+1} &= \frac{\sqrt{\theta_k^4 + 4\theta_k^2} - \theta_k^2}{2} \end{aligned}$$

Question: what is the difference between θ_k and γ_k? Show $\theta_k \leq \frac{2}{k+2}$ for all k.

Complexity:

$$F(x^k) - F(x^*) \leq \frac{4L}{(k+1)^2} \|x^* - x^0\|_2^2$$

APG Variant 2

Another version of APG:

$$\begin{aligned}
y^k &= (1-\theta_k)x^k + \theta_k z^k \\
z^{k+1} &= \operatorname{prox}_{th}(z^k - t\nabla f(y^k)) \\
x^{k+1} &= (1-\theta_k)x^k + \theta_k z^{k+1} \\
\theta_{k+1} &= \frac{\sqrt{\theta_k^4 + 4\theta_k^2} - \theta_k^2}{2}
\end{aligned}$$

y^k is a convex combination of x^k and z^k,
x^{k+1} is a convex combination of x^k and z^{k+1}.

Complexity:

$$F(x^k) - F(x^*) \leq \frac{4L}{(k+1)^2}\|x^* - z^0\|_2^2$$

Outline: ADMM

- Alternating direction augmented Lagrangian methods
- Variable splitting method
- Convergence for problems with two blocks of variables

References:

- Wotao Yin, Stanley Osher, Donald Goldfarb, Jerome Darbon, *Bregman Iterative Algorithms for l1-Minimization with Applications to Compressed Sensing*
- Junfeng Yang, Yin Zhang, *Alternating direction algorithms for l1-problems in Compressed Sensing*
- Tom Goldstein, Stanely Osher, *The Split Bregman Method for L1-Regularized Problems*
- B.S. He, H. Yang, S.L. Wang, *Alternating Direction Method with Self-Adaptive Penalty Parameters for Monotone Variational Inequalities*

Basis pursuit problem

$$\begin{aligned} \text{Primal:} \quad & \min \quad \|x\|_1, \ \ \text{s.t. } Ax = b \\ \text{Dual:} \quad & \max \quad b^\top \lambda, \ \ \text{s.t. } \|A^\top \lambda\|_\infty \le 1 \end{aligned}$$

The dual problem is equivalent to

$$\max \ b^\top \lambda, \ \ \text{s.t. } A^\top \lambda = s, \ \ \|s\|_\infty \le 1.$$

Augmented Lagrangian (Bregman) framework

Augmented Lagrangian function:

$$\mathcal{L}(\lambda, s, x) := -b^\top \lambda + x^\top (A^\top \lambda - s) + \frac{1}{2\mu}\|A^\top \lambda - s\|^2$$

Algorithmic framework

- Compute λ^{k+1} and s^{k+1} at k-th iteration

  (DL) $\quad \min_{\lambda,s} \mathcal{L}(\lambda, s, x^k),\;$ s.t. $\|s\|_\infty \le 1$
- Update the Lagrangian multiplier:

$$x^{k+1} = x^k + \frac{A^\top \lambda^{k+1} - s^{k+1}}{\mu}$$

Pros and Cons:

- Pros: rich theory, well understood and a lot of algorithms
- Cons: $\mathcal{L}(\lambda, s, x^k)$ is not separable in λ and s, and the subproblem (DL) is difficult to minimize

An alternating direction minimization scheme

- Divide variables into different blocks according to their roles
- Minimize the augmented Lagrangian function with respect to one block at a time while all other blocks are fixed

ADMM

$$\begin{aligned} \lambda^{k+1} &= \arg\min_{\lambda} \ \mathcal{L}(\lambda, s^k, x^k) \\ s^{k+1} &= \arg\min_{s} \ \mathcal{L}(\lambda^{k+1}, s, x^k), \quad \text{s.t. } \|s\|_\infty \le 1 \\ x^{k+1} &= x^k + \frac{A^\top \lambda^{k+1} - s^{k+1}}{\mu} \end{aligned}$$

An alternating direction minimization scheme

Explicit solutions:

$$
\begin{aligned}
\lambda^{k+1} &= (AA^\top)^{-1}\left(\mu(Ax^k - b) + As^k\right)\\
s^{k+1} &= \arg\min \ \|s - A^\top\lambda^{k+1} - \mu x^k\|^2, \ \ \text{s.t. } \|s\|_\infty \le 1\\
&= \mathcal{P}_{[-1,1]}(A^\top\lambda^{k+1} + \mu x^k)\\
x^{k+1} &= x^k + \frac{A^\top\lambda^{k+1} - s^{k+1}}{\mu}
\end{aligned}
$$

ADMM for BP-denoising

Primal:

$$\min \ \|x\|_1, \text{ s.t. } \|Ax-b\|_2 \le \sigma$$

which is equivalent to

$$\min \ \|x\|_1, \text{ s.t. } Ax-b+r=0, \ \|r\|_2 \le \sigma$$

Lagrangian function:

$$\begin{aligned}\mathcal{L}(x,r,\lambda) &:= \|x\|_1 - \lambda^\top(Ax-b+r) + \pi(\|r\|_2-\sigma) \\ &= \|x\|_1 - (A^\top\lambda)^\top x + \pi\|r\|_2 - \lambda^\top r + b^\top\lambda - \pi\sigma\end{aligned}$$

Hence, the dual problem is:

$$\max b^\top\lambda - \pi\sigma, \ \text{ s.t. } \|A^\top\lambda\|_\infty \le 1, \ \|\lambda\|_2 \le \pi$$

which is equivalent to

$$\max b^\top\lambda - \sigma\|\lambda\|_2, \ \text{ s.t. } \|A^\top\lambda\|_\infty \le 1$$

ADMM for BP-denoising

The dual problem is equivalent to:

$$\max b^\top \lambda - \sigma\|u\|_2, \quad \text{s.t. } A^\top \lambda = s, \ \|s\|_\infty \le 1, \lambda = u$$

Augmented Lagrangian function is:

$$\mathcal{L} = -b^\top \lambda + \sigma\|u\|_2 + x^\top (A^\top \lambda - s) + \frac{1}{2\mu}\|A^\top \lambda - s\|^2 + \pi^\top (\lambda - u) + \frac{1}{2\mu}\|\lambda - u\|^2$$

ADMM scheme:

$$\begin{aligned}
\lambda^{k+1} &= \arg\min_\lambda \ \frac{1}{2\mu}\|A^\top \lambda - s^k\|^2 + (Ax^k - b + \pi^k)^\top \lambda + \frac{1}{2\mu}\|\lambda - u^k\|^2, \\
u^{k+1} &= \arg\min_u \ \sigma\|u\|_2 + (\pi^k)^\top (\lambda^{k+1} - u) + \frac{1}{2\mu}\|\lambda^{k+1} - u\|^2, \\
s^{k+1} &= \arg\min_s \ \|s - A^\top \lambda^{k+1} - \mu x^k\|^2, \ \text{s.t. } \|s\|_\infty \le 1 \\
&= \mathcal{P}_{[-1,1]}(A^\top \lambda^{k+1} + \mu x^k) \\
x^{k+1} &= x^k + \frac{A^\top \lambda^{k+1} - s^{k+1}}{\mu}, \quad \pi^{k+1} = \pi^k + \frac{1}{\mu}(\lambda^{k+1} - u^{k+1})
\end{aligned}$$

ADMM for ℓ_1-regularized problem

Primal:

$$\min \ \mu\|x\|_1 + \frac{1}{2}\|Ax-b\|_2^2$$

which is equivalent to

$$\min \ \mu\|x\|_1 + \frac{1}{2}\|r\|_2^2, \ \text{ s.t. } Ax-b=r.$$

Lagrangian function:

$$\begin{aligned}\mathcal{L}(x,r,\lambda) \ &:= \ \mu\|x\|_1 + \frac{1}{2}\|r\|_2^2 - \lambda^\top(Ax-b-r) \\ &= \ \mu\|x\|_1 - (A^\top\lambda)^\top x + \frac{1}{2}\|r\|_2^2 + \lambda^\top r + b^\top\lambda\end{aligned}$$

Hence, the dual problem is:

$$\max b^\top\lambda - \frac{1}{2}\|\lambda\|^2, \ \text{ s.t. } \|A^\top\lambda\|_\infty \le \mu$$

ADMM for ℓ_1-regularized problem

The dual problem is equivalent to

$$\max\ b^\top \lambda - \frac{1}{2}\|\lambda\|^2, \ \ \text{s.t. } A^\top \lambda = s, \ \ \|s\|_\infty \le \mu.$$

Augmented Lagrangian function is:

$$\mathcal{L}(\lambda, s, x) := -b^\top \lambda + \frac{1}{2}\|\lambda\|^2 + x^\top (A^\top \lambda - s) + \frac{1}{2\mu}\|A^\top \lambda - s\|^2$$

ADMM scheme:

$$\begin{aligned}
\lambda^{k+1} &= (AA^\top + \mu I)^{-1} \left(\mu(Ax^k - b) + As^k\right) \\
s^{k+1} &= \arg\min\ \|s - A^\top \lambda^{k+1} - \mu x^k\|^2, \ \ \text{s.t. } \|s\|_\infty \le \mu \\
&= \mathcal{P}_{[-\mu,\mu]}(A^\top \lambda^{k+1} + \mu x^k) \\
x^{k+1} &= x^k + \frac{A^\top \lambda^{k+1} - s^{k+1}}{\mu}
\end{aligned}$$

YALL1

Derive ADMM for the following problems:

$$\begin{array}{rll} \text{BP:} & \min_{x\in\mathbb{C}^n} & \|Wx\|_{w,1}, \ \text{ s.t. } Ax=b \\ \text{L1/L1:} & \min_{x\in\mathbb{C}^n} & \|Wx\|_{w,1}+\frac{1}{\nu}\|Ax-b\|_1 \\ \text{L1/L2:} & \min_{x\in\mathbb{C}^n} & \|Wx\|_{w,1}+\frac{1}{2\rho}\|Ax-b\|_2^2 \\ \text{BP+:} & \min_{x\in\mathbb{R}^n} & \|x\|_{w,1}, \ \text{ s.t. } Ax=b, \ x\geq 0 \\ \text{L1/L1+:} & \min_{x\in\mathbb{R}^n} & \|x\|_{w,1}+\frac{1}{\nu}\|Ax-b\|_1, \ \text{ s.t. } x\geq 0 \\ \text{L1/L2+:} & \min_{x\in\mathbb{R}^n} & \|x\|_{w,1}+\frac{1}{2\rho}\|Ax-b\|_2^2, \ \text{ s.t. } x\geq 0 \end{array}$$

$\nu, \rho \geq 0$, $A \in \mathbb{C}^{m\times n}$, $b \in \mathbb{C}^m$,$x \in \mathbb{C}^n$ for the first three and $x \in \mathbb{R}^n$ for the last three, $W \in \mathbb{C}^{n\times n}$ is an unitary matrix serving as a sparsifying basis, and $\|x\|_{w,1} := \sum_{i=1}^n w_i|x_i|$.

Variable splitting

Given $A \in \mathbb{R}^{m\times n}$, consider $\min\; f(x) + g(Ax)$, which is

$$\min\; f(x) + g(y), \quad \text{s.t. } Ax = y$$

Augmented Lagrangian function:

$$\mathcal{L}(x, y, \lambda) = f(x) + g(y) - \lambda^\top (Ax - y) + \frac{1}{2\mu}\|Ax - y\|_2^2$$

ADMM

$$\begin{aligned}
(P_x):&\quad x^{k+1} := \arg\min_{x\in\mathcal{X}} \quad \mathcal{L}(x, y^k, \lambda^k),\\
(P_y):&\quad y^{k+1} := \arg\min_{y\in\mathcal{Y}} \quad \mathcal{L}(x^{k+1}, y, \lambda^k),\\
(P_\lambda):&\quad \lambda^{k+1} := \lambda^k - \gamma \frac{Ax^{k+1} - y^{k+1}}{\mu}
\end{aligned}$$

Variable splitting

split Bregman (Goldstein and Osher) for anisotropic TV:

$$\min\ \alpha\|Du\|_1 + \beta\|\Psi u\|_1 + \frac{1}{2}\|Au - f\|_2^2$$

Introduce $y = Du$ and $w = \Psi u$, obtain

$$\min\ \alpha\|y\|_1 + \beta\|w\|_1 + \frac{1}{2}\|Au - f\|_2^2, \ \text{ s.t. } y = Du,\ w = \Psi u$$

Augmented Lagrangian function:

$$\begin{aligned} \mathcal{L} \quad := \quad & \alpha\|y\|_1 + \beta\|w\|_1 + \frac{1}{2}\|Au - f\|_2^2 - p^\top(Du - y) + \frac{1}{2\mu}\|Du - y\|_2^2 \\ & -q^\top(\Psi u - w) + \frac{1}{2\mu}\|\Psi u - w\|_2^2 \end{aligned}$$

Variable splitting

- The variable u can be otained by

$$\left(A^\top A + \frac{1}{\mu}(D^\top D + I)\right) u = A^\top f + \frac{1}{\mu}(D^\top y + \Psi^\top w) + D^\top p + \Psi^\top q$$

  If A and D are diagonalizable by FFT, then the computational cost is very cheap. For example, $A = R\mathcal{F}$, both R and D are circulant matrices.

- Variables y and w:

$$\begin{aligned} y &:= \mathcal{S}(Du - \mu p, \alpha\mu) \\ w &:= \mathcal{S}(\Psi u - \mu q, \alpha\mu) \end{aligned}$$

- apply a few iterations before updating the Lagrangian multipliers p and q

Exercise: isotropic TV

$$\min \ \alpha\|Du\|_2 + \beta\|\Psi u\|_1 + \frac{1}{2}\|Au - f\|_2^2$$

FTVd: Fast TV deconvolution

Wang-Yang-Yin-Zhang consider:

$$\min_{u} \sum \|D_i u\|_2 + \frac{1}{2\mu}\|Ku - f\|_2^2$$

Introducing w and quadratic penalty:

$$\min_{u,w} \sum \left(\|w_i\|_2 + \frac{1}{2\beta}\|w_i - D_i u\|_2^2 \right) + \frac{1}{2\mu}\|Ku - f\|_2^2$$

Alternating minimization:

- For fixed u, $\{w_i\}$ can be solved by shrinkage at $O(N)$
- For fixed $\{w_i\}$, u can be solved by FFT at $O(N \log N)$

Outline: Linearized ADMM

- Linearized Bregman and Bregmanized operator splitting
- ADMM + proximal point method
- Xiaoqun Zhang, Martin Burgerz, Stanley Osher, *A unified primal-dual algorithm framework based on Bregman iteration*

Review of Bregman method

Consider BP:

$$\min \ \|x\|_1, \ \ \text{s.t. } Ax = b$$

Bregman method:

- $D_J^{p^k}(x, x^k) := \|x\|_1 - \|x^k\|_1 - \langle p^k, x - x^k \rangle$
- $x^{k+1} := \arg\min_x \ \mu D_J^{p^k}(x, x^k) + \frac{1}{2}\|Ax - b\|_2^2$
- $p^{k+1} = p^k + \frac{1}{\mu} A^\top (b - Ax^{k+1})$

Augmented Lagrangian (updating multiplier or b):

- $x^{k+1} := \arg\min_x \ \mu\|x\|_1 + \frac{1}{2}\|Ax - b^k\|_2^2$
- $b^{k+1} = b + (b^k - Ax^{k+1})$

They are equivalent, see Yin-Osher-Goldfarb-Darbon

Linearized approaches

Linearized Bregman method:

$$\begin{aligned} x^{k+1} &:= \arg\min\ \mu D_J^{p^k}(x, x^k) + (A^\top(Ax^k - b))^\top(x - x^k) + \frac{1}{2\delta}\|x - x^k\|_2^2, \\ p^{k+1} &:= p^k - \frac{1}{\mu\delta}(x^{k+1} - x^k) - \frac{1}{\mu}A^\top(Ax^k - b), \end{aligned}$$

which is equivalent to

$$\begin{aligned} x^{k+1} &:= \arg\min\ \mu\|x\|_1 + \frac{1}{2\delta}\|x - v^k\|_2^2 \\ v^{k+1} &:= v^k - \delta A^\top(Ax^{k+1} - b) \end{aligned}$$

Bregmanized operator splitting:

$$\begin{aligned} x^{k+1} &:= \arg\min\ \mu\|x\|_1 + (A^\top(Ax^k - b^k))^\top(x - x^k) + \frac{1}{2\delta}\|x - x^k\|_2^2 \\ b^{k+1} &= b + (b^k - Ax^{k+1}) \end{aligned}$$

Are they equivalent?

Linearized approaches

Linearized Bregman method:

$$\begin{aligned} x^{k+1} &:= \arg\min \ \mu D_J^{p^k}(x, x^k) + (A^\top(Ax^k - b))^\top(x - x^k) + \frac{1}{2\delta}\|x - x^k\|_2^2, \\ p^{k+1} &:= p^k - \frac{1}{\mu\delta}(x^{k+1} - x^k) - \frac{1}{\mu}A^\top(Ax^k - b), \end{aligned}$$

which is equivalent to

$$\begin{aligned} x^{k+1} &:= \mathcal{S}(v^k, \mu\delta) \\ v^{k+1} &:= v^k - \delta A^\top(Ax^{k+1} - b) \end{aligned} \quad \text{or} \quad \begin{aligned} x^{k+1} &:= \mathcal{S}(\delta A^\top b^k, \mu\delta) \\ b^{k+1} &:= b + (b^k - Ax^{k+1}) \end{aligned}$$

Bregmanized operator splitting:

$$\begin{aligned} x^{k+1} &:= \mathcal{S}(x^k - \delta(A^\top(Ax^k - b^k)), \mu\delta) = \mathcal{S}(\delta A^\top b^k + x^k - \delta A^\top A x^k, \mu\delta) \\ b^{k+1} &= b + (b^k - Ax^{k+1}) \end{aligned}$$

Linearized approaches

Linearized Bregman:

- If the sequence x^k converges and p^k is bounded, then the limit of x^k is the unique solution of

$$\min\ \mu\|x\|_1 + \frac{1}{2\delta}\|x\|_2^2 \text{ s.t. } Ax = b.$$

- For μ large enough, the limit solution solves BP.
- Exact regularization if $\delta > \bar{\delta}$

What about Bregmanized operator splitting?

Primal ADMM for ℓ_1-regularized problem

Primal: $\min \ \mu\|x\|_1 + \frac{1}{2}\|Ax-b\|_2^2$ which is equivalent to

$$\min \ \mu\|x\|_1 + \frac{1}{2}\|r\|_2^2, \ \text{s.t. } Ax - b = r.$$

Augmented Lagrangian function:

$$\mathcal{L}(x,r,\lambda) = \mu\|x\|_1 + \frac{1}{2}\|r\|_2^2 - \lambda^\top(Ax-b-r) + \frac{1}{2\delta}\|Ax-b-r\|_2^2$$

ADMM scheme:

$$\begin{aligned} x^{k+1} &= \arg\min_x \ \mu\|x\|_1 + \frac{1}{2\delta}\|Ax-b-r^k-\delta\lambda^k\|_2^2 \quad \text{original problem} \\ r^{k+1} &= \arg\min_r \ \frac{1}{2}\|r\|_2^2 + \frac{1}{2\delta}\|Ax^{k+1}-b-r-\delta\lambda^k\|_2^2 \\ \lambda^{k+1} &= \lambda^k + \frac{Ax^{k+1}-b-r^{k+1}}{\delta} \end{aligned}$$

Primal ADMM for ℓ_1-regularized problem

Primal: $\min \ \mu\|x\|_1 + \frac{1}{2}\|Ax-b\|_2^2$ which is equivalent to

$$\min \ \mu\|x\|_1 + \frac{1}{2}\|r\|_2^2, \ \text{ s.t. } Ax - b = r.$$

Augmented Lagrangian function:

$$\mathcal{L}(x,r,\lambda) = \mu\|x\|_1 + \frac{1}{2}\|r\|_2^2 - \lambda^\top(Ax-b-r) + \frac{1}{2\delta}\|Ax-b-r\|_2^2$$

ADMM scheme:

$$\begin{aligned}
x^{k+1} &= \arg\min_x \ \mu\|x\|_1 + (g^k)^\top(x-x^k) + \frac{1}{2\tau}\|x-x^k\|_2^2 \\
r^{k+1} &= \arg\min_r \ \frac{1}{2}\|r\|_2^2 + \frac{1}{2\delta}\|Ax^{k+1} - b - r - \delta\lambda^k\|_2^2 \\
\lambda^{k+1} &= \lambda^k + \frac{Ax^{k+1} - b - r^{k+1}}{\delta}
\end{aligned}$$

Convergence of the linearized scheme?

Outline: Greedy Methods

- Orthogonal matching pursuit
- CoSaOMP

Orthogonal Matching Pursuit, OMP

- $g^k = A^\top(Ax^{k-1} - b)$
- $x^k = \mathrm{argmin}_x\{\|Ax - b\|_2 : \mathrm{supp}(x) \subseteq \mathcal{S}^k\}$. 如果矩阵$A$满足相关RIP条件，则$A_{\mathcal{S}^k}^\top A_{\mathcal{S}^k}$实际上是可逆的. 则等价于$\mathrm{argmin}_x\{\|A_{\mathcal{S}^k} x_{\mathcal{S}^k} - b\|_2 : \mathrm{supp}(x) \subseteq \mathcal{S}^k\}$. 显式解为$x_{\mathcal{S}^k} = (A_{\mathcal{S}^k}^\top A_{\mathcal{S}^k})^{-1} A_{\mathcal{S}^k}^\top b$

Algorithm 1 OMP算法框架

```
输入：$A, b, x^0 \in \mathbb{R}^n, \mathcal{S}^0 = \oslash, k = 1,$最大迭代次数$k_{\max}$.
while $k < k_{\max}$ do
    计算$r^k = Ax^{k-1} - b$.
    计算$g^k = A^\top r^k$.
    计算$\mathcal{S}^k = \mathcal{S}^{k-1} \cup \mathrm{argmax}_i |g_i^k|$.
    计算$x^k = \mathrm{argmin}_x\{\|Ax - b\|_2 : \mathrm{supp}(x) \subseteq \mathcal{S}^k\}$.
    $k \leftarrow k + 1\cdot$
end while
```

CoSaOMP

- $g^k_{2s} = \operatorname{argmin}\{\|x - g^k\|_2 : \|x\|_0 \le 2s\}$是$g^k$的$2s$-逼近

Algorithm 2 CoSaOMP算法框架

```
输入：$A, r^0, x^0 \in \mathbb{R}^n, \mathcal{S}^0 = \emptyset, k = 1$,终止条件$\varepsilon$.
while $\|r^k\| \le \varepsilon$ do
    计算$r^k = Ax^{k-1} - b$.
    计算$g^k = A^\top r^k$.
    计算$\mathcal{S}^k = \operatorname{supp}(x^{k-1}) \cup \operatorname{supp}(g^k_{2s})$.
    计算$c = \operatorname{argmin}_x\{\|Ax - b\|_2 : \operatorname{supp}(x) \subseteq \mathcal{S}^k\}$.
    计算$x^k = c_s$.
    $k \leftarrow k + 1$.
end while
```

Outline: Algorithm Unrolling

- A Brief Introduction to Algorithm Unrolling
- Learned ISTA

Algorithm Unrolling (AU)

AU consists of two steps

- Pick a classic iteration and unroll it to an Neural Network (NN)
- Select a set of NN parameters to learn

LASSO example: assume $b = Ax^{\text{true}} + \text{noise}$; recover x^{true} by

$$x^{\text{lasso}} \leftarrow \underset{x}{\text{minimize}} \frac{1}{2}\|Ax - b\|_2^2 + \lambda\|x\|_1$$

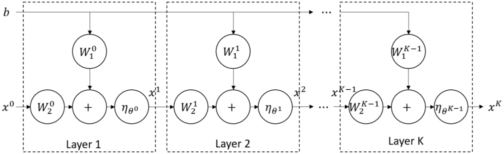
Iterative soft-thresholding algorithm (ISTA):

$$x^{k+1} = \eta_{\lambda\alpha}\left(x^k - \alpha A^T\left(Ax^k - b\right)\right)$$

- convergence requires a proper stepsize α or line search
- the gradient-descent step reduces $\frac{1}{2}\|Ax - b\|^2$
- the soft-thresholding step $\eta_{\lambda\alpha}(\cdot)$ reduces $\lambda\|x\|_1$

Unrolled ISTA

- Introduce scalar $\theta = \lambda\alpha$ and matrices $W_1 = \alpha A^T$ and $W_2 = I - \alpha A^T A$
- Rewrite ISTA as
$$x^{k+1} = \eta_\theta \left(W_1 b + W_2 x^k \right)$$
- Unrolling: introduce $\theta^k, W_1^k, W_2^k, k = 0, 1, \ldots,$ as free parameters and re-define
$$x^{k+1} = \eta_{\theta^k} \left(W_1^k b + W_2^k x^k \right)$$
which resembles a DNN:



- Once θ^k, W_1^k, W_2^k are chosen, the algorithm is defined

Train the Unrolled ISTA

- **Objective:** Find θ^k, W_1^k, W_2^k for $k = 0, 1, \ldots,$ such that the algorithm converges quickly for LASSO instances with the same matrix A.
- **Setup and Training:**
  - Fix a random matrix A, generate sparse vectors x_i^{true} with varying supports, and compute $b_i = Ax_i^{\text{true}} + \text{noise}_i$. Form the training set $D = \{(x_i^{\text{true}}, b_i)\}$.
  - Fix a small $K > 0$, and train the parameters $\{\theta^k, W_1^k, W_2^k\}_{k=0}^K$ using SGD to minimize:

$$\underset{\{\theta^k, W_1^k, W_2^k\}_{k=0}^K}{\text{minimize}} \sum_{(x^*, b) \in D} \left\| x^K(b) - x^* \right\|_2^2,$$

where $x^K(b)$ is the K-layer output of the neural network.

Performance of the Learned ISTA (LISTA)

After the NN is trained with $K = 16$, the test performance is pretty good:

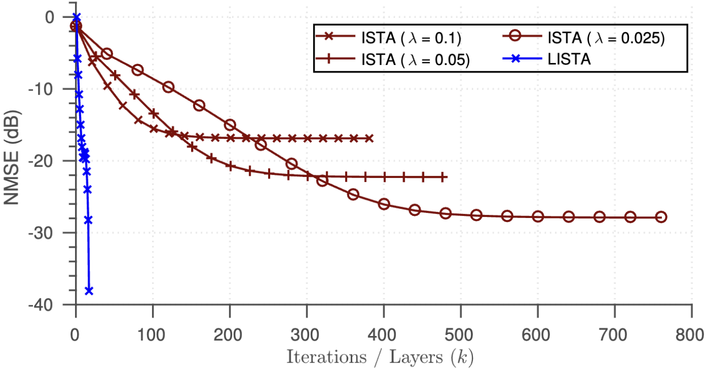


Figure: The trained unrolled ISTA is called Learned ISTA (LISTA)

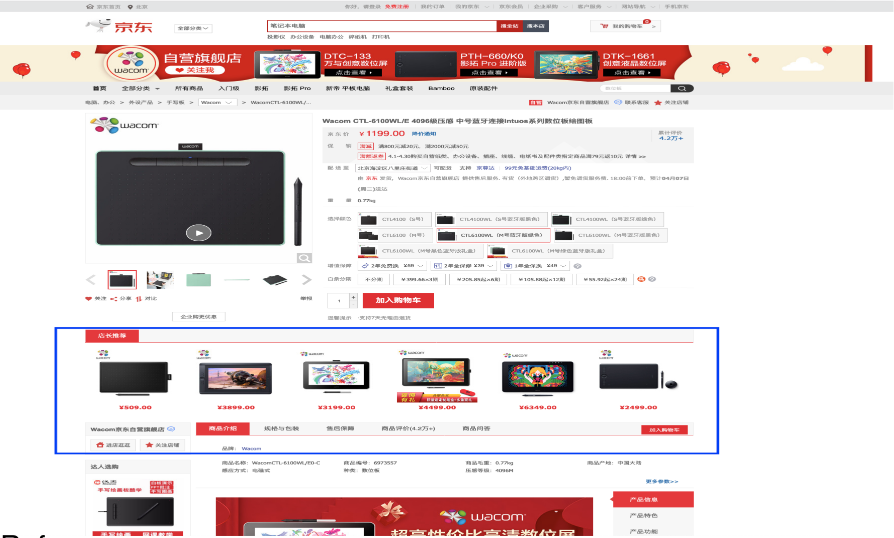
LISTA is better than ISTA at any λ and using a theoretical stepsize

Lecture: Matrix Completion

`http://bicmr.pku.edu.cn/~wenzw/bigdata2020.html`

Acknowledgement: this slides is based on Prof. Jure Leskovec and Prof. Emmanuel Candes's lecture notes

Recommendation systems



Wacom CTL-6100WL/E 4096级压感 中号蓝牙连接intuos系列数位板绘图板

京 东 价 ¥1199.00 降价通知

累计评价 4.2万+

促 销 满减 满800元减20元，满2000元减50元

满额返券 4.1-4.30购买自营纸类、办公设备、插座、线缆、电纸书及配件类指定商品满79元返10元 详情 >>

配 送 至 北京海淀区八里庄街道 可配货 支持 京尊达 99元免基础运费(20kg内)

由 京东 发货，Wacom京东自营旗舰店 提供售后服务. 有货（外地跨区调货），暂免调货服务费. 18:00前下单，预计04月07日(周二)送达

重 量 0.77kg

选择颜色 CTL4100（S号） CTL4100WL（S号蓝牙版黑色） CTL4100WL（S号蓝牙版绿色） CTL6100（M号） CTL6100WL（M号蓝牙版绿色） CTL6100WL（M号蓝牙版黑色） CTL6100WL（M号黑色蓝牙版礼盒） CTL6100WL（M号绿色蓝牙版礼盒）

增值保障 2年免费换 ¥59 2年全保修 ¥39 1年全保换 ¥49

白条分期 不分期 ¥399.66×3期 ¥205.85起×6期 ¥105.88起×12期 ¥55.92起×24期

温馨提示 ·支持7天无理由退货

品牌：Wacom

商品名称：WacomCTL-6100WL/E0-C 商品编号：6973557 商品毛重：0.77kg 商品产地：中国大陆

感应方式：电磁式 种类：数位板 压感等级：4096M

更多参数>>

References:

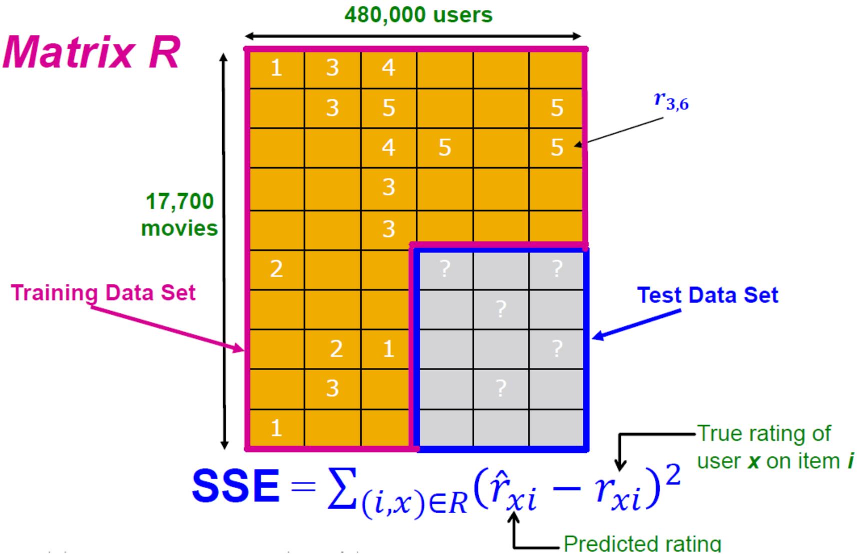
`http://bicmr.pku.edu.cn/~wenzw/bigdata/07-recsys1.pdf`

`http://bicmr.pku.edu.cn/~wenzw/bigdata/08-recsys2.pdf`

The Netflix Prize

- Training data
  - 100 million ratings, 480,000 users, 17,770 movies
  - 6 years of data: 2000-2005
- Test data
  - Last few ratings of each user (2.8 million)
  - Evaluation criterion: root mean squared error (RMSE): $\sqrt{\sum_{xi}(r_{xi} - r_{xi}^*)^2}$: r_{xi} and r_{xi}^* are the predicted and true rating of x on i
  - Netflix Cinematch RMSE: 0.9514
- Competition
  - 2700+ teams
  - \$1 million prize for 10% improvement on Cinematch

Netflix: evaluation



Matrix R

480,000 users

17,700 movies

1	3	4			
	3	5			5
		4	5		5
		3			
		3			
2			?		?
				?	
	2	1			?
	3			?	
1					

$r_{3,6}$

Training Data Set

Test Data Set

$$\mathbf{SSE} = \sum_{(i,x)\in R}(\hat{r}_{xi} - r_{xi})^2$$

True rating of user ***x*** on item ***i***

Predicted rating

Collaborative Filtering: weighted sum model

$$\hat{r}_{xi} = b_{xi} + \sum_{j \in N(i;x)} w_{ij}(r_{xj} - b_{xj})$$

- baseline estimate for r_{xi}: $b_{xi} = \mu + b_x + b_i$
  μ: overall mean rating
  b_x: rating deviation of user x = (avg. rating of user x) - μ
  b_i: (avg. rating of movie i) - μ
- We sum over all movies j that are similar to i and were rated by x
- w_{ij} is the interpolation weight (some real number). We allow: $\sum_{j \in N(i,x)} w_{ij} \neq 1$
- w_{ij} models interaction between pairs of movies (it does not depend on user x)
- $N(i;x)$: set of movies rated by user x that are similar to movie i

Finding weights w_{ij}?

Find w_{ij} such that they work well on known (user, item) ratings:

$$\min_{w_{ij}} \quad F(w) := \sum_x \left(\left[b_{xi} + \sum_{j \in N(i;x)} w_{ij}(r_{xj} - b_{xj}) \right] - r_{xi} \right)^2$$

- Unconstrained optimization: quadratic function

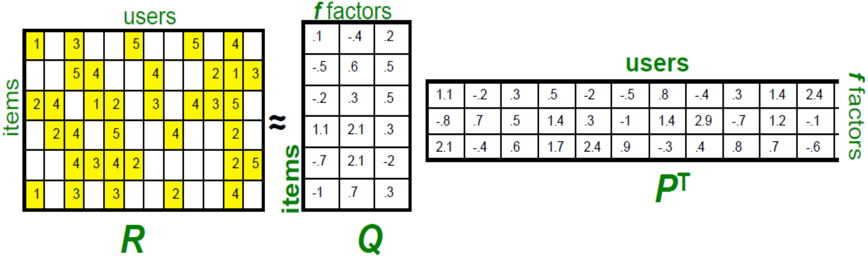
$$\nabla_{w_{ij}} F(w) = 2 \sum_x \left(\left[b_{xi} + \sum_{k \in N(i;x)} w_{ik}(r_{xk} - b_{xk}) \right] - r_{xi} \right) (r_{xj} - b_{xj}) = 0$$

for $j \in \{N(i,x), \forall i, x\}$

- Equivalent to solving a system of linear equations?
- Steepest gradient descent method: $w^{k+1} = w^k - \tau \nabla F(w)$
- Conjugate gradient method

Latent factor models

- low rank factorization on Netflix data: $R \approx Q \cdot P^T$



1		3			5			5		4	
		5	4			4			2	1	3
2	4		1	2		3		4	3	5	
	2	4		5			4			2	
		4	3	4	2					2	5
1		3		3			2			4	

.1	-.4	.2
-.5	.6	.5
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

1.1	-.2	.3	.5	-2	-.5	.8	-.4	.3	1.4	2.4
-.8	.7	.5	1.4	.3	-1	1.4	2.9	-.7	1.2	-.1
2.1	-.4	.6	1.7	2.4	.9	-.3	.4	.8	.7	-.6

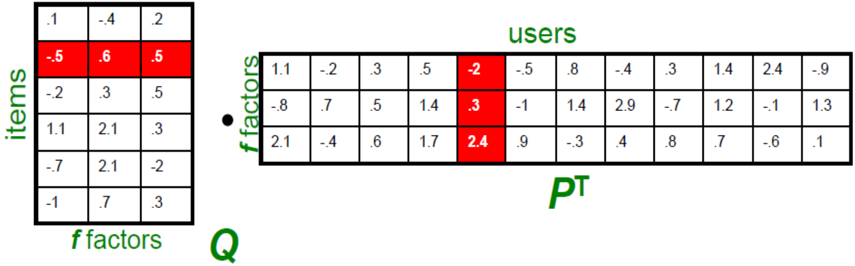
- For now let's assume we can approximate the rating matrix R as a product of “thin” $Q \cdot P^T$
  R has missing entries but let's ignore that for now! Basically, we will want the reconstruction error to be small on known ratings and we don't care about the values on the missing ones

Ratings as products of factors

- How to estimate the missing rating of user x for item i?

$$\hat{r}_{xi} = q_i \cdot p_x^T = \sum_f q_{if} p_{xf},$$

where q_i is row i of Q and p_x is column x of P^T



.1	-.4	.2
-.5	**.6**	**.5**
-.2	.3	.5
1.1	2.1	.3
-.7	2.1	-2
-1	.7	.3

1.1	-.2	.3	.5	**-2**	-.5	.8	-.4	.3	1.4	2.4	-.9
-.8	.7	.5	1.4	**.3**	-1	1.4	2.9	-.7	1.2	-.1	1.3
2.1	-.4	.6	1.7	**2.4**	.9	-.3	.4	.8	.7	-.6	.1

Latent factor models

- Minimize SSE on training data!
- Use specialized methods to find P, Q such that $\hat{r}_{xi} = q_i \cdot p_x^T$

$$\min_{P,Q} \sum_{(i,x)\in\text{training}} (r_{xi} - q_i \cdot p_x^T)^2$$

We don't require cols of P, Q to be orthogonal/unit length

- P, Q map users/movies to a latent space
- Add regularization:

$$\min_{P,Q} \sum_{(i,x)\in\text{training}} (r_{xi} - q_i \cdot p_x^T)^2 + \lambda \left[\sum_x \|p_x\|_2^2 + \sum_i \|q_i\|_2^2 \right]$$

λ is called regularization parameters

Gradient descent method

$$\min_{P,Q} \quad F(P,Q) := \sum_{(i,x)\in\text{training}} (r_{xi} - q_i \cdot p_x^T)^2 + \lambda \left[\sum_x \|p_x\|_2^2 + \sum_i \|q_i\|_2^2 \right]$$

Gradient decent:

- Initialize P and Q (using SVD, pretend missing ratings are 0)
- Do gradient descent:
  $P^{k+1} \leftarrow P^k - \tau \nabla_P F(P^k, Q^k)$,
  $Q^{k+1} \leftarrow Q^k - \tau \nabla_Q F(P^k, Q^k)$,
  where $(\nabla_Q F)_{if} = -2\sum_{x,i}(r_{xi} - q_i p_x^T)p_{xf} + 2\lambda q_{if}$. Here q_{if} is entry f of row q_i of matrix Q
- Computing gradients is slow when the dimension is huge

Stochastic gradient descent method

Observation: Letq_{if} be entry f of row q_i of matrix Q

$$\begin{aligned}(\nabla_Q F)_{if} &= \sum_{x,i} \left(-2(r_{xi} - q_i p_x^T) p_{xf} + 2\lambda q_{if}\right) = \sum_{x,i} \nabla_Q F(r_{xi}) \\ (\nabla_P F)_{xf} &= \sum_{x,i} \left(-2(r_{xi} - q_i p_x^T) q_{xf} + 2\lambda p_{if}\right) = \sum_{x,i} \nabla_P F(r_{xi})\end{aligned}$$

Stochastic gradient decent:

- Instead of evaluating gradient over all ratings, evaluate it for each individual rating and make a step
- $P \leftarrow P - \tau \nabla_P F(r_{xi})$
  $Q \leftarrow Q - \tau \nabla_Q F(r_{xi})$
- Need more steps but each step is computed much faster

Latent factor models with biases

predicted models:

$$\hat{r}_{xi} = \mu + b_x + b_i + q_i \cdot p_x^T$$

μ: overall mean rating, b_x: Bias for user x, b_i: Bias for movie i

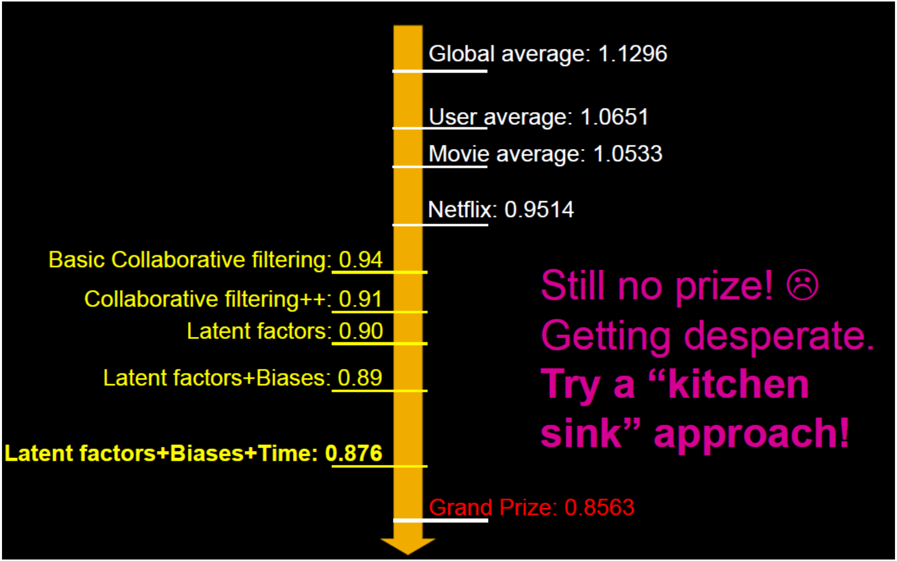
New model:

$$\min_{P,Q,b_x,b_i} \sum_{(i,x)\in \text{training}} (r_{xi} - (\mu + b_x + b_i + q_i \cdot p_x^T))^2 + \lambda \left[\sum_x \|p_x\|_2^2 + \sum_i \|q_i\|_2^2 + \|b_x\|_2^2 + \|b_i\|_2^2 \right]$$

- Both biases b_x, b_i as well as interactions q_i, p_x are treated as parameters (we estimate them)
- Add time dependence to biases:

$$\hat{r}_{xi} = \mu + b_x(t) + b_i(t) + q_i \cdot p_x^T$$

Netflix: performance


Global average: 1.1296

User average: 1.0651

Movie average: 1.0533

Netflix: 0.9514

Basic Collaborative filtering: 0.94

Collaborative filtering++: 0.91

Latent factors: 0.90

Latent factors+Biases: 0.89

Latent factors+Biases+Time: 0.876

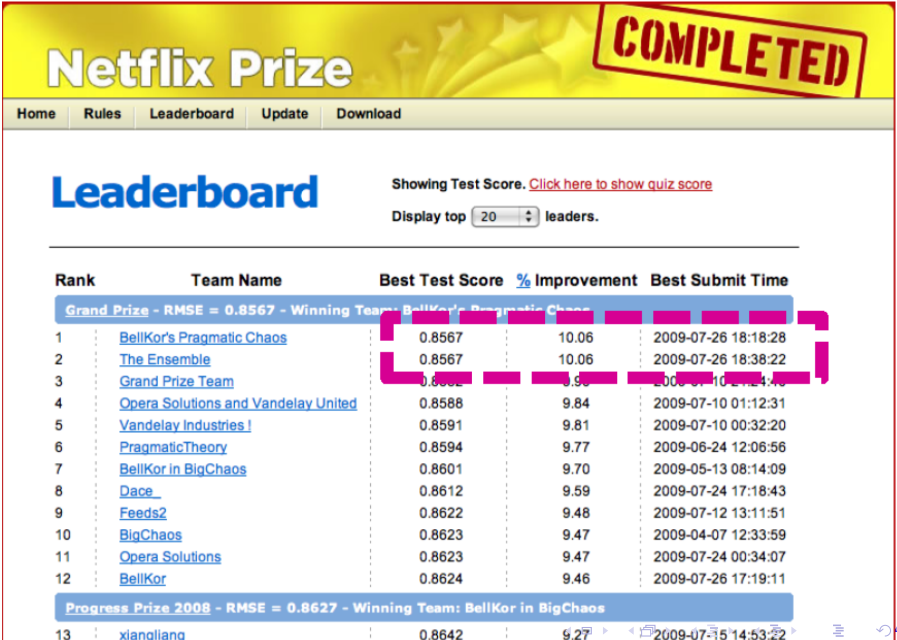
Still no prize! ☹

Getting desperate.

Try a “kitchen sink” approach!

Grand Prize: 0.8563

Netflix: performance



Leaderboard

Showing Test Score. Click here to show quiz score

Display top 20 leaders.

Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	0.8582	9.90	2009-07-10 21:24:40
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries !	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace_	0.8612	9.59	2009-07-24 17:18:43
9	Feeds2	0.8622	9.48	2009-07-12 13:11:51
10	BigChaos	0.8623	9.47	2009-04-07 12:33:59
11	Opera Solutions	0.8623	9.47	2009-07-24 00:34:07
12	BellKor	0.8624	9.46	2009-07-26 17:19:11
Progress Prize 2008 - RMSE = 0.8627 - Winning Team: BellKor in BigChaos				
13	xiangliang	0.8642	9.27	2009-07-15 14:53:22

General matrix completion

Matrix completion

- Matrix $M \in \mathbb{R}^{n_1 \times n_2}$
- Observe subset of entries
- Can we guess the missing entries?

$$\begin{bmatrix} \times & ? & ? & ? & \times & ? \\ ? & ? & \times & \times & ? & ? \\ \times & ? & ? & \times & ? & ? \\ ? & ? & \times & ? & ? & \times \\ \times & ? & ? & ? & ? & ? \\ ? & ? & \times & \times & ? & ? \end{bmatrix}$$

Which algorithm ?

Hope: only one low-rank matrix consistent with the sampled entries

Recovery by minimum complexity

$$\begin{array}{rl} \text{minimize} & \operatorname{rank}(X) \\ \text{subject to} & X_{ij} = M_{ij}, \quad (i,j) \in \Omega \end{array}$$

Problem

- This is NP-hard
- Doubly exponential in n (?)

SVD - Properties

Theorem: SVD

If A is a real m-by-n matrix, then there exits

$$U = [u_1, \ldots, u_m] \in \mathbb{R}^{m\times m} \text{ and } V = [v_1, \ldots, v_n] \in \mathbb{R}^{n\times n}$$

such that $U^TU = I$, $V^TV = I$ and

$$U^TAV = \mathrm{diag}(\sigma_1, \ldots, \sigma_p) \in \mathbb{R}^{m\times n}, \quad p = \min(m, n),$$

where $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p \geq 0$.

- Proof: Let $V_1 \in \mathbb{R}^{n\times r}$ has orthonormal columns, then exits $V_2 \in \mathbb{R}^{n\times(n-r)}$ such that $V = [V_1, V_2]$ is orthogonal.
- Let $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$ be unit 2-norm vectors: $Ax = \sigma y$ with $\sigma = \|A\|_2$. Then exists $V_2 \in \mathbb{R}^{n\times(n-1)}$ and $U_2 \in \mathbb{R}^{m\times(m-1)}$ so $V = [x, V_2] \in \mathbb{R}^{n\times n}$ and $U = [y, U_2] \in \mathbb{R}^{m\times m}$ are orthogonal.

- Then it can be proved that U^TAV has the following structure

  $$U^TAV = \begin{pmatrix} \sigma & w^T \\ 0 & B \end{pmatrix} \equiv A_1.$$

  Since

  $$\left\| A_1 \begin{pmatrix} \sigma \\ w \end{pmatrix} \right\|_2^2 \geq (\sigma^2 + w^T w)^2,$$

  we have $\|A_1\|_2^2 \geq (\sigma^2 + w^T w)$. But $\sigma^2 = \|A\|_2^2 = \|A_1\|_2^2$, and so we must have $w = 0$. An induction gives the proof.

Properties:

- $AV = U\Sigma$, $A^TU = V\Sigma^T$: $Av_i = \sigma u_i$, $A^Tu_i = \sigma_i v_i$, $i = 1, \ldots, p$.
- $rank(A) = r$, $null(A) = span\{v_{r+1}, \ldots, v_n\}$, $ran(A) = span\{u_1, \ldots, u_r\}$
- $A = \sum_{i=1}^r \sigma_i u_i v_i^T$
- $\|A\|_F^2 = \sigma_1^2 + \ldots + \sigma_p^2$, $\|A\|_2 = \sigma_1$

SVD - Best Low Rank Approximation

Theorem

Let the SVD of $A \in \mathbb{R}^{m \times n}$ be given in Theorem: SVD. If $k < r = rank(A)$ and $A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T$, then

$$\min_{rank(B)=k} \quad \|A - B\|_2 = \|A - A_k\|_2 = \sigma_{k+1}.$$

- Proof: Since $U^T A_k V = \text{diag}(\sigma_1, \ldots, \sigma_k, 0, \ldots, 0)$ it follows that $rank(A_k) = k$ and $U^T(A - A_k)V = \text{diag}(0, \ldots, 0, \sigma_{k+1}, \ldots, \sigma_p)$. Hence $\|A - A_k\|_2 = \sigma_{k+1}$.

- Suppose $rank(B) = k$ for some $B \in \mathbb{R}^{m \times n}$. We can find orthonormal vectors $x_1, \ldots, x_{n-k}$ so $null(B) = span\{x_1, \ldots, x_{n-k}\}$. A dimension argument shows:

$$span\{x_1, \ldots, x_{n-k}\} \cap span\{v_1, \ldots, v_{k+1}\} \neq \{0\}$$

- Let z be a unit 2-norm vector in this intersection. Since $Bz = 0$ and

  $$Az = \sum_{i=1}^{k+1} \sigma_i (v_i^T z) u_i,$$

  we have

$$\|A - B\|_2^2 \geq \|(A - B)z\|_2^2 = \|Az\|_2^2 = \sum_{i=1}^{k+1} \sigma_i^2 (v_i^T z)^2 \geq \sigma_{k+1}^2$$

Comments:

- So zeroing small σ_i introduces less error
- How many σs to keep? Rule of thumb: keep 80-90% of ’energy’ $(= \sum \sigma_i^2)$

SVD - Complexity

- To compute SVD: $O(nm^2)$ or $O(n^2m)$
- But:
  - Less work, if we just want singular values
  - or if we want first k singular vectors
  - or if the matrix is sparse
- Implemented in linear algebra packages like
  - Dense matrix: LAPACK
  - Sparse Matrix: ARPACK, PROPACK
  - High Level Software packages: Matlab, SPlus, Mathematica ...

Relation to Eigen-decomposition

- SVD gives us $A = U\Sigma V^\top$
- Eigen-decomposition: $A = X\Lambda X^\top$
  - A is symmetric
  - U, V, X are orthonormal
  - Λ, Σ are diagonal
- $AA^\top = U\Sigma\Sigma^\top U^\top$
- $A^\top A = V\Sigma\Sigma^\top V^\top$
- $\lambda_i(A^\top A) = \sigma_i^2(A)$

Nuclear-norm minimization

Singular value decomposition

$$X = \sum_{k=1}^{r} \sigma_k u_k v_k^*$$

- $\{\sigma_k\}$: singular values, $\{u_k\}, \{v_k\}$: singular vectors

Nuclear norm ($\sigma_i(X)$ is ith largest singular value of X)

$$\|X\|_* = \sum_{i=1}^{n} \sigma_i(X)$$

Heuristic

$$\begin{array}{ll} \text{minimize} & \|X\|_* \\ \text{subject to} & X_{ij} = M_{ij}, \quad (i,j) \in \Omega \end{array}$$

- Convex relaxation of the rank minimization program

Connections with compressed sensing

General setup

Rank minimization

$$\begin{array}{ll}\text{minimize} & \text{rank}(X) \\ \text{subject to} & \mathcal{A}(X) = b\end{array}$$

Convex relaxation

$$\begin{array}{ll}\text{minimize} & \|X\|_* \\ \text{subject to} & \mathcal{A}(X) = b\end{array}$$

Suppose $X = \text{diag}(x), x \in \mathbb{R}^n$

- $\text{rank}(X) = \sum_i 1_{(x_i \neq 0)} = \|x\|_{\ell_0}$
- $\|X\|_* = \sum_i |x_i| = \|x\|_{\ell_1}$

Rank minimization

$$\begin{array}{ll}\text{minimize} & \|x\|_{\ell_0} \\ \text{subject to} & Ax = b\end{array}$$

Convex relaxation

$$\begin{array}{ll}\text{minimize} & \|x\|_{\ell_1} \\ \text{subject to} & Ax = b\end{array}$$

This is compressed sensing!

SOCP/SDP Duality

(P) $\min \quad c^\top x$

s.t. $\quad Ax = b, x_Q \succeq 0$

(D) $\max \quad b^\top y$

s.t. $\quad A^\top y + s = c, s_Q \succeq 0$

(P) $\min \quad \langle C, X \rangle$

s.t.
$$\begin{aligned} &\langle A_1, X \rangle = b_1 \\ &\dots \\ &\langle A_m, X \rangle = b_m \\ &X \succeq 0 \end{aligned}$$

(D) $\max \quad b^\top y$

s.t.
$$\begin{aligned} &\sum_i y_i A_i + S = C \\ &S \succeq 0 \end{aligned}$$

Strong duality

- If $p^* > -\infty$, (P) is **strictly** feasible, then (D) is feasible and $p^* = d^*$
- If $d^* < +\infty$, (D) is **strictly** feasible, then (P) is feasible and $p^* = d^*$
- If (P) and (D) has **strictly** feasible solutions, then both have optimal solutions.

Semidefinite program

$$(D) \quad \begin{array}{ll} \min & -b^\top y \\ \text{s.t.} & y_1 A_1 + \ldots + y_m A_m \preceq C \end{array}$$

- $A_i, C \in \mathcal{S}^k$, multiplier is matrix $X \in \mathcal{S}^k$
- Lagrangian $\mathcal{L}(y, X) = -b^\top y + \langle X, y_1 A_1 + \ldots + y_m A_m - C \rangle$
- dual function

$$g(X) = \inf_y \quad \mathcal{L}(y, X) = \begin{cases} -\langle C, X \rangle, & \langle A_i, X \rangle = b_i \\ -\infty & \text{otherwise} \end{cases}$$

The dual of (D) is

$$\begin{array}{ll} \min & \langle C, X \rangle \\ \text{s.t.} & \langle A_i, X \rangle = b_i, X \succeq 0 \end{array}$$

$p^* = d^*$ if primal SDP is strictly feasible.

Positive semidefinite unknown: SDP formulation

Suppose unknown matrix X is positive semidefinite

$$\begin{array}{ll} \min & \sum_{i=1}^{n} \sigma_i(X) \\ \text{s.t.} & X_{ij} = M_{ij} \quad (i,j) \in \Omega \\ & X \succcurlyeq 0 \end{array} \quad \Leftrightarrow \quad \begin{array}{ll} \min & \text{trace}(X) \\ \text{s.t.} & X_{ij} = M_{ij} \quad (i,j) \in \Omega \\ & X \succcurlyeq 0 \end{array}$$

Trace heuristic: Mesbahi & Papavassilopoulos (1997), Beck & D'Andrea (1998)

General SDP formulation

Let $X \in \mathbb{R}^{m\times n}$. For a given norm $\|\cdot\|$, the dual norm $\|\cdot\|_d$ is defined as

$$\|X\|_d := \sup\{\langle X, Y\rangle : Y \in \mathbb{R}^{m\times n}, \|Y\| \leq 1\}$$

Nuclear norm and spectral norms are dual:

$$\|X\| := \sigma_1(X), \quad \|X\|_* = \sum_i \sigma_i(X).$$

(P) $\begin{aligned} \max_Y \quad & \langle X, Y\rangle \\ \text{s.t.} \quad & \|Y\|_2 \leq 1 \end{aligned}$ $\Leftrightarrow$ $\begin{aligned} \max_Y \; & 2\langle X, Y\rangle \\ \text{s.t.} \; & \begin{bmatrix} I_m & Y \\ Y^\top & I_n \end{bmatrix} \succcurlyeq 0 \end{aligned}$ $\Leftrightarrow$ $\begin{aligned} \min_Z \; & -\left\langle Z, \begin{bmatrix} 0 & X \\ X^\top & 0 \end{bmatrix}\right\rangle \\ \text{s.t.} \; & Z_1 = I_m \\ & Z_2 = I_n \\ & Z = \begin{bmatrix} Z_1 & Z_3 \\ Z_3^\top & Z_2 \end{bmatrix} \succeq 0 \end{aligned}$

General SDP formulation

The Lagrangian dual problem is:

$$\max_{W_1, W_2} \min_{Z \succeq 0} \quad -\left\langle Z, \begin{bmatrix} 0 & X \\ X^* & 0 \end{bmatrix} \right\rangle + \langle Z_1 - I_m, W_1 \rangle + \langle Z_2 - I_n, W_2 \rangle$$

strong duality after a scaling of $1/2$ and change of variables X to $-X$

$$\text{(D)} \quad \begin{array}{ll} \text{minimize} & \frac{1}{2}(\text{trace}(W_1) + \text{trace}(W_2)) \\ \text{subject to} & \begin{bmatrix} W_1 & X \\ X^\top & W_2 \end{bmatrix} \succcurlyeq 0 \end{array}$$

Optimization variables: $W_1 \in \mathbb{R}^{n_1 \times n_1}, W_2 \in \mathbb{R}^{n_2 \times n_2}$.

Proposition 2.1 in "Guaranteed Minimum-Rank Solutions of Linear Matrix Equations via Nuclear Norm Minimization", Benjamin Recht, Maryam Fazel, Pablo A. Parrilo

General SDP formulation

Nuclear norm minimization

$$\begin{aligned} \min\ & \|X\|_* \\ \text{s.t. } & \mathcal{A}(X) = b \end{aligned} \quad \Longleftrightarrow \quad \begin{aligned} \max\ & b^\top y \\ \text{s.t. } & \|\mathcal{A}^*(y)\| \le 1 \end{aligned}$$

SDP Reformulation

$$\begin{aligned} \min\ & \frac{1}{2}(\text{trace}(W_1) + \text{trace}(W_2)) \\ \text{s.t. } & \mathcal{A}(X) = b \\ & \begin{bmatrix} W_1 & X \\ X^\top & W_2 \end{bmatrix} \succcurlyeq 0 \end{aligned} \quad \Longleftrightarrow \quad \begin{aligned} \max\ & b^\top y \\ \text{s.t. } & \begin{bmatrix} I & \mathcal{A}^*(y) \\ (\mathcal{A}^*(y))^\top & I \end{bmatrix} \succcurlyeq 0 \end{aligned}$$

Matrix recovery

$$M = \sum_{k=1}^{2} \sigma_k u_k u_k^*, \quad \begin{aligned} u_1 &= (e_1 + e_2)/\sqrt{2}, \\ u_2 &= (e_1 - e_2)/\sqrt{2} \end{aligned}$$

$$M = \begin{bmatrix} * & * & 0 & \dots & 0 & 0 \\ * & * & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 0 & 0 \end{bmatrix}$$

Cannot be recovered from a small set of entries

Rank-1 matrix $M = xy^*$

$$M_{ij} = x_i y_j$$

$$\begin{bmatrix} \times & \times & \times & \times & \times & \times \\ & & & & & \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \end{bmatrix}$$

If single row (or column) is not sampled $\to$ recovery is not possible

What happens for almost all sampling sets?

Ω subset of m entries selected uniformly at random

References

- Jianfeng Cai, Emmanuel Candes, Zuowei Shen, *Singular value thresholding algorithm for matrix completion*
- Shiqian Ma, Donald Goldfarb, Lifeng Chen, *Fixed point and Bregman iterative methods for matrix rank minimization*
- Zaiwen Wen, Wotao Yin, Yin Zhang, *Solving a low-rank factorization model for matrix completion by a nonlinear successive over-relaxation algorithm*
- Onureena Banerjee, Laurent El Ghaoui, Alexandre d'Aspremont, *Model Selection Through Sparse Maximum Likelihood Estimation for Multivariate Gaussian or Binary Data*
- Zhaosong Lu, *Smooth optimization approach for sparse covariance selection*

Matrix Rank Minimization

Given $X \in \mathbb{R}^{m\times n}$, $\mathcal{A} : \mathbb{R}^{m\times n} \rightarrow \mathbb{R}^p$, $b \in \mathbb{R}^p$, we consider

- the matrix rank minimization problem:

$$\min \ \operatorname{rank}(X), \ \text{s.t. } \mathcal{A}(X) = b$$

- matrix completion problem:

$$\min \ \operatorname{rank}(X), \ \text{s.t. } X_{ij} = M_{ij}, (i,j) \in \Omega$$

- nuclear norm minimization:

$$\min \ \|X\|_* \ \text{ s.t. } \mathcal{A}(X) = b$$

where $\|X\|_* = \sum_i \sigma_i$ and $\sigma_i = i$th singular value of matrix X.

Quadratic penalty framework

- Unconstrained Nuclear Norm Minimization:

$$\min \ F(X) := \mu\|X\|_* + \frac{1}{2}\|\mathcal{A}(X) - b\|_2^2.$$

- Optimality condition:

$$\mathbf{0} \in \mu\partial\|X^*\|_* + \mathcal{A}^*(\mathcal{A}(X^*) - b),$$

  where $\partial\|X\|_* = \{UV^\top + W : U^\top W = 0, WV = 0, \|W\|_2 \leq 1\}$.

- Linearization approach (g is the gradient of $\frac{1}{2}\|\mathcal{A}(X) - b\|_2^2$):

$$\begin{aligned} X^{k+1} &:= \arg\min_X \ \mu\|X\|_* + \langle g^k, X - X^k \rangle + \frac{1}{2\tau}\|X - X^k\|_F^2 \\ &= \arg\min_X \ \mu\|X\|_* + \frac{1}{2\tau}\|X - (X^k - \tau g^k)\|_F^2 \end{aligned}$$

Matrix Shrinkage Operator

For a matrix $Y \in \mathbb{R}^{m \times n}$, consider:

$$\min_{X \in \mathbb{R}^{m \times n}} \ \nu \|X\|_* + \frac{1}{2}\|X - Y\|_F^2.$$

The optimal solution is:

$$X := S(Y, \nu) = U \mathrm{Diag}(s(\sigma, \nu)) V^\top,$$

- SVD: $Y = U \mathrm{Diag}(\sigma) V^\top$
- Thresholding operator:

$$s(x, \nu) := \bar{x}, \ \text{with } \bar{x}_i = \begin{cases} x_i - \nu, & \text{if } x_i - \nu > 0 \\ 0, & \text{o.w.} \end{cases}$$

Fixed Point Method (Proximal gradient method)

Fixed Point Iterative Scheme

$$\begin{cases} Y^k = X^k - \tau \mathcal{A}^*(\mathcal{A}(X^k) - b) \\ X^{k+1} = S(Y^k, \tau\mu). \end{cases}$$

Lemma: Matrix shrinkage operator is non-expansive. i.e.,

$$\|S(Y_1, \nu) - S(Y_2, \nu)\|_F \leq \|Y_1 - Y_2\|_F.$$

Complexity of the fixed point method:

$$F(X^k) - F(X^*) \leq \frac{L_f \|X^0 - X^*\|^2}{2k}$$

Accelerated proximal gradient (APG) method

APG algorithm ($t^{-1} = t^0 = 1$):

$$\begin{aligned} Y^k &= X^k + \frac{t^{k-1}-1}{t^k}(X^k - X^{k-1}) \\ G^k &= Y^k - (\tau^k)^{-1}\mathcal{A}^*(\mathcal{A}(Y^k) - b) \\ X^{k+1} &= S_{\tau^k}(G^k), \quad t^{k+1} = \frac{1+\sqrt{1+4(t^k)^2}}{2} \end{aligned}$$

Complexity:

$$F(X^k) - F(X^*) \le \frac{2L_f\|X^0 - X^*\|^2}{(k+1)^2}$$

SVT

Linearized Bregman method:

$$\begin{aligned} V^{k+1} &:= V^k - \tau \mathcal{A}^*(\mathcal{A}(X^k) - b) \\ X^{k+1} &:= S_{\tau\mu}(V^{k+1}) \end{aligned}$$

Convergence to

$$\min \; \tau \|X\|_* + \frac{1}{2}\|X\|_F^2, \text{ s.t. } \mathcal{A}(X) = b$$

Review of Bregman method

Consider the problem:

$$\min \ \|X\|_*, \ \text{ s.t. } \mathcal{A}(X) = b$$

Bregman method:

- $D^{P^k}(X, X^k) := \|X\|_* - \|X^k\|_* - \langle P^k, X - X^k \rangle$
- $X^{k+1} := \arg\min_X \ \mu D^{P^k}(X, X^k) + \frac{1}{2}\|\mathcal{A}(X) - b\|_2^2$
- $P^{k+1} = P^k + \frac{1}{\mu}\mathcal{A}^\top(b - \mathcal{A}(X^{k+1}))$

Augmented Lagrangian (updating multiplier or b):

- $X^{k+1} := \arg\min_X \ \mu\|X\|_* + \frac{1}{2}\|\mathcal{A}(X) - b^k\|_2^2$
- $b^{k+1} = b + (b^k - \mathcal{A}(X^{k+1}))$

They are equivalent, see Yin-Osher-Goldfarb-Darbon

Linearized approaches

Linearized Bregman method:

$$\begin{aligned} X^{k+1} &:= \arg\min \ \mu D^{p^k}(X, X^k) + \left\langle \mathcal{A}^\top(\mathcal{A}(X^k) - b), X - X^k \right\rangle + \frac{1}{2\delta}\|X - X^k\|_F^2, \\ P^{k+1} &:= P^k - \frac{1}{\mu\delta}(X^{k+1} - X^k) - \frac{1}{\mu}\mathcal{A}^\top(\mathcal{A}(X^k) - b), \end{aligned}$$

which is equivalent to

$$\begin{aligned} X^{k+1} &:= \arg\min \ \mu\|X\|_* + \frac{1}{2\delta}\|X - V^k\|_F^2 \\ V^{k+1} &:= V^k - \delta\mathcal{A}^\top(\mathcal{A}(X^{k+1}) - b) \end{aligned}$$

Bregmanized operator splitting:

$$\begin{aligned} X^{k+1} &:= \arg\min \ \mu\|X\|_* + \left\langle \mathcal{A}^\top(\mathcal{A}(X^k) - b^k), X - X^k \right\rangle + \frac{1}{2\delta}\|X - X^k\|_F^2 \\ b^{k+1} &= b + (b^k - \mathcal{A}(X^{k+1})) \end{aligned}$$

Are they equivalent?

Linearized approaches

Linearized Bregman method:

$$\begin{aligned} X^{k+1} &:= \arg\min \ \mu D^{p^k}(X, X^k) + \left\langle \mathcal{A}^\top(\mathcal{A}(X^k) - b), X - X^k \right\rangle + \frac{1}{2\delta}\|X - X^k\|_F^2, \\ P^{k+1} &:= P^k - \frac{1}{\mu\delta}(X^{k+1} - X^k) - \frac{1}{\mu}\mathcal{A}^\top(\mathcal{A}(X^k) - b), \end{aligned}$$

which is equivalent to

$$\begin{aligned} X^{k+1} &:= \mathcal{S}(V^k, \mu\delta) \\ V^{k+1} &:= V^k - \delta\mathcal{A}^\top(\mathcal{A}(X^{k+1}) - b) \end{aligned} \quad \text{or} \quad \begin{aligned} X^{k+1} &:= \mathcal{S}(\delta A^\top(b^k), \mu\delta) \\ b^{k+1} &:= b + (b^k - \mathcal{A}(X^{k+1})) \end{aligned}$$

Bregmanized operator splitting:

$$\begin{aligned} X^{k+1} &:= \mathcal{S}(X^k - \delta(\mathcal{A}^\top(\mathcal{A}(X^k) - b^k)), \mu\delta) = \mathcal{S}(\delta\mathcal{A}^\top(b^k) + X^k - \delta\mathcal{A}^\top(\mathcal{A}(X^k)), \mu\delta) \\ b^{k+1} &= b + (b^k - \mathcal{A}(X^{k+1})) \end{aligned}$$

Low-rank factorization model

- Finding a low-rank matrix W so that $\|\mathcal{P}_\Omega(W-M)\|_F^2$ or the distance between W and $\{Z \in \mathbb{R}^{m\times n}, Z_{ij} = M_{ij}, \forall (i,j) \in \Omega\}$ is minimized.
- Any matrix $W \in \mathbb{R}^{m\times n}$ with $\mathrm{rank}(W) \le K$ can be expressed as $W = XY$ where $X \in \mathbb{R}^{m\times K}$ and $Y \in \mathbb{R}^{K\times n}$.

New model

$$\min_{X,Y,Z} \frac{1}{2}\|XY - Z\|_F^2 \;\text{ s.t. }\; Z_{ij} = M_{ij}, \forall (i,j) \in \Omega$$

- Advantage: SVD is no longer needed!
- Related work: the solver `OptSpace` based on optimization on manifold

Nonlinear Gauss-Seidel scheme

First variant of alternating minimization:

$$\begin{aligned} X_+ &\leftarrow ZY^\dagger \equiv ZY^\top(YY^\top)^\dagger, \\ Y_+ &\leftarrow (X_+)^\dagger Z \equiv (X_+^\top X_+)^\dagger (X_+^\top Z), \\ Z_+ &\leftarrow X_+Y_+ + \mathcal{P}_\Omega(M - X_+Y_+). \end{aligned}$$

Let $\mathcal{P}_A$ be the orthogonal projection onto the range space $\mathcal{R}(A)$

- $X_+Y_+ = \left(X_+(X_+^\top X_+)^\dagger X_+^\top\right) Z = \mathcal{P}_{X_+} Z$
- One can verify that $\mathcal{R}(X_+) = \mathcal{R}(ZY^\top)$.
- $X_+Y_+ = \mathcal{P}_{ZY^\top} Z = ZY^\top(YZ^\top ZY^\top)^\dagger(YZ^\top)Z.$
- idea: modify X_+ or Y_+ to obtain the same product X_+Y_+

Nonlinear Gauss-Seidel scheme

Second variant of alternating minimization:

$$\begin{aligned} X_+ &\leftarrow ZY^\top, \\ Y_+ &\leftarrow (X_+)^\dagger Z \equiv (X_+^\top X_+)^\dagger (X_+^\top Z), \\ Z_+ &\leftarrow X_+ Y_+ + \mathcal{P}_\Omega(M - X_+ Y_+). \end{aligned}$$

Third variant of alternating minimization: $V = \texttt{orth}(ZY^\top)$

$$\begin{aligned} X_+ &\leftarrow V, \\ Y_+ &\leftarrow V^\top Z, \\ Z_+ &\leftarrow X_+ Y_+ + \mathcal{P}_\Omega(M - X_+ Y_+). \end{aligned}$$

Sparse and low-rank matrix separation

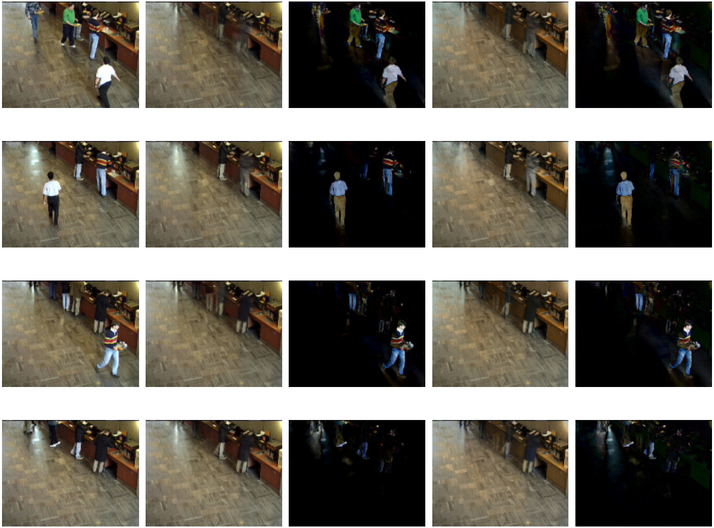
- Given a matrix M, we want to find a low rank matrix W and a sparse matrix E, so that $W + E = M$.
- Convex approximation:

$$\min_{W,E} \ \|W\|_* + \mu\|E\|_1, \text{ s.t. } W + E = M$$

- Robust PCA

Video separation

- Partition the video into moving and static parts



ADMM

Convex approximation:

$$\min_{W,E} \ \|W\|_* + \mu\|E\|_1, \text{ s.t. } W + E = M$$

Augmented Lagrangian function:

$$L(W, E, \Lambda) := \|W\|_* + \mu\|E\|_1 + \langle \Lambda, W + E - M \rangle + \frac{1}{2\beta}\|W + E - M\|_F^2$$

Alternating direction Augmented Lagrangian method

$$\begin{aligned} W^{j+1} \quad &:= \quad \arg\min_{W} \ L(W,\ E^j,\ \Lambda^j), \\ E^{j+1} \quad &:= \quad \arg\min_{E} \ L(W^{j+1},\ E,\ \Lambda^j), \\ \Lambda^{j+1} \quad &:= \quad \Lambda^j + \frac{\gamma}{\beta}(W^{j+1} + E^{j+1} - M). \end{aligned}$$

W-subproblem

Convex approximation:

$$\begin{aligned} W^{j+1} &:= \arg\min_{W} \ L(W,\ E^j,\ \Lambda^j) \\ &= \arg\min_{W} \|W\|_* + \frac{1}{2\beta} \left\| W - \left(M - E^j - \beta\Lambda^j\right) \right\|_F^2 \\ &= S_\beta(M - E^j - \beta\Lambda^j) := U\mathrm{Diag}(s_\beta(\sigma))V^\top \end{aligned}$$

- SVD: $M - E^j - \beta\Lambda^j = U\mathrm{Diag}(\sigma)V^\top$
- Thresholding operator:

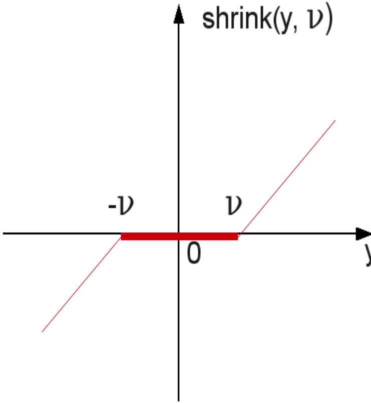
$$s_\nu(x) := \bar{x}, \ \text{with } \bar{x}_i = \begin{cases} x_i - \nu, & \text{if } x_i - \nu > 0 \\ 0, & \text{o.w.} \end{cases}$$

E-subproblem

Convex approximation:

$$\begin{aligned} W^{j+1} &:= \arg\min_{E} \; L(W^{j+1},\; E,\; \Lambda^{j}) \\ &= \arg\min_{E} \|E\|_1 + \frac{1}{2\beta\mu} \left\|E - \left(M - W^{j+1} - \beta\Lambda^{j}\right)\right\|_F^2 \\ &= s_{\beta\mu}(M - W^{j+1} - \beta\Lambda^{j}) \end{aligned}$$

$$\begin{aligned} s_\nu(y) : &= \arg\min_{x\in\mathbb{R}} \; \nu\|x\|_1 + \frac{1}{2}\|x-y\|_2^2 \\ &= \begin{cases} y - \nu\,\text{sgn}(y), & \text{if } |y| > \nu \\ 0, & \text{otherwise} \end{cases} \end{aligned}$$



Low-rank factorization model for matrix separation

- Consider the model

$$\min_{Z,S} \ \|S\|_1 \ \text{ s.t. } \ Z+S=D, \ \operatorname{rank}(Z)\le K$$

- Low-rank factorization: $Z = UV$

$$\min_{U,V,Z} \ \|Z-D\|_1 \ \text{ s.t. } \ UV-Z=0$$

- Only the entries D_{ij}, $(i,j)\in\Omega$, are given. $\mathcal{P}_\Omega(D)$ is the projection of D onto Ω.

New model

$$\min_{U,V,Z} \quad \|\mathcal{P}_\Omega(Z-D)\|_1 \quad \text{ s.t. } \quad UV-Z=0$$

- Advantage: SVD is no longer needed!

ADMM

Consider:

$$\min_{U,V,Z} \quad \|\mathcal{P}_\Omega(Z-D)\|_1 \quad \text{s.t.} \quad UV-Z=0$$

Introduce the augmented Lagrangian function

$$\mathcal{L}_\beta(U,V,Z,\Lambda)=\|\mathcal{P}_\Omega(Z-D)\|_1+\langle\Lambda, UV-Z\rangle+\frac{\beta}{2}\|UV-Z\|_F^2,$$

Alternating direction augmented Lagrangian framework (Bregman):

$$\begin{aligned}
U^{j+1} &:= \arg\min_{U\in\mathbb{R}^{m\times k}} \mathcal{L}_\beta(U,\ V^j,\ Z^j,\ \Lambda^j),\\
V^{j+1} &:= \arg\min_{V\in\mathbb{R}^{k\times n}} \mathcal{L}_\beta(U^{j+1},\ V,\ Z^j,\ \Lambda^j),\\
Z^{j+1} &:= \arg\min_{Z\in\mathbb{R}^{m\times n}} \mathcal{L}_\beta(U^{j+1},\ V^{j+1},\ Z,\ \Lambda^j),\\
\Lambda^{j+1} &:= \Lambda^j+\gamma\beta(U^{j+1}V^{j+1}-Z^{j+1}).
\end{aligned}$$

ADMM subproblems

- Let $B = Z - \Lambda/\beta$, then

  $$U_+ = BV^\top (VV^\top)^\dagger \text{ and } V_+ = (U_+^\top U_+)^\dagger U_+^\top B$$

  Since $U_+ V_+ = U_+ (U_+^\top U_+)^\dagger U_+^\top B = \mathcal{P}_{U_+} B$, then:

  $$Q := \texttt{orth}(BV^\top), \quad U_+ = Q \text{ and } V_+ = Q^\top B$$

- Variable Z:

  $$\begin{aligned} \mathcal{P}_\Omega(Z_+) &= \mathcal{P}_\Omega \left(\mathcal{S}\left(U_+ V_+ - D + \frac{\Lambda}{\beta}, \frac{1}{\beta} \right) + D \right) \\ \mathcal{P}_{\Omega^c}(Z_+) &= \mathcal{P}_{\Omega^c} \left(U_+ V_+ + \frac{\Lambda}{\beta} \right) \end{aligned}$$

Optimal Transport

`http://bicmr.pku.edu.cn/~wenzw/bigdata2023.html`

Acknowledgement: this slides is based on Prof. Gabriel Peyré's lecture notes

Outline

1. Problem Formulation
2. Applications
3. Entropic Regularization
4. Sinkhorn's Algorithm
5. Sinkhorn-Newton method
6. Wasserstein barycenter

A Geometric Motivation

Setting: Probability measures $\mathcal{P}(\mathcal{X})$ on a metric space $(\mathcal{X}, \mathrm{dist})$.

distance between μ and ν:

- $\mu = \delta_{x_1}$ and $\nu = \delta_{y_1}$
  $\mathrm{dist}(\mu, \nu) = \mathrm{dist}(x_1, y_1)$

- $\mu = \frac{1}{n}\sum_{i=1}^{n} \delta_{x_i}$ and $\nu = \frac{1}{n}\sum_{i=1}^{n} \delta_{y_i}$
  $\mathrm{dist}(\mu, \nu) = \frac{1}{n^2}\sum_{ij} \mathrm{dist}(x_i, y_j)$? or
  $\mathrm{dist}(\mu, \nu) = \min_{\sigma \text{ permutation}} \ \frac{1}{n}\sum_{i} \mathrm{dist}(x_i, y_{\sigma(i)})$

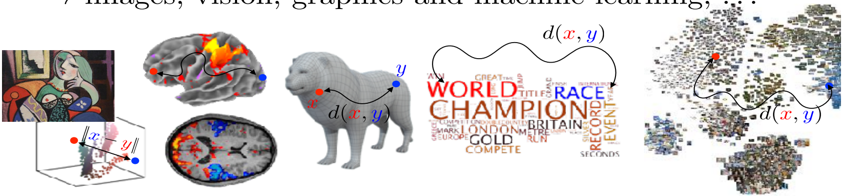
- What if $\mu, \nu \in \mathcal{P}(\mathcal{X})$?

Goal: Build a metric on $\mathcal{P}(\mathcal{X})$ with the geometry of $(\mathcal{X}, \mathrm{dist})$.

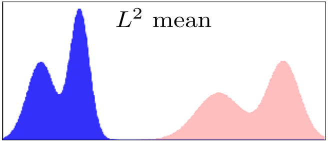
Applications: comparing measures

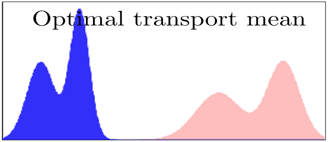
Comparing Measures

→ images, vision, graphics and machine learning, … .



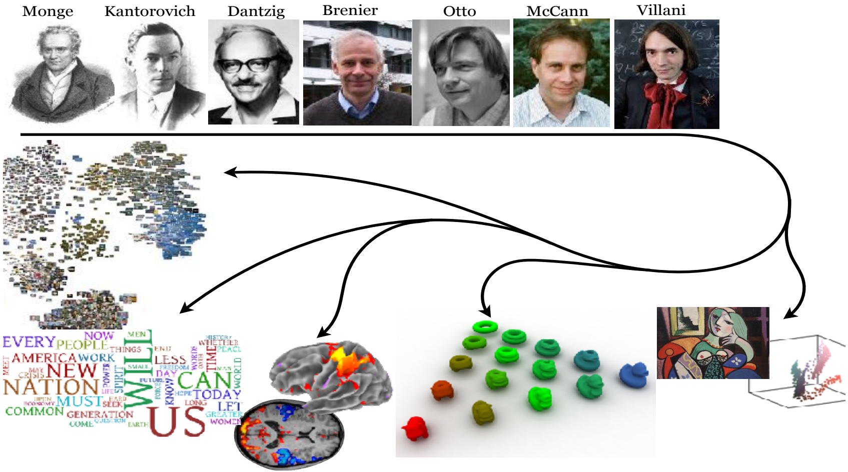
- *Optimal transport*
  → takes into account a metric d.





Applications: toward high-dimensional OT

Toward High-dimensional OT


Monge
Kantorovich
Dantzig
Brenier
Otto
McCann
Villani
EVERY PEOPLE NOW MEN THINGS AMERICA WORK NEW NATION MUST COMMON GENERATION WILL LESS CAN TODAY TIME US LET

Kantorovitch's Formulation

Discrete Optimal Transport

Input two discrete probability measures

$$\alpha = \sum_{i=1}^{m} a_i \delta_{x_i}, \quad \beta = \sum_{j=1}^{n} b_j \delta_{y_j}. \tag{1}$$

- $X = \{x_i\}_i$, $Y = \{x_j\}_j$: are given points clouds, x_i, y_i are vectors.
- a_i, b_j : positive weights, $\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j = 1$.
- C_{ij}: costs, $C_{ij} = c(x_i, y_j) \geq 0$.

Couplings

$$\mathbf{U}(\alpha, \beta) \overset{\text{def}}{=} \{\Pi \in \mathbb{R}_{+}^{m\times n}; \Pi \mathbf{1}_n = a, \Pi^{\top} \mathbf{1}_m = b\} \tag{2}$$

is called the set of couplings with respect to α and β.

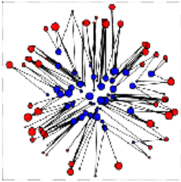
Kantorovitch's Formulation

Discrete Optimal Transport

In the optimal transport, we want to compute the following quantity [Kantorovich 1942]

Optimal transport distance

$$\mathcal{L}(\alpha, \beta, C) \overset{\text{def}}{=} \min \left\{ \sum_{i,j} C_{i,j} \Pi_{i,j}; \Pi \in \mathbf{U}(a, b) \right\}. \tag{3}$$



Push Forward

- Radon measures (α, β) on $(\mathcal{X}, \mathcal{Y})$.
- Transfer of measure by $T : \mathcal{X} \rightarrow \mathcal{Y}$: push forward.
- The measure $T_{\#}\alpha$ on $\mathcal{Y}$ is defined by

$$T_{\#}\alpha(Y) = \alpha(T^{-1}(Y)), \quad \text{for all measurable } Y \in \mathcal{Y}. \tag{4}$$

Equivalently,

$$\int_{\mathcal{Y}} g(y)\mathrm{d}T_{\#}\alpha(y) \overset{\text{def}}{=} \int_{\mathcal{X}} g(T(x))\mathrm{d}\alpha(x). \tag{5}$$

- Discrete measures: $T_{\#}\alpha = \sum_i \alpha_i \delta_{T(x_i)}$
- Smooth densities: $\mathrm{d}\alpha = \rho(x)\mathrm{d}x$, $\mathrm{d}\beta = \xi(x)\mathrm{d}x$.

$$T_{\#}\alpha = \beta \iff \rho(T(x))|\mathbf{det}(\partial T(x))| = \xi(x). \tag{6}$$

Monge problem

- Monge problem seeks for a map that associates to each point x_i a single point y_j, and which must push the mass of α toward the mass of β, namely:

$$\forall j, \quad b_j = \sum_{i: T(x_i) = y_j} a_i$$

- Discrete case:

$$\min_T \quad \sum_i c(x_i, T(x_i)), \quad \text{s.t.} \quad T_{\#}\alpha = \beta$$

- Arbitrary measures:

$$\min_T \quad \int_{\mathcal{X}} c(x, T(x)) d\alpha(x), \quad \text{s.t.} \quad T_{\#}\alpha = \beta$$

Couplings between General Measures

Projectors:

$$\begin{aligned} P_{\mathcal{X}} &: (x,y) \in \mathcal{X} \times \mathcal{Y} \to x \in \mathcal{X}, \\ P_{\mathcal{Y}} &: (x,y) \in \mathcal{X} \times \mathcal{Y} \to y \in \mathcal{Y}. \end{aligned} \tag{7}$$

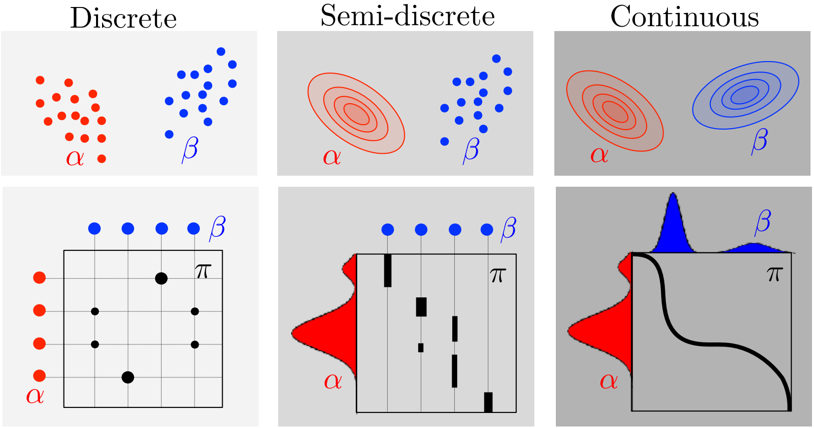
Couplings between General Measures

$$\mathcal{U}(\alpha,\beta) \overset{\text{def}}{=} \{\pi \in \mathcal{M}_+(\mathcal{X} \times \mathcal{Y}); P_{\mathcal{X}\#}\pi = \alpha, P_{\mathcal{Y}\#}\pi = \beta\}. \tag{8}$$

is called the set of couplings with respect to α and β.

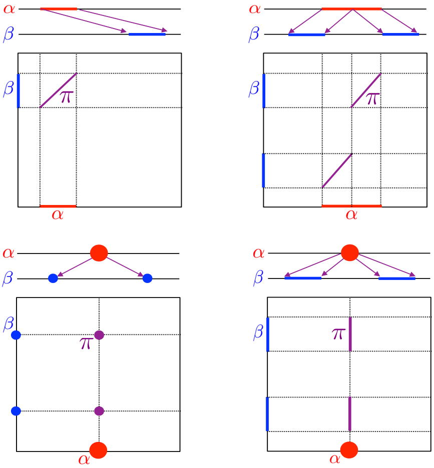
Cases of Couplings

Couplings: the 3 Settings



More Examples

Examples of Couplings



Kantorovich Problem for General Measures

Optimal transport distance between General Measures

$$\mathcal{L}(\alpha, \beta, c) \overset{\text{def}}{=} \min_{\pi \in \mathcal{U}(\alpha,\beta)} \int_{\mathcal{X}\times\mathcal{Y}} c(x, y)\mathrm{d}\pi(x, y). \tag{9}$$

Probability interpretation:

$$\min_{(X,Y)} \{\mathbb{E}_{(X,Y)}(c(X, Y)), X \sim \alpha, Y \sim \beta\}. \tag{10}$$

Wasserstein Distance

Metric Space $\mathcal{X} = \mathcal{Y}$.
Distance $d(x, y)$ (nonegative, symmetric, identity, triangle inequality).
Cost $c(x, y) = d(x, y)^p$, $p \geq 1$.

Wasserstein Distance

$$\mathcal{W}_p(\alpha, \beta) \overset{\text{def}}{=} \mathcal{L}(\alpha, \beta, d^p)^{1/p}. \tag{11}$$

Theorem

$\mathcal{W}_p$ is a distance, and

$$\mathcal{W}_p(\alpha_n, \alpha) \to 0 \iff \alpha_n \overset{\text{weak}}{\to} \alpha. \tag{12}$$

Example

$$\mathcal{W}_p(\delta_x, \delta_y) = d(x, y). \tag{13}$$

Dual form

Dual problem (discrete case)

$$\begin{aligned} \max_{w\in\mathbb{R}^m, r\in\mathbb{R}^n} \quad & w^\top a + r^\top b, \\ \text{s.t.} \quad & w_i + r_j \leq C_{ij}, \quad \forall (i,j) \end{aligned} \tag{14}$$

Relation between any primal and dual solutions:

$$P_{ij} > 0 \Rightarrow w_i + r_j = C_{ij}.$$

Wasserstein barycenter

- Define $C \overset{\text{def}}{=} M_{XY}$, where $(M_{XY})_{ij} = d(x_i, y_i)^p$. The Wasserstein distance as

$$\mathcal{L}(a, b, C) \overset{\text{def}}{=} \min \left\{ \sum_{i,j} C_{i,j}\Pi_{i,j}; \Pi \in \mathbf{U}(a, b) \right\}. \tag{15}$$

- Given a set of point clouds and their corresponding probability vector $\{(Y^i, b^i)\}$, $i = 1, \ldots, N$.
- Find a support $X = \{x_i\}$ with a probability vector a such that (X, a) is the optimal solution of the following problem

$$\min_{X,a} \sum_{k=1}^{N} \lambda_k \mathcal{L}(a, b^k, M_{XY^k}),$$

where $\sum_k \lambda_k = 1$ and $\lambda_k \geq 0$.

Outline

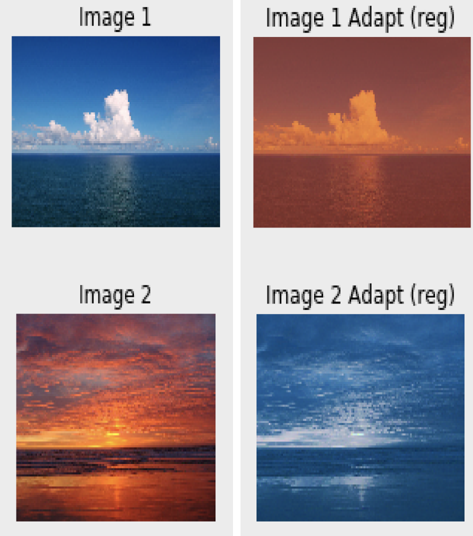
1. Problem Formulation
2. Applications
3. Entropic Regularization
4. Sinkhorn's Algorithm
5. Sinkhorn-Newton method
6. Wasserstein barycenter

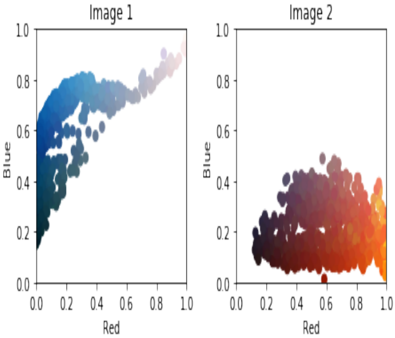
Applications: image color adaptation

Example: `https://github.com/rflamary/POT/blob/master/notebooks/plot_otda_color_images.ipynb`

```
Given color image stored in the RGB format: I1, I2
# Converts an image to matrix (one pixel per line)
X1 = im2mat(I1), X2 = im2mat(I2)
# Take samples
Xs = X1[idx1, :], Xt = X2[idx2, :]
# Scatter plot of colors
pl.scatter(Xs[:, 0], Xs[:, 2], c=Xs)
# Sinkhorn Transport
ot_sinkhorn = ot.da.SinkhornTransport(reg_e=1e-1)
ot_sinkhorn.fit(Xs=Xs, Xt=Xt)
# prediction between images
transp_Xs_sinkhorn = ot_sinkhorn.transform(Xs=X1)
transp_Xt_sinkhorn = ot_sinkhorn.inverse_transform(Xt=X2)
```

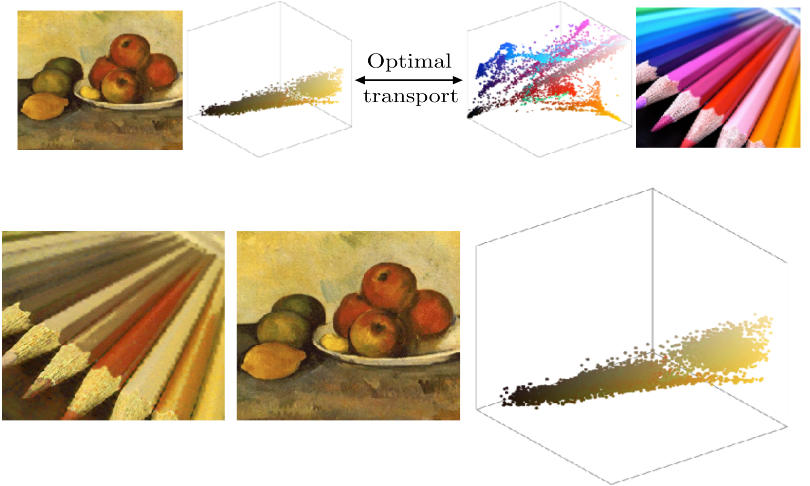
Applications: image color adaptation





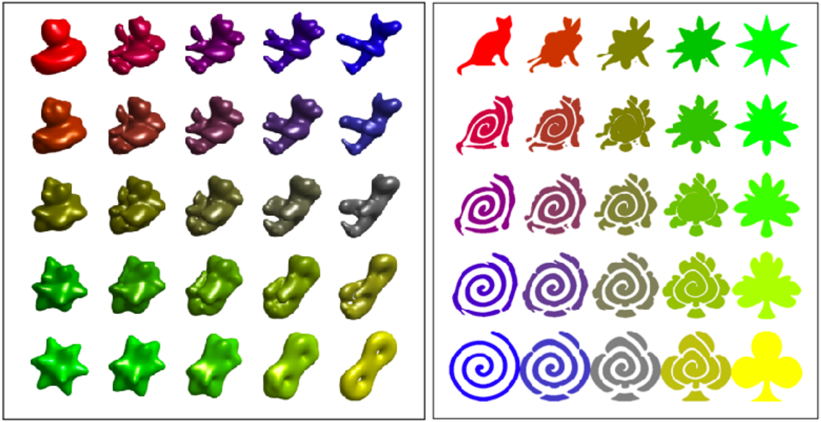
Applications: image color palette equalization

Image Color Palette Equalization



Applications: shape interpolation

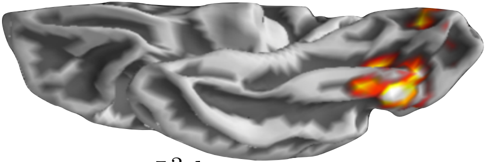
Shape Interpolation



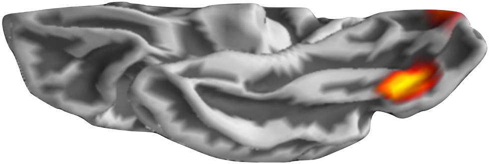
Applications: MRI Data Processing

MRI Data Procesing [with A. Gramfort]

Ground cost $c = d_M$: geodesic on cortical surface M.



L^2 barycenter

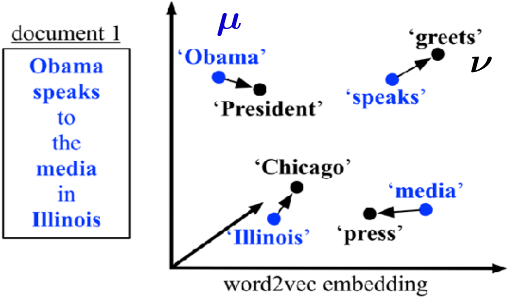


W_2^2 barycenter

Applications: word mover's distance

normalized bag-of-words (nBOW), word travel cost (word2vec distance), document distance $T_{ij}c(i,j)$, transportation problem

Bag of Words



[Kusner'15] $\mathrm{dist}(D_1, D_2) = W_2(\boldsymbol{\mu}, \boldsymbol{\nu})$

Applications: word mover's distance

$$
\begin{aligned}
\min_{\Pi \geq 0} \quad & \sum_{ij} \Pi_{ij} c_{ij} \\
\text{s.t.} \quad & \sum_{j=1}^{n} \Pi_{ij} = d_i \\
& \sum_{i=1}^{n} \Pi_{ij} = d'_j
\end{aligned}
$$

- x_i: word2vec embedding
- $c_{ij} = \|x_i - x_j\|_2$
- if word i appears w_i times in the document, we denote $d_i = \frac{w_i}{\sum w_j}$

Distributional Robust Optimization (DRO)

- stochastic optimization:

  $$\inf_{\beta \in B} E_{P^*}[\ell(\beta^\top X)],$$

  where B is a convex set, ℓ is a loss function, $E_{P^*}[\cdot]$ represents the expectation operator associated to the probability model P^*, which describes the random element X.

- The DRO model:

  $$\inf_{\beta \in B} \sup_{P \in \mathcal{U}_\delta(P_0)} E_P[\ell(\beta^\top X)],$$

  where $\mathcal{U}_\delta(P_0)$ is a so-called distributional uncertainty region "centered" around some benchmark model, P_0, which may be data-driven (for example, an empirical distribution) and δ >0 parameterizes the sizeof the distributional uncertainty.

- Wasserstein distance: $\mathcal{U}_\delta(P_0) = \{P \mid \mathcal{W}(P, P_0) \leq \delta\}$.

Outline

1 Problem Formulation

2 Applications

3 Entropic Regularization

4 Sinkhorn's Algorithm

5 Sinkhorn-Newton method

6 Wasserstein barycenter

Discrete OT Review

Given an integer $n \geqslant 1$, we write Σ_n for the discrete probability simplex

$$\Sigma_n \overset{\text{def}}{=} \left\{ a \in \mathbb{R}_n^+ ; \sum_{i=1}^{n} a_i = 1. \right\} \tag{16}$$

Given $a \in \Sigma_m$, $b \in \Sigma_n$, the Optimal Transport problem is to compute

$$L(a, b, C) \overset{\text{def}}{=} \min\{ \sum_{i,j} C_{i,j} \mathbf{P}_{i,j}; \quad \text{s.t.} \quad \mathbf{P} \in \mathbf{U}(a, b) \}. \tag{17}$$

Where $\mathbf{U}(a, b)$ is the set of couplings between a and b.

Entropy

The discrete entropy of a positive matrix $\mathbf{P}$ ($\sum_{ij} \mathbf{P}_{ij} = 1$) is defined as

$$H(\mathbf{P}) \overset{\text{def}}{=} -\sum_{i,j} \mathbf{P}_{i,j}(\log(\mathbf{P}_{i,j}) - 1). \tag{18}$$

For a positive vector $u \in \Sigma_n$, the entropy is defined analogously:

$$H(\mathbf{u}) \overset{\text{def}}{=} -\sum_{i} \mathbf{u}_i(\log(\mathbf{u}_i) - 1). \tag{19}$$

For two positive vector $u, v \in \Sigma_n$, the Kullback-Leibler divergence (or, KL divergence) is defined to be

$$\textbf{KL}(u\|v) = -\sum_{i=1}^{n} u_i \log(\frac{v_i}{u_i}). \tag{20}$$

The KL divergence is always non-negative: $\textbf{KL}(u\|v) \geq 0$ (Jensen's inequality: $E[f(g(X))] \geq f(E[g(X)])$).

Entropic regularization

- Given $a \in \Sigma_m$, $b \in \Sigma_n$ and cost matrix $\mathbf{C} \in \mathbb{R}_+^{m\times n}$. The entropic regularization of the transportation problem reads

$$L^\varepsilon(a, b, \mathbf{C}) = \min_{\mathbf{P}\in\mathbf{U}(a,b)} \langle \mathbf{P}, \mathbf{C}\rangle - \varepsilon H(\mathbf{P}). \tag{21}$$

- The case $\varepsilon = 0$ corresponds to the classic (linear) optimal transport problem.
- For $\varepsilon > 0$, problem (21) has an ε-strongly convex objective and therefore admits a unique optimal solution $\mathbf{P}_\varepsilon^\star$.

- This is not (necessarily) true for $\varepsilon = 0$. But we have the following proposition.

Entropic regularization

Proposition

When $\varepsilon \to 0$, the unique solution P_ε of (21) converges to the optimal solution with maximal entropy within the set of all optimal solutions of the unregularized transportation problem, namely,

$$\mathbf{P}_\varepsilon \overset{\varepsilon \to 0}{\to} \text{argmax}_{\mathbf{P}}\{H(\mathbf{P}); \mathbf{P} \in U(a,b), \langle \mathbf{P}, \mathbf{C} \rangle = L^0(a,b,\mathbf{C})\} \tag{22}$$

The above proposition motivates us to solve the problems in (21) sequentially and then take $\epsilon \to 0$.

Entropic regularization

Proof

We consider a sequence $(\varepsilon_\ell)_\ell$ such that $\varepsilon_\ell \to 0$ and $\varepsilon_\ell > 0$. We denote $\mathbf{P}_\ell = \mathbf{P}^\star_{\varepsilon_\ell}$. Since $\mathbf{U}(a, b)$ is bounded, we can extract a sequence (that we do not relabel for the sake of simplicity) such that $\mathbf{P}_\ell \to \mathbf{P}^\star$. Since $\mathbf{U}(a, b)$ is closed, $\mathbf{P}^\star \in \mathbf{U}(a, b)$. We consider any $\mathbf{P}$ such that $\langle \mathbf{C}, \mathbf{P} \rangle = L^0(a, b, \mathbf{C})$. By optimality of $\mathbf{P}$ and $\mathbf{P}_\ell$ for their respective optimization problems (for $\varepsilon = 0$ and $\varepsilon = \varepsilon_\ell$), one has

$$0 \leq \langle \mathbf{C}, \mathbf{P}_\ell \rangle - \langle \mathbf{C}, \mathbf{P} \rangle \leq \varepsilon_\ell (H(\mathbf{P}_\ell) - H(\mathbf{P})). \tag{23}$$

Since H is continuous, taking the limit $\ell \to +\infty$ in this expression shows that $\langle C, \mathbf{P}^\star \rangle = \langle C, \mathbf{P} \rangle$. Furthermore, dividing by ε_ℓ and taking the limit shows that $H(\mathbf{P}) \leqslant H(\mathbf{P}^\star)$. Now the result follows from the strictly convexity of $-H$.

Entropic regularization

By the concavity of entropy, for $\alpha > 0$, we introduce the convex set

$$\begin{aligned}\mathbf{U}_\alpha(a,b) &\overset{\text{def}}{=} \{\mathbf{P} \in \mathbf{U}(a,b) | \mathbf{KL}(\mathbf{P} \| ab^\top) \le \alpha\} \\ &= \{\mathbf{P} \in \mathbf{U}(a,b) | H(\mathbf{P}) \ge H(a) + H(b) - 1 - \alpha\}.\end{aligned} \tag{24}$$

Definition: Sinkhorn Distance

$$d_{\mathbf{C},\alpha}(a,b) \overset{\text{def}}{=} \min_{\mathbf{P} \in \mathbf{U}_\alpha(a,b)} \langle \mathbf{C}, \mathbf{P} \rangle. \tag{25}$$

Proposition

For $\alpha \ge 0$, $d_{\mathbf{C},\alpha}(a,b)$ is symmetric and satisfies all triangle inequalities. Moreover, $\mathbf{1}_{a \ne b} d_{\mathbf{C},\alpha}(a,b)$ satisfies all three distance axioms.

Entropic regularization

Proposition

For α large enough, the Sinkhorn distance $d_{C,\alpha}$ is the transport distance $d_{\mathbf{C}}$.

Proof.

Note that for any $\mathbf{P} \in U(a,b)$, we have

$$H(\mathbf{P}) \geq \frac{1}{2}(H(a) + H(b)), \tag{26}$$

so for $\alpha \geq \frac{1}{2}(H(a) + H(b)) - 1$, we have

$$U_\alpha(a,b) = U(a,b).$$

Outline

1. Problem Formulation
2. Applications
3. Entropic Regularization
4. Sinkhorn's Algorithm
5. Sinkhorn-Newton method
6. Wasserstein barycenter

Sinkhorn's algorithm

For solving (21), consider its Lagrangian dual function

$$\mathcal{L}_{\mathbf{C}}^{\varepsilon}(\mathbf{P}, w, r) = \langle \mathbf{C}, \mathbf{P} \rangle - \varepsilon H(\mathbf{P}) + w^{\top}(\mathbf{P}\mathbf{1}_n - a) + r^{\top}(\mathbf{P}^{\top}\mathbf{1}_m - b). \tag{27}$$

Now let $\partial \mathcal{L}_{\mathbf{C}}^{\varepsilon} / \partial \mathbf{P}_{ij} = 0$, i.e.,

$$\mathbf{P}_{ij} = e^{-\frac{c_{ij} + w_i + r_j}{\varepsilon}}, \tag{28}$$

so we can write

$$\mathbf{P}_{\varepsilon} = \textbf{diag}(e^{-\frac{w}{\varepsilon}}) e^{-\frac{\mathbf{C}}{\varepsilon}} \textbf{diag}(e^{-\frac{r}{\varepsilon}}). \tag{29}$$

Note that

$$\mathbf{P}_{\varepsilon}\mathbf{1}_n = a, \quad \mathbf{P}_{\varepsilon}^{\top}\mathbf{1}_m = b, \tag{30}$$

we can then use Sinkhorn's algorithm to find $\mathbf{P}_{\varepsilon}$!

Sinkhorn's algorithm

Let $u = e^{-\frac{w}{\varepsilon}}$, $v = e^{-\frac{r}{\varepsilon}}$ and $\mathbf{K} = e^{-\mathbf{C}/\varepsilon}$. We again state the KKT system of (21):

$$\begin{aligned} \mathbf{P}_\varepsilon &= \textbf{diag}(u)\mathbf{K}\textbf{diag}(v), \\ a &= \textbf{diag}(u)\mathbf{K}v, \\ b &= \textbf{diag}(v)\mathbf{K}^\top u. \end{aligned} \tag{31}$$

Then the Sinkhorn's algorithm amounts to alternating updates in the form of

$$\begin{aligned} u^{(k+1)} &= \textbf{diag}(\mathbf{K}v^{(k)})^{-1}a, \\ v^{(k+1)} &= \textbf{diag}(\mathbf{K}^\top u^{(k+1)})^{-1}b. \end{aligned} \tag{32}$$

Sinkhorn's algorithm

Sinkhorn's algorithm

1. Compute $\mathbf{K} = e^{-\frac{\mathbf{C}}{\varepsilon}}$.
2. Compute $\hat{\mathbf{K}} = \textbf{diag}(a^{-1})\mathbf{K}$.
3. Initial scale factor $u \in \mathbb{R}^m$.
4. Iteratively update u:

   $$u = 1./(\hat{\mathbf{K}}(b./(\mathbf{K}^\top u))),$$

   until reaches certain stopping criterion.
5. Compute

   $$v = b./(\mathbf{K}^\top u),$$

   and eventually

   $$\mathbf{P}_\varepsilon = \textbf{diag}(u)\mathbf{K}\textbf{diag}(v).$$

Outline

1. Problem Formulation
2. Applications
3. Entropic Regularization
4. Sinkhorn's Algorithm
5. Sinkhorn-Newton method
6. Wasserstein barycenter

Sinkhorn-Newton method

The dual problem of (21) is

$$\begin{aligned} \min_{w,r} \quad & \langle a, w\rangle + \langle b, r\rangle + \varepsilon\langle e^{-\frac{w}{\varepsilon}}, \mathbf{K}e^{-\frac{r}{\varepsilon}}\rangle, \\ \text{s.t.} \quad & \textbf{diag}(e^{-\frac{w}{\varepsilon}})\mathbf{K}e^{-\frac{r}{\varepsilon}} = a, \\ & \textbf{diag}(e^{-\frac{r}{\varepsilon}})\mathbf{K}^\top e^{-\frac{w}{\varepsilon}} = b. \end{aligned} \tag{33}$$

with w, r being the dual variables.

Sinkhorn-Newton method

Let

$$F(w,r) = \begin{pmatrix} \textbf{diag}(e^{-\frac{w}{\varepsilon}})\mathbf{K}e^{-\frac{r}{\varepsilon}} - a \\ \textbf{diag}(e^{-\frac{r}{\varepsilon}})\mathbf{K}^\top e^{-\frac{w}{\varepsilon}} - b \end{pmatrix}. \tag{34}$$

We want to find w, r such that $F(w,r) = 0$ so that

$$\mathbf{P}_\varepsilon = \textbf{diag}(e^{-\frac{w}{\varepsilon}})e^{-\frac{\mathbf{C}}{\varepsilon}}\textbf{diag}(e^{-\frac{r}{\varepsilon}}). \tag{35}$$

The Newton iteration is given by

$$\begin{pmatrix} w^{(k+1)} \\ r^{(k+1)} \end{pmatrix} = \begin{pmatrix} w^{(k)} \\ r^{(k)} \end{pmatrix} - J_F^{-1}(w^{(k)}, r^{(k)})F(w^{(k)}, r^{(k)}), \tag{36}$$

where

$$J_F = \frac{1}{\varepsilon}\begin{pmatrix} \textbf{diag}(\mathbf{P}\mathbf{1}_n) & \mathbf{P} \\ \mathbf{P}^\top & \text{diag}(\mathbf{P}^\top\mathbf{1}_m) \end{pmatrix}. \tag{37}$$

Sinkhorn-Newton method: Convergence

Proposition

For $w \in \mathbb{R}^m$ and $r \in \mathbb{R}^n$, the Jacobian matrix $J_F(w,r)$ is symmetric positive semidefinite, and its kernel is given by

$$\mathbf{ker}(J_F(w,r)) = \mathbf{span}\left\{ \begin{pmatrix} \mathbf{1}_m \\ -\mathbf{1}_n \end{pmatrix} \right\}. \tag{38}$$

Proof

J_F is clearly symmetric. For arbitrary $\gamma \in \mathbb{R}^m$ and $\phi \in \mathbb{R}^n$, one has

$$\begin{pmatrix} \gamma^\top & \phi^\top \end{pmatrix} J_F \begin{pmatrix} \gamma \\ \phi \end{pmatrix} = \frac{1}{\varepsilon} \sum_{ij} \mathbf{P}_{ij} (\gamma_i + \phi_j)^2 \geq 0,$$

which holds with equality if and only if $\gamma_i + \phi_j = 0$ for all i, j, leading us to (38).

Sinkhorn-Newton method: Convergence

Lemma

Let $F : D \to \mathbb{R}^n$ be a continuously differentiable mapping with $D \subset \mathbb{R}^n$ open and convex. Suppose that $F(x)$ is invertible for each $x \in D$. Assume that the following affine covariant Lipschitz condition holds

$$\|F'(x)^{-1}(F'(y) - F'(x))(y - x)\| \le \omega \|y - x\|^2 \tag{39}$$

for $x, y \in D$. Let $F(x) = 0$ have a solution x^*. For the initial guess $x^{(0)}$ assume that $B(x^*, \|x^{(0)} - x^*\|) \subset D$ and that

$$\omega \|x^{(0)} - x^*\| < 2.$$

Then the ordinary Newton iterates remain in the open ball $B(x^*, \|x^{(0)} - x^*\|)$ and converge to x^* at an estimated quadratic rate

$$\|x^{(k+1)} - x^*\| \le \frac{\omega}{2} \|x^{(k)} - x^*\|^2. \tag{40}$$

Moreover, the solution x^* is unique in the open ball $B(x^*, 2/\omega)$.

Sinkhorn-Newton method: Convergence

Proof

Denote $e^{(k)} = x^{(k)} - x^*$. Let us prove the lemma by induction:

$$\begin{aligned} \|e^{(k+1)})\| &= \|x^{(k)} - (F'(x^{(k)}))^{-1}(F(x^{(k)} - F(x^*)) - x^*\| \\ &= \|e^{(k)} - (F'(x^{(k)}))^{-1}(F(x^{(k)} - F(x^*))\| \\ &= \|(F'(x^{(k)}))^{-1}((F(x^*) - F(x^{(k)})) + F'(x^{(k)})e^{(k)})\| \\ &= \|(F'(x^{(k)}))^{-1} \int_{s=0}^{-1} (F'(x^{(k)} + se^{(k)}) - F'(x^{(k)}))e^{(k)} \, \mathrm{d}s\| \\ &\leq \omega\| \int_{s=0}^{-1} s \, \mathrm{d}s \|e^{(k)}\|^2 = \frac{\omega}{2}\|e^{(k)}\|^2 < \|e^{(k)}\|. \end{aligned} \tag{41}$$

Also

$$\omega\|e^{(k+1)}\| \leq \omega\|e^{(k)}\| < 2. \tag{42}$$

For the uniqueness part, let $x^{(0)} = x^{**} \neq x^*$ be a different solution,then $x^{(1)} = x^{**}$, then consider (40) when $k = 0$.

Sinkhorn-Newton method: Convergence

Proposition

For any $k \in \mathbb{N}$ with $P_{\varepsilon,ij}^{(k)} > 0$, the affine covariante Lipschitz condition holds in the ℓ_∞-norm for

$$\omega \leq (e^{\frac{1}{\varepsilon}} - 1)\left(1 + 2e^{\frac{1}{\varepsilon}} \frac{\max\{\|\mathbf{P}_\varepsilon^{(k)}\mathbf{1}_n\|_\infty, \|(\mathbf{P}_\varepsilon^{(k)})^\top \mathbf{1}_m\|_\infty\}}{\min_{ij} \mathbf{P}_{\varepsilon,ij}^{(k)}}\right) \tag{43}$$

when $\|y - x\|_\infty \leq 1$.

The proof for this proposition is tedious and therefore we refer the interested readers to the paper [**?**].

Relationship with Sinkhorn's algorithm

Let $u = e^{-\frac{w}{\varepsilon}}$, $v = e^{-\frac{r}{\varepsilon}}$ and $\mathbf{K} = e^{-\mathbf{C}/\varepsilon}$. We again state the KKT system of (21):

$$\begin{aligned} \mathbf{P}_\varepsilon &= \textbf{diag}(u)\mathbf{K}\textbf{diag}(v), \\ a &= \textbf{diag}(u)\mathbf{K}v, \\ b &= \textbf{diag}(v)\mathbf{K}^\top u. \end{aligned} \tag{44}$$

Then the Sinkhorn's algorithm amounts to alternating updates in the form of

$$\begin{aligned} u^{(k+1)} &= \textbf{diag}(\mathbf{K}v^{(k)})^{-1}a, \\ v^{(k+1)} &= \textbf{diag}(\mathbf{K}^\top u^{(k+1)})^{-1}b. \end{aligned} \tag{45}$$

Relationship with Sinkhorn's algorithm

Define

$$G(u,v) = \begin{pmatrix} \mathbf{diag}(u)\mathbf{K}v - a \\ \mathbf{diag}(v)\mathbf{K}^\top u - b \end{pmatrix}. \tag{46}$$

Process analogously to the Sinkhorn-Newton method we just discussed, note that

$$J_G(u,v) = \begin{pmatrix} \mathbf{diag}(\mathbf{K}v) & \mathbf{diag}(u)\mathbf{K} \\ \mathbf{diag}(v)\mathbf{K}^\top & \mathbf{diag}(\mathbf{K}^\top u) \end{pmatrix}. \tag{47}$$

If we neglect the off-diagonal blocks above, i.e.,

$$\hat{J}_G(u,v) = \begin{pmatrix} \mathbf{diag}(\mathbf{K}v) & \mathbf{0} \\ \mathbf{0} & \mathbf{diag}(\mathbf{K}^\top u) \end{pmatrix}, \tag{48}$$

and perform the Newton iteration

$$\begin{pmatrix} u^{(k+1)} \\ v^{(k+1)} \end{pmatrix} = \begin{pmatrix} u^{(k)} \\ v^{(k)} \end{pmatrix} - \hat{J}_G^{-1}(u^{(k)}, v^{(k)}) G(u^{(k)}, v^{(k)}), \tag{49}$$

Relationship with Sinkhorn's algorithm

We get

$$\begin{aligned} u^{(k+1)} &= \textbf{diag}(\mathbf{K}v^{(k)})^{-1}a, \\ v^{(k+1)} &= \textbf{diag}(\mathbf{K}^{\top}u^{(k)})^{-1}b. \end{aligned} \tag{50}$$

So the Sinkhorn's algorithm simply approximates one Newton step by neglecting the off-diagonal blocks and replacing $u^{(k)}$ by $u^{(k+1)}$.

Outline

1. Problem Formulation
2. Applications
3. Entropic Regularization
4. Sinkhorn's Algorithm
5. Sinkhorn-Newton method
6. Wasserstein barycenter

Wasserstein barycenter

- Define $C \overset{\text{def}}{=} M_{XY}$, where $(M_{XY})_{ij} = d(x_i, y_i)^p$. The Wasserstein distance as

$$\mathcal{L}(a, b, C) \overset{\text{def}}{=} \min \left\{ \sum_{i,j} C_{i,j}\Pi_{i,j}; \Pi \in \mathbf{U}(a, b) \right\}. \tag{51}$$

- Given a set of point clouds and their corresponding probability vector $\{(Y^i, b^i)\}$, $i = 1, \ldots, N$.
- Find a support $X = \{x_i\}$ with a probability vector a such that (X, a) is the optimal solution of the following problem

$$\min_{X,a} \ \psi(a, X) = \sum_{k=1}^{N} \lambda_k \mathcal{L}(a, b^k, M_{XY^k}), \text{ s.t. } \sum_i a_i = 1, a \geq 0.$$

where $\sum_k \lambda_k = 1$ and $\lambda_k \geq 0$.

Differentiability of $\mathcal{L}(a, b, C)$ w.r.t. a

- The primal problem:

$$\mathcal{L}(a, b, C) \overset{\text{def}}{=} \min_{\Pi} \sum_{i,j} C_{i,j}\Pi_{i,j} \quad \text{s.t.} \quad \Pi \mathbf{1}_n = a, \Pi^\top \mathbf{1}_m = b, \Pi \geq 0.$$

- Let u^* is the optimal dual vector of the dual problem:

$$\max_{u \in \mathbb{R}^m, v \in \mathbb{R}^n} \quad u^\top a + v^\top b, \text{ s.t.} \quad u_i + v_j \leq C_{ij}, \quad \forall (i, j)$$

- Suppose $\mathcal{L}(a, b, C)$ is finite, the strong duality holds. Then u^* is a subgradient of $\mathcal{L}(a, b, C)$ w.r.t. a.

Subgradient of optimal value function

define $h(u, v)$ as the optimal value of convex problem

$$\begin{aligned} \min \quad & f_0(x) \\ \text{s.t.} \quad & f_i(x) \leq u_i,\ i = 1, \cdots, m \\ & Ax = b + v \end{aligned}$$

(functions f_i are convex; optimization variable is x)

weak result: suppose $h(\hat{u}, \hat{v})$ is finite, strong duality holds with the dual

$$\begin{aligned} \max \quad & \inf_x \left(f_0(x) + \sum_i \lambda_i (f_i(x) - \hat{u}_i) + \nu^\top (Ax - b - \hat{v}) \right) \\ \text{s.t.} \quad & \lambda \geq 0 \end{aligned}$$

if $\hat{\lambda}, \hat{\nu}$ are optimal dual variables (for r.h.s. $\hat{u}, \hat{v}$) then $(\hat{\lambda}, \hat{\nu}) \in \partial h(\hat{u}, \hat{v})$

proof : by weak duality for problem with r.h.s. u, v

$$\begin{aligned} h(u,v) &\geq \inf_x \left(f_0(x) + \sum_i \hat{\lambda}_i (f_i(x - u_i) + \hat{\nu}^\top (Ax - b - v) \right) \\ &= \inf_x \left(f_0(x) + \sum_i \hat{\lambda}_i (f_i(x - \hat{u}_i) + \hat{\nu}^\top (Ax - b - \hat{v}) \right) \\ &- \hat{\lambda}^\top (u - \hat{u}) - \hat{\nu}^\top (v - \hat{v}) \\ &= h(\hat{u}, \hat{v}) - \hat{\lambda}^\top (u - \hat{u}) - \hat{\nu}^\top (v - \hat{v}) \end{aligned}$$

minimizing $\psi(a, X)$ w.r.t a

For a fixed X, consider the problem

$$\min_{a} \quad \psi(a, X) = \sum_{k=1}^{N} \lambda_k \mathcal{L}(a, b^k, M_{XY^k}), \text{ s.t. } \sum a_i = 1, a \geq 0$$

- Let u^k be the optimal dual variable of $\mathcal{L}(a, b^k, M_{XY^k})$ w.r.t. a. Then

$$g = \sum_{k=1}^{N} \lambda_k u^k \in \partial_a \psi(a, X)$$

- Let $h(a) = \sum_{i=1}^{m} a_i \log a_i$. The associated Bregman divergence is

$$D_h(y, x) = h(y) - h(x) - \nabla h(x)^T (y - x)$$

- The mirror descent method is

$$a^{j+1} = \underset{\sum a_i = 1, a \geq 0}{\operatorname{argmin}} \left\{ g^T (a - a^j) + \frac{1}{\alpha} D_h(a, a^j) \right\}$$

Minimizing $\psi(a, X)$ w.r.t. X

Denote $X = [x_1, \ldots, x_m]$ and $Y = [y_1, \ldots, y_n]$.

- Consider $(M_{XY})_{ij} = \|x_i - y_i\|_2^2$. Let $\mathrm{x} = \mathrm{diag}(X^\top X)$ and $\mathrm{y} = \mathrm{diag}(Y^\top Y)$. Then we have:

$$M_{XY} = \mathrm{x}1_n^\top + 1_m^\top \mathrm{y} - 2X^\top Y \in \mathbb{R}^{m\times n}$$

- Let Π be the optimal matrix corresponding to a

$$\begin{aligned}
\mathcal{L}(a, b, M_{XY}) &= \langle \Pi, M_{XY} \rangle \\
&= \left\langle \Pi, \mathrm{x}1_n^\top + 1_m^\top \mathrm{y} - 2X^\top Y \right\rangle \\
&= \langle \mathrm{x}, \Pi 1_n \rangle + \left\langle \mathrm{y}, \Pi^\top 1_m \right\rangle - 2 \left\langle \Pi, X^\top Y \right\rangle \\
&= \mathrm{x}^\top a + \mathrm{y}^\top b - 2 \left\langle \Pi, X^\top Y \right\rangle \\
&= \|X \mathrm{diag}(a^{1/2}) - Y \Pi^\top \mathrm{diag}(a^{-1/2})\|_F^2 + \mathrm{const}.
\end{aligned}$$

Minimizing $\psi(a, X)$ w.r.t. X

For a fixed a, consider the problem

$$\min_{X} \quad \psi(a, X) = \sum_{k=1}^{N} \lambda_k \mathcal{L}(a, b^k, M_{XY^k}).$$

Then, it is equivalent to

$$\min{}_X \quad \sum_{k=1}^{N} \lambda_k \left(\mathrm{x}^\top a - 2 \left\langle \Pi^k, X^\top Y^k \right\rangle \right)$$

$$\min{}_X \quad \mathrm{x}^\top a - 2 \left\langle \sum_{k=1}^{N} \lambda_k \Pi^k, X^\top Y^k \right\rangle$$

$$\min{}_X \quad \|X \mathrm{diag}(a^{1/2}) - \sum_{k=1}^{N} \lambda_k Y^k (\Pi^k)^\top \mathrm{diag}(a^{-1/2})\|_F^2$$

The optimal solution is:

$$X = \sum_{k=1}^{N} \lambda_k Y^k (\Pi^k)^\top \mathrm{diag}(a^{-1})$$

Discrete Optimization: Modelling

https://bicmr.pku.edu.cn/~wenzw/bigdata2022.html

Acknowledgement: this slides is based on Prof. Ted Ralphs's lecture notes

Outline

1. Introduction to Integer Programming

2. Integer Programming Modeling and Formulation

3. Constraint Programming (CP)

Mixed Integer Linear Programming

- Consider linear programming with additionally constraints

$$X = \mathbb{Z}_+^p \times \mathbb{R}_+^{n-p}.$$

- The general form of such a mathematical optimization problem is

$$z_{IP} = \max\{c^\top x \mid Ax \le b, x \in \mathbb{Z}_+^p \times \mathbb{R}_+^{n-p}\},$$

  where for $A \in \mathbb{Q}^{m \times n}, b \in \mathbb{Q}^m, c \in \mathbb{Q}^n$.

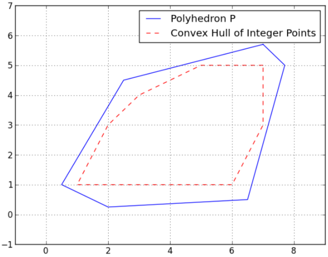
- This type of optimization problem is called a mixed integer linear programming (MILP) problem.
- If $p = n$, then we have a pure integer linear optimization problem.
- Special case: the integer variables are binary, i.e., 0 or 1.

The Geometry of Integer Programming

- Let's consider an integer linear program

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \mathbb{Z}^n_+ \end{aligned}$$

- The feasible region is the integer points inside a polyhedron.



- Why does solving the LP relaxation not necessarily yield a good solution?

How Hard is Integer Programming?

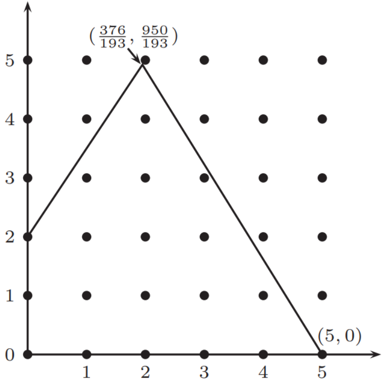
- Solving general integer programs can be much more difficult than solving linear programs.
- There in no known polynomial-time algorithm for solving general MIPs.
- Solving the associated linear programming relaxation results in an upper bound on the optimal solution to the MIP.
- In general, solving the LP relaxation, an LP obtained by dropping the integerality restrictions, does not tell us much.
  - Rounding to a feasible integer solution may be difficult.
  - The optimal solution to the LP relaxation can be arbitrarily far away from the optimal solution to the MIP.
  - Rounding may result in a solution far from optimal.
  - We can bound the difference between the optimal solution to the LP and the optimal solution to the MIP (how?).

How Hard is Integer Programming?

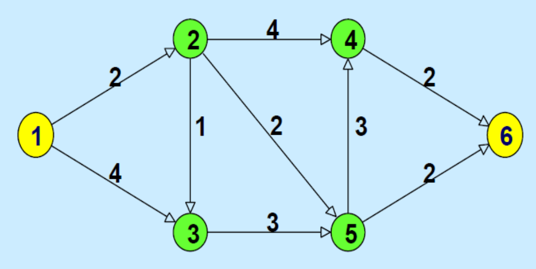
Consider the integer program

$$\begin{aligned} \max \quad & 50x_1 + 32x_2, \\ \text{s.t.} \quad & 50x_1 + 31x_2 \leq 250, \\ & 3x_1 - 2x_2 \geq -4, \\ & x_1, x_2 \geq 0 \text{ and integer.} \end{aligned}$$

The linear programming solution $(376\backslash 193, 950\backslash 193)$ is a long way from the optimal integer solution $(5, 0)$.

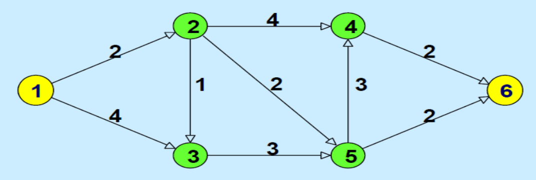


The shortest path problem



- Consider a network G = (N, A) with cost c_{ij} on each edge $(i, j) \in A$. There is an origin node s and a destination node t.
- Standard notation: n = |N|, m = |A|
- cost of of a path: $c(P) = \sum_{(i,j) \in P} c_{ij}$
- What is the shortest path from s to t?

The shortest path problem



$$\min \quad \sum_{(i,j)\in A} c_{ij}x_{ij}$$

$$\text{s.t. } \sum_j x_{sj} = 1$$

$$\sum_j x_{ij} - \sum_j x_{ji} = 0, \text{ for each i } \neq s \text{ or } t$$

$$-\sum_i x_{it} = -1$$

$$x_{ij} \in \{0,1\} \text{ for all } (i,j)$$

Conjunction versus Disjunction

- A more general mathematical view that ties integer programming to logic is to think of integer variables as expressing **disjunction**.
- The constraints of a standard mathematical program are **conjunctive**.
  - All constraints must be satisfied.

$$g_1(x) \leq b_1 \text{ AND } g_2(x) \leq b_1 \text{ AND } \cdots \text{ AND } g_m(x) \leq b_m$$

  - This corresponds to intersection of the regions associated with each constraint.
- Integer variables introduce the possibility to model disjunction.
  - At least one constraint must be satisfied.

$$g_1(x) \leq b_1 \text{ OR } g_2(x) \leq b_1 \text{ OR } \cdots \text{ OR } g_m(x) \leq b_m$$

  - This corresponds to union of the regions associated with each constraint.

Representability Theorem

The connection between integer programming and disjunction is captured most elegantly by the following theorem.

Theorem

A set $\mathcal{F} \subseteq \mathbb{R}^n$ *is MIP representable if and only if there exist rational polytopes* $\mathcal{P}_1, \cdots, \mathcal{P}_k$ *and vectors* $r^1, \cdots, r^t \in \mathbb{Z}^n$ *such that*

$$\mathcal{F} = \bigcup_{i=1}^{n} \mathcal{P}_i + \operatorname{intcone}\{r^1, \cdots, r^t\}.$$

where $\operatorname{intcone}\{r^1, \cdots, r^t\} = \left\{\sum_{i=1}^{t} \lambda_i r_i \mid \lambda \in \mathbb{Z}^t_+\right\}$

Roughly speaking, we are optimizing over a union of polyhedra, which can be obtained simply by introducing a disjunctive logical operator to the language of linear programming.

Outline

1. Introduction to Integer Programming
2. Integer Programming Modeling and Formulation
3. Constraint Programming (CP)

Modeling with Integer Variables

- From a practical standpoint, why do we need integer variables?
- Integer variable essentially allow us to introduce disjunctive logic.
- If the variable is associated with a physical entity that is indivisible, then the value must be integer.
- At its heart, integrality is a kind of disjunctive constraint.
- 0-1 (binary) variables are often used to model more abstract kinds of disjunctions (non-numerical).
    - Modeling yes/no decisions.
    - Enforcing logical conditions.
    - Modeling fixed costs.
    - Modeling piecewise linear functions.

Modeling Binary Choice

- We use binary variables to model yes/no decisions.
- Example: Integer knapsack problem
  - We are given a set of items with associated values and weights.
  - We wish to select a subset of maximum value such that the total weight is less than a constant K.
  - We associate a 0-1 variable with each item indicating whether it is selected or not.

$$\begin{array}{ll} \max & \sum_{j=1}^{m} c_j x_j \\ \text{s.t.} & \sum_{j=1}^{m} w_j x_j \leq K \\ & x \in \{0,1\}^n \end{array}$$

Modeling Dependent Decisions

- We can also use binary variables to enforce the condition that a certain action can only be taken if some other action is also taken.
- Suppose x and y are binary variables representing whether or not to take certain actions.
- The constraint $x \leq y$ says "only take action x if action y is also taken"

MIP reformulation of ℓ_0-minimization

- Big-M assumption: $\forall i$, $|x_i| \leq M$

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{s.t.} \quad \|x\|_0 \leq k, \, |x_i| \leq M$$

- MIP formulation:

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{s.t.} \quad \sum_{i=1}^{n} y_i \leq k, \, |x_i| \leq M y_i, y_i \in \{0, 1\}$$

Example: Facility Location Problem

- We are given n potential facility locations and m customers.
- There is a fixed cost c_j of opening facility j.
- There is a cost d_{ij} associated with serving customer i from facility j.
- We have two sets of binary variables.
  - y_j is 1 if facility j is opened, 0 otherwise.
  - x_{ij} is 1 if customer i is served by facility j, 0 otherwise.
- Here is one formulation:

$$\begin{array}{lll} \min & \sum_{j=1}^{n} c_j y_j + \sum_{i=1}^{m} \sum_{j=1}^{n} d_{ij} x_{ij} & \\ \text{s.t.} & \sum_{j=1}^{n} x_{ij} = 1 & \forall i \\ & \sum_{i=1}^{m} x_{ij} \le m y_j & \forall j \\ & x_{ij}, y_i \in \{0,1\} & \forall i,j \end{array}$$

Selecting from a Set

- We can use constraints of the form $\sum_{j\in T} x_j \geq 1$ to represent that at least one item should be chosen from a set T.

- Similarly, we can also model that at most one or exactly one item should be chosen.

- Example: Set covering problem
  - A set covering problem is any problem of the form.

$$\min \quad \{c^\top x \mid Ax \geq 1, x_j \in \{0,1\}\}$$

    where A is a 0-1 matrix.
  - Each row of A represents an item from a set S.
  - Each column A_j represents a subset S_j of the items.
  - Each variable x_j represents selecting subset S_j.
  - In other words, each item must appear in at least one selected subset.

Modeling Disjunctive Constraints

- We are given two constraints $a^\top x \geq b$ and $c^\top x \geq d$ with nonnegative coefficients.
- Instead of insisting both constraints be satisfied, we want at least one of the two constraints to be satisfied.
- To model this, we define a binary variable y and impose

$$\begin{aligned} a^\top x &\geq yb, \\ c^\top x &\geq (1-y)d, \\ y &\in \{0,1\}. \end{aligned}$$

- More generally, we can impose that at least k out of m constraints be satisfied with

$$\begin{aligned} a_i^\top x &\geq y_i b_i, \\ \sum_{i=1}^{m} y_i &\geq k, \\ y_i &\in \{0,1\}. \end{aligned}$$

Modeling Disjunctive Constraints (cont'd)

- Consider the disjunctive constraints $a^\top x \geq b$ and $c^\top x \geq d$ where the coefficients are allowed to be negative.

- To model this, we use the Big-M Reformulation. we define a binary variable y and impose

$$\begin{aligned} a^\top x &\geq b - My, \\ c^\top x &\geq d - M(1-y), \\ y &\in \{0,1\}. \end{aligned}$$

  where M is a sufficiently large positive number.

Modeling a Restricted Set of Values

- We may want variable x to only take on values in the set $\{a_1, \cdots, a_m\}$.
- We introduce m binary variables y_j, $j = 1, \cdots, m$ and the constraints

$$x = \sum_{j=1}^{m} a_j y_j,$$

$$\sum_{j=1}^{m} y_j = 1,$$

$$y_j \in \{0, 1\}.$$

Fixed-charge Problems

- In many instances, there is a fixed cost and a variable cost associated with a particular decision.
- Example: Fixed-charge Network Flow Problem
  - We are given a directed graph $G = (N, A)$.
  - There is a fixed cost c_{ij} associated with "opening" arc (i, j) (think of this as the cost to "build" the link).
  - There is also a variable cost d_{ij} associated with each unit of flow along arc (i, j).
  - Consider an instance with a single supply node.
    - Minimizing the fixed cost by itself is a minimum spanning tree problem (easy).
    - Minimizing the variable cost by itself is a minimum cost network flow problem (easy).
    - We want to minimize the sum of these two costs (difficult).

Modeling the Fixed-charge Network Flow Problem

- To model the FCNFP, we associate two variables with each arc.
  - x_{ij} (fixed-charge variable) indicates whether arc (i,j) is open.
  - f_{ij} (flow variable) represents the flow on arc (i,j).
  - Note that we have to ensure that $f_{ij} > 0 \Rightarrow x_{ij} = 1$.

$$
\begin{array}{lll}
\min & \displaystyle\sum_{(i,j)\in A} c_{ij}x_{ij} + d_{ij}f_{ij} & \\
\textsf{s.t.} & \displaystyle\sum_{j\in O(i)} f_{ij} - \sum_{j\in I(i)} f_{ji} = b_i, & \forall i \in N \\
& f_{ij} \leq Cx_{ij}, & \forall (i,j) \in A \\
& f_{ij} \geq 0, & \forall (i,j) \in A \\
& x_{ij} \in \{0,1\}, & \forall (i,j) \in A
\end{array}
$$

Alternative Formulations

- A key concept in the rest of the course will be that every mathematical model has many alternative formulations.
- Many of the key methodologies in integer programming are essentially automatic methods of reformulating a given model.
- The goal of the reformulation is to make the model easier to solve.

Simple Example: Knapsack Problem

- We are given a set $N = \{1, \cdots, n\}$ of items and a capacity K.
- There is a profit c_i and a size w_i associated with each item $i \in N$.
- We want to choose the set of items that maximizes profit subject to the constraint that their total size does not exceed the capacity.
- The most straightforward formulation is to introduce a binary variable x_i associated with each item.
- x_i takes value 1 if item i is chosen and 0 otherwise.
- Then the formulation is

$$
\begin{aligned}
\min \quad & \sum_{j=1}^{n} c_j x_j \\
\text{s.t.} \quad & \sum_{j=1}^{n} w_j x_j \leq K, \\
& x_i \in \{0, 1\}, \quad \forall i
\end{aligned}
$$

An Alternative Formulation

- Let us call a set $C \subseteq N$ a cover is $\sum_{i \in C} w_i > K$.
- Further, a cover C is minimal if $\sum_{i \in C \setminus \{j\}} w_i \leq K$ for all $j \in C$.
- Then we claim that the following is also a valid formulation of the original problem.

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} c_j x_j, \\ \text{s.t.} \quad & \sum_{j \in C} x_j \leq |C| - 1, \quad \text{for all minimal covers } C \\ & x_i \in \{0, 1\}, \quad i \in N \end{aligned}$$

- Which formulation is "better"?

Back to the Facility Location Problem

- Here is another formulation for the same problem:

$$\begin{array}{lll}
\min & \displaystyle\sum_{j=1}^{n} c_j y_j + \sum_{i=1}^{m}\sum_{j=1}^{n} d_{ij} x_{ij} & \\
\text{s.t.} & \displaystyle\sum_{j=1}^{n} x_{ij} = 1, & \forall i, \\
 & x_{ij} \leq y_j, & \forall i, j, \\
 & x_{ij}, y_j \in \{0, 1\}, & \forall i, j.
\end{array}$$

- Notice that the set of integer solutions contained in each of the polyhedra is the same (why?).
- However, the second polyhedron is strictly included in the first one (how do we prove this?).
- Therefore, the second polyhedron will yield a better lower bound.
- The second polyhedron is a better approximation to the convex hull of integer solutions.

Formulation Strength and Ideal Formulations

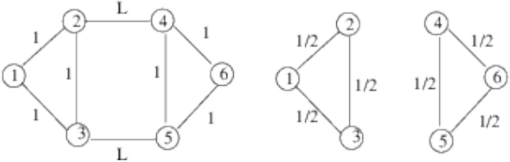
- Consider two formulations A and B for the same ILP.
- Denote the feasible regions corresponding to their LP relaxations as $\mathcal{P}_A$ and $\mathcal{P}_B$.
- Formulation A is said to be at least as strong as formulation B if $\mathcal{P}_A \subseteq \mathcal{P}_B$
- If the inclusion is strict, then A is stronger than B.
- If $\mathcal{S}$ is the set of all feasible integer solutions for the ILP, then we must have $\mathrm{conv}(\mathcal{S}) \subseteq \mathcal{P}_A$ (why?).
- A is ideal if $\mathrm{conv}(\mathcal{S}) = \mathcal{P}_A$.
- If we know an ideal formulation, we can solve the IP (why?).
- How do our formulations of the knapsack problem compare by this measure?

Strengthening Formulations

- Often, a given formulation can be strengthened with additional inequalities satisfied by all feasible integer solutions.
- Example: given a graph G = (V, E), a perfect matching in G is a subset M of edge set E, such that every vertex in V is adjacent to exactly one edge in M.
  - We are given a set of n people that need to paired in teams of two.
  - Let c_{ij} represent the "cost" of the team formed by person i and person j.
  - The nodes represent the people and the edges represent pairings.
  - We have $x_e = 1$ if the endpoints of e are matched, $x_e = 0$ otherwise.

$$
\begin{aligned}
\min \quad & \sum_{e=\{i,j\}\in E} c_e x_e \\
\text{s.t.} \quad & \sum_{\{j|\{i,j\}\in E\}} x_{ij} = 1, && \forall i \in N \\
& x_e \in \{0,1\}, && \forall e = \{i,j\} \in E
\end{aligned}
$$

Valid Inequalities for Matching



- Consider the graph on the left above.
- The optimal perfect matching has value L + 2.
- The optimal solution to the LP relaxation has value 3.
- This formulation can be extremely weak.
- Add the valid inequality $x_{24} + x_{35} \geq 1$.
- Every perfect matching satisfies this inequality.

The Odd Set Inequalities

- We can generalize the inequality from the last slide.
- Consider the cut S corresponding to any odd set of nodes.
- The cutset corresponding to S is

$$\delta(S) = \{\{i,j\} \in E | i \in S, j \notin S\}.$$

- An odd cutset is any $\delta(S)$ for which the $|S|$ is odd.
- Note that every perfect matching contains at least one edge from every odd cutset.
- Hence, each odd cutset induces a possible valid inequality.

$$\sum_{e \in \delta(S)} x_e \geq 1, S \subset N, |S| \text{ odd.}$$

Using the New Formulation

- If we add all of the odd set inequalities, the new formulation is ideal.
- Hence, we can solve this LP and get a solution to the IP.
- However, the number of inequalities is exponential in size, so this is not really practical.
- Recall that only a small number of these inequalities will be active at the optimal solution.
- Later, we will see how we can efficiently generate these inequalities on the fly to solve the IP

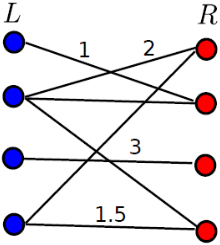
Contrast with Linear Programming

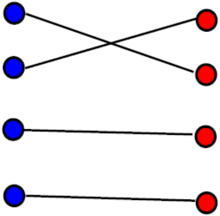
- In linear programming, the same problem can also have multiple formulations.
- In LP, however, conventional wisdom is that bigger formulations take longer to solve.
- In IP, this conventional wisdom does not hold.
- We have already seen two examples where it is not valid.
- Generally speaking, the size of the formulation does not determine how difficult the IP is.

The Max-Weight Bipartite Matching Problem

Given a bipartite graph G = (N, A), with N = L $\cup$ R, and weights w_{ij} on edges (i,j), find a maximum weight matching.

- Matching: a set of edges covering each node at most once
- Let n=|N| and m = |A|.
- Equivalent to maximum weight / minimum cost perfect matching.





The Max-Weight Bipartite Matching

Integer Programming (IP) formulation

$$\begin{aligned} \max \quad & \sum_{ij} w_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j} x_{ij} \leq 1, \forall i \in L \\ & \sum_{i} x_{ij} \leq 1, \forall j \in R \\ & x_{ij} \in \{0, 1\}, \forall (i,j) \in A \end{aligned}$$

- $x_{ij} = 1$ indicate that we include edge (i, j) in the matching
- IP: non-convex feasible set

The Max-Weight Bipartite Matching

Integer program (IP)

$$\begin{aligned} \max \quad & \sum_{ij} w_{ij}x_{ij} \\ \text{s.t.} \quad & \sum_{j} x_{ij} \leq 1, \forall i \in L \\ & \sum_{i} x_{ij} \leq 1, \forall j \in R \\ & x_{ij} \in \{0,1\}, \forall (i,j) \in A \end{aligned}$$

LP relaxation

$$\begin{aligned} \max \quad & \sum_{ij} w_{ij}x_{ij} \\ \text{s.t.} \quad & \sum_{j} x_{ij} \leq 1, \forall i \in L \\ & \sum_{i} x_{ij} \leq 1, \forall j \in R \\ & x_{ij} \geq 0, \forall (i,j) \in A \end{aligned}$$

- Theorem. The feasible region of the matching LP is the convex hull of indicator vectors of matchings.
- This is the strongest guarantee you could hope for an LP relaxation of a combinatorial problem
- Solving LP is equivalent to solving the combinatorial problem

Primal-Dual Interpretation

Primal LP relaxation

$$
\begin{aligned}
\max \quad & \sum_{ij} w_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{j} x_{ij} \le 1, \forall i \in L \\
& \sum_{i} x_{ij} \le 1, \forall j \in R \\
& x_{ij} \ge 0, \forall (i,j) \in A
\end{aligned}
$$

Dual

$$
\begin{aligned}
\min \quad & \sum_{i} y_i \\
\text{s.t.} \quad & y_i + y_j \ge w_{ij}, \forall (i,j) \in A \\
& y \ge 0
\end{aligned}
$$

- Dual problem is solving minimum vertex cover: find smallest set of nodes S such that at least one end of each edge is in S
- From strong duality theorem, we know $P^*_{LP} = D^*_{LP}$

Primal-Dual Interpretation

Suppose edge weights $w_{ij} = 1$, then binary solutions to the dual are node covers.

Dual

$$\begin{aligned} \min \quad & \sum_i y_i \\ \text{s.t. } & y_i + y_j \geq 1, \forall (i,j) \in A \\ & y \geq 0 \end{aligned}$$

Dual Integer Program

$$\begin{aligned} \min \quad & \sum_i y_i \\ \text{s.t. } & y_i + y_j \geq 1, \forall (i,j) \in A \\ & y \in \{0, 1\} \end{aligned}$$

- Dual problem is solving minimum vertex cover: find smallest set of nodes S such that at least one end of each edge is in S
- From strong duality theorem, we know $P^*_{LP} = D^*_{LP}$
- Consider IP formulation of the dual, then

$$P^*_{IP} \leq P^*_{LP} = D^*_{LP} \leq D^*_{IP}$$

Total Unimodularity

Defintion: A matrix A is Totally Unimodular if every square submatrix has determinant 0, +1 or -1.

Theorem: If $A \in \mathbb{R}^{m \times n}$ is totally unimodular, and b is an integer vector, then $\{x : Ax \le b; x \ge 0\}$ has integer vertices.

- Non-zero entries of vertex x are solution of $A'x' = b'$ for some nonsignular square submatrix A' and corresponding sub-vector b'
- Cramer's rule:

$$x_i = \frac{\det(A'_i \mid b')}{\det A'}$$

Claim: The constraint matrix of the bipartite matching LP is totally unimodular.

The Minimum weight vertex cover

- undirected graph G = (N, A) with node weights $w_i \geq 0$
- A vertex cover is a set of nodes S such that each edge has at least one end in S
- The weight of a vertex cover is sum of all weights of nodes in the cover
- Find the vertex cover with minimum weight

Integer Program

$$\min \quad \sum_i w_i y_i$$

$$\text{s.t. } y_i + y_j \geq 1, \forall (i,j) \in A$$

$$y \in \{0,1\}$$

LP Relaxation

$$\min \quad \sum_i w_i y_i$$

$$\text{s.t. } y_i + y_j \geq 1, \forall (i,j) \in A$$

$$y \geq 0$$

LP Relaxation for the Minimum weight vertex cover

- In the LP relaxation, we do not need $y \leq 1$, since the optimal solution y^* of the LP does not change if $y \leq 1$ is added.
  Proof: suppose that there exists an index i such that the optimal solution of the LP y_i^* is strictly larger than one. Then, let y' be a vector which is same as y^* except for $y'_i = 1 < y_i^*$. This y' satisfies all the constraints, and the objective function is smaller.
- The solution of the relaxed LP may not be integer, i.e., $0 < y_i^* < 1$
- rounding technique:

$$y'_i = \begin{cases} 0, & \text{if } y_i^* < 0.5 \\ 1, & \text{if } y_i^* \geq 0.5 \end{cases}$$

- The rounded solution y' is feasible to the original problem

LP Relaxation for the Minimum weight vertex cover

The weight of the vertex cover we get from rounding is at most twice as large as the minimum weight vertex cover.

- Note that $y_i' = \min(\lfloor 2y_i^* \rfloor, 1)$
- Let P_{IP}^* be the optimal solution for IP, and P_{LP}^* be the optimal solution for the LP relaxation
- Since any feasible solution for IP is also feasible in LP, $P_{LP}^* \leq P_{IP}^*$
- The rounded solution y' satisfy

$$\sum_i y_i' w_i \quad = \quad \sum_i \min(\lfloor 2y_i^* \rfloor, 1) w_i \leq \sum_i 2y_i^* w_i = 2P_{LP}^* \leq 2P_{IP}^*$$

Outline

1. Introduction to Integer Programming

2. Integer Programming Modeling and Formulation

3. Constraint Programming (CP)

Difference with mathematical programming

Problem:

- Solving combinatorial optimization problems (Decision Optimization)

Modeling:

- Declarative modeling paradigm
- Logical constraints & global constraints
- Integer, interval & boolean variables (maybe double variables)

Solving:

- Constructive search & domain reduction (propagation)
- Based on computer science (logic programming, graph theory, ...)

Solution:

- Feasible solution & optimal solution

A simple example: n-Queen Problem

The classic queens problem: placing n queens on an nxn checkerboard so that no two queens can attack each other, i.e., no two queens are on the same row, column, or diagonals.

Integer variable params:

- N: number of variables
- 0: minimum value
- N-1: maximum value
- "X": name prefix

constraints:

- all_diff: $x_i \neq x_j, \forall i \neq j$

DOcplex for n-Queen problem

```
# Create model
mdl = CpoModel()

# Create column index of each queen
x = mdl. integer_var_list (N, 0, N – 1, "X")

# One queen per row
mdl.add(mdl. all_diff (x))

# One queen per diagonal xi – xj != i – j
mdl.add(mdl. all_diff (x[i] + i for i in range(N)))

# One queen per diagonal xi – xj != j – i
mdl.add(mdl. all_diff (x[i] – i for i in range(N)))
```

High-level constraints: Global constraints

Global constraint captures complex relationships among multiple variables in a concise and efficient way.

alldifferent

- All variables in a set take distinct values
- Assignment problems
- alldifferent([x, y, z])
- $x \neq y, x \neq z, y \neq z$

table

- Tuple of variables takes values from predefined set
- table([x, y, z], [(1, 2, 3), (4, 5, 6)])
- (x, y, z) = (1, 2, 3) or (4, 5, 6)

circuit

- Sequence of variables forms a Hamitonian cycle
- Routing problems
- circuit(x)
- x=[0, 1, 3, 2, 0] means $0- > 1- > 3- > 2- > 0$

Arithmetic expressions and constraints

CP Optimizer supports integer variables and is possible to contain float-point expressions in constraints or objective function.

- operator +, -, *, /
- Sum
- Diff
- ScalProd
- Div
- Modulo(%)

- StandardDeviation
- Min
- Max
- Count
- CountDifferent
- Abs
- Element

- ==
- !=
- $<$
- $>$
- $<=$
- $>=$

Diff and Element

Diff & operator-

Automatic linearization by slack variables

```
IloNumExpr e1 = x * y;
IloNumExpr e2 = z / w;
IloNumExpr diff =  IloDiff (e1, e2);
model.add(diff == 0);
```

Element: y=array[x]

```
// Create the model
IloModel model(env);

// Define an array
 IloIntArray  array(env, 4);
array[0]  = 10;
array[1]  = 20;
array[2]  = 30;
array[3]  = 40;

// Define variables
 IloIntVar  x(env, 0, 3, "x");
 IloIntVar  y(env, 0, 100, "y");

// Add the element constraint
model.add(y == IloElement(array, x));
```

Logical and compatibility constraints

- &&
- ||
- Not
- IfThen
- AllowedAssignments
- ForbiddenAssignments

IfThen

```
model.add(IloIfThen(x >= 5, y <= 3));

// Nested structure
model.add(IloIfThen(x >= 5, IloIfThen(y <= 3, z == 0)));

// Combined with global constraints
model.add(IloIfThen(x != y,  IloAllDiff (env, x, y, z)));
```

AllowedAssignments

```
IloIntTupleSet allowed(env);
allowed.add(IloIntArray(env, 2, 1, 2));
allowed.add(IloIntArray(env, 2, 2, 3));

model.add(IloAllowedAssignments(vars, allowed));
```

Special constraints on integer variables

Theoretically, these special constraints can be written from arithmetic constraints and expressions, but they can also be designed and implemented to reduce domains efficiently during a search.

- AllDiff
- AllMinDistance
- Pack
- Inverse
- Lexicographic
- Distribute

Pack

```
IloIntVarArray bin(env, 3, 0, 1); // 3 items, 2 bins
IloIntArray size(env, 3); // Size of each item
size[0] = 2;
size[1] = 3;
size[2] = 4;
IloIntVarArray load(env, 2, 0, 5); // Max capacity is 5

model.add(IloPack(bin, size, load));
```

Interval variables

Interval variables

Static

- StartOf
- EndOf
- LengthOf
- SizeOf

Dynamic

- StartEval
- EndEval
- LengthEval
- SizeEval

StartOf

```
IloIntervalVar  task(env, 10);  //  Task with duration  10
IloIntExpr  start  =  IloStartOf(task);  //  Static  start  time
model.add(start >= 5);  //  Task must start  after  time 5
```

StartEval

```
IloIntervalVar  task(env, 10);
IloIntVar  condition(env, 0,  1);  //  0 or  1

//  Dynamic start time:  if  condition=1,  start  time increases by 5
IloIntExpr  dynamicStart = IloStartEval(task)  + condition  *  5;

//  Constraint  depends on runtime value of ‘condition‘
model.add(dynamicStart <= 20);
```

Special constraints on interval variables

Forbidden constraints

- ForbidStart
- ForbidEnd
- ForbidExtent

Precedence constraints

- End/Start
  + Before/At
  + End/Start
- e.g. EndBeforeStart

Groups of interval variables

- PresenceOf
- Isomorphism
- Span
- Alternative
- Synchronize

Sequence constraints

- First
- Last
- Before
- Prev

Special constraints on interval variables

- * CumulFunctionExpr
  - AlwaysIn/AlwaysEqual
  - AlwaysConstant
  - AlwaysNoState
  - operator$<=, >=$
- Pulse
- Step
- StepAtStart
- StepAtEnd
- HeightAtStart
- HeightAtEnd

Resource usage constraints

```
IloIntervalVar  task1(env, 10, "Task1");
IloIntervalVar  task2(env, 5,  "Task2");

IloCumulFunctionExpr resourceUsage(env);
resourceUsage += IloPulse(task1, 2);
resourceUsage += IloPulse(task2, 1);
resourceUsage += IloStep(5, 3);
resourceUsage += IloStepAtStart(task1, 1);
resourceUsage += IloStepAtEnd(task2, –1);

model.add(resourceUsage <= 5);
```

Algorithms for Mixed Integer Linear Programming

https://bicmr.pku.edu.cn/~wenzw/bigdata2022.html

Acknowledgement: this slides is based on Prof. Ted Ralphs's lecture notes

Outline

1. Branch and Bound
2. Bounding
3. Branching
4. Cutting Plane

Computational Integer Optimization

- Computationally, the most important aspects of solving integer optimization problems are
  - A method for obtaining good bounds on the value of the optimal solution (usually by solving a relaxation or dual; and
  - A method for generating valid disjunctions violated by a given (infeasible) solution.
- In this lecture, we will motivate this fact by introducing the branch and bound algorithm.
- We will then look at various methods of obtaining bounds.
- Later, we will examine branch and bound in more detail.

Integer Optimization and Disjunction

- The difficulty arises from the requirement that certain variables take on integer values.
- Such requirements can be described in terms of logical disjunctions, constraints of the form

$$x \in \bigcup_{1 \le i \le k} X_i, \quad X_i \subseteq \mathbb{R}^n.$$

- The integer variables in a given formulation may represent logical conditions that were originally expressed in terms of disjunction.
- In fact, the MILP Representability Theorem tells us that any MILP can be re-formulated as an optimization problem whose feasible region is

$$\mathcal{F} = \bigcup_{1 \le i \le k} \mathcal{P}_i + \text{intcone}\{r^1, \cdots, r^t\}$$

  is the disjunctive set $\mathcal{F}$ defined above, for some appropriately chosen polytopes $\mathcal{P}_1, \cdots, \mathcal{P}_k$ and vectors $r^1, \cdots, r^t \in \mathbb{Z}^n$.

Two Conceptual Reformulations

- We have two conceptual reformulations of a given integer optimization problem.
- The first is in terms of disjunction:

$$\max \left\{ c^\top x \mid x \in \bigcup_{1 \le i \le k} \mathcal{P}_i + \text{intcone}\{r^1, \cdots, r^t\} \right\}$$

- The second is in terms of valid inequalities

$$\max\{c^\top x \mid x \in \text{conv}(\mathcal{S})\}$$

  where $\mathcal{S}$ is the feasible region.
- In principle, if we had a method for generating either of these reformulations, this would lead to a practical method of solution.
- Unfortunately, these reformulations are necessarily of exponential size in general, so there can be no way of generating them efficiently.

Valid Disjunctions

- In practice, we dynamically generate parts of the reformulations (CP) and (DIS) in order to obtain a proof of optimality for a particular instance.

- The concept of valid disjunction, arises from a desire to approximate the feasible region of (DIS).
  - **Definition 1.** Let $\{X_i\}_{i=1}^k$ be a collection of subset of $\mathbb{R}^n$. Then if $\mathcal{S} \subseteq \cup_{1 \le i \le k} X_i$, the disjunction associated with $\{X_i\}_{i=1}^k$ is said to be valid for an MILP with feasible set $\mathcal{S}$.
  - **Definition 2.** Let $\{X_i\}_{i=1}^k$ is a disjunction valid for $\mathcal{S}$, and X_i is polyhedral for all i, then we say the disjunction is linear.
  - **Definition 3.** Let $\{X_i\}_{i=1}^k$ is a disjunction valid for $\mathcal{S}$, and $X_i \cap X_j = \emptyset$ for all i, j then we say the disjunction is partitive.
  - **Definition 4.** Let $\{X_i\}_{i=1}^k$ is a disjunction valid for $\mathcal{S}$ that is both linear and partitive, we call it admissible.

Valid Inequalities

- Likewise, we can think of the concept of a valid inequality as arising from our desire to approximate $\text{conv}(\mathcal{S})$ (the feasible region of (CP)).

- The inequality denoted by (π, π_0) is called a valid inequality for $\mathcal{S}$ if $\pi^\top x \leq \pi_0, \forall x \in \mathcal{S}$.

- Note (π, π_0) is a valid inequality if and only if $\mathcal{S} \subseteq \{x \in \mathbb{R}^n \mid \pi^\top x \leq \pi_0\}$.

Optimality Conditions

- Let us now consider an MILP (A, b, c, p) with feasible set $\mathcal{S} = \mathcal{P} \cap (\mathbb{Z}^p_+ \times \mathbb{R}^{n-p}_+)$, where $\mathcal{P}$ is the given formulation.
- Further, let $\{X_i\}_{i=1}^k$ be a linear disjunction valid for this MILP so that $X_i \cap \mathcal{P} \subseteq \mathbb{R}^n$ is a polyhedral.
- Then $\max_{X_i \cap \mathcal{S}} c^\top x$ is an MILP for all $i \in 1, \dots, k$.
- For each i, let $\mathcal{P}_i$ be a polyhedron such that $X_i \cap \mathcal{S} \subseteq \mathcal{P}_i \subseteq \mathcal{P} \cap X_i$.
- In other words, $\mathcal{P}_i$ is a valid formulation for subproblem i, possibly strengthened by additional valid inequalities.
- Note that $\{\mathcal{P}_i\}$ is itself a valid linear disjunction.

Optimality Conditions

- From the disjunction on the previous slide, we obtain a relaxation of a general MILP.
- This relaxation yields a practical set of optimality conditions.
- In particular,

$$\max_{i \in 1,\cdots,k} \max_{x \in \mathcal{P}_i \cap \mathbb{R}^n_+} c^\top x \geq z_{\text{IP}}.$$

- If we have $x^* \in \mathcal{S}$ such that

$$\max_{i \in 1,\cdots,k} \max_{x \in \mathcal{P}_i \cap \mathbb{R}^n_+} c^\top x = c^\top x^*$$

then x^* must be optimal.

Branch and Bound

- Branch and bound is the most commonly-used algorithm for solving MILPs. It is a recursive, divide-and-conquer approach.
- Suppose $\mathcal{S}$ is the feasible set for an MILP and we wish to compute $\max_{x \in \mathcal{S}} c^\top x$.
- Consider a partition of $\mathcal{S}$ into subsets $\mathcal{S}_1, \cdots, \mathcal{S}_k$. Then

$$\max_{x \in \mathcal{S}} c^\top x = \max_{1 \leq i \leq k} \{\max_{x \in \mathcal{S}_i} c^\top x\}.$$

- Idea: If we can't solve the original problem directly, we might be able to solve the smaller subproblems recursively.
- Dividing the original problem into subproblems is called branching.
- Taken to the extreme, this scheme is equivalent to complete enumeration.

Branching in Branch and Bound

- Branching is achieved by selecting an admissible disjunction $\{X_i\}_{i=1}^k$ and using it to partition $\mathcal{S}$, e.g., $\mathcal{S}_i = \mathcal{S} \cap X_i$.
- We only consider linear disjunctions so that the subproblem remain MILPs after branching.
- The way this disjunction is selected is called the branching method and is a topic we will examine in some depth.
- Generally speaking, we want $x^* \notin \cup_i X_i$, where x^* is the (infeasible) solution produced by solving the bounding problem associated with a given subproblem.
- A typical disjunction is

$$
\begin{aligned}
X_1 &= \{x_j \geq \lceil x_j^* \rceil\} \\
X_2 &= \{x_j \leq \lfloor x_j^* \rfloor\}
\end{aligned}
$$

where $x^* \in \operatorname{argmax}_{x \in \mathcal{P}} c^\top x$.

Bounding in Branch and Bound

- The bounding problem is a problem solved to obtain a bound on the optimal solution value of a subproblem $\max_{\mathcal{S}_i} c^\top x$.
- Typically, the bounding problem is either a relaxation or a dual of the subproblem.
- Solving the bounding problem serves two purposes.
  - In some cases, the solution x^* to the relaxation may actually be a feasible solution, in which case $c^\top x^*$ is a global lower bound $l(\mathcal{S})$.
  - Bounding enables us to inexpensively obtain a bound $b(\mathcal{S}_i)$ on the optimal solution value of subproblem i.
- If $b(\mathcal{S}_i) \leq l(\mathcal{S})$, then $\mathcal{S}_i$ can't contain a solution strictly better than the best one found so far.
- Thus, we may discard or prune subproblem i.
- For the rest of the lecture, assume all variables have finite upper and lower bounds.

LP-based Branch and Bound: Initial Subproblem

- In LP-based branch and bound, we first solve the LP relaxation of the original problem. The result is one of the following:
  - The LP is infeasible $\Rightarrow$ MILP is infeasible.
  - We obtain a feasible solution for the MILP $\Rightarrow$ optimal solution.
  - We obtain an optimal solution to the LP that is not feasible for the MILP $\Rightarrow$ upper bound.
- In the first two cases, we are finished.
- In the third case, we must branch and recursively solve the resulting subproblems.

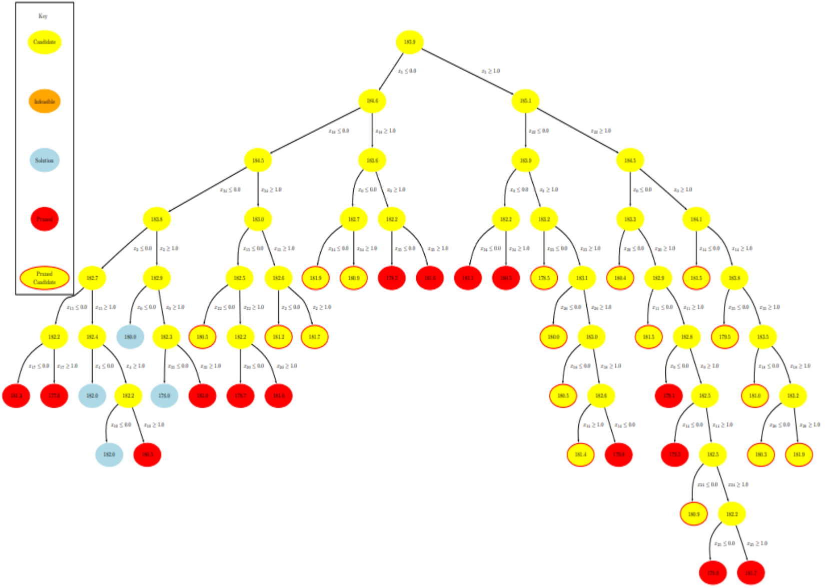
Branching in LP-based Branch and Bound

- In LP-based branch and bound, the most commonly used disjunctions are the variable disjunctions, imposed as follows:
  - Select a variable i whose value $\hat{x}_i$ is fractional in the LP solution.
  - Create two subproblems.
  - In one subproblem, impose the constraint $x_i \leq \lfloor \hat{x}_i \rfloor$.
  - In the other subproblem, impose the constraint $x_i \geq \lceil \hat{x}_i \rceil$.
- What does it mean in a 0-1 problem?

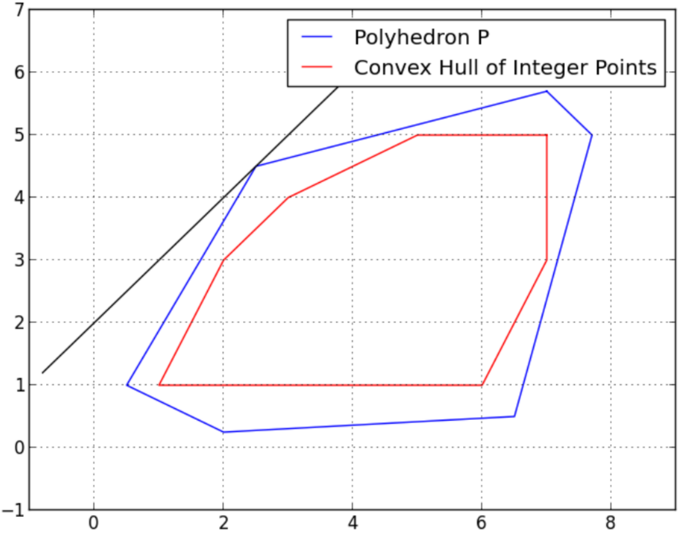
LP-based Branch and Bound Algorithm

- To start, derive a lower bound L using a heuristic method.
- Put the original problem on the candidate list.
- Select a problem $\mathcal{S}$ from the candidate list and solve the LP relaxation to obtain the bound $b(\mathcal{S})$.
  - If the LP is infeasible $\Rightarrow$ node can be pruned.
  - Otherwise, if $b(\mathcal{S}) \leq L \Rightarrow$ node can be pruned.
  - Otherwise, if $b(\mathcal{S}) > L$ and the solution is feasible for the MILP $\Rightarrow$ set $L \leftarrow b(\mathcal{S})$.
  - Otherwise, branch and add the new subproblem to the candidate list.
- If the candidate list in nonempty, go to Step 2. Otherwise, the algorithm is completed.

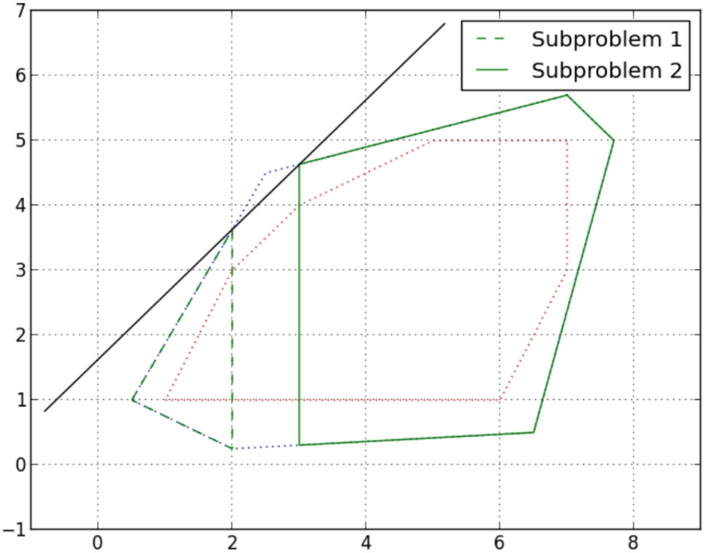
Branch and Bound Tree



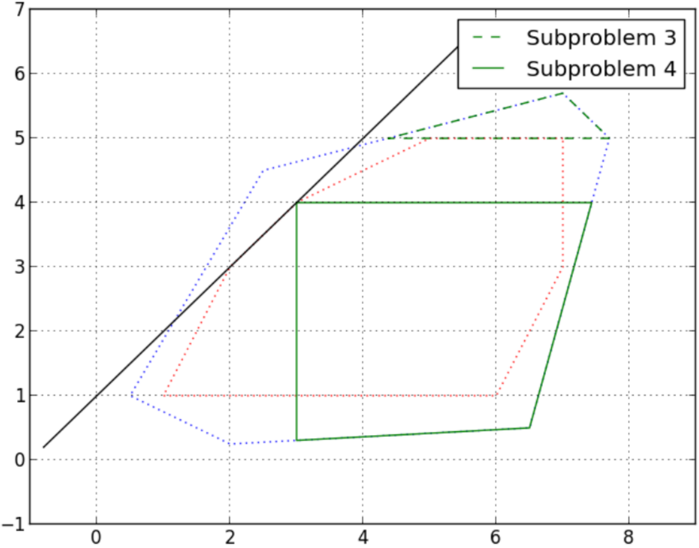
The Geometry of Branching



The Geometry of Branching (cont'd)



The Geometry of Branching



Continuing the Algorithm After Branching

- After branching, we solve each of the subproblems recursively.
- As mentioned earlier, if the optimal solution value to the LP relaxation is smaller than the current lower bound, we need not consider the subproblem further. This is the key to the efficiency of the algorithm.
- Terminology
    - If we picture the subproblems graphically, they form a search tree.
    - Each subproblem is linked to its parent and eventually to its children.
    - Eliminating a problem from further consideration is called pruning.
    - The act of bounding and then branching is called processing.
    - A subproblem that has not yet been considered is called a candidate for processing.
    - The set of candidates for processing is called the candidate list.

Ensuring Finite Convergence

- For LP-based branch and bound, ensuring convergence requires a convergent branching method.
- Roughly speaking, a convergent branching method is one which will
  - produce a violated admissible disjunction whenever the solution to the bounding problem is infeasible; and
  - if applied recursively, guarantee that at some finite depth, any resulting bounding problem will either
    - produce a feasible solution (to the original MILP); or
    - be proven infeasible; or
    - be pruned by bound.
- Typically, we achieve this by ensuring that at some finite depth, the feasible region of the bounding problem contains at most one feasible solution.

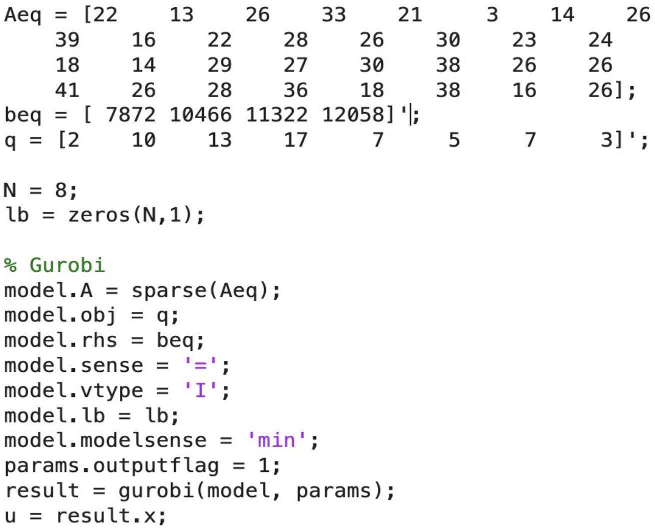
Algorithmic Choices in Branch and Bound

- Although the basic algorithm is straightforward, the efficiency of it in practice depends strongly on making good algorithmic choices.

- These algorithmic choices are made largely by heuristics that guide the algorithm.

- Basic decisions to be made include
  - The bounding method(s).
  - The method of selecting the next candidate to process.
    - "Best-first" always chooses the candidate with the highest upper bound.
    - This rule minimizes the size of the tree (why?).
    - There may be practical reasons to deviate from this rule.
  - The method of branching.
    - Branching wisely is extremely important.
    - A "poor" branching can slow the algorithm significantly.

An example solved by Gurobi

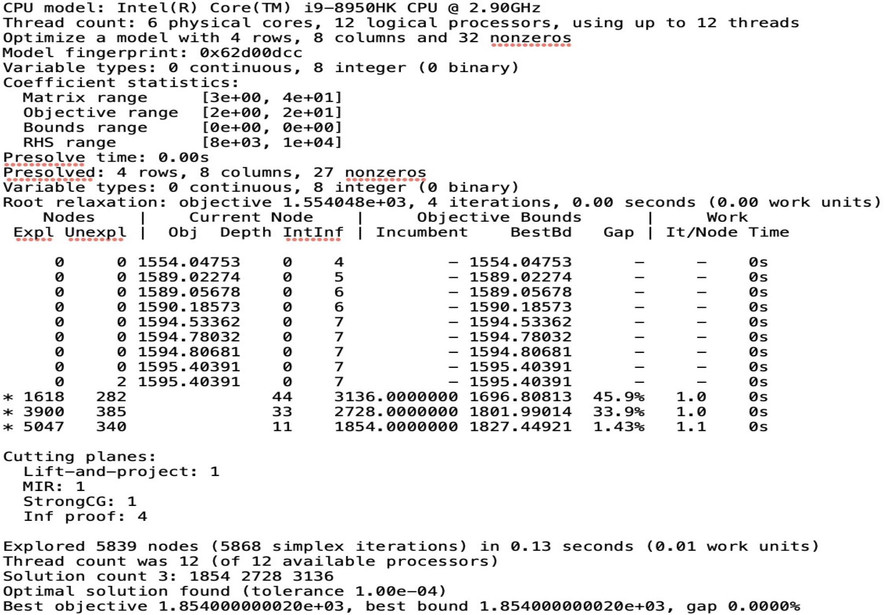


```
Aeq = [22     13     26     33     21      3     14     26
    39     16     22     28     26     30     23     24
    18     14     29     27     30     38     26     26
    41     26     28     36     18     38     16     26];
beq = [ 7872 10466 11322 12058]';
q = [2     10     13     17      7      5      7      3]';

N = 8;
lb = zeros(N,1);

% Gurobi
model.A = sparse(Aeq);
model.obj = q;
model.rhs = beq;
model.sense = '=';
model.vtype = 'I';
model.lb = lb;
model.modelsense = 'min';
params.outputflag = 1;
result = gurobi(model, params);
u = result.x;
```

An example solved by Gurobi



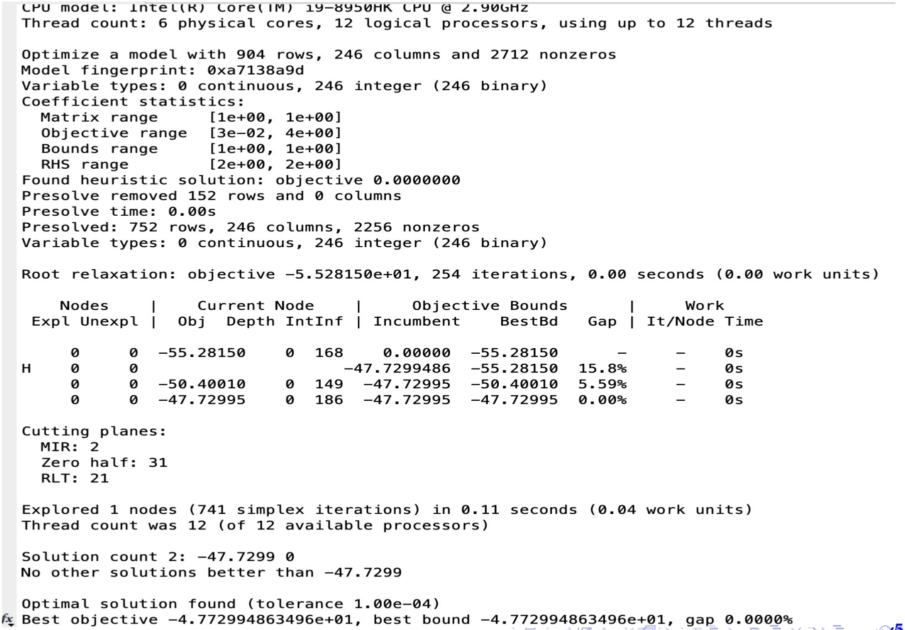
```
CPU model: Intel(R) Core(TM) i9-8950HK CPU @ 2.90GHz
Thread count: 6 physical cores, 12 logical processors, using up to 12 threads
Optimize a model with 4 rows, 8 columns and 32 nonzeros
Model fingerprint: 0x62d00dcc
Variable types: 0 continuous, 8 integer (0 binary)
Coefficient statistics:
  Matrix range     [3e+00, 4e+01]
  Objective range  [2e+00, 2e+01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [8e+03, 1e+04]
Presolve time: 0.00s
Presolved: 4 rows, 8 columns, 27 nonzeros
Variable types: 0 continuous, 8 integer (0 binary)
Root relaxation: objective 1.554048e+03, 4 iterations, 0.00 seconds (0.00 work units)
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 1554.04753    0    4          - 1554.04753      -     -    0s
     0     0 1589.02274    0    5          - 1589.02274      -     -    0s
     0     0 1589.05678    0    6          - 1589.05678      -     -    0s
     0     0 1590.18573    0    6          - 1590.18573      -     -    0s
     0     0 1594.53362    0    7          - 1594.53362      -     -    0s
     0     0 1594.78032    0    7          - 1594.78032      -     -    0s
     0     0 1594.80681    0    7          - 1594.80681      -     -    0s
     0     0 1595.40391    0    7          - 1595.40391      -     -    0s
     0     2 1595.40391    0    7          - 1595.40391      -     -    0s
* 1618   282              44    3136.0000000 1696.80813  45.9%   1.0    0s
* 3900   385              33    2728.0000000 1801.99014  33.9%   1.0    0s
* 5047   340              11    1854.0000000 1827.44921  1.43%   1.1    0s

Cutting planes:
  Lift-and-project: 1
  MIR: 1
  StrongCG: 1
  Inf proof: 4

Explored 5839 nodes (5868 simplex iterations) in 0.13 seconds (0.01 work units)
Thread count was 12 (of 12 available processors)
Solution count 3: 1854 2728 3136
Optimal solution found (tolerance 1.00e-04)
Best objective 1.854000000020e+03, best bound 1.854000000020e+03, gap 0.0000%
```

Another example solved by Gurobi



```
CPU model: Intel(R) Core(TM) i9-8950HK CPU @ 2.90GHz
Thread count: 6 physical cores, 12 logical processors, using up to 12 threads

Optimize a model with 904 rows, 246 columns and 2712 nonzeros
Model fingerprint: 0xa7138a9d
Variable types: 0 continuous, 246 integer (246 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [3e-02, 4e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [2e+00, 2e+00]
Found heuristic solution: objective 0.0000000
Presolve removed 152 rows and 0 columns
Presolve time: 0.00s
Presolved: 752 rows, 246 columns, 2256 nonzeros
Variable types: 0 continuous, 246 integer (246 binary)

Root relaxation: objective -5.528150e+01, 254 iterations, 0.00 seconds (0.00 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0  -55.28150    0  168    0.00000  -55.28150      -      -    0s
H    0     0                       -47.7299486  -55.28150  15.8%     -    0s
     0     0  -50.40010    0  149  -47.72995  -50.40010  5.59%     -    0s
     0     0  -47.72995    0  186  -47.72995  -47.72995  0.00%     -    0s

Cutting planes:
  MIR: 2
  Zero half: 31
  RLT: 21

Explored 1 nodes (741 simplex iterations) in 0.11 seconds (0.04 work units)
Thread count was 12 (of 12 available processors)

Solution count 2: -47.7299 0
No other solutions better than -47.7299

Optimal solution found (tolerance 1.00e-04)
Best objective -4.772994863496e+01, best bound -4.772994863496e+01, gap 0.0000%
```

Outline

1. Branch and Bound
2. Bounding
3. Branching
4. Cutting Plane

The Efficiency of Branch and Bound

- The overall solution time is the product of the number of nodes enumerated and the time to process each node.
- Typically, by spending more time in processing, we can achieve a reduction in tree size by computing stronger bounds.
- This highlights another of the many tradeoffs we must navigate.
- Our goal in bounding is to achieve a balance between the strength of the bound and the efficiency.
- How do we compute bounds?
  - Relaxation: Relax some of the constraints and solve the resulting mathematical optimization problem.
  - Duality: Formulate a "dual" problem and find a feasible to it.
- In practice, we will use both of these two approaches.

Relaxation

- As usual, we consider the MILP

  $$z_{\mathrm{IP}} = \max\{c^\top x \mid x \in \mathcal{S}\}$$

  where

  $$\mathcal{P} = \{x \in \mathbb{R}^n \mid Ax \leq b\}$$
  $$\mathcal{S} = \mathcal{P} \cap (\mathbb{Z}_+^p \times \mathbb{R}_+^{n-p}).$$

- **Definition 1.** A relaxation of IP is a maximization problem defined as

  $$z_R = \max\{z_R(x) | x \in \mathcal{S}_R\}$$

  with the following two properties:

  $$\mathcal{S} \subseteq \mathcal{S}_R$$
  $$c^\top x \leq z_R(x), \quad \forall x \in \mathcal{S}$$

Importance of Relaxations

- The main purpose of a relaxation is to obtain an upper bound on z_{IP} .
- Solving a relaxation is one simple method of bounding in branch and bound.
- The idea is to choose a relaxation that is much easier to solve than the original problem, but still yields a bound that is "strong enough."
- Note that the relaxation must be solved to optimality to yield a valid bound.
- We consider three types of "formulation-based" relaxations.
  - LP relaxation
  - Combinatorial relaxation
  - Lagrangian relaxation
- Relaxations are also used in some other bounding schemes we'll look at.

Obtaining and Using Relaxations

- Properties of relaxations
    - If a relaxation of (MILP) is infeasible, then so is (MILP).
    - If $z_R(x) = c^\top x$, then for $x^* \in \operatorname{argmax}_{x \in \mathcal{S}_R} z_R(x)$, if $x^* \in \mathcal{S}$, then x^* is optimal for (MILP).
- The easiest way to obtain relaxations of IP is to drop some of the constraints defining the feasible set $\mathcal{S}$.
- It is "obvious" how to obtain an LP relaxation, but combinatorial relaxations are not as obvious.

Lagrangian Relaxation

- The idea is again based on relaxing a set of constraints from the original formulation.
- We try to push the solution towards feasibility by penalizing violation of the dropped constraints.
- Suppose our IP is defined by

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & A^1 x \leq b^1 \\ & A^2 x \leq b^2 \\ & x \in \mathbb{Z}^n_+ \end{aligned}$$

where optimizing over $Q = \{x \in \mathbb{Z}^n_+ \mid A^2 x \leq b^2\}$ is "easy."

- Lagrangian Relaxation:

$$LR(\lambda) : Z_R(\lambda) = \max_{x \in Q} \{(c - (A^1)^\top \lambda)^\top x + \lambda^\top b^1)\}.$$

Properties of the Lagrangian Relaxation

- For any $\lambda \geq 0$, $LR(\lambda)$ is a relaxation of IP (why?).
- Solving $LR(\lambda)$ yields an upper bound on the value of the optimal solution.
- Because of our assumptions, $LR(\lambda)$ can be solved easily.
- Recalling LP duality, one can think of λ as a vector of "dual variables."
- If the solution to the relaxation is integral, it is optimal if the primal and dual solutions are complementary, as in LP.

Outline

1. Branch and Bound
2. Bounding
3. **Branching**
4. Cutting Plane

Disjunctions and Branching

- Recall that branching is generally achieved by selecting an admissible disjunction $\{X_i\}_{i=1}^k$ and using it to partition $\mathcal{S}$, e.g., $\mathcal{S}_i = \mathcal{S} \cap X_i$.
- The way this disjunction is selected is called the branching method.
- Generally speaking, we want $x^* \notin \cup_{1 \leq i \leq k} X_i$, where x^* is the (infeasible) solution produced by solving the bounding problem associated with a given subproblem.

Split Disjunctions

- The most easily handled disjunctions are those based on dividing the feasible region using a given hyperplane.
- In such cases, each term of the disjunction can be imposed by addition of a single inequality.
- A hyperplane defined by a vector $\pi \in \mathbb{R}^n$ is said to be integer if $\pi_i \in \mathbb{Z}$ for $0 \leq i \leq p$ and $\pi_i = 0$ for $p + 1 \leq i \leq n$.
- Note that if π is integer, then we have $\pi^\top x \in \mathbb{Z}$ whenever $x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}$.
- Then the disjunction defined by

  $$X_1 = \{x \in \mathbb{R}^n \mid \pi^\top x \leq \pi_0\}, X_2 = \{x \in \mathbb{R}^n \mid \pi^\top x \geq \pi_0 + 1\},$$

  is valid when $\pi_0 \in \mathbb{Z}$.
- This is known as a split disjunction.

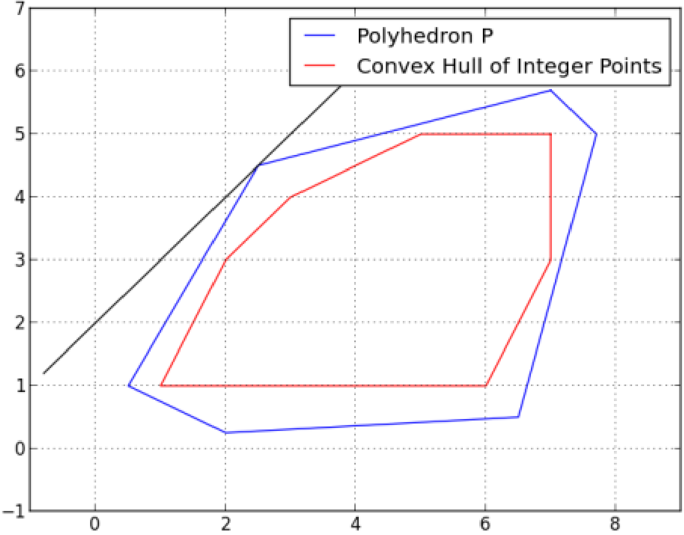
Variable Disjunctions

- The simplest split disjunction is to take $\pi = e_i$ for $0 \leq i \leq p$, where e_i is the ith unit vector.
- If we branch using such a disjunction, we simply say we are branching on x_i.
- For such a branching disjunction to be admissible, we should have $\pi_0 < x_i^* < \pi_0 + 1$.
- In the special case of a 0-1 IP, this dichotomy reduces to

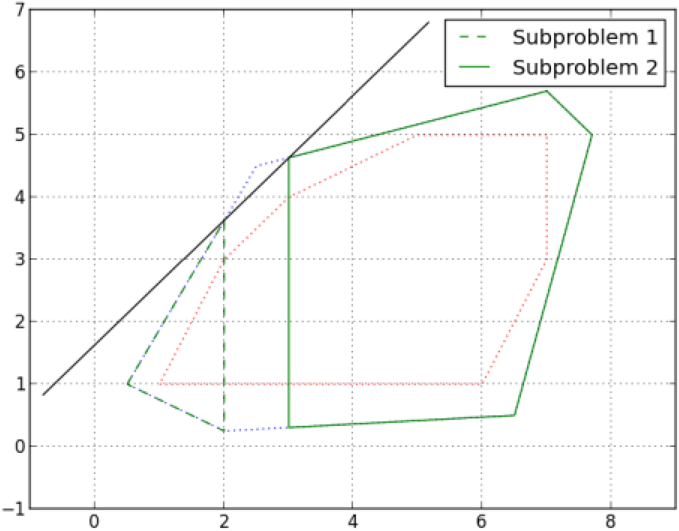
$$x_j = 0 \quad \text{OR} \quad x_j = 1$$

- In general IP, branching on a variable involves imposing new bound constraints in each one of the subproblems.
- This is is the most common method of branching and is easily handled implicitly in most cases.
- What are the benefits of such a scheme?

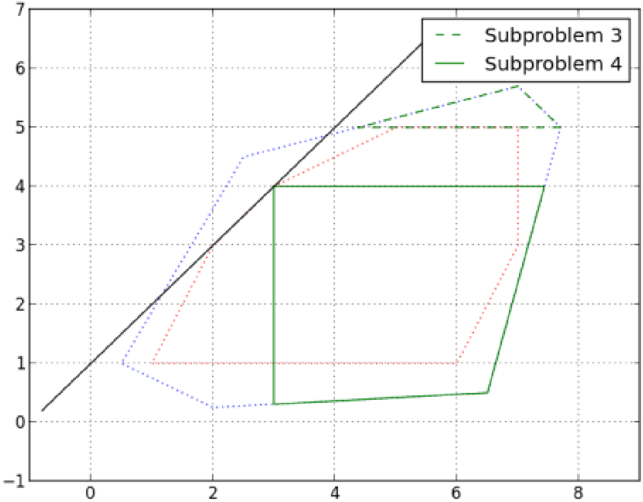
The Geometry of Branching



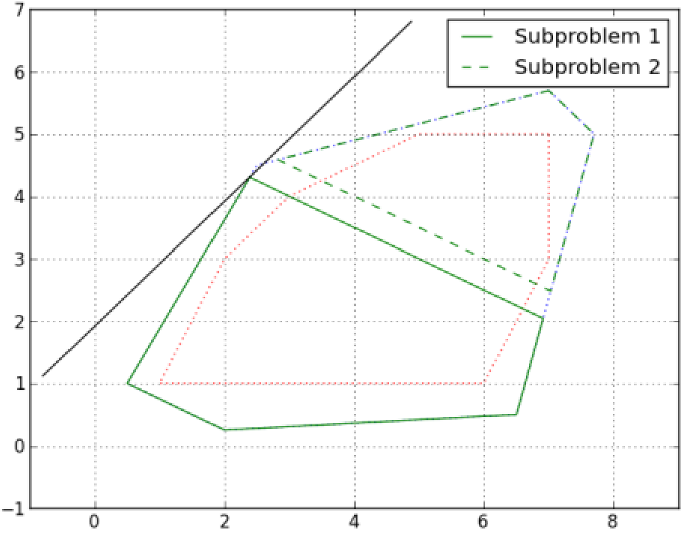
The Geometry of Branching (Variable Disjunction)



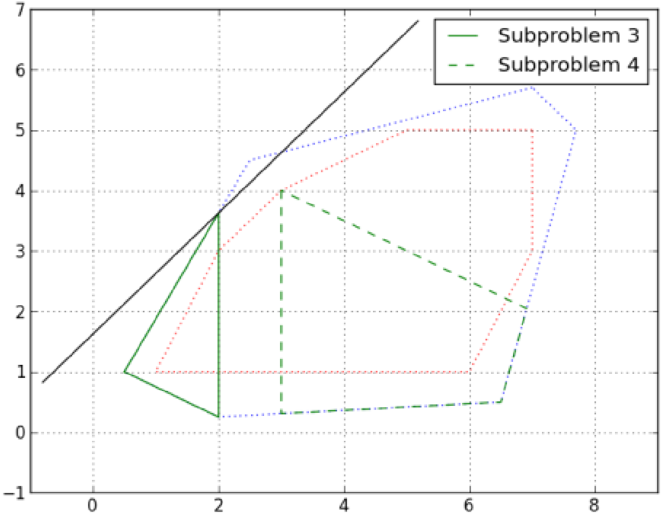
The Geometry of Branching (Variable Disjunction)



The Geometry of Branching (General Split Disjunction)



The Geometry of Branching (General Split Disjunction)



Outline

1. Branch and Bound

2. Bounding

3. Branching

4. Cutting Plane

Describing $\mathrm{conv}(\mathcal{S})$

- We have seen that, in theory, $\mathrm{conv}(\mathcal{S})$ is a polyhedron and has a finite description.

- If we "simply" construct that description, we could turn our MILP into an LP.

- So why aren't IPs easy to solve?
  - The size of the description is generally HUGE!
  - The number of facets of the TSP polytope for an instance with 120 nodes is more than 10^{100} times the number of atoms in the universe.
  - It is physically impossible to write down a description of this polytope.
  - Not only that, but it is very difficult in general to generate these facets (this problem is not polynomially solvable in general).

Cutting Planes

- Recall that the inequality denoted by (π, π_0) is valid for a polyhedron $\mathcal{P}$ if $\pi^\top x \leq \pi_0, \forall x \in \mathcal{P}$.
- The term cutting plane usually refers to an inequality valid for $\text{conv}(\mathcal{S})$, but which is violated by the solution obtained by solving the (current) LP relaxation.
- Cutting plane methods attempt to improve the bound produced by the LP relaxation by iteratively adding cutting planes to the initial LP relaxation.
- Adding such inequalities to the LP relaxation may improve the bound (this is not a guarantee).
- Note that when π and π_0 are integer, then π, π_0 is a split disjunction for which $X_2 = \emptyset$.

The Separation Problem

- The problem of generating a cutting plane can be stated as: **Separation Problem**: Given a polyhedron $\mathcal{Q} \in \mathbb{R}^n$ and $x^* \in \mathbb{R}^n$ determine whether $x^* \in \mathcal{Q}$ and if not, determine (π, π_0), a valid inequality for $\mathcal{Q}$ such that $\pi^\top x^* > \pi_0$.
- This problem is stated here independent of any solution algorithm.
- However, it is typically used as a subroutine inside an iterative method for improving the LP relaxation.
- In such a case, x^* is the solution to the LP relaxation (of the current formulation, including previously generated cuts).
- We will see later that the difficulty of solving this problem exactly is strongly tied to the difficulty of the optimization problem itself.

Generic Cutting Plane Method

Let $\mathcal{P} = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ be the initial formulation for

$$\max\{c^\top x \mid x \in \mathcal{S}\}, \quad \mathcal{S} = \mathcal{P} \cap \mathbb{Z}^p_+ \times \mathbb{R}^{n-p}_+.$$

Algorithm 1: Cutting plane method

```
$\mathcal{P}_0 \leftarrow \mathcal{P}$, $k \leftarrow 0$.
while TRUE do
    Solve the LP relaxation $\max\{c^\top x | x \in \mathcal{P}_k\}$ to obtain solution $x^k$.
    Solve the problem of separating $x^k$ from $\mathrm{conv}(\mathcal{S})$.
    if $x^k \in \mathrm{conv}(\mathcal{S})$ then STOP;
    else Get an inequality $(\pi^k, \pi^k_0)$ valid for $\mathrm{conv}(\mathcal{S})$ but
      $(\pi^k)^\top x^k > \pi^k_0$ ;
    $\mathcal{P}_{k+1} \leftarrow \mathcal{P}_k \cap \{x \in \mathbb{R}^n \mid (\pi^k)^\top x \leq \pi^k_0\}$.
    $k \leftarrow k + 1$.
```

Generating Valid Inequalities for $\mathrm{conv}(\mathcal{S})$

Consider the MILP

$$z_{IP} = \max c^\top x, \text{ s.t. } x \in S,$$

where $\mathcal{P} = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ and $S = \mathcal{P} \cap (\mathbb{Z}_+^p \times \mathbb{R}_+^{n-p})$

- All inequalities valid for $\mathcal{P}$ are also valid for $\mathrm{conv}(\mathcal{S})$, but they are not cutting planes.
- We need the following simple principle: if $a \leq b$ and a is an integer, then $a \leq \lfloor b \rfloor$.
- This simple fact is all we need to generate all valid inequalities for $\mathrm{conv}(\mathcal{S})$!
- Example: suppose that $2x_1 + x_2 \leq 3/2$ is valid for $\mathcal{P}$, then $2x_1 + x_2 \leq 1$ is also valid for $\mathrm{conv}(\mathcal{S})$.

Chvátal Inequalities

- split $A = [A_I, A_C]$ according to integer and continuous variables
- Suppose we can find a $u \in \mathbb{R}^m_+$ such that $\pi = A^\top u$ is integer ($A_I^\top u \in \mathbb{Z}^p$ and $A_C^\top u = 0$) and $\pi_0 = u^\top b \notin \mathbb{Z}$.
- In this case, we have $\pi^\top x \in \mathbb{Z}$ for all $x \in \mathcal{S}$, and so $\pi^\top x \leq \lfloor \pi_0 \rfloor$ for all $x \in \mathcal{S}$.
- In other words, $(\pi, \lfloor \pi_0 \rfloor)$ is both a valid inequality and a split disjunction

$$\{x \in \mathcal{P} \mid \pi^\top x \geq \lfloor \pi_0 \rfloor + 1\} = \emptyset$$

- Such an inequality is called a Chvátal inequality
- Note that we have not used the non-negativity constraints in deriving this inequality

Chvátal-Gomory Inequalities

- Assume that $\mathcal{P} \subset \mathbb{R}^n_+$ and let $u \in \mathbb{R}^n_+$ be such that $A_C^\top u \geq 0$
- Since the variables are nonnegative, we have $u^\top A_C x_C \geq 0$ and

$$\sum_{i=1}^{p} (u^\top A_i) x_i \leq u^\top b, \quad \forall x \in \mathcal{P}$$

- Again, because the variables are nonnegative, we have

$$\sum_{i=1}^{p} \lfloor u^\top A_i \rfloor x_i \leq u^\top b, \quad \forall x \in \mathcal{P}$$

- Finally, we have:

$$\sum_{i=1}^{p} \lfloor u^\top A_i \rfloor x_i \leq \lfloor u^\top b \rfloor, \quad \forall x \in \mathcal{S}$$

- This is the Chvátal-Gomory inequality

Chvátal-Gomory Inequalities: another derivation

- We explicitly add the non-negativity constraints to the formulation along the other constraints with associated multipliers $v \in \mathbb{R}^n_+$
- We cannot round the coefficients to make them integral, so we require π integral

$$\begin{aligned} \pi_i &= u^\top A_i - v_i \in \mathbb{Z} \quad \text{for } 1 \leq i \leq p \\ \pi_i &= u^\top A_i - v_i = 0 \quad \text{for } p+1 \leq i \leq n \end{aligned}$$

- v_i will be non-negative as as long as we have

$$\begin{aligned} v_i &\geq u^\top A_i - \lfloor u^\top A_i \rfloor, && \text{for } 0 \leq i \leq p, \\ v_i &= u^\top A_i \geq 0, && \text{for } \; p+1 \leq i \leq n. \end{aligned}$$

- Taking $v_i = u^\top A_i - \lfloor uA_i \rfloor$ for $1 \leq i \leq p$, we obtain

$$\sum_{i=1}^{p} \pi_i x_i = \sum_{i=1}^{p} \lfloor uA_i \rfloor x_i \leq \lfloor ub \rfloor = \pi_0$$

is a C-G inequality for all $u \in \mathbb{R}^m_+$ such that $A_C^\top u \geq 0$

The Chvátal-Gomory Procedure

1. Choose a weight vector $u \in \mathbb{R}^m_+$ such that $A_C^\top u \geq 0$.
2. Obtain the valid inequality $\sum_{i=1}^{p} (u^\top A_i) x_i \leq u^\top b$.
3. Round the coefficients down to obtain $\sum_{i=1}^{p} \lfloor u^\top A_i \rfloor x_i \leq u^\top b$.
4. Finally, round the right hand side down to obtain the valid inequality

$$\sum_{i=1}^{p} \lfloor u^\top A_i \rfloor x_i \leq \lfloor u^\top b \rfloor$$

- This procedure is called the Chvátal-Gomory rounding procedure, or simply the C-G procedure.
- Surprisingly, for pure ILPs $(p = n)$, any inequality valid for $\operatorname{conv}(\mathcal{S})$ can be produced by a finite number of iterations of this procedure!
- This is not true for the general mixed case.

Gomory Inequalities

- Consider the set of solutions to a pure ILP with one equation:

$$T = \left\{ x \in \mathbb{Z}^n_+ \mid \sum_{j=1}^{n} a_j x_j = a_0 \right\}$$

- For each j, let $f_j = a_j - \lfloor a_j \rfloor$. Then equivalently

$$T = \left\{ x \in \mathbb{Z}^n_+ \mid \sum_{j=1}^{n} f_j x_j = f_0 + \lfloor a_0 \rfloor - \sum_{j=1}^{n} \lfloor a_j \rfloor x_j \right\}$$

- Since $\sum_{j=1}^{n} f_j x_j \geq 0$ and $f_0 < 1$, then $\lfloor a_0 \rfloor - \sum_{j=1}^{n} \lfloor a_j \rfloor x_j \geq 0$ and so

$$\sum_{j=1}^{n} f_j x_j \geq f_0$$

is a valid inequality for $\mathcal{S}$ called a Gomory inequality.

Large-scale Integer Linear Programming

https://bicmr.pku.edu.cn/~wenzw/bigdata2024.html

Outline

1. Lagrangian Relaxation
2. Dantzig-Wolfe decomposition
3. Bender's Decomposition

Lagrangian Relaxation

- Consider the integer programming problem

$$\begin{aligned} \max \quad & c^\top x, \\ \text{s.t.} \quad & Ax \le b,\ Dx \le d, \\ & x \in \mathbb{Z}^n, \end{aligned} \tag{1}$$

  and assume that A, D, b, c, d have integer entries.

- Let Z_{IP} the optimal cost and let

$$X = \{x \in \mathbb{Z}^n \mid Dx \le d\}. \tag{2}$$

  We assume that optimizing over the set X can be done efficiently.

- Let $\lambda \ge 0$ be a vector of dual variables. We introduce the problem

$$\begin{aligned} \max \quad & c^\top x + \lambda^\top (b - Ax), \\ \text{s.t.} \quad & x \in X, \end{aligned} \tag{3}$$

  and denote its optimal cost by $Z(\lambda)$.

Lagrangian Relaxation

Lemma

If the problem (1) *has an optimal solution and if* $\lambda \geq 0$, *then* $Z(\lambda) \geq Z_{IP}$

- **Proof:** Let x^* denote an optimal solution to (1).
  Then, $b - Ax^* \geq 0$ and, therefore

  $$c^\top x^* + \lambda^\top (b - Ax^*) \geq c^\top x^* = Z_{IP}.$$

  Since $x^* \in X$,

  $$Z(\lambda) \geq c^\top x^* + \lambda^\top (b - Ax^*) \geq c^\top x^* = Z_{IP}.$$

- Problem (3) provides an upper bound to (1). It is natural to consider the tightest such bound.

Lagrangian Dual

- We introduce the problem

$$\min\ Z(\lambda), \quad \text{s.t.}\ \ \lambda \geq 0. \tag{4}$$

- We will refer to problem (4) as the Lagrangian dual. Let

$$Z_D = \min_{\lambda \geq 0}\ Z(\lambda).$$

- Suppose $X = \{x^1, \cdots, x^m\}$. Then $Z(\lambda)$ can be written as

$$Z(\lambda) = \max_{i=1,\cdots,m} (c^\top x^i + \lambda^\top (b - Ax^i)).$$

- The function $Z(\lambda)$ is convex and piecewise linear.
- Computing Z_D can be recast as a linear programming problem with a very large number of constraints.

Weak Duality

Theorem (Weak Duality)

We have $Z_D = \min_{\lambda \geq 0} \; Z(\lambda) \geq Z_{IP}$*.*

- The previous theorem represents the weak duality theory of integer programming.
- Unlike linear programming, integer programming does not have a strong duality theory. It is possible to have $Z_D > Z_{IP}$.
- The procedure of obtaining bounds for integer programming problems by calculating Z_D is called *Lagrangian relaxation*.

Strength of the Lagrangian Dual

Theorem

The optimal value Z_D *of the Lagrangian dual is equal to the optimal cost of the following linear programming problem:*

$$\begin{aligned} \max \quad & c^\top x, \\ \textit{s.t.} \quad & Ax \le b, x \in \mathrm{conv}(X). \end{aligned} \tag{5}$$

where $\mathrm{conv}(X)$ *be the convex hull of the set* $X = \{x \in \mathbb{Z}^n \mid Dx \le d\}$.

Proof:

$$Z(\lambda) = \max_{x \in X} \quad (c^\top x + \lambda^\top (b - Ax)).$$

- The optimal cost remains same if we allow convex combinations of the elements of X.

$$Z(\lambda) = \max_{x \in \mathrm{conv}(X)} \quad (c^\top x + \lambda^\top (b - Ax)).$$

Proof

- By definition, we have

$$Z_D = \min_{\lambda \geq 0} Z(\lambda) = \min_{\lambda \geq 0} \max_{x \in \mathrm{conv}(X)} (c^\top x + \lambda^\top (b - Ax)).$$

- Let $\{v^k, k \in K\}$ be the extreme points, and $\{r^j, j \in J\}$ be the complete set of extreme rays of $\mathrm{conv}(X)$.
- Then, for any fixed λ, we have

$$Z(\lambda) = \begin{cases} +\infty, & \exists\, j \in J, (c^\top - \lambda^\top A) r^j > 0, \\ \max_{k \in K} (c^\top v^k + \lambda^\top (b - Av^k)), & \text{otherwise.} \end{cases} \tag{6}$$

Proof

- According to (6), the Lagrangian dual is equivalent to and has the same optimal value as the problem

$$\begin{aligned} \min_{\lambda \geq 0} \quad & \max_{k \in K} (c^\top v^k + \lambda^\top (b - Av^k)), \\ \text{s.t.} \quad & (c^\top - \lambda^\top A) r^j \leq 0, \quad j \in J. \end{aligned} \tag{7}$$

- Problem (7) is equivalent to the linear programming problem

$$\begin{aligned} \min_{\lambda \geq 0} \quad & y, \\ \text{s.t.} \quad & y + \lambda^\top (Av^k - b) \geq c^\top v^k, \quad k \in K, \\ & \lambda^\top A r^j \geq c^\top r^j, \quad j \in J. \end{aligned} \tag{8}$$

Proof

- Taking the linear programming dual of problem (8), and using strong duality, Z_D is equal to the optimal cost of the problem

$$\begin{aligned} \max \quad & c^\top \left(\sum_{k \in K} \alpha_k v^k + \sum_{j \in J} \beta_j r^j \right), \\ \text{s.t.} \quad & A \left(\sum_{k \in K} \alpha_k v^k + \sum_{j \in J} \beta_j r^j \right) \leq b, \\ & \sum_{k \in K} \alpha_k = 1, \quad \alpha_k, \beta_j \geq 0. \end{aligned}$$

- The result follows since

$$\operatorname{conv}(X) = \left\{ \sum_{k \in K} \alpha_k v^k + \sum_{j \in J} \beta_j r^j \;\middle|\; \sum_{k \in K} \alpha_k = 1, \alpha_k, \beta_j \geq 0 \right\}$$

Linear Relaxation

- We have characterized the optimal value of the Lagrangian dual as solution to a linear programming problem.
- It is natural to compare the optimal cost Z_{IP} and the optimal cost Z_{LP} of the linear relaxation

$$\begin{array}{ll} \max & c^\top x, \\ \textbf{s.t.} & Ax \le b,\ Dx \le d. \end{array}$$

- In general, the following ordering holds among Z_{LP}, Z_{IP}, and Z_D:

$$Z_{LP} \ge Z_D \ge Z_{IP}.$$

Linear Relaxation

- We have $Z_{IP} = Z_D$ for all cost vector c, if and only if

$$\operatorname{conv}\left(X \cap \{x \mid Ax \leq b\}\right) = \operatorname{conv}(X) \cap \{x \mid Ax \leq b\}\,.$$

- We have $Z_{LP} = Z_D$ for all cost vector c, if

$$\operatorname{conv}(X) = \{x \mid Dx \leq d\}.$$

Solution of the Lagrangian Dual

- We outline a method for finding the optimal Lagrangian multipliers λ^*, that solve the Lagrangian dual problem

$$\min \ Z(\lambda), \quad \text{s.t. } \lambda \geq 0.$$

- To keep the presentation simple, we assume that X is finite and $X = \{x^1, \cdots, x^m\}$.
- Given a particular value of λ, we assume that we can calculate $Z(\lambda)$, which we have defined as follows:

$$Z(\lambda) = \max_{i=1,\cdots,m} (c^\top x^i + \lambda^\top (b - Ax^i)).$$

Subgradient

- Let $\mathrm{f}_i = b - Ax^i$ and $h_i = c^\top x^i$. Then,

$$Z(\lambda) = \max_{i=1,\cdots,m} (h_i + \mathrm{f}_i^\top \lambda).$$

- Let $E(\lambda) = \{i \mid Z(\lambda) = h_i + \mathrm{f}_i^\top \lambda\}$.
- For every $i \in E(\lambda^*)$, f_i is a subgradient of the function $Z(\cdot)$ at λ^*.
- $\partial Z(\lambda^*) = \mathrm{conv}(\{\mathrm{f}_i, i \in E(\lambda^*)\})$, i.e., a vector s is a subgradient of the function $Z(\cdot)$ at λ^* if and only if s is a convex combination of the vectors $\mathrm{f}_i, i \in E(\lambda^*)$.

Subgradient Optimization Algorithm

The following algorithm generalizes the steepest ascent algorithm to maximize a nondifferentiable concave function $Z(\cdot)$.

1. Choose a starting point λ^1; let $t = 1$.
2. Given λ^t, choose a subgradient s^t of the function $Z(\cdot)$ at λ^t.
3. If $s^t = 0$, then λ^t is optimal and the algorithm terminates. Else, continue.
4. Let $\lambda_j^{t+1} = \max\{\lambda_j^t - \theta_t s_j^t, 0\}$, where θ_t is a positive stepwise parameter. Increment t and go to Step 2.

- Typically, only the extreme subgradients f_i are used.
- The stopping criterion $0 \in \partial Z(\lambda^t)$ is rarely met. Typically, the algorithm is stopped after a fixed number of iterations.

Stepsize

- It can be proved that $Z(\lambda^t)$ converges for any stepsize sequence θ_t such that
$$\sum_{t=1}^{\infty} \theta_t = \infty, \quad \text{and} \quad \lim_{t\to\infty} \theta_t = 0.$$
- An example of the stepsize sequence is $\theta_t = 1/t$, which leads to slow convergence in practical. Another example is
$$\theta_t = \theta_0 \alpha^t, \quad t = 1, 2, \cdots,$$
where α is a scalar satisfying $0 < \alpha < 1$.
- A more sophisticated and popular rule is to let
$$\theta_t = \frac{Z(\lambda^t) - \hat{Z}_D}{\|s^t\|^2} \alpha$$
where α is a scalar satisfying $0 < \alpha < 1$ and $\hat{Z}_D$ is an estimate of the optimal value Z_D.

Outline

1 Lagrangian Relaxation

2 Dantzig-Wolfe decomposition

3 Bender's Decomposition

Mixed Integer Program

- Let us consider a mixed integer program (MIP)

$$\begin{aligned} z_I \quad = \quad \max \quad & c^T x, \\ \text{s.t.} \quad & Ax \leq b, Dx \leq d, \\ & x \in \mathbb{Z}_+^d \times R_+^p. \end{aligned} \tag{9}$$

- Let X be defined as

$$X = \left\{ x \in \mathbb{Z}_+^d \times R_+^p \; : \; Dx \leq d \right\}.$$

We assume that X is nonempty and D, d have rational entries.

Lagrangian dual

- Let m be the number of rows of A, and take $\lambda \in \mathbb{R}^m_+$. The Lagrangian relaxation with respect to λ as follows.

$$\begin{aligned} z_{LR}(\lambda) \quad = \quad \max \quad & c^\top x + \lambda^\top (b - Ax), \\ \text{s.t.} \quad & Dx \leq d, \\ & x \in \mathbb{Z}^q_+ \times R^p_+. \end{aligned} \tag{10}$$

- Moreover, recall that the Lagrangian dual is defined as

$$z_{LD} \quad = \quad \min\{Z_{LR}(\lambda) : \ \lambda \geq 0\}. \tag{11}$$

- (10) and (11) are related according to the following characterization of z_{LD}.

$$z_{LD} = \max\{c^\top x : \ Ax \leq b, x \in \text{conv}(X)\}.$$

Decomposition of $\mathrm{conv}(X)$

- $\mathrm{conv}(X)$ can be expressed as

$$\mathrm{conv}(X) = \mathrm{conv}\left\{v^1, \ldots, v^n\right\} + \mathrm{cone}\left\{r^1, \ldots, r^\ell\right\},$$

  where $v^1, \ldots, v^n$ are the extreme points of $\mathrm{conv}(X)$ and $r^1, \ldots, r^\ell$ are the extreme rays of $\mathrm{conv}(X)$.

- Any point x in $\mathrm{conv}(X)$ can be written as

$$x = \sum_{k \in [n]} \alpha_k v^k + \sum_{h \in [\ell]} \beta_h r^h$$

  for some $\alpha \in \mathbb{R}^k_+$ and $\beta \in \mathbb{R}^\ell_+$ such that $\sum_{k \in [n]} \alpha_k = 1$.

Dantzig-Wolfe Relaxation

Based on the decomposition of $\operatorname{conv}(X)$, it follows that

$$
\begin{aligned}
z_{\text{LD}} = \max \quad & \sum_{k \in [n]} \left(c^\top v^k\right) \alpha_k + \sum_{h \in [\ell]} \left(c^\top r^h\right) \beta_k, \\
\textbf{s.t.} \quad & \sum_{k \in [n]} \left(A v^k\right) \alpha_k + \sum_{h \in [\ell]} \left(A r^h\right) \beta_k \le b, \\
& \sum_{k \in [n]} \alpha_k = 1, \alpha \in \mathbb{R}^k_+, \beta \in \mathbb{R}^\ell_+.
\end{aligned}
\tag{12}
$$

We refer to (12) as the Dantzig-Wolfe relaxation.

Dantzig-Wolfe Reformulation

- Moreover, we have

$$z_I = \max\left\{c^\top x : Ax \le b, x \in \operatorname{conv}(X), x_j \in \mathbb{Z},\ \forall j \in [q]\right\}.$$

- Therefore, we deduce

$$\begin{aligned} z_I = \max & \sum_{k\in[n]} \left(c^\top v^k\right)\alpha_k + \sum_{h\in[\ell]} \left(c^\top r^h\right)\beta_k, \\ \text{s.t.} & \sum_{k\in[n]} \left(Av^k\right)\alpha_k + \sum_{h\in[\ell]} \left(Ar^h\right)\beta_k \le b, \\ & \sum_{k\in[n]} \alpha_k = 1, \\ & \alpha \in \mathbb{R}^k_+, \beta \in \mathbb{R}^\ell_+, \\ & \sum_{k\in[n]} \alpha_k v^k_j + \sum_{h\in[\ell]} \beta_h r^h_j \in \mathbb{Z}, \quad j \in [q]. \end{aligned} \tag{13}$$

- Here, (13) is referred to as the Dantzig-Wolfe reformulation.

Pure Binary Programs

- Let us consider a pure binary integer program as follows.

$$\begin{aligned} z_I = \max \quad & c^\top x, \\ \text{s.t.} \quad & Ax \leq b, Dx \leq d, \\ & x \in \{0,1\}^p. \end{aligned}$$

- We define X as

$$X = \{x \in \{0,1\}^p : Dx \leq d\}.$$

- Since X is bounded and finite, $X = \{v^1, \ldots, v^n\}$
- Any point x in X can be expressed as

$$x = \sum_{k \in [n]} \alpha_k v^k, \quad \sum_{k \in [n]} \alpha_k = 1, \quad \alpha \in \{0,1\}^n.$$

Pure Binary Programs

- Then we obtain the Dantzig-Wolfe reformulation.

$$\begin{aligned} z_I = \max \quad & \sum_{k\in[n]} \left(c^\top v^k\right) \alpha_k, \\ \text{s.t.} \quad & \sum_{k\in[n]} \left(Av^k\right) \alpha_k \le b, \\ & \sum_{k\in[n]} \alpha_k = 1, \quad \alpha \in \{0,1\}^n. \end{aligned}$$

- The Dantzig-Wolfe relaxation

$$\begin{aligned} \max & \sum_{k\in[n]} \left(c^\top v^k\right) \alpha_k, \\ \text{s.t.} & \sum_{k\in[n]} \left(Av^k\right) \alpha_k \le b, \\ & \sum_{k\in[n]} \alpha_k = 1, \quad \alpha \ge 0. \end{aligned}$$

Block Diagonal Structure

- We consider the following optimization model with block diagonal structure.

$$\begin{array}{llllll}
\max & c^{1\top}x^1 + & c^{2\top}x^2 + \cdots & +c^{p\top}x^p, & & \\
\text{s.t.} & D^1x^1 & & & & \leq d^1, \\
 & & D^2x^2 & & & \leq d^2, \\
 & & & \ddots & & \\
 & & & & D^px^p & \leq d^p, \\
 & A^1x^1 + & A^2x^2 + \cdots & & +A^px^p & \leq b, \\
 & x^j \in \{0,1\}^{q_j}, & j \in [p]. & & &
\end{array}$$

- For $j \in [p]$, let X_j be defined as $X_j = \left\{x^j \in \{0,1\}^{q_j} : D^jx^j \leq d^j\right\}$.
- X_j is bounded and finite. Any point x^j in X_j can be written as

$$x^j = \sum_{v \in X_j} \alpha_v^j v, \quad \sum_{v \in X_j} \alpha_v^j = 1, \quad \alpha^j \in \{0,1\}^{|X_j|}.$$

Block Diagonal Structure

- The Dantzig-Wolfe reformulation is given by

$$\begin{aligned} \max \quad & \sum_{v \in X_1} \left(c^{1\top} v\right) \alpha_v^1 + \sum_{v \in X_2} \left(c^{2\top} v\right) \alpha_v^2 + \cdots + \sum_{v \in X_p} \left(c^{p\top} v\right) \alpha_v^p, \\ \text{s.t.} \quad & \sum_{v \in X_1} (A^1 v) \alpha_v^1 + \sum_{v \in X_2} (A^2 v) \alpha_v^2 + \cdots + \sum_{v \in X_p} (A^p v) \alpha_v^p \le b, \\ & \sum_{v \in X_j} \alpha_v^j = 1, \quad \alpha^j \in \{0,1\}^{|X_j|}, \quad j \in [p]. \end{aligned}$$

- The Dantzig-Wolfe relaxation is given by

$$\begin{aligned} \max \quad & \sum_{v \in X_1} \left(c^{1\top} v\right) \alpha_v^1 + \sum_{v \in X_2} \left(c^{2\top} v\right) \alpha_v^2 + \cdots + \sum_{v \in X_p} \left(c^{p\top} v\right) \alpha_v^p, \\ \text{s.t.} \quad & \sum_{v \in X_1} (A^1 v) \alpha_v^1 + \sum_{v \in X_2} (A^2 v) \alpha_v^2 + \cdots + \sum_{v \in X_p} (A^p v) \alpha_v^p \le b, \\ & \sum_{v \in X_j} \alpha_v^j = 1, \quad \alpha^j \ge 0, \quad j \in [p]. \end{aligned}$$

Column Generation: Master Problem

- The Dantzig-Wolfe relaxation has variables $\alpha_1, \ldots, \alpha_n$ for the extreme points of $\mathrm{conv}(X)$ and variables $\beta_1, \ldots, \beta_\ell$ for the extreme rays of $\mathrm{conv}(X)$.
- n and ℓ are potentially very large. In this case, we may apply the column generation technique.
- The column generation procedure works as follows. We start with $N \subseteq [n]$ and $L \subseteq [\ell]$. Then we have the master problem

$$\begin{aligned} \max \quad & \sum_{k \in N} \left(c^\top v^k\right) \alpha_k + \sum_{h \in L} \left(c^\top r^h\right) \beta_k, \\ \text{s.t.} \quad & \sum_{k \in N} \left(Av^k\right) \alpha_k + \sum_{h \in L} \left(Ar^h\right) \beta_k \leq b, \\ & \sum_{k \in N} \alpha_k = 1, \quad \alpha \in \mathbb{R}^k_+, \beta \in \mathbb{R}^\ell_+. \end{aligned}$$

Column Generation: Subproblem — 原理解析

- 子问题的构造基于对偶理论，其目标是在当前主问题解的对偶向量λ下，检验是否存在未加入的列（变量）仍可提升目标函数。
- 设当前主问题对偶解为λ，则任意潜在变量$x \in \mathrm{conv}(X)$的reduced cost为：

$$\bar{c}(x) = c^\top x - \lambda^\top A x.$$

- 若存在$x \in \mathrm{conv}(X)$使得$\bar{c}(x) > 0$，则当前主问题并非最优。
- 因此，构造如下子问题以判定最优性并生成新列：

$$\max_{x \in \mathrm{conv}(X)} \left(c^\top + \lambda^\top (b - Ax) \right).$$

- 三种情形：
  - 若最优值> 0：存在可提升目标的极点/极射线；
  - 若最优值$= 0$：当前已最优，无需再加列；
  - 若最优值为$+\infty$：存在无界方向（极射线）可继续改进。

Column Generation: Subproblem — 算法流程

1. 解主问题（restricted master problem），得到对偶变量λ；
2. 构造子问题：
$$\max_{x \in \mathrm{conv}(X)} \quad c^\top x + \lambda^\top (b - Ax);$$
3. 若子问题最优值> 0，则：
    - 若最优解为极点v^k，则有
    $$(Av^k - b)^\top \lambda < c^\top v^k,$$
    加入v^k 到主问题；
    - 若子问题无界，存在极射线r^h，则有
    $$(Ar^h)^\top \lambda < c^\top r^h,$$
    加入r^h 到主问题；
4. 若子问题最优值≤ 0，则当前解已为全局最优，算法终止。

注： 子问题等价于在X 上求解线性目标$\left(c - A^\top \lambda\right)^\top x$ 的最优解，是pricing problem 的具体实现。

Outline

1. Lagrangian Relaxation
2. Dantzig-Wolfe decomposition
3. Bender's Decomposition

Bender's Decomposition

- We use the Lagrangian relaxation framework to deal with complicating constraints.
- In this section, we learn the Bender's reformulation technique that can deal with complicating variables.
- Consider the following mixed-integer program.

$$\begin{aligned} z_I = \max \quad & c^\top x + q^\top y, \\ \text{s.t.} \quad & Ax + Gy \leq b, \\ & x \in \mathbb{Z}_+^d, y \in \mathbb{R}_+^p. \end{aligned}$$

Bender's Decomposition

- Here, the integer variables x are complicating variables. If we fix the x part, then the optimization problem becomes

$$\begin{aligned} z_{LP}(x) = \max \quad & q^\top y, \\ \text{s.t.} \quad & Gy \leq b - Ax, \\ & y \in \mathbb{R}^p_+. \end{aligned}$$

- Taking the dual of it, we deduce

$$\begin{aligned} \min \quad & u^\top (b - Ax), \\ \text{s.t.} \quad & G^\top u \geq q, \\ & u \geq 0. \end{aligned}$$

- Here, the feasible set of the dual does not depend on x.

Bender's Decomposition

- Let Q denote the feasible set of the dual:

$$Q = \left\{ u : G^\top u \geq q, u \geq 0 \right\}.$$

- Suppose that Q can be expressed as

$$Q = \operatorname{conv}\left\{v^1, \ldots, v^n\right\} + \operatorname{cone}\left\{r^1, \ldots, r^\ell\right\}.$$

  for some vectors $v^1, \ldots, v^n$ and $r^1, \ldots, r^\ell$.
- We will prove the following theorem.

Theorem (Bender's Decomposition)

The mixed integer program can be reformulated as

$$\begin{aligned} z_I = \max \quad & \eta, \\ \textit{s.t.} \quad & \eta \leq c^\top x + (b - Ax)^\top v^k, \quad k \in [n], \\ & (b - Ax)^\top r^h \geq 0, \quad h \in [\ell], \\ & x \in \mathbb{Z}^d_+, \quad \eta \in \mathbb{R}. \end{aligned}$$

Projection Theorem of Egon Balas

Theorem

Let $P = \left\{(x,y) \in \mathbb{R}^d \times \mathbb{R}^p : Ax + Gy \leq b, y \geq 0\right\}$*. Suppose that* $C = \left\{u : G^\top u \geq 0, u \geq 0\right\}$ *can be expressed as* $C = \operatorname{cone}\left\{r^1, \ldots, r^\ell\right\}$*. Then* $\operatorname{proj}_x(P)$*, the projection of* P *onto the* x*-space, is given by*

$$\operatorname{proj}_x(P) = \left\{x \in \mathbb{R}^d : (b - Ax)^\top r^h \geq 0, h \in [\ell]\right\}.$$

- Let $\bar{x} \in \mathbb{R}^d$. Note that $\bar{x} \notin \operatorname{proj}_x(P)$ holds if and only if there is no $y \in \mathbb{R}^p$ that satisfies $Gy \leq b - A\bar{x}$ and $y \geq 0$.
- By Farkas' Lemma, the system $Gy \leq b - A\bar{x}, y \geq 0$ is infeasible if and only if there exists $u \in C$ such that $u^\top(b - A\bar{x}) < 0$.
- Since $C = \operatorname{cone}\left\{r^1, \ldots, r^\ell\right\}$, such a vector u exists if and only if $(b - A\bar{x})^\top r^h \leq 0$ for some $h \in [\ell]$, in which case, $\bar{x} \notin \left\{x \in \mathbb{R}^d : (b - Ax)^\top r^h \geq 0, h \in [\ell]\right\}$.

Proof of Bender's Decomposition

- Let $P = \left\{(x,y) \in \mathbb{R}^d \times \mathbb{R}^p : Ax + Gy \leq b, y \geq 0\right\}$. Note that

$$\begin{aligned} z_I = \max \quad & c^\top x + z_{LP}(x), \\ \text{s.t.} \quad & x \in \mathbb{Z}^d_+. \end{aligned}$$

- Here, $z_{LP}(x) > -\infty$ if and only if there exists some $y \geq 0$ such that $Gy \leq b - Ax$, which is equivalent to $x \in \mathrm{proj}_x(P)$.
- Therefore, it follows that

$$\begin{aligned} z_I = \max \quad & c^\top x + z_{LP}(x), \\ \text{s.t.} \quad & x \in \mathrm{proj}_x(P) \cap \mathbb{Z}^d_+. \end{aligned}$$

Proof of Bender's Decomposition

- Recall that $Q = \left\{u : G^\top u \geq q, u \geq 0\right\}$ and

$$Q = \text{conv}\left\{v^1, \ldots, v^n\right\} + \text{cone}\left\{r^1, \ldots, r^\ell\right\}.$$

- Then $C = \left\{u : G^\top u \geq 0, u \geq 0\right\}$ is the recession cone of Q, so we have $C = \text{cone}\left\{r^1, \ldots, r^\ell\right\}$.
- Then it follows from projection theorem of Egon Balas that $\text{proj}_x(P) = \left\{x \in \mathbb{R}^d : (b - Ax)^\top r^h \geq 0, h \in [\ell]\right\}$.
- Therefore, we deduce that

$$\begin{aligned} z_I = \max \quad & c^\top x + z_{LP}(x), \\ \text{s.t.} \quad & (b - Ax)^\top r^h \geq 0, \quad h \in [\ell], \\ & x \in \mathbb{Z}_+^d. \end{aligned}$$

Proof of Bender's Decomposition

- Moreover, note that for any $x \in \text{proj}_x(P), z_{LP}(x) > -\infty$, so strong duality implies that

$$\begin{aligned} z_{LP}(x) = \min \quad & u^\top (b - Ax), \\ \text{s.t.} \quad & G^\top u \geq q, \\ & u \geq 0. \end{aligned}$$

- If $z_{LP}(x)$ is finite, then it means that Q is non-empty and

$$z_{LP}(x) = \min_{k \in [n]} \left\{ (b - Ax)^\top v^k \right\}.$$

- If $z_{LP}(x) = +\infty$, then Q is empty, so the above equation also holds. Hence,

$$\begin{aligned} z_I = \max \quad & c^\top x + \min_{k \in [n]} \left\{ (b - Ax)^\top v^k \right\}, \\ \text{s.t.} \quad & (b - Ax)^\top r^h \geq 0, \quad h \in [\ell], \\ & x \in \mathbb{Z}_+^d. \end{aligned}$$

Proof of Bender's Decomposition

- We may move the term $\min_{k\in[n]}\left\{(b-Ax)^\top v^k\right\}$ in the objective to constraints, after which we deduce that

$$\begin{aligned} z_I = \max \quad & \eta, \\ \text{s.t.} \quad & \eta \le c^\top x + \min_{k\in[n]}\left\{(b-Ax)^\top v^k\right\}, \\ & (b-Ax)^\top r^h \ge 0, \quad h \in [\ell], \\ & x \in \mathbb{Z}_+^d, \quad \eta \in \mathbb{R}. \end{aligned}$$

which is equivalent to Bender's reformulation as required.

Bender’s Decomposition Algorithm

- The Bender’s reformulation has an enormous number of constraints.
- A natural approach is to work with a small subset of the constraints and add new ones as cutting planes.
- The Bender’s decomposition algorithm is the row generation framework for Bender’s reformulation.

Master Problem

- At iteration t, we have $N_t \subseteq [n]$ and $L_t \subseteq [\ell]$. Then we solve

$$\begin{aligned} z_I^t = \max \quad & \eta, \\ \textbf{s.t.} \quad & \eta \leq c^\top x + (b - Ax)^\top v^k, \quad k \in N_t, \\ & (b - Ax)^\top r^h \geq 0, \quad h \in L_t, \\ & x \in \mathbb{Z}_+^d, \eta \in \mathbb{R}. \end{aligned}$$

  This is the **master problem.**

- Assume that we get a solution (x^t, η^t) after solving the master problem at iteration t. Then we attempts to find a violated inequality among

$$\begin{aligned} & \eta \leq c^\top x + (b - Ax)^\top v^k, \quad k \in [n] \backslash N_t, \\ & (b - Ax)^\top r^h \geq 0, \quad h \in [\ell] \backslash L_t. \end{aligned}$$

Subproblem

- The question is
    - does there exists $k_t \in [n]$ such that
    $$\eta^t > c^\top x^t + (b - Ax^t)^\top v^{k_t}?$$
    - does there exists $h_t \in [\ell]$ such that
    $$(b - Ax^t)^\top r^{h_t} < 0?$$
- To answer this, we solve
$$\begin{aligned} z_{LP}(x^t) = \max \quad & q^\top y, \\ \textbf{s.t.} \quad & Gy \leq b - Ax^t, \\ & y \in \mathbb{R}^p_+. \end{aligned}$$
- This is the **subproblem** for the Bender's decomposition algorithm.

Solving the Subproblem

- If $z_{LP}\left(x^t\right) = +\infty$, then for any $M > 0$, there exists $y \geq 0$ such that $Ax^t + Gy \leq b$ and $c^\top x^t + q^\top y > M$, in which case $z_I = +\infty$.
- If $z_{LP}\left(x^t\right)$ is finite, then

$$z_{LP}\left(x^t\right) = \min_{k \in [n]} \left(b - Ax^t\right)^\top v^k = \left(b - Ax^t\right)^\top v^{k_t}$$

  for some k_t.
- Hence, we deduce that

$$c^\top x^t + z_{LP}\left(x^t\right) = c^\top x^t + \left(b - Ax^t\right)^\top v^{k_t}.$$

- Moreover, if $z_{LP}\left(x^t\right) = -\infty$, then the subproblem is infeasible, in which case, there exists $h_t \in [\ell]$

$$\left(b - Ax^t\right)^\top r^{h_t} < 0.$$

Bender's decomposition algorithm

1. At iteration t, solve the master problem with $N_t \subseteq [n]$ and $L_t \subseteq [\ell]$.
2. If $z_I^t = -\infty$, then the mixed-integer program is infeasible.
3. Let (x^t, η^t) be an optimal solution to the master problem. Solve the subproblem with x^t.
4. If $z_{LP}(x^t) = +\infty$ then the mixed-integer program is unbounded.
5. If $z_{LP}(x^t) = -\infty$ then there exists $h_t \in [\ell]$ such that $(b - Ax)^\top r^{h_t} < 0$.
   Add constraint $(b - Ax)^\top r^{h_t} \geq 0$ and update $L_{t+1} = L_t \cup \{h_t\}$.
6. If $z_{LP}(x^t)$ is finite. Let y^t be an optimal solution and $k_t \in \operatorname{argmin}_{k \in [n]} \{(b - Ax^t)^\top > v^k\}$.
   If $c^\top x^t + q^\top y^t \geq \eta^t$, then we conclude that (x^t, y^t) is an optimal solution.
   If $c^\top x^t + q^\top y^t < \eta^t$, then we add constraint $\eta \leq c^\top x + (b - Ax)^\top v^{k_t}$ and update $N_{t+1} = N_t \cup \{k_t\}$.

Machine Learning for Solving Large-scale Integer Programming Problems

Zaiwen Wen

Beijing International Center For Mathematical Research
Peking University

Outline

1. Introduction
2. Machine learning for binary optimization
3. Machine learning for general MILPs
4. Machine learning for routing problems

Maxcut: 0.878 bounds

- For graph (V,E) and weights $w_{ij} = w_{ji} \geq 0$, the maxcut problem is

$$(Q) \quad \max_{x} \sum_{i<j} w_{ij}(1 - x_i x_j), \quad \text{s.t.} \quad x_i \in \{-1, 1\}$$

- SDP relaxation

$$(SDP) \quad \max_{X \in S^n} \sum_{i<j} w_{ij}(1 - X_{ij}), \quad \text{s.t.} \quad X_{ii} = 1, X \succeq 0$$

  Compute the decomposition $X = V^\top V$, where $V = [v_1, v_2, \dots, v_n]$

- Rounding: generate a vector r uniformly distributed on the unit sphere, i.e., $\|r\|_2 = 1$, set

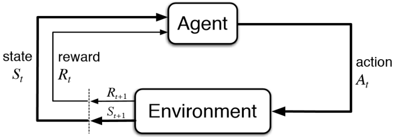
$$x_i = \begin{cases} 1 & v_i^\top r \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

- Let $Z^*_{(SDP)}$ and $Z^*_{(Q)}$ be the optimal values of (SDP) and (Q)

$$E(W) \geq 0.878 Z^*_{(SDP)} \geq 0.878 Z^*_{(Q)}$$

Reinforcement Learning

Consider an infinite-horizon discounted Markov decision process (MDP), usually defined by a tuple $(\mathcal{S}, \mathcal{A}, P, R, \rho_0, \gamma)$;



Agent

Environment

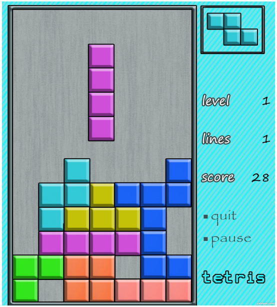
state S_t

reward R_t

action A_t

R_{t+1}

S_{t+1}



- The policy is supposed to maximize the total expected reward:

$$\max_{\pi} \mathrm{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right], \text{ with } s_0 \sim \rho_0, a_t \sim \pi(\cdot|s_t), s_{t+1} \sim P(\cdot|s_t, a_t).$$

Erdos Goes Neural

- The probability distribution $\mathcal{D}$ in *Erdos* is learned by a GNN.
- A "good" probability distribution leads to higher quality solutions.

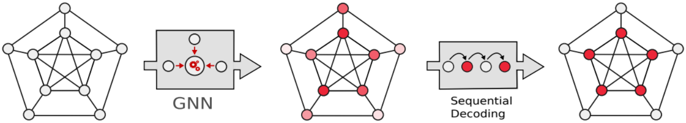


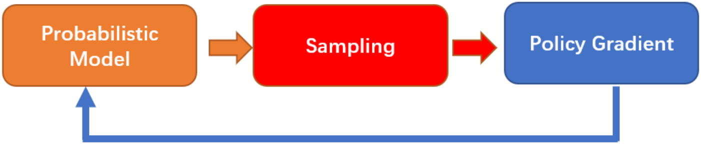
Figure: Illustration of the "Erdos goes neural" pipeline.

- Optimization on explicit formulation of the expectation.
- Maximum clique problem:

$$\ell(\mathcal{D}) = \gamma - (\beta + 1) \sum_{(v_i,v_j)\in E} w_{ij} p_i p_j + \frac{\beta}{2} \sum_{v_i \neq v_j} p_i p_j.$$

Parameterized Probabilistic Model

- **MCPG**: construct a parameterized model with parameter θ to output p_θ and generate $x \sim p_\theta$ by Monte Carlo sampling



- MCPG: optimization over the probabilistic space.
- Erdos: optimization on the expectation of objective function.

Outline

1. Introduction
2. Machine learning for binary optimization
3. Machine learning for general MILPs
4. Machine learning for routing problems

Binary Optimization

Let f be arbitrary (even non-smooth) cost function:

$$\min \quad f(x), \quad \text{s.t.} \quad x \in \mathcal{B}_n = \{-1,1\}^n.$$

- Example: maxcut problem on $G = (V,E)$

$$\max \quad \sum_{(i,j)\in E} w_{ij}(1 - x_i x_j), \quad \text{s.t.} \quad x \in \{-1,1\}^n.$$

- Example: maxSAT problem:

$$\max_{x\in\{-1,1\}^n} \quad \sum_{c^i\in C_1} \max\{c_1^i x_1, c_2^i x_2, \cdots, c_n^i x_n, 0\},$$
$$\text{s.t.} \quad \max\{c_1^i x_1, c_2^i x_2, \cdots, c_n^i x_n, 0\} = 1, \quad \text{for } c^i \in C_2$$

- Binary optimization is NP-hard due to the combinatorial structure.

Probabilistic Approach

Let $\mathcal{X}^*$ be the set of optimal solutions and consider the distribution,

$$q^*(x) = \frac{1}{|\mathcal{X}^*|}\mathbf{1}_{\mathcal{X}^*}(x) = \begin{cases} \frac{1}{|\mathcal{X}^*|}, & x \in \mathcal{X}^*, \\ 0, & x \notin \mathcal{X}^*. \end{cases}$$

Motivation: Searching for optimal points $\mathcal{X}^* \Rightarrow$ Constructing a distribution $p_\theta(x)$ converging to $q^*(x)$.

- A universal approach for various binary optimization problems.
- Algorithms for continuous optimization can be applied.
- The optimal points set $\mathcal{X}^*$ is unknown.

Gibbs distributions

- To approximate q^*, we introduce Gibbs distributions,

$$q_\lambda(x) = \frac{1}{Z_\lambda} \exp\left(-\frac{f(x)}{\lambda}\right), \quad x \in \mathcal{B}_n,$$

  where $Z_\lambda = \sum_{x \in \mathcal{B}_n} \exp\left(-\frac{f(x)}{\lambda}\right)$ is the normalizer.

- Given the optimal objective value f^*, for any $x \in \mathcal{B}_n$,

$$\begin{aligned} q_\lambda(x) = \frac{\exp\left(\frac{f^* - f(x)}{\lambda}\right)}{\sum_{x \in \mathcal{B}_n} \exp\left(\frac{f^* - f(x)}{\lambda}\right)} &= \frac{\exp\left(\frac{f^* - f(x)}{\lambda}\right)}{|\mathcal{X}^*| + \sum_{x \in \mathcal{B}_n / \mathcal{X}^*} \exp\left(\frac{f^* - f(x)}{\lambda}\right)} \\ \to \frac{1}{|\mathcal{X}^*|} \mathbf{1}_{\mathcal{X}^*}(x) = q^*, &\quad \text{as } \lambda \to 0. \end{aligned}$$

- The calculation of q_λ does not require knowledge of $\mathcal{X}^*$.

Parameterized Probabilistic Model

- KL divergence:

$$\mathrm{KL}\left(p_\theta \,\|\, q_\lambda\right) = \sum_{x \in \mathbf{B}_n} p_\theta(x) \log \frac{p_\theta(x)}{q_\lambda(x)}.$$

- In order to reduce the discrepancy between p_θ and q_λ, the KL divergence is supposed to be minimized:

$$\begin{aligned}\mathrm{KL}\left(p_\theta \,\|\, q_\lambda\right) &= \frac{1}{\lambda} \sum_{x \in \mathbf{B}_n} p_\theta(x) f(x) + \sum_{x \in \mathbf{B}_n} p_\theta(x) \log p_\theta(x) + \log Z_\lambda \\ &= \frac{1}{\lambda} \left(\mathbb{E}_{p_\theta}\left[f(x)\right] + \lambda \mathbb{E}_{p_\theta}\left[\log p_\theta(x)\right]\right) + \log Z_\lambda.\end{aligned}$$

- Loss Function (Z_λ is a constant):

$$\boxed{\min_\theta \quad L_\lambda(\theta) = \mathbb{E}_{p_\theta}\left[f(x)\right] + \lambda \mathbb{E}_{p_\theta}\left[\log p_\theta(x)\right]}$$

Gradient for the Loss Function

Lemma 1

Suppose for any $x \in \mathcal{B}_n$, $p_\theta(x)$ *is differentiable with respect to* θ. *For any constant* $c \in \mathrm{Re}$, *we denote the advantage function*

$$A_\lambda(x; \theta, c) := f(x) + \lambda \log p_\theta(x) - c.$$

Then, the gradient of the loss function is given by

$$\nabla_\theta L_\lambda(\theta) = \mathbb{E}_{p_\theta}\left[A_\lambda(x; \theta, c) \nabla_\theta \log p_\theta(x)\right].$$

One candidate for c *is*

$$c = \mathbb{E}_{p_\theta}[f(x)].$$

Very similar to the policy gradient in reinforcement learning!

Extension: general constrained problem

- Consider

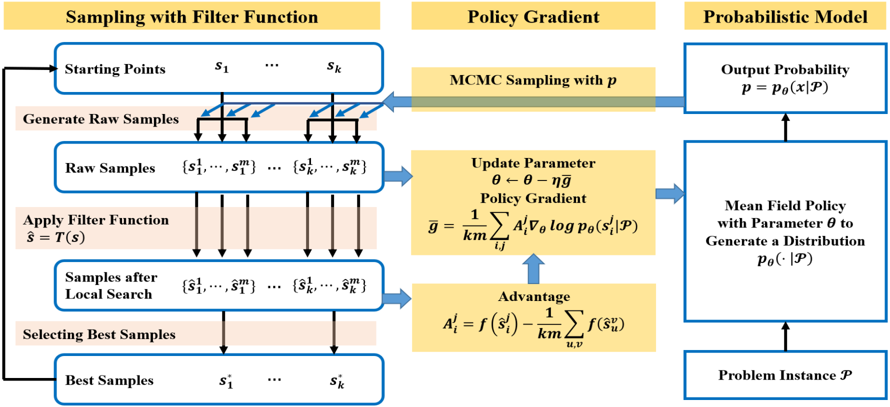
$$x^* = \arg\min_{x} \quad f(x), \quad \text{s.t.} \;\; c(x) = 0, \; x \in \mathcal{B}_n$$

- L1 exact penalty problem

$$x_\sigma^* = \arg\min_{x \in \mathcal{B}_n} f_\sigma(x) := f(x) + \sigma \|c(x)\|_1$$

- Let $\varpi := \min_{x \in \mathcal{B}_n} \{\|c(x)\|_1 \mid \|c(x)\|_1 \neq 0\}$ and $f^* = \min_{x \in \mathcal{B}_n} f(x)$. Define $\bar{\sigma} = (f_\sigma(x^*) - f^*)/\varpi \geq 0$.
- For all $\sigma \geq \bar{\sigma}$, x^* is a global minima of the penalty problem and x_σ^* is also a global minima of the constrained problem.

Pipeline of MCPG



Sampling with Filter Function

Starting Points s_1 ⋯ s_k

Generate Raw Samples

Raw Samples $\{s_1^1, \cdots, s_1^m\}$ ⋯ $\{s_k^1, \cdots, s_k^m\}$

Apply Filter Function
$\hat{s} = T(s)$

Samples after Local Search $\{\hat{s}_1^1, \cdots, \hat{s}_1^m\}$ ⋯ $\{\hat{s}_k^1, \cdots, \hat{s}_k^m\}$

Selecting Best Samples

Best Samples s_1^* ⋯ s_k^*

Policy Gradient

MCMC Sampling with p

Update Parameter
$\theta \leftarrow \theta - \eta \bar{g}$
Policy Gradient

$$\bar{g} = \frac{1}{km}\sum_{i,j} A_i^j \nabla_\theta \log p_\theta(s_i^j|\mathcal{P})$$

Advantage

$$A_i^j = f\left(\hat{s}_i^j\right) - \frac{1}{km}\sum_{u,v} f(\hat{s}_u^v)$$

Probabilistic Model

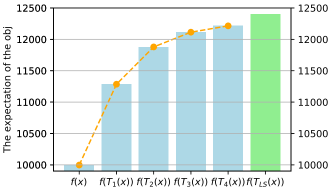
Output Probability
$p = p_\theta(x|\mathcal{P})$

Mean Field Policy
with Parameter θ to
Generate a Distribution
$p_\theta(\cdot\,|\mathcal{P})$

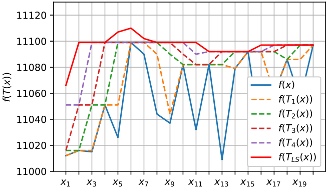
Problem Instance $\mathcal{P}$

Filter Function

- The filter function T projects x to a better one in the neighborhood.
- Applied with the filter function, $f(T(x))$ has fewer local minima and the same global minimum as the original one.



(a) Expectation of the objective function.



(b) A selected sequence of solutions.

Filter Function

Definition 2 (Filter Function)

For each $x \in \mathcal{B}_n$, let $\mathcal{N}(x) \subset \mathcal{B}_n$ be a neighborhood of x such that $x \in \mathcal{N}(x)$, $|\mathcal{N}(x)| \geq 2$ and any point in $\mathcal{N}(x)$ can be reached by applying a series of “simple" operations to x. A filter function $T(x)$ is defined as

$$T(x) \in \arg\min_{\hat{x} \in \mathcal{N}(x)} f(\hat{x}),$$

where $T(x)$ is arbitrarily chosen if there exists multiple solutions.

- Projection to the best solution on the neighborhood:

$$T_k(x) = \arg\min_{\|\hat{x}-x\|_1 \leq 2k} f(\hat{x}), \quad \mathcal{N}(x) = \{\hat{x} \mid \|\hat{x} - x\|_1 \leq 2k\}.$$

- Algorithms serves as the filter function:

$$T_{LS}(x) = \texttt{LocalSearch}_f(x).$$

Local Search

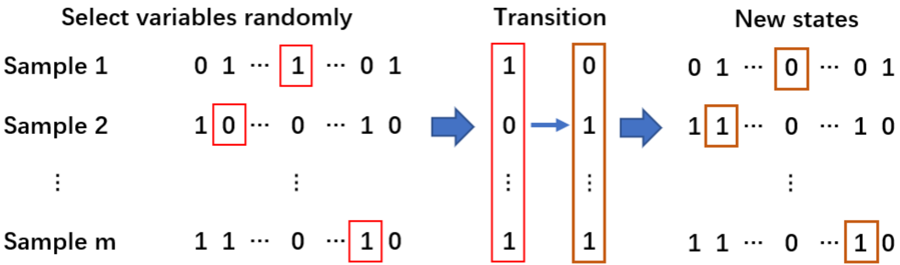
Local Search:

- **Generality**: Local search works for various kinds of problem.
- **Efficiency**: GPUs allow parallel access to the same indexed variable for a large number of samples.

Pipeline of Local Search with flipping operation:

1. Choose a single variable from the current solution x.
2. Flip the variable to its opposite value.
3. Evaluate the new solution to determine if it is improvement.
4. If it is, the variable is flipped to its opposite value
5. Back to Step 1 and continues to the next index in I.

Large-Scale Parallel Sampling on GPU



- GPU: quick for parallel accessing but slow for memory copying.

Sampling in MCPG

- constructs large number of short chains,
- discards all previous states in transition (no memory copying),
- outputs the last states for all chains.

Probabilistic Model Applied with Filter Function

- MCPG focuses on the following modified binary optimization:

$$\min \quad f(T(x)), \quad \text{s.t.} \quad x \in \mathcal{B}_n.$$

- The probabilistic model is equivalent to

$$\min_{\theta} \quad L_\lambda(\theta;\mathcal{P}) = \mathbb{E}_{p_\theta}\left[f(T(x))\right] + \lambda \mathbb{E}_{p_\theta}\left[\log p_\theta(x|\mathcal{P})\right].$$

- Empirical gradient:

$$\bar{g}_\lambda(\theta) = \frac{1}{|S|}\sum_{x\in S} A_\lambda(x;\theta)\nabla_\theta \log p(x|\theta;\mathcal{P}).$$

where S is the sample set extracted from distribution $p_\theta(\cdot|\mathcal{P})$ and

$$A_\lambda(x;\theta) := f(T(x)) + \lambda \log p_\theta(x|\mathcal{P}) - \frac{1}{|S|}\sum_{x\in S} f(T(x)).$$

Binary Optimization and Probabilistic model

For an arbitrary function f on $\mathcal{B}_n$, we define the B as

$$\mathcal{G}(f) = \min_{x \in \mathcal{B}_n \setminus \mathcal{X}^*} f(x) - f^*. \tag{1}$$

Proposition 1

For any $0 < \delta < 1$, suppose $L_\lambda(\theta) - f^* < (1 - \delta)\mathcal{G}(f)$, then

$$\mathbb{P}(x \in \mathcal{X}^*) > \delta.$$

Therefore, for $x^1, \dots, x^m$ independently sampled from p_θ, $\min_k f(x^k) = f^*$ with probability at least $1 - (1 - \delta)^m$.

The above proposition shows that with a optimized probabilistic model, the obtained probability from the optimal solutions is linearly dependent on the gap between the expectation and the minimum of f.

Impact of the Filter Function

- When $T(x) = x$, it means that x is a local minimum point.
- For any given $x \in \mathcal{B}_n$, there exists a corresponding local minimum point by applying the filter function T to x for many times.
- We can divide the set $\mathcal{B}_n$ into subsets with respect to the classification of local minima.

Let $X_1, X_2, ..., X_r$ be a partition of $\mathcal{B}_n$ such that for any $j \in \{1, \ldots, r\}$, every $x \in X_j$ has the same corresponding local minimum point.

Proposition 2

If there exists some $x \in \mathcal{B}_n$ *such that* $p_\theta(x) > 0$ *and* $f(x) > f(T(x))$, *then for any sufficiently small* $\lambda > 0$ *satisfying*

$$\mathbb{E}_{p_\theta}[f(x) - f(T(x))] \geq \lambda \log(\max_{1 \leq i \leq r} |X_i|),$$

it holds that

$$\mathrm{KL}\left(p_\theta \,\|\, \hat{q}_\lambda\right) \leq \mathrm{KL}\left(p_\theta \,\|\, q_\lambda\right).$$

Boundedness of $f(T(x))$

Denote $N = 2^n$ and sort all possible points in $\mathcal{B}_n = \{s_1, \ldots, s_N\}$ such that $f(s_1) \leq f(s_2) \leq \cdots \leq f(s_N)$. The bounds of $f(T(x))$ and $\mathbb{E}_{p_\theta}[f(T(x))]$, for a large probability, are not related to samples $s_{M+1}, s_{M+2}, ..., s_N$ for an integer M.

Proposition 3

Suppose that the cardinality of each neighborhood $\mathcal{N}(s_i)$ is fixed to be $|\mathcal{N}(s_i)| \geq X \geq n+1$ and all elements in $\mathcal{N}(s_i)$ except s_i are chosen uniformly at random from $\mathcal{B}_n \backslash \{s_i\}$. For $\delta \in (0,1)$, let $M = \left\lceil \frac{\log(N/\delta)}{X-1} N \right\rceil + 1$. Then, with probability at least $1-\delta$ over the choice of $T(x)$, it holds:

1) $f(T(x)) \in [f(s_1), f(s_M)], \ \forall x \in \mathcal{B}_n$;
2) $\mathbb{E}_{p_\theta}[f(T(x))] \leq \sum_{i=1}^{M-1} p_\theta(s_i) f(s_i) + (1 - \sum_{i=1}^{M-1} p_\theta(s_i)) f(s_M) \leq f(s_M)$.

Convergence of MCPG

Assumption: Let $\phi(x;\theta) = \log p_\theta(x|\mathcal{P})$. There exists some constants $M_1, M_2, M_3 > 0$ such that, for any $x \in \mathcal{B}_n$,

1. $\sup_{\theta \in \mathrm{Re}^d} |\phi(x;\theta)| \le M_1$,
2. $\sup_{\theta \in \mathrm{Re}^d} \|\nabla_\theta \phi(x;\theta)\| \le M_2$,
3. $\|\nabla_{\theta_1}\phi(x;\theta) - \nabla_{\theta_2}\phi(x;\theta)\| \le M_3 \|\theta_1 - \theta_2\|, \forall \theta_1, \theta_2, \in \mathrm{Re}^d$.

Theorem 3

Let the assumption holds and $\{\theta_t\}$ *be generated by MCPG. If the stepsize is chosen as* $\eta^t = \frac{c\sqrt{mk}}{\sqrt{t}}$ *with* $c \le \frac{1}{2l}$, *then we have*

$$\min_{1 \le t \le \tau} \mathbb{E}\left[\|\nabla_\theta L_\lambda(\theta^t)\|^2\right] \le O\left(\frac{\log \tau}{\sqrt{mk\tau}} + \frac{1}{m^2}\right).$$

Parameterization of sampling policy

- Mean field (MF) approximation:

$$p_\theta(x|\mathcal{P}) = \prod_{i=1}^{n} \mu_i^{(1+x_i)/2}(1-\mu_i)^{(1-x_i)/2}, \quad \mu_i = \phi_i(\theta;\mathcal{P})$$

- Parameterization of μ_i:

$$\mu_i = \phi_i(\theta_i) = \frac{1-2\alpha}{1+\exp(-\theta_i)} + \alpha, \quad 1 \le i \le n.$$

  The probability is scaled to the range $(\alpha, 1-\alpha)$, where $0 < \alpha < 0.5$ is given.

- For problems graph structures, combining advanced neural networks such as GNN can also be a good choice.

Maxcut

- We use the results reported by BLS as benchmark. Denoting UB as the results achieved by BLS and obj as the cut size, the gap reported is defined as follows:

$$\mathrm{gap} = \frac{\mathrm{UB} - \mathrm{obj}}{\mathrm{UB}} \times 100\%.$$

Graph	Nodes	Edges	BLS	MCPG	DSDP	RUN-CSP	PI-GNN	EO	EMADM
G14	800	4,694	**3,064**	**3,064**	2,922	2,943	3,026	3047	3045
G15	800	4,661	**3,050**	**3,050**	2,938	2,928	2,990	3028	3034
G22	2,000	19,990	**13,359**	**13,359**	12,960	13,028	13,181	13215	13297
G49	3,000	6,000	**6,000**	**6,000**	**6,000**	**6,000**	5,918	**6000**	**6000**
G50	3,000	6,000	**5,880**	**5,880**	**5,880**	**5,880**	5,820	5878	5870
G55	5,000	12,468	10,294	**10,296**	9,960	10,116	10,138	10107	10208
G70	10,000	9,999	9,541	**9595**	9,456	-	9,421	8513	9557

Table: Computational results on selected Gset instances. The result is sourced from references.

Outline

1. Introduction
2. Machine learning for binary optimization
3. Machine learning for general MILPs
4. Machine learning for routing problems

Mixed integer linear program

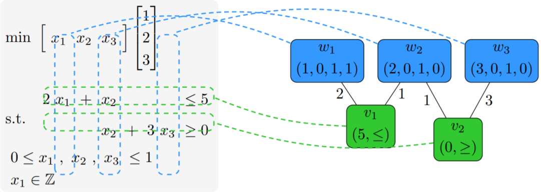
- Mixed-Integer Linear Programs (MILPs) are utilized to solve a myriad of decision-making problems across various practical applications.

$$\begin{aligned} \min \quad & c^{\mathrm{T}} x, \\ \text{s.t.} \quad & Ax \leq b, \\ & l \leq x \leq u, \\ & x \in \mathbb{R}^{n-p} \times \mathbb{Z}^{p}. \end{aligned}$$

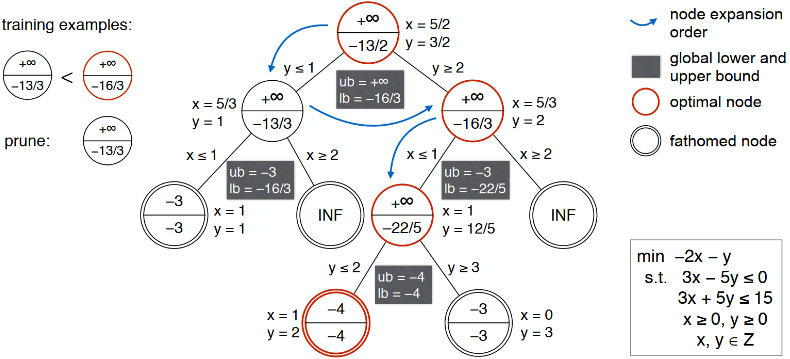
- **Feasibility**: The feasible region is discrete and non-convex, which makes it difficult to analyze and optimization methods hard to design.
- **Complexity**: Even with a relatively small number of variables, the solution space can be exponentially vast due to the integer constraints.
- **Algorithmic strategy**: Preprocessing, Branching Strategies, Bounding Strategies, Cut Generation, Heuristic ...

Graph Representation of MILP Instances

- An MILP instance can be represented as a bipartite graph $G = (V \cup W, E)$:
  - **Variable nodes** $w_j \in W$: correspond to variables x_j, each with features:
    - Type of variable (e.g., binary, integer, continuous);
    - Objective coefficient c_j;
    - Bounds $[l_j, u_j]$.
  - **Constraint nodes** $v_i \in V$: represent constraints δ_i, each with features:
    - Constraint type ($\leq$, $=$, or $\geq$);
    - Right-hand side value b_i.
  - **Edges** $(v_i, w_j) \in E$: exist if x_j appears in constraint δ_i, with weight a_{ij}.



Branch and bound



$$
\begin{array}{rl}
\min & -2x - y \\
\text{s.t.} & 3x - 5y \le 0 \\
 & 3x + 5y \le 15 \\
 & x \ge 0,\ y \ge 0 \\
 & x, y \in Z
\end{array}
$$

Learning the exact methods

- **Branching Variable Selection:**
  - Branch variable selection determines which fractional variables (also known as candidates) to branch the current node into two child nodes.
  - Nair et al.(2021) encode MIP to the GCN as a bipartite graph and compute an initial feasible solution (Neural Diving), then train a GCN to imitate ADMM-based policy for branching (Neural Branching).
- **Node Selection:**
  - The branch-and-bound algorithm recursively divides the feasible set of a problem into disjoint subsets, organized in a tree structure.
  - He et al.(2014) uses imitation learning to train a node selection and a node pruning policy to speed up the tree search in the B&B process.
- **Cutting Plane:**
  - Cuts serve as the purpose of reducing the LP solution space, which might lead to a smaller tree in the branch-and-cut algorithm.
  - Tang et al. (2020) train a RL agent for sequentially selecting cutting planes.

Outline

1. Introduction
2. Machine learning for binary optimization
3. Machine learning for general MILPs
4. Machine learning for routing problems

Routing problems

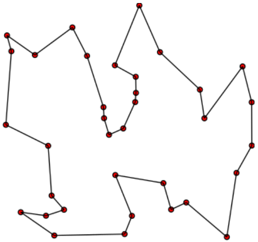
- Travelling Salesman Problem (TSP)
  - Given a fully connected graph with node coordinates $\{x_i\}_{i=1}^{n}$, the goal is to find a tour that visits each node exactly once and returns to the starting point, while minimizing the total travel distance.
  - Permutaion formulation

$$\min_{\pi} L(\pi) := \sum_{i=1}^{n-1} \|x_{\pi(i+1)} - x_{\pi(i)}\| + \|x_{\pi(1)} - x_{\pi(n)}\|.$$

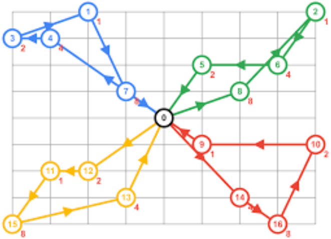
- Capacitated Vehicle Routing Problem (CVRP)
  There are n customers, each with a demand δ_i, to be served by a fleet of identical vehicles with capacity D, all starting and ending at a common depot. The objective is to find the shortest possible set of routes such that every customer is visited exactly once, and the total demand on each route does not exceed the vehicle capacity.

Example tours

- Travelling Salesman Problem (TSP)



- Vehicle Routing Problem (VRP)



- NP-hard combinatorial problem with a wide range of applications!

Overview: machine learning for routing problems

- **Learning to construct**: iteratively add nodes to the partial solution.
  - Pointer Network was first proposed by Vinyals et al. based on Recurrent Neural Networks and supervised learning.
  - The Graph Neural Networks were then leveraged for graph embedding (Dai et al.) and faster encoding (Drori et al.) under reinforcement learning framework.
  - Later, the Attention Model (AM) was proposed by Kool et al.
  - Policy Optimization with Multiple Optima (POMO) significantly improved AM with diverse rollouts and data augmentations (Kwon et al.).
  - Efficient Active Search (EAS) helps to get out of local optima by updating a small subset of pre-trained model parameters on each test instance (Hottung et al.), which could be further boosted if coupled with Simulation Guided Beam Search (SGBS) by Choo et al., achieving better generalization performance.
  - Light Encoder and Heavy Decoder (LEHD) model is proposed by Luo et al. with stronger generalization to large-scale instances sizes.

Overview: machine learning for routing problems

- **Learning to search**: iteratively refine a solution to a new one — a search process.
  - NeuRewriter (Chen et al.) and L2I (Lu et al.) relied heavily on traditional local search algorithms with long run time.
  - Hottung and Tierney proposed the Neural large neighborhood search (NLNS) solver improving upon them by controlling a ruin-and-repair process using a deep model.
  - Several L2S solvers focused on controlling k-opt heuristic within RL training: self-attention-based policy (Wu et al.), Dual-Aspect Collaborative Attention (Ma et al.), Synthesis Attention (Ma et al.) , GNN+RNN-based policy (Costa et al.).
- **Learning to predict**: guide the search by predicting critical information.
  - Joshi et al. proposed using GNN models to predict heatmaps that indicate probabilities of the presence of an edge, which then uses beam search to solve TSP.
  - The GLS solver (Hudson et al.) used GNN to guide the local search heuristics.
  - The DIFUSCO solver (Sun et al.) proposed to replace those GNN models with diffusion models in generating heatmaps.

Comparison

- The L2C solvers can produce high-quality solutions within seconds using greedy rollouts; however, they are shown to get trapped in local optima, even when equipped with post-hoc methods, such as sampling, beam search, etc.
- Although L2S solvers strive to surpass L2C solvers by directly learning to search, they are still inferior to those state-of-the-art L2C solvers even when given prolonged run time.
- Compared to L2C or L2S solvers, L2P solvers exhibit better scalability for large instances; however, L2P solvers are mostly limited to supervised learning and TSP only, due to challenges in preparing training data and the ineffectiveness of heatmaps in handling VRP constraints.

Construct a path

- A solution $\pi = (\pi_1, \ldots, \pi_n)$ is a permutation of the nodes $\{1, \ldots, n\}$.
- Given a problem instance s, the stochastic policy for selecting a solution π is parameterized by θ as

$$p_\theta(\pi|s) = \prod_{t=1}^{n} p_\theta(\pi_t|s, \pi_{1:t-1}).$$

- The encoder produces embeddings of all input nodes, where an instance s is encoded by features x_i on each node i.
- The decoder produces the sequence π of input nodes, one nodes at a time, which takes as input the encoder embeddings and a problem specific mask and context.

Multi-head attention mechanism

- The multi-head attention mechanism starts by linearly projecting input sequences Q, K, V into H distinct subspaces using learned projection matrices W_j^Q, W_j^K, W_j^V:

$$Q_j = QW_j^Q, \quad K_j = KW_j^K, \quad V_j = VW_j^V, \quad j = 1, \ldots, H.$$

- Attention weights are obtained via a scaled dot-product between projected queries and keys, followed by a softmax operation:

$$A_j = \text{Softmax}\left(\frac{Q_j K_j^{\mathrm{T}}}{\sqrt{d_k}} + M\right), \quad j = 1, \ldots, H,$$

  where d_k represents the dimension of the keys and M is an optional attention mask that can be used to prevent attending to certain positions.

Multi-head attention mechanism

- Using these attention weights, the mechanism computes a weighted sum of the projected values, yielding the output of each attention head:

$$Z_j = A_j V_j, \quad j = 1, 2, \ldots, H.$$

- Finally, the outputs from all attention heads are concatenated and linearly projected using a learned output matrix W^O, forming the final multi-head attention output:

$$\mathrm{MHA}(Q, K, V; M) = \mathrm{Concat}(Z_1, \ldots, Z_H) W^O.$$

Encoder

- The encoder computes the initial embeddings $h_i^{(0)} \in \mathbb{R}^{d_h}$ from node features x_i using a linear transformation:

$$h_i^{(0)} = W^{(0)} x_i + b^{(0)}, \quad i = 1, \ldots, n.$$

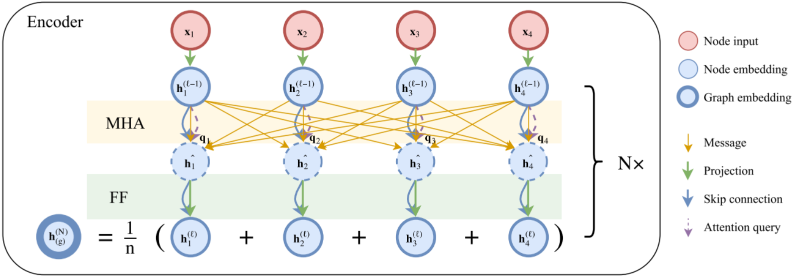
- Stacking these embeddings forms $\boldsymbol{h}^{(0)} \in \mathbb{R}^{n \times d_h}$. The encoder then refines them through L attention layers, each consisting of a multi-head attention (MHA) layer and a node-wise fully connected feed-forward (FF) layer:

$$\begin{aligned} \hat{\boldsymbol{h}}^{(\ell)} &= \mathrm{BN}^{l}\left(\boldsymbol{h}^{(\ell-1)} + \mathrm{MHA}^{(\ell)}\left(\boldsymbol{h}^{(\ell-1)}, \boldsymbol{h}^{(\ell-1)}, \boldsymbol{h}^{(\ell-1)}\right)\right), \\ \boldsymbol{h}^{(\ell)} &= \mathrm{BN}^{\ell}\left(\hat{\boldsymbol{h}}^{(\ell)} + \mathrm{FF}^{(\ell)}\left(\hat{\boldsymbol{h}}^{(\ell)}\right)\right). \end{aligned}$$

- The graph-level representation is the mean of the final node embeddings:

$$\bar{h}^{(L)} = \frac{1}{n} \sum_{i=1}^{n} h_i^{(L)}.$$

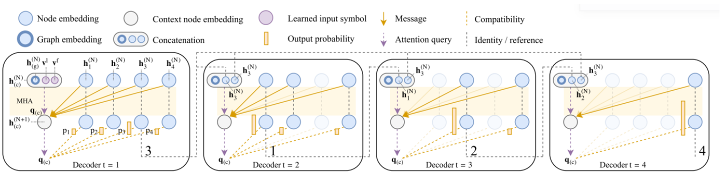
Encoder



Encoder
x_1
x_2
x_3
x_4
$h_1^{(\ell-1)}$
$h_2^{(\ell-1)}$
$h_3^{(\ell-1)}$
$h_4^{(\ell-1)}$
MHA
q_1
q_2
q_3
q_4
$\hat{h}_1$
$\hat{h}_2$
$\hat{h}_3$
$\hat{h}_4$
FF
$h_{(g)}^{(N)} = \frac{1}{n}\left(h_1^{(\ell)} + h_2^{(\ell)} + h_3^{(\ell)} + h_4^{(\ell)}\right)$
N×
Node input
Node embedding
Graph embedding
Message
Projection
Skip connection
Attention query

Decoder

- During decoding, the graph is augmented with a special context node (c) to represent the decoding context.
- The decoder computes an attention (sub)layer on top of the encoder, but with messages only to the context node for efficiency.
- The final probabilities are computed using a single-head attention mechanism.



Context embedding

- The context vector of the decoder at time t consists of the embedding of the graph $\bar{h}^{(L)}$, the previous (last) node π_{t-1} and the first node π_1:

$$h_{(c)} = \begin{cases} [\bar{h}^{(L)}, h^{(L)}_{\pi_{t-1}}, h^{(L)}_{\pi_1}], & t > 1, \\ [\bar{h}^{(L)}, v^1, v^2] & t = 1. \end{cases}$$

- The context embedding $h'_{(c)}$ is computed using a single masked cross-attention layer, where the context vector serves as the query, while the node embeddings provide the keys and values:

$$h'_{(c)} = \mathrm{MHA}(h_{(c)}, \boldsymbol{h}, \boldsymbol{h}; M_t).$$

  The mask vector M_t encodes node availability at time t, with $M_t(i) = 0$ for unvisited nodes and $M_t(i) = -\infty$ for visited nodes.

Calculation of probabilities

- The logits are obtained by a single attention head:

$$z = \frac{(\boldsymbol{h}^{(L)} W^K) h'_{(c)}}{\sqrt{d_k}},$$

  where the matrix $\boldsymbol{h}^{(L)} W^K$ is precomputed only once as cache during the overall decoding process.

- The conditional probability distribution over available nodes is computed using a softmax:

$$p_\theta(\cdot \mid s, \pi_{1:t-1}) = \text{Softmax}\left(C \cdot \tanh(z) + M_t\right),$$

  where the tanh clipping constant $C > 0$ serves in improving the exploration.

Attention model for the CVRP

- **Encoder**: Let $\hat{\delta}_i$ be the normalized demand of the node i.

$$h_i^{(0)} = \begin{cases} W_0^{(0)} x_i + b_0^0, & i = 0, \\ W^{(0)}[x_i, \hat{\delta}_i], & i = 1, \dots, n. \end{cases}$$

- **Capacity constraints**: Keep track of the remaining demands $\hat{\delta}_{i,t}$ for the nodes $i \in \{1, \dots, n\}$ and remaining vehicle capacity $\hat{D}_t$ at time t. At $t = 1$, these are initialized as $\hat{\delta}_{i,t} = \hat{\delta}_i$ and $\hat{D}_t = 1$.

$$\hat{\delta}_{i,t+1} = \begin{cases} \max(0, \hat{\delta}_{i,t} - \hat{D}_t), & \pi_t = i, \\ \hat{\delta}_{i,t}, & \pi_t \neq i. \end{cases} \qquad \hat{D}_{t+1} = \begin{cases} \max(0, \hat{D}_t - \hat{\delta}_{\pi_t,t}), & \pi_t \neq 0, \\ 1, & \pi_t = 0. \end{cases}$$

Attention model for the CVRP

- **Decoder context**: The context for the decoder for the VRP at time t is the current/last location π_{t-1} and the remaining capacity $\hat{D}_t$.

$$h_{(c)} = \begin{cases} [\bar{h}^{(L)}, h^{(L)}_{\pi_{t-1}}, \hat{D}_t], & t > 1, \\ [\bar{h}^{(L)}, h^{(L)}_{0}, \hat{D}_t] & t = 1. \end{cases}$$

- **Masking**: In the decoder layers, the masking rules are defined as follows: for the depot node 0, it is masked (i.e., $M_t(0) = -\infty$) if and only if the current step $t = 1$ or the previous node π_{t-1} is the depot itself. For any customer node $j \neq 0$, it is masked (i.e., $M_t(j) = -\infty$) if it has been visited ($\hat{\delta}_{i,t} = 0$) or its demand exceeds the remaining capacity ($\hat{\delta}_{i,t} > \hat{D}_t$).

Reinforcement learning

- **Loss function**:

$$\mathcal{L}(\theta|s) = \mathbb{E}_{p_\theta(\pi|s)}[L(\pi)],$$

where $L(\pi)$ is the tour length for TSP.

- **Policy gradient**:

$$\nabla_\theta \mathcal{L}(\theta|s) = \mathbb{E}_{p_\theta(\pi|s)}[(L(\pi) - b(s))\nabla_\theta \log p_\theta(\pi|s)].$$

- **Rollout baseline**:

$$b(s) = L(\pi^{\mathrm{BL}}),$$

where π^{BL} is a solution from a deterministic greedy rollout of the policy p_θ.

- **Optimizer**: Adam.

Policy Optimization with Multiple Optima

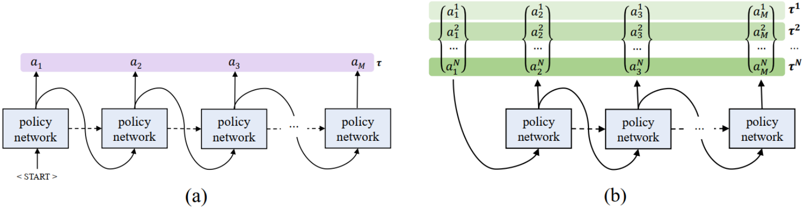
- Symmetry in solving CO problems leads to multiple optima.
- A routing problem contains a loop rather than a sequence, where $(\pi_1, \pi_2, \pi_3, \pi_4)$ is the same as $(\pi_2, \pi_3, \pi_4, \pi_1)$.
- Let a solution trajectory denoted by $\pi = (\pi_1, \ldots, \pi_n)$ and the policy

$$p_\theta(\pi|s) = \prod_{t=1}^{n} p_\theta(\pi_t|s, \pi_{1:t-1}).$$

- In the above equation, the starting nodes π_1 heavily influences the rest of the sequence $(\pi_2, \ldots, \pi_n)$, when in fact any choice for π_1 should be equally good.

Explorations from multiple starting nodes

- Designate N different nodes $\{\pi_1^1, \dots, \pi_1^N\}$ as starting points for exploration.
- Sample N different solution trajectories $\pi^1, \dots, \pi^N$ from the policy.
- Apply entropy maximization techniques to improve exploration of the first moves.



Policy gradient with a shared baseline

- A set of solution trajectories $\pi^1, \dots, \pi^N$ is sampled from the policy $p_\theta(\pi|s)$.
- The policy gradient is approximated by

$$\hat{\nabla}_\theta \mathcal{L}(\theta|s) = \frac{1}{N}\sum_{i=1}^{N}\left(L(\pi^i) - b(s)\right)\nabla_\theta \log p_\theta(\pi^i|s),$$

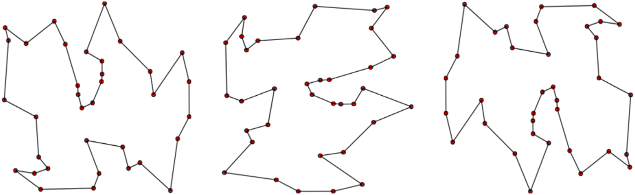
  where $p_\theta(\pi^i|s) = \prod_{t=2}^{n} p_\theta(\pi_t^i|s, \pi_{1:t-1}^i)$.
- The shared baseline is taken as the approximation of $\mathbb{E}_{p_\theta(\pi|s)}[L(\pi)]$,

$$b(s) = \frac{1}{N}\sum_{j=1}^{N} L(\pi^j).$$

- The shared baseline makes RL training highly resistant to local minima.

Instance augmentation

- **Drawback**: N, the number of greedy rollouts one can utilize, cannot be arbitrarily large, as it is limited to a finite number of possible starting nodes.
- **Reformulate the problem**: meet a different problem but arrive at the same solution.
- One can flip or rotate the coordinates of all the nodes in a 2D routing problem and generate another instance, from which more greedy trajectories can be acquired.



0° rotation 90° rotation 180° rotation

Multi-Task vehicle routing problems

- Prevailing neural solvers still need network structures tailored and trained independently for each specific VRP.
- Several VRP variants involve additional practical constraints:
  - **Open route (O):** The vehicle does not need to return to the depot after visiting customers.
  - **Backhaul (B):** We name the customer nodes with $\delta_i > 0$ as linehauls and the ones with $\delta_i < 0$ as backhauls. VRP with backhaul allows the vehicle traverses linehauls and backhauls in a mixed manner, without strict precedence between them.
  - **Duration Limit (L):** To maintain a reasonable workload, the cost (i.e., length) of each route is upper bounded by a predefined threshold.
  - **Time Window (TW):** Each node $v_i \in V$ is associated with a time window $[e_i, l_i]$ and a service time s_i. A vehicle must start serving customer v_i in the time slot from e_i to l_i.

Mixture of Experts

- An MoE layer consists of
    1. m experts $\{E_1, E_2, \ldots, E_m\}$, each of which is a linear layer or FFN with independent trainable parameters.
    2. A gating network G parameterized by W_G, which decides how the inputs are distributed to experts.

$$\text{MoE}(x) = \sum_{j=1}^{m} G(x)_j E_j(x).$$

- A sparse vector $G(x)$ only activates a small subset of experts with partial model parameters, and hence saves the computation.
- A TopK operator can achieve such sparsity by only keeping the K-largest values while setting others as the negative infinity.

$$G(x) = \text{Softmax}(\text{TopK}(x \cdot W_G)).$$

MVMoE

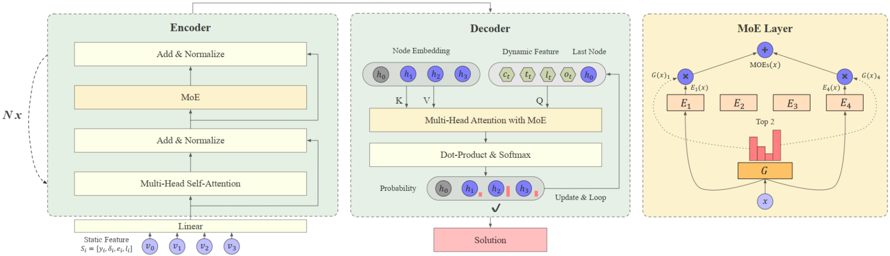
- It jointly optimizes all trainable parameters θ, with the objective formulated as follows

$$\min_{\theta} \mathcal{L} = \mathcal{L}_a + \alpha \mathcal{L}_b.$$

- $\mathcal{L}_a = \mathbb{E}_{\pi \sim p_\theta}\left[L(\pi)\right]$ denotes the original loss function of the VRP solver.
- $\mathcal{L}_b$ denotes the the auxiliary loss used to ensure load-balancing in MoEs.

$$I(X) = \sum_{x \in X} G(x),$$
$$D(X)_j = \sum_{x \in X} \Phi\left(\frac{(x \cdot W_G) - \phi(H'_x, k, j)}{\text{Softplus}((x \cdot W_{noise})_j)}\right),$$
$$\mathcal{L}_b = \text{Var}(I(X))^2 + \text{Var}(D(X))^2.$$

MVMoE



- Despite MVMoE presents the first attempt towards a large VRP model, the scale of parameters is still far less than LLMs.

Failure in TSPTW

- The success of the masking mechanism in routing problems relies on
  - the feasibility of the entire solution can be properly decomposed into the feasibility of each node selection step;
  - ground truth masks are easily obtainable for each step.
- However, such assumptions may fail in some routing problems, such as travelling salesman problem with time windows (TSPTW).
- Once a node is selected, the decision becomes irreversible, potentially leading to infeasible situations after several steps.

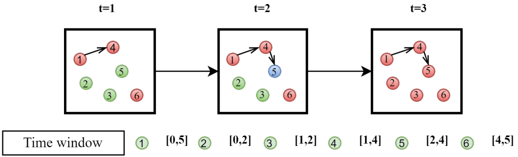
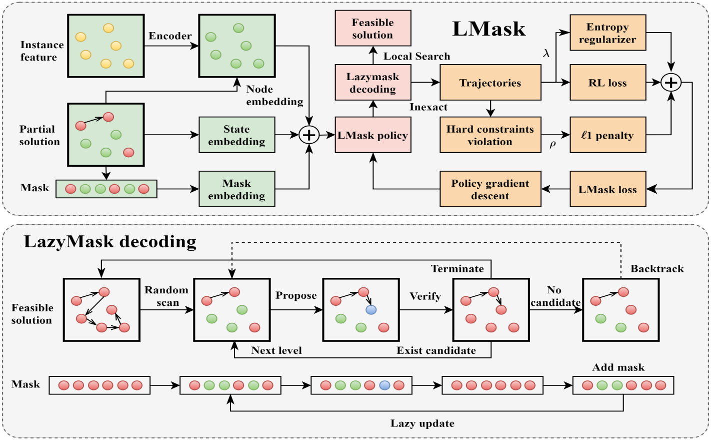


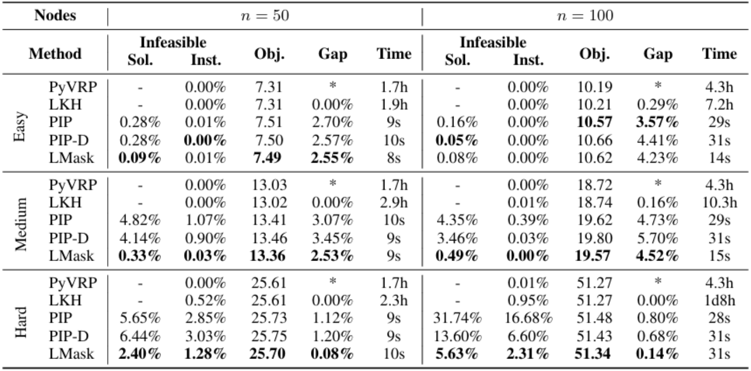
Figure: No node can be selected to satisfy the time windows.

LMask


LMask
Instance feature
Encoder
Node embedding
Partial solution
State embedding
Mask
Mask embedding
Feasible solution
Local Search
Lazymask decoding
LMask policy
Inexact
Trajectories
Hard constraints violation
Entropy regularizer
RL loss
ℓ1 penalty
λ
ρ
Policy gradient descent
LMask loss
LazyMask decoding
Feasible solution
Random scan
Propose
Verify
No candidate
Terminate
Backtrack
Next level
Exist candidate
Mask
Add mask
Lazy update

Numerical results

Table 1: Results on synthetic TSPTW datasets.



	Nodes	$n = 50$					$n = 100$				
	Method	**Infeasible Sol.**	**Inst.**	**Obj.**	**Gap**	**Time**	**Infeasible Sol.**	**Inst.**	**Obj.**	**Gap**	**Time**
Easy	PyVRP	-	0.00%	7.31	*	1.7h	-	0.00%	10.19	*	4.3h
	LKH	-	0.00%	7.31	0.00%	1.9h	-	0.00%	10.21	0.29%	7.2h
	PIP	0.28%	0.01%	7.51	2.70%	9s	0.16%	0.00%	**10.57**	**3.57%**	29s
	PIP-D	0.28%	**0.00%**	7.50	2.57%	10s	**0.05%**	0.00%	10.66	4.41%	31s
	LMask	**0.09%**	0.01%	**7.49**	**2.55%**	8s	0.08%	0.00%	10.62	4.23%	14s
Medium	PyVRP	-	0.00%	13.03	*	1.7h	-	0.00%	18.72	*	4.3h
	LKH	-	0.00%	13.02	0.00%	2.9h	-	0.01%	18.74	0.16%	10.3h
	PIP	4.82%	1.07%	13.41	3.07%	10s	4.35%	0.39%	19.62	4.73%	29s
	PIP-D	4.14%	0.90%	13.46	3.45%	9s	3.46%	0.03%	19.80	5.70%	31s
	LMask	**0.33%**	**0.03%**	**13.36**	**2.53%**	9s	**0.49%**	**0.00%**	**19.57**	**4.52%**	15s
Hard	PyVRP	-	0.00%	25.61	*	1.7h	-	0.01%	51.27	*	4.3h
	LKH	-	0.52%	25.61	0.00%	2.3h	-	0.95%	51.27	0.00%	1d8h
	PIP	5.65%	2.85%	25.73	1.12%	9s	31.74%	16.68%	51.48	0.80%	28s
	PIP-D	6.44%	3.03%	25.75	1.20%	9s	13.60%	6.60%	51.43	0.68%	31s
	LMask	**2.40%**	**1.28%**	**25.70**	**0.08%**	10s	**5.63%**	**2.31%**	**51.34**	**0.14%**	31s

Submodular Function Optimization

`http://bicmr.pku.edu.cn/~wenzw/bigdata2024.html`

Acknowledgement: this slides is based on Prof. Andreas Krause's, Prof. Jeff Bilmes, Prof. Francis Bach and Prof. Shaddin Dughmi lecture notes

Outline

1. What is submodularity?

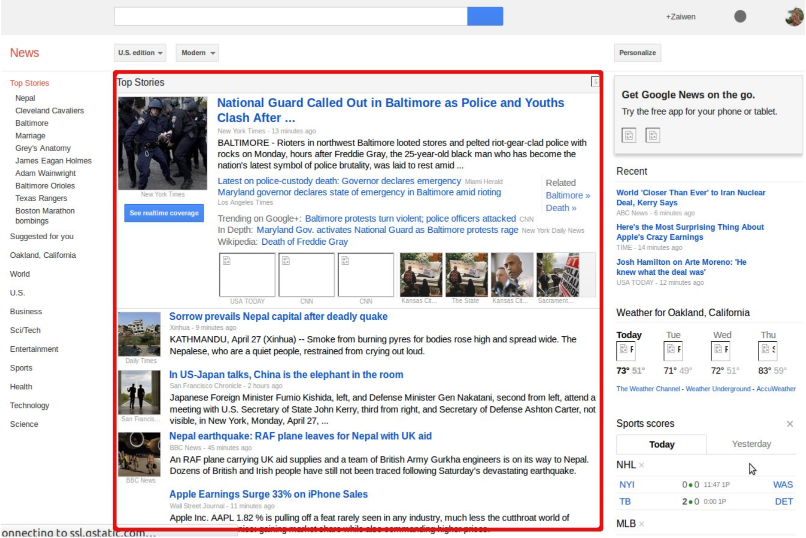
2. Submodular maximization

3. Submodular minimization

Interactive recommendation

- Number of recommendations k to choose from large data.
  - Similar articles → similar click-through rates!
- Performance depends on query / context.
  - Similar users → similar click-through rates!
- Need to compile sets of k recommendations(instead of only one).
  - Similar sets→ similar click-through rates!

News recommendation



+Zaiwen

News

U.S. edition

Modern

Personalize

Top Stories
- Nepal
- Cleveland Cavaliers
- Baltimore
- Marriage
- Grey's Anatomy
- James Eagan Holmes
- Adam Wainwright
- Baltimore Orioles
- Texas Rangers
- Boston Marathon bombings

Suggested for you

Oakland, California

World

U.S.

Business

Sci/Tech

Entertainment

Sports

Health

Technology

Science

Top Stories

National Guard Called Out in Baltimore as Police and Youths Clash After ...

New York Times - 13 minutes ago

BALTIMORE - Rioters in northwest Baltimore looted stores and pelted riot-gear-clad police with rocks on Monday, hours after Freddie Gray, the 25-year-old black man who has become the nation's latest symbol of police brutality, was laid to rest amid ...

New York Times

See realtime coverage

Latest on police-custody death: Governor declares emergency Miami Herald

Maryland governor declares state of emergency in Baltimore amid rioting Los Angeles Times

Related
Baltimore »
Death »

Trending on Google+: Baltimore protests turn violent; police officers attacked CNN

In Depth: Maryland Gov. activates National Guard as Baltimore protests rage New York Daily News

Wikipedia: Death of Freddie Gray

USA TODAY · CNN · CNN · Kansas Cit... · The State · Kansas Cit... · Sacrament...

Sorrow prevails Nepal capital after deadly quake

Xinhua - 9 minutes ago

KATHMANDU, April 27 (Xinhua) -- Smoke from burning pyres for bodies rose high and spread wide. The Nepalese, who are a quiet people, restrained from crying out loud.

Daily Times

In US-Japan talks, China is the elephant in the room

San Francisco Chronicle - 2 hours ago

Japanese Foreign Minister Fumio Kishida, left, and Defense Minister Gen Nakatani, second from left, attend a meeting with U.S. Secretary of State John Kerry, third from right, and Secretary of Defense Ashton Carter, not visible, in New York, Monday, April 27, ...

San Francis...

Nepal earthquake: RAF plane leaves for Nepal with UK aid

BBC News - 45 minutes ago

An RAF plane carrying UK aid supplies and a team of British Army Gurkha engineers is on its way to Nepal. Dozens of British and Irish people have still not been traced following Saturday's devastating earthquake.

BBC News

Apple Earnings Surge 33% on iPhone Sales

Wall Street Journal - 11 minutes ago

Apple Inc. AAPL 1.82 % is pulling off a feat rarely seen in any industry, much less the cutthroat world of

onnecting to ssl.gstatic.com...

Get Google News on the go.

Try the free app for your phone or tablet.

Recent

World 'Closer Than Ever' to Iran Nuclear Deal, Kerry Says

ABC News - 6 minutes ago

Here's the Most Surprising Thing About Apple's Crazy Earnings

TIME - 14 minutes ago

Josh Hamilton on Arte Moreno: 'He knew what the deal was'

USA TODAY - 12 minutes ago

Weather for Oakland, California

Today	Tue	Wed	Thu
73° 51°	71° 49°	72° 51°	83° 59°

The Weather Channel - Weather Underground - AccuWeather

Sports scores

Today · Yesterday

NHL

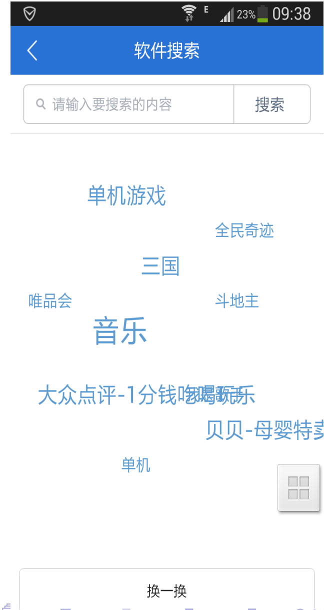
NYI	0 ● 0	11:47 1P	WAS
TB	2 ● 0	0:00 1P	DET

MLB

Which set of articles satisfies most users?

Relevance vs. Diversity

- Users may have different interests / queries may be ambiguous.
  - E.g., "jaguar", "squash",$\cdots$.
- Want to choose a set that is relevant to as many users as possible.
  - Users may choose from the set the article they're most interested in.
- Want to optimize both relevance and diversity.



Simple abstract model

- Given a set W of users and a collection V of articles/ads.
- Each article i is relevant to a set of users S_i.
  - For now suppose this is known!
- For each set A of articles, define

$$F(A) = | \cup_{i \in A} S_i |.$$

- Want to select k articles from V to maximize "users covered"

$$\max_{A \subseteq V, |A| < k} F(A).$$

- Number of sets A grows exponential in k!
- Finding optimal A is NP-hard.

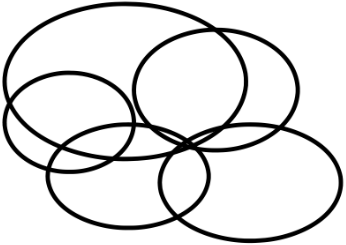
Maximum coverage

- **Given:** Collection V of sets, utility function $F(.)$.

Want: $A^* \subseteq V$ such that

$$\mathcal{A}^* = \text{argmax}_{|\mathcal{A}| \leq k} F(\mathcal{A})$$

NP-hard!



Set Functions

- Ground set $X := \{x_1, x_2, ..., x_n\}$ is the domain of interest or the universe of elements.
  - In sensor network, the ground set might consist of all possible locations where sensors could be placed.
- The solution space $V := 2^X = \{A \mid A \subseteq X\}$.
- A set function takes as input a set, and outputs a real number.
  - Inputs are some subsets of ground set X.
  - $F : 2^X \to \mathbb{R}$.
- It is common in the literature to use either X or V as the ground set.
- We will follow this inconsistency in the literature and will inconsistently use either X or V as our ground set (hopefully not in the same equation, if so, please point this out).

Modular Functions

- If F is a modular function, then for any $A, B \subseteq X$, we have

$$F(A) + F(B) = F(A \cap B) + F(A \cup B).$$

- If F is a modular function, it may be written as

$$F(A) = F(\emptyset) + \sum_{a \in A} \left(F(\{a\}) - F(\emptyset) \right).$$

- Modular set functions
  - Associate a weight w_i with each $i \in X$, and set $F(S) = \sum_{i \in S} w_i$.
  - Discrete analogue of linear functions.
- Other possibly useful properties a set function may have:
  - Monotone: if $A \subseteq B \subseteq X$, then $F(A) \leq F(B)$.
  - Nonnegative: $F(S) \geq 0$ for all $S \subseteq X$.
  - Normalized: $F(\emptyset) = 0$.

Submodular Functions

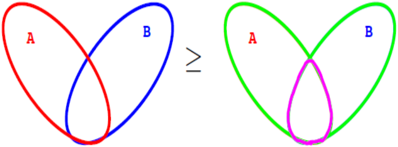
Definition 1

A set function $F : 2^X \to \mathbb{R}$ is submodular if and only if

$$F(A) + F(B) \geq F(A \cap B) + F(A \cup B)$$

for all $A, B \subseteq X$.

- "Uncrossing" two sets reduces their total function value.



Definition

A set function $F : 2^X \to \mathbb{R}$ is supmodular if and only if $-F$ is submodular.

Submodular Functions

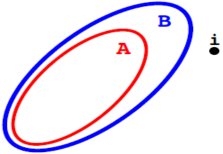
Definition 2 (diminishing returns)

A set function $F : 2^X \to \mathbb{R}$ is submodular if and only if

$$\underbrace{F(B \cup \{s\}) - F(B)}_{\text{Gain of adding an element } s \text{ to a large set}} \leq \underbrace{F(A \cup \{s\}) - F(A)}_{\text{Gain of adding an element } s \text{ to a small set}}$$

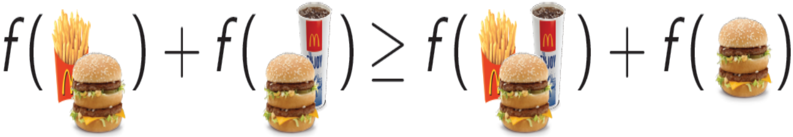
for all $A \subseteq B \subseteq X$ and $s \in X \setminus B$.

- The marginal value of the added element exhibits “diminishing marginal returns”.
- This means that the incremental “value”, “gain”, or “cost” of s decreases (diminishes) as the context in which s is considered grows from A to B.

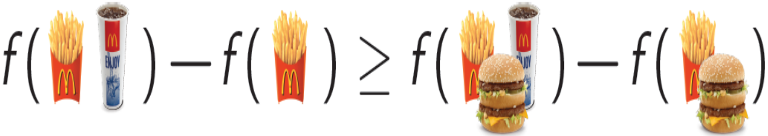


Submodular: Consumer Costs of Living

- Consumer costs are very often submodular.
    - For example:



    - When seen as diminishing returns:



Submodular Functions

Definition 3 (group diminishing returns)

A set function $F : 2^X \to \mathbb{R}$ is submodular if and only if

$$F(B \cup C) - F(B) \leq F(A \cup C) - F(A)$$

for all $A \subseteq B \subseteq X$ and $C \subseteq X \backslash B$.

- This means that the incremental “value”, “gain”, or “cost” of set C decreases (diminishes) as the context in which C is considered grows from A to B.

Equivalence of Definitions

Definition 2 $\Longrightarrow$ Definition 3

Let $C = \{c_1, \ldots, c_k\}$. The Definition 2 implies

$$\begin{aligned}
& F(A \cup C) - F(A) \\
= \; & F(A \cup C) - \sum_{i=1}^{k-1} (F(A \cup \{c_1, \ldots, c_i\}) - F(A \cup \{c_1, \ldots, c_i\})) - F(A) \\
= \; & \sum_{i=1}^{k} (F(A \cup \{c_1, \ldots, c_i\}) - F(A \cup \{c_1, \ldots, c_{i-1}\})) \\
\geq \; & \sum_{i=1}^{k} (F(B \cup \{c_1, \ldots, c_i\}) - F(B \cup \{c_1, \ldots, c_{i-1}\})) \\
= \; & F(B \cup C) - F(B)
\end{aligned}$$

Equivalence of Definitions

Definition 1 $\Longrightarrow$ Definition 2
Let $A' = A \cup \{s\}$, $B' = B$, from Definition 1, we have

$$\begin{aligned} F(A \cup \{s\}) + F(B) &= F(A') + F(B') \\ &\geq F(A' \cap B') + F(A' \cup B') \\ &= F(A) + F(B \cup \{s\}) \end{aligned}$$

Definition 2 $\Longrightarrow$ Definition 1
Assume $A \neq B$. Define $A' = A \cap B$, $C = A \backslash B$ and $B' = B$. Then

$$\begin{aligned} & F(A' \cup C) - F(A') \geq F(B' \cup C) - F(B') \\ \Longleftrightarrow\quad & F((A \cap B) \cup (A \backslash B)) + F(B) \geq F(B \cup (A \backslash B)) + F(A') \\ \Longleftrightarrow\quad & F(A) + F(B) \geq F(A \cup B) + F(A \cap B) \end{aligned}$$

Submodularity

- Submodular functions have a long history in economics, game theory, combinatorial optimization, electrical networks, and operations research.
- They are gaining importance in machine learning as well.
- Arbitrary set functions are hopelessly difficult to optimize, while the minimum of submodular functions can be found in polynomial time, and the maximum can be constant-factor approximated in low-order polynomial time.
- Submodular functions share properties in common with both convex and concave functions.

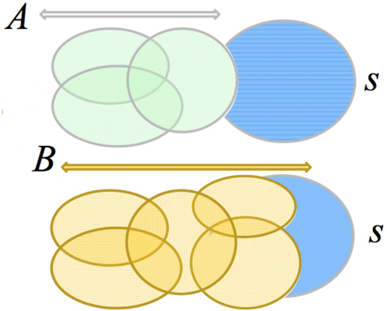
Example: Set cover

- F is submodular: $A \subseteq B$

$$\underbrace{F(A \cup \{s\}) - F(A)}_{\text{Gain of adding an element } s \text{ to a small set}} \geq \underbrace{F(B \cup \{s\}) - F(B)}_{\text{Gain of adding an element } s \text{ to a large set}}$$

- Natural example:
  - Set $S_1, S_2, \cdots, S_n$
  - F(A)=size of union of S_i (e.g., number of satisfied users)

$$F(A) = |\cup_{i \in A} S_i|$$



Closedness properties

- $F_1, \cdots, F_m$ are submodular functions on V and $\lambda_1, \cdots, \lambda_m \geq 0$.
- Then: $F(A) = \sum_i \lambda_i F_i(A)$ is submodular!

- Submodularity closed under nonnegative linear combinations
- Extremely useful fact:
  - $F_\theta(A)$ is submodular $\Rightarrow \sum_\theta P(\theta) F_\theta(A)$ is submodular!
  - Multi-objective optimization:
    $F_1, \cdots, F_m$ are submodular, $\lambda_i > 0 \Rightarrow \sum_i \lambda_i F_i(A)$ is submodular.

Probabilistic set cover

- Document coverage function:
  cover$_d(c)$=probability document d covers concept c, e.g., how strongly d covers c.
  It can model how relevant is concept c for user u.
- Set coverage function:

$$\text{cover}_A(c) = 1 - \Pi_{d \in A}(1 - \text{cover}_d(c)).$$

  Probability that at least one document in A covers c.
- Objective:

$$\max_{|A| \leq k} \quad F(A) = \sum_c w_c.\text{cover}_A(c)$$

  w_c is the concept weights.
- The objective function is submodular.

The value of a friend

- Let X be a group of individuals. How valuable to you is a given friend $x \in X$?
- It depends on how many friends you have.
- Given a group of friends $S \subseteq X$, can you valuate them with a function $F(S)$ and how?
- Let $F(S)$ be the value of the set of friends S. Is submodular or supermodular a good model?

Information and Summarization

- Let X be a set of information containing elements
  - X might say be either words, sentences, documents, web pages, or blogs.
  - Each $x \in X$ is one element, so x might be a word, a sentence, a document, etc.
  - The total amount of information in X is measure by a function $F(X)$; subset $S \subseteq X$ measures the amount of information in S, given by $F(S)$.
- How informative is any given item x in different sized contexts? Any such real-world information function would exhibit diminishing returns, i.e., the value of x decreases when it is considered in a larger context.
- So a submodular function would likely be a good model.

Restriction

Restriction

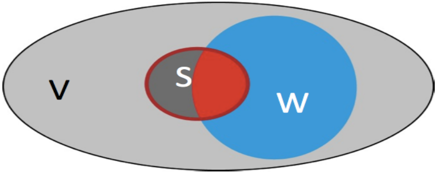
If $F(S)$ is submodular on V and $W \subseteq V$. Then $F'(S) = F(S \cap W)$ is submodular.

Proof: Given $A \subseteq B \subseteq V \backslash \{i\}$, prove:

$$F((A \cup \{i\}) \cap W) - F(A \cap W) \geq F((B \cup \{i\}) \cap W) - F(B \cap W).$$

If $i \notin W$, then both differences on each size are zero. Suppose that $i \in W$, then $(A \cup \{i\}) \cap W = (A \cap W) \cup \{i\}$ and $(B \cup \{i\}) \cap W = (B \cap W) \cup \{i\}$. We have $A \cap W \subseteq B \cap W$, the submodularity of F yields

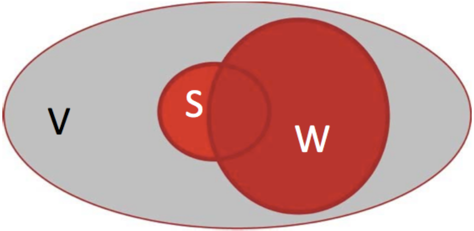
$$F((A \cap W) \cup \{i\}) - F(A \cap W) \geq F((B \cap W) \cup \{i\}) - F(B \cap W).$$



Conditioning

Conditioning

If $F(S)$ is submodular on V and $W \subseteq V$. Then $F'(S) = F(S \cup W)$ is submodular



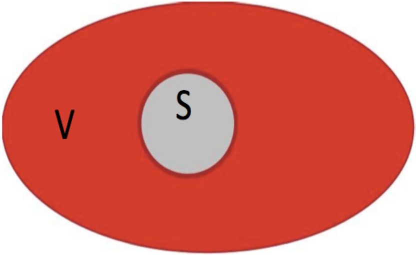
Reflection

Reflection

If $F(S)$ is submodular on V. Then $F'(S) = F(V \setminus S)$ is submodular.

Proof: Since $V\backslash(A \cup B) = (V\backslash A) \cap (V\backslash B)$ and $V\backslash(A \cap B) = (V\backslash A) \cup (V\backslash B)$, then

$$F(V\backslash A) + F(V\backslash B) \geq F(V\backslash(A \cup B)) + F(V\backslash(A \cap B)))$$



Contraction

Let $F : 2^X \rightarrow \mathbb{R}$ and $A \subseteq X$. Define $F_A(S) = F(A \cup S) - F(A)$.
Lemma: If F is monotone and submodular, then F_A is monotone, submodular, and normalized for any A.

- Proof: Monotone:
  - Let $S \subseteq T$, then $F_A(S) = F(A \cup S) - F(A) \leq F(A \cup T) - F(A) = F_A(T)$
- Submodular. Let $S, T \subseteq X$:

$$\begin{aligned} F_A(S) + F_A(T) &= F(S \cup A) - F(A) + F(T \cup A) - F(A) \\ &\geq F(S \cup T \cup A) - F(A) + F((S \cap T) \cup A) - F(A) \\ &= F_A(S \cup T) + F_A(S \cap T) \end{aligned}$$

Lemma

If F is normalized and submodular, and $A \subseteq X$, then there is $j \in A$ such that $F(\{j\}) \geq \frac{1}{|A|}F(A)$

- Proof. If A_1 and A_2 partition A, i.e., $A = A_1 \cup A_2$ and $A_1 \cap A_2 = \emptyset$, then

$$F(A_1) + F(A_2) \geq F(A_1 \cup A_2) + F(A_1 \cap A_2) = F(A)$$

- Applying recursively, we get

$$\sum_{j \in A} F(\{j\}) \geq F(A)$$

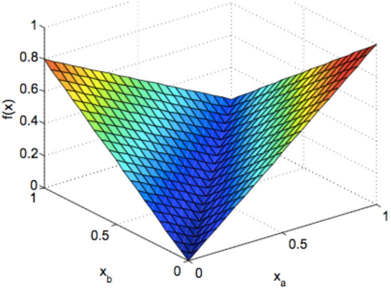
- Therefore, $\max_{j \in A} F(\{j\}) \geq \frac{1}{|A|}F(A)$

Convex aspects

- Submodularity as discrete analogue of convexity

- Convex extension
- Duality
- Polynomial time minimization!

$$A^* = \arg\min_{A \subseteq V} F(A)$$



- Many applications (computer vision,ML, $\cdots$)

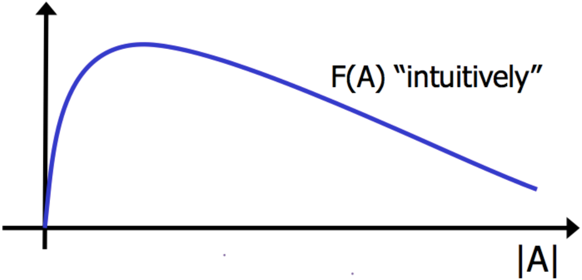
Concave aspects

- Marginal gain $\triangle_F(s|A) = F(\{s\} \cup A) - F(A)$
- Submodular:

$$\forall A \subseteq B, s \notin B : \quad F(A \cup \{s\}) - F(A) \geq F(B \cup \{s\}) - F(B)$$

- Concave:

$$\forall a \leq b, s > 0 \qquad g(a+s) - g(a) \geq g(b+s) - g(b)$$



$$\forall a \leq b, s > 0 \qquad g(a+s) - g(a) \geq g(b+s) - g(b)$$

- Suppose that $a + s \in [a, b]$
- Apply the concavity of $g(x)$ to $[a, a+s, b+s]$:

$$\begin{aligned} g(a+s) &\geq \frac{b-a}{b+s-a} g(a) + \frac{s}{b+s-a} g(b+s) \\ \iff g(a+s) - g(a) &\geq \frac{-s}{b+s-a} g(a) + \frac{s}{b+s-a} g(b+s) \end{aligned}$$

- Apply the concavity of $g(x)$ to $[a+s, b, b+s]$:

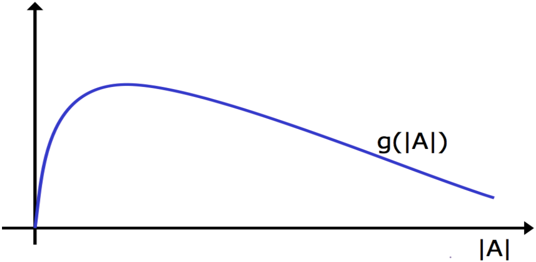
$$\begin{aligned} g(b) &\geq \frac{s}{b+s-a} g(a) + \frac{b-a}{b+s-a} g(b+s) \\ \iff g(b+s) - g(b) &\leq \frac{-s}{b+s-a} g(a) + \frac{s}{b+s-a} g(b+s) \end{aligned}$$

Submodularity and Concavity

Let $m \in \mathbb{R}_+^X$ be a modular function, and g a concave function over $\mathbb{R}$. Define $F(A) = g(m(A))$. Then F(A) is submodular.

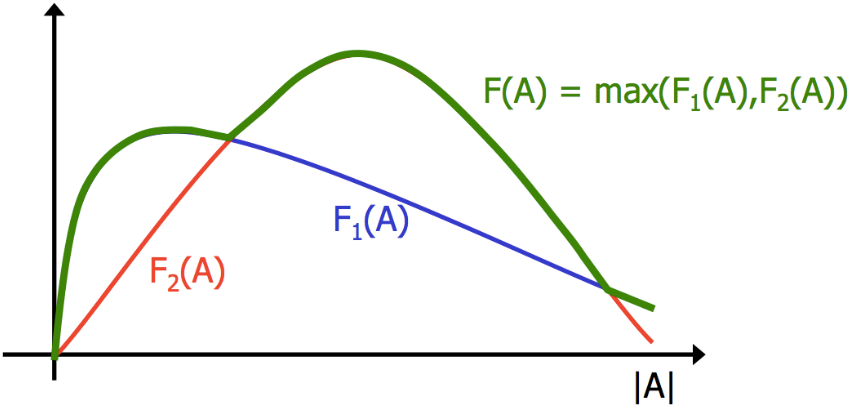
Proof: Given $A \subseteq B \subseteq X \backslash v$, we have $0 \leq a = m(A) \leq b = m(B)$, and $0 \leq s = m(v)$. For g concave, we have $g(a+s) - g(a) \geq g(b+s) - g(b)$, which implies

$$g(m(A) + m(v)) - g(m(A)) \geq g(m(B) + m(v)) - g(m(B))$$



Maximum of submodular functions

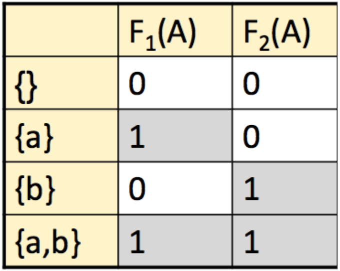
Suppose $F_1(A)$ and $F_2(A)$ submodular.
Is $F(A) = \max(F_1(A), F_2(A))$ submodular?



$\max(F_1, F_2)$ not submodular in general!

Minimum of submodular functions

Well,maybe $F(A) = \min(F_1(A), F_2(A))$ instead?



	$F_1(A)$	$F_2(A)$
{}	0	0
{a}	1	0
{b}	0	1
{a,b}	1	1

F({b}) – F({})=0

<

F({a,b}) – F({a})=1

$\min(F_1, F_2)$ not submodular in general!

Max - normalized

Given V, let $c \in \mathbb{R}_+^V$ be a given fixed vector. Then $F : 2^V \to \mathbb{R}_+$, where

$$F(A) = \max_{j \in A} c_j$$

is submodular and normalized (we take $F(\emptyset) = 0$).

Proof: Since

$$\max(\max_{j \in A} c_j, \max_{j \in B} c_j) = \max_{j \in A \cup B} c_j$$

and

$$\min(\max_{j \in A} c_j, \max_{j \in B} c_j) \geq \max_{j \in A \cap B} c_j,$$

we have

$$\max_{j \in A} c_j + \max_{j \in B} c_j \geq \max_{j \in A \cup B} c_j + \max_{j \in A \cap B} c_j$$

Monotone difference of two functions

Let F and G both be submodular functions on subsets of V and let $(F-G)(\cdot)$ be either monotone increasing. Then $h: 2^V \to R$ defined by $h(A) = \min(F(A), G(A))$ is submodular.

- If $h(A)$ agrees with either f or g on both X and Y , the result follows since

$$\begin{array}{l} F(X)+F(Y) \\ G(X)+G(Y) \end{array} \geq \min(F(X\cup Y), G(X\cup Y)) + \min(F(X\cap Y), G(X\cap Y))$$

- otherwise, w.l.o.g., $h(X) = F(X)$ and $h(Y) = G(Y)$, giving

$$h(X)+h(Y) = F(X)+G(Y) \geq F(X\cup Y)+F(X\cap Y)+G(Y)-F(Y)$$

Assume $F-G$ is monotonic increasing. Hence, $F(X\cup Y)+G(Y)-F(Y) \geq G(X\cup Y)$ giving

$$h(X)+h(Y) \geq G(X\cup Y)+F(X\cap Y) \geq h(X\cup Y)+h(X\cap Y)$$

Min

- Let $F : 2^V \to \mathbb{R}$ be an increasing or decreasing submodular function and let k be a constant. Then the function $h : 2^V \to \mathbb{R}$ defined by
$$h(A) = \min(k; F(A))$$
is submodular

- In general, the minimum of two submodular functions is not submodular. However, when wishing to maximize two monotone non-decreasing submodular functions, we can define function $h : 2^V \to R$ as
$$h(A) = \frac{1}{2}(\min(k, F) + \min(k, G))$$
then h is submodular, and $h(A) \geq k$ if and only if both $F(A) \geq k$ and $G(A) \geq k$

Outline

1. What is submodularity?

2. Submodular maximization

3. Submodular minimization

Submodular maximization with Cardinality Constraint

Problem Definition

Given a non-decreasing and normalized submodular function $F : 2^X \to \mathbb{R}^+$ on a finite ground set X with $|X| = n$, and an integer $k \leq n$:

$$\max \quad F(A), \text{ s.t. } |A| \leq k$$

Greedy Algorithm

- $A_0 \leftarrow \emptyset$, set $i = 0$
- While $|A_i| \leq k$
  - Choose $s \in X$ maximizing $F(A_i \cup \{s\})$
  - $A_{i+1} \leftarrow A_i \cup \{s\}$

Greedy maximization is near-optimal

Theorem[Nemhauser, Fisher& Wolsey'78]

For monotonic submodular functions, Greedy algorithm gives constant factor approximation

$$F(A_{\text{greedy}}) \geq \underbrace{(1-1/e)}_{\sim 63\%} F(A^*)$$

- Greedy algorithm gives near-optimal solution!
- For many submodular objectives: Guarantees best possible unless P=NP
- Can also handle more complex constraints.

Greedy maximization is near-optimal

Theorem[Nemhauser, Fisher& Wolsey'78]

For monotonic submodular functions, Greedy algorithm gives constant factor approximation

$$F(A_{\text{greedy}}) \geq (1 - 1/e)F(A^*)$$

- Proof: Let A_i be the working set in the algorithm
- Let A^* be optimal solution.
- We will show that the suboptimality $F(A^*) - F(A)$ shrinks by a factor of $(1 - 1/k)$ each iteration
- After k iterations, it has shrunk to $(1 - 1/k)^k \leq 1/e$ from its original value
- The algorithm choose $s \in X$ maximizing $F(A_i \cup \{s\})$. Hence:

$$F(A_{i+1}) = F(A_i) + F(A_i \cup \{s\}) - F(A_i) = F(A_i) + \max_j F_{A_i}(\{j\})$$

- By our lemmas, there is $j \in A^*$ s.t.

$$\begin{aligned} F_{A_i}(\{j\}) &\geq \frac{1}{|A^*|} F_{A_i}(A^*) \quad \text{(apply lemma to } F_{A_i}\text{)} \\ &= \frac{1}{k}(F(A_i \cup A^*) - F(A_i)) \\ &\geq \frac{1}{k}(F(A^*) - F(A_i)) \end{aligned}$$

- Therefore

$$\begin{aligned} F(A^*) - F(A_{i+1}) &= F(A^*) - F(A_i) - \max_j F_{A_i}(\{j\}) \\ &\leq \left(1 - \frac{1}{k}\right)(F(A^*) - F(A_i)) \\ &\leq \left(1 - \frac{1}{k}\right)^k (F(A^*) - F(\emptyset)) \end{aligned}$$

Scaling up the greedy algorithm [Minoux'78]

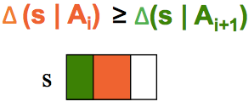
In round i+1,

- have picked $A_i = s_1, \cdots, s_i$
- pick $s_{i+1} = \arg\max_s F(A_i \cup \{s\}) - F(A_i)$.
- Update the gain of other elements affected by the addition of s_{i+1}.

The core of the algorithm is maximize "marginal benefit" $\triangle(s|A_i)$

$$\triangle(s|A_i) = F(A_i \cup \{s\}) - F(A_i)$$

Key observation: Submodularity implies

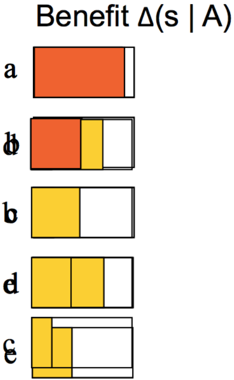


Marginal benefits can never increase!

"Lazy" greedy algorithm [Minoux'78]

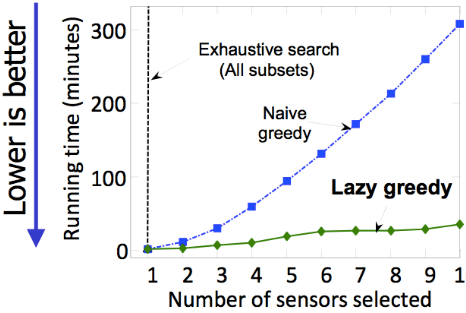
Lazy greedy algorithm:

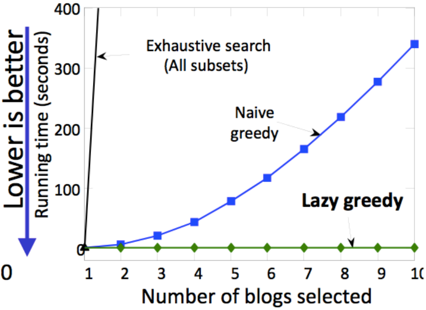
- First iteration as usual
- Keep an ordered list of marginal benefits $\triangle_i$ from previous iteration
- Re-evaluate $\triangle_i$ only for top element
- If $\triangle_i$ stays on top, use it, otherwise re-sort



Note: Very easy to compute online bounds, lazy evaluations, etc. [Leskovec,Krause et al.'07]

Empirical improvements [Leskovec, Krause et al'06]





Sensor placement

Blog selection

30x speedup

700x speedup

Stochastic-greedy algorithm [[Mirzasoleimanet al'14]

In round i+1,

- have picked $A_i = s_1, \cdots, s_i$.
- R is a random subset obtained by sampling s random elements from $X \setminus A$.
- pick $s_{i+1} = \arg\max_{s \in R} F(A_i \cup \{s\}) - F(A_i)$.

The algorithm at each step selects a random subset R of size $s = \frac{n}{k} \log \frac{1}{\epsilon}$, choosing the element from R that provides the maximum marginal gain to the current solution A.
It achieves a $(1 - \frac{1}{e} - \epsilon)$ approximation guarantee with $O(n \log \frac{1}{\epsilon})$ function evaluations, where ϵ is an acceptable error bound for the algorithm.

Outline

1 What is submodularity?

2 Submodular maximization

3 Submodular minimization

Optimizing Submodular Functions

- As our examples suggest, optimization problems involving submodular functions are very common

- These can be classified on two axes: constrained/unconstrained and maximization/minimization

	Maximization	Minimization
Unconstrained	NP-hard $\frac{1}{2}$ approximation	Polynomial time via convex opt
Constrained	Usually NP-hard $1 - 1/e$ (mono, matroid) O(1) (“nice” constriants)	Usually NP-hard to apx. Few easy special cases

Representation

In order to generalize all our examples, algorithmic results are often posed in the value oracle model. Namely, we only assume we have access to a subroutine evaluating $F(S)$.

Problem Definition

Given a submodular function $f : 2^X \to \mathbb{R}$ on a finite ground set X,

$$\begin{array}{ll} \min & F(S) \\ \text{s.t.} & S \subseteq X \end{array}$$

- We denote $n = |X|$
- We assume $F(S)$ is a rational number with at most b bits
- Representation: in order to generalize all our examples, algorithmic results are often posed in the value oracle model. Namely, we only assume we have access to a subroutine evaluating $F(S)$ in constant time.

Goal

An algorithm which runs in time polynomial in n and b.

Some more notations

- $E = \{1, 2, \ldots, n\}$
- $\mathbb{R}^E = \{x = (x_j \in \mathbb{R} : j \in E)\}$
- $\mathbb{R}^E_+ = \{x = (x_j \in \mathbb{R} : j \in E) : x \geq 0\}$
- Any vector $x \in \mathbb{R}^E$ can be treated as a normalized modular function, and vice verse. That is

  $$x(A) = \sum_{a \in A} x_a.$$

  Note that x is said to be normalized since $x(\emptyset) = 0$.
- Given $A \subseteq E$, define the vector $1_A \in \mathbb{R}^E_+$ to be

  $$1_A(j) = \begin{cases} 1 & \text{if } j \in A \\ 0 & \text{if } j \notin A \end{cases}$$

- given modular function $x \in \mathbb{R}^E$, we can write $x(A)$ in a variety of ways, i.e., $x(A) = x \cdot 1_A = \sum_{i \in A} x_i$

Continuous Extensions of a Set Function

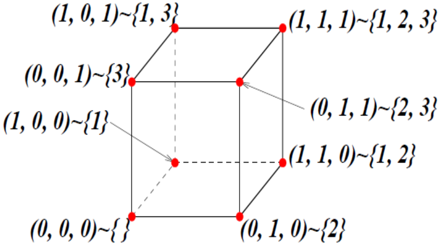
- A set function F on $X = \{1, \ldots, n\}$ can be thought of as a map from the vertices $\{0, 1\}^n$ of the n-dimensional hypercube to the real numbers.

Extension of a Set Function

Given a set function $F : \{0, 1\}^n \to \mathbb{R}$, an extension of F to the hypercube $[0, 1]^n$ is a function $g : [0, 1]^n \to \mathbb{R}$ satisfying $g(x) = F(x)$ for every $x \in \{0, 1\}^n$.

$$\min_{w \in \{0,1\}^n} \quad F(w)$$

with $\forall A \subseteq X$, $F(1_A) = F(A)$



Choquet integral - Lovász extension

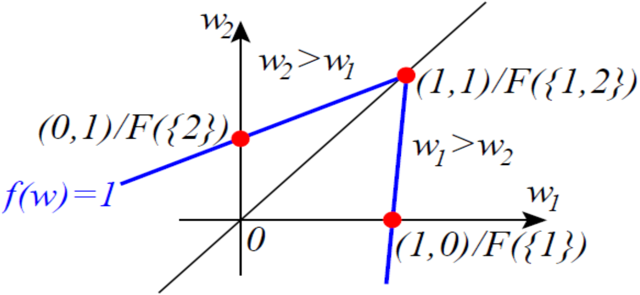
- Subsets may be identified with elements of $\{0,1\}^n$
- Given any set-function F and w such that $w_{j_1} \geq \ldots \geq w_{j_n}$, define

$$\begin{aligned} f(w) &= \sum_{k=1}^{n} w_{j_k} [F(\{j_1,\ldots,j_k\}) - F(\{j_1,\ldots,j_{k-1}\}) \\ &= \sum_{k=1}^{n-1} (w_{j_k} - w_{j_{k+1}}) F(\{j_1,\ldots,j_k\}) + w_{j_n} F(\{j_1,\ldots,j_n\}) \end{aligned}$$

- If $w = 1_A$, $f(w) = F(A) \Longrightarrow$ extension from $\{0,1\}^n$ to $\mathbb{R}^n$

Choquet integral - Lovász extension, example: $p = 2$

- If $w_1 \geq w_2, f(w) = F(\{1\})w_1 + [F(\{1,2\}) - F(\{1\})]w_2$
- If $w_1 \leq w_2, f(w) = F(\{2\})w_2 + [F(\{1,2\}) - F(\{2\})]w_1$



level set $\{w \in \mathbb{R}^2, f(w) = 1\}$ is displayed in blue

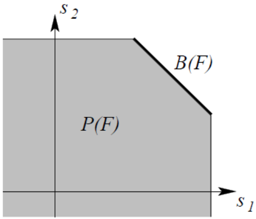
- Compact formulation: $f(w) = [F(\{1,2\}) - F(\{1\}) - F(\{2\})] \min(w_1, w_2) + F(\{1\})w_1 + F(\{2\})w_2$

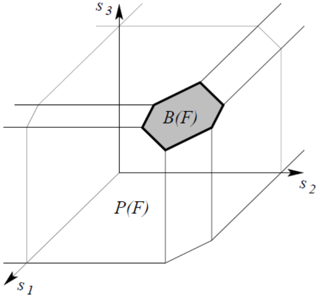
Links with convexity

Theorem (Lovász, 1982)

F is submodular if and only if f is convex

- Proof requires: Submodular and base polyhedra
- Submodular polyhedron: $P(F) = \{s \in \mathbb{R}^n, \forall A \subseteq V, s(A) \leq F(A)\}$
- Base polyhedron: $B(F) = P(F) \cap \{s(V) = F(V)\}$





Submodular and base polyhedra

- $P(F)$ has non-empty interior
- Many facets (up to 2^n), many extreme points (up to $n!$)

Fundamental property (Edmonds, 1970): If F is submodular, maximizing linear functions may be done by a "greedy algorithm"

- Let $w \in \mathbb{R}_+^n$ such that $w_{j_1} \geq \ldots \geq w_{j_n}$
- Let $s_{j_k} = F(\{j_1, \ldots, j_k\}) - F(\{j_1, \ldots, j_{k-1}\})$ for $k \in \{1, \ldots, n\}$
- Then

$$f(w) = \max_{s \in P(F)} w^\top s = \max_{s \in B(F)} w^\top s$$

- Both problems attained at s defined as above.
- proofs: pages 41-44 in `http://bicmr.pku.edu.cn/~wenzw/bigdata/submodular_fbach_mlss2012.pdf`

Links with convexity

Theorem (Lovász, 1982)

F is submodular if and only if f is convex

- If F is submodular, f is the maximum of linear functions. Then f is convex
- If f is convex, let $A, B \subseteq V$
  - $1_{A\cup B} + 1_{A\cap B} = 1_A + 1_B$ has components equal to 0 (on $V\backslash(A\cup B)$), 2 (on $A\cap B$) and 1 (on $A\Delta B = (A\backslash B)\cup(B\backslash A)$)
  - Thus $f(1_{A\cup B} + 1_{A\cap B}) = F(A\cup B) + F(A\cap B)$. Proof by writing out $f(1_{A\cup B} + 1_{A\cap B})$ and the definition of $f(w)$.
  - By homogeneity and convexity, $f(1_A + 1_B) \leq f(1_A) + f(1_B)$, which is equal to $F(A) + F(B)$, and thus F is submodular.

Links with convexity

Theorem (Lovász, 1982)

If F is submodular, then

$$\min_{A\subseteq V} F(A) = \min_{w\in\{0,1\}^n} f(w) = \min_{w\in[0,1]^n} f(w)$$

- Since f is an extension of F,

$$\min_{A\subseteq V} F(A) = \min_{w\in\{0,1\}^n} f(w) \geq \min_{w\in[0,1]^n} f(w)$$

- Any $w \in [0,1]^n$ can be decomposed as $w = \sum_{i=1}^m \lambda_i 1_{B_i}$, where $B_1 \subseteq \ldots \subseteq B_m = V$, where $\lambda \geq 0$ and $\lambda(V) \leq 1$:
    - Since $\min_{A\subseteq V} F(A) \leq 0$ ($F(\emptyset) = 0$),

$$f(w) = \sum_{i=1}^m \lambda_i F(B_i) \geq \sum_{i=1}^m \lambda_i \min_{A\subseteq V} F(A) \geq \min_{A\subseteq V} F(A)$$

    - Thus $\min_{w\in[0,1]^n} f(w) \geq \min_{A\subseteq V} F(A)$.

Links with convexity

- Any $w \in [0,1]^n$, sort $w_{j_1} \geq \ldots \geq w_{j_n}$. Find λ such that

$$\sum_{k=1}^{n} \lambda_{j_k} = w_{j_1}, \sum_{k=2}^{n} \lambda_{j_k} = w_{j_2}, \ldots, \lambda_{j_n} = w_{j_n},$$
$$B_1 = \{j_1\}, B_2 = \{j_1, j_2\}, \ldots, B_n = \{j_1, j_2, \ldots, j_n\}$$

  Then we have $w = \sum_{i=1}^{n} \lambda_i 1_{B_i}$, where $B_1 \subseteq \ldots \subseteq B_n = V$, where $\lambda \geq 0$ and $\lambda(V) = \sum_{i \in V} \lambda_i \leq 1$.

Submodular function minimization

- Let $F : 2^V \to \mathbb{R}$ be a submodular function (such that $F(\emptyset) = 0$)
- **convex duality**:

$$\begin{aligned} \min_{A \subseteq V} F(A) &= \min_{w \in [0,1]^n} f(w) \\ &= \min_{w \in [0,1]^n} \max_{s \in B(F)} w^\top s \\ &= \max_{s \in B(F)} \min_{w \in [0,1]^n} w^\top s = \max_{s \in B(F)} s_-(V) \end{aligned}$$

Submodular function minimization

Convex optimization

If F is submodular, then

$$\min_{A \subseteq V} F(A) = \min_{w \in \{0,1\}^n} f(w) = \min_{w \in [0,1]^n} f(w)$$

Using projected subgradient descent to minimize f on $[0,1]^n$

- Iteration: $w_t = \Pi_{[0,1]^n}(w_{t-1} - \frac{C}{\sqrt{t}} s_t)$, where $s_t \in \partial f(w_{t-1})$
- $f(w) = \max_{s \in B(F)} w^\top s$
- Standard convergence results from convex optimization

$$f(w_t) - \min_{w \in [0,1]^n} f(w) \leq \frac{C}{\sqrt{t}}$$

Summary

- Many problems of recommending sets can be cast as submodular maximization
- Greedy algorithm gives best set of size k
- Can use lazy evaluations to speed up
- Approximate submodular maximization possible under a variety of constraints:
  - Matroid
  - Knapsack
  - Multiple matroid and knapsack constraints
  - Path constraints (Submodular orienteering)
  - Connectedness (Submodular Steiner)
  - Robustness (minimax)

Stochastic Gradient Methods for Large-scale Machine Learning

Zaiwen Wen

http://bicmr.pku.edu.cn/~wenzw/bigdata2023.html

Thanks Yongfeng Li and Zhanwang Deng for preparing part of this slides

Why Optimization in Machine Learning?

Many problems in ML can be written as

$$\min_{\theta \in \mathcal{W}} \quad \sum_{i=1}^{N} \frac{1}{2} \|x_i^\top \theta - y_i\|_2^2 + \mu \|\theta\|_2^2 \quad \text{linear regression}$$

$$\min_{\theta \in \mathcal{W}} \quad \frac{1}{N} \sum_{i=1}^{N} \log(1 + \exp(-y_i x_i^\top \theta)) + \mu \|\theta\|_2^2 \quad \text{logistic regression}$$

$$\min_{\theta \in \mathcal{W}} \quad \sum_{i=1}^{N} \ell(h(\theta, x_i), y_i) + \mu \varphi(\theta) \quad \text{general formulation}$$

- The pairs (x_i, y_i) are given data, y_i is the label of the data point x_i
- $\ell(\cdot)$: measures how model fit for data points (avoids under-fitting)
- $\varphi(\theta)$: regularization term (avoids over-fitting)
- $h(\theta, x)$: linear function or models constructed from deep neural networks

Sparse Logistic Regression

The logistic regression problem:

$$\min_{\theta \in \mathbb{R}^n} \quad \frac{1}{N} \sum_{i=1}^{N} \log(1 + \exp(-y_i x_i^T \theta)) + \mu \|\theta\|_2^2.$$

- The data pair $\{x_i, y_i\} \in \mathbb{R}^n \times \{-1, 1\}, i \in [N]$,

Data Set	# data N	# features n	sparsity(%)
cina	16,033	132	70.49
a9a	32,561	123	88.72
ijcnn1	49,990	22	40.91
covtype	581,012	54	77.88
url	2,396,130	3,231,961	99.99
susy	5,000,000	18	1.18
higgs	11,000,000	28	7.89
news20	19,996	1,355,191	99.97
rcv1	20,242	47,236	99.84
kdda	8,407,752	20,216,830	99.99

Deep Learning

The objective function is the CrossEntropy function plus regularization term:

$$\min_{\theta} \quad \frac{1}{N}\sum_{i=1}^{N} -\log\left(\frac{\exp(h(\theta,x_i)[y_i])}{\sum_j \exp(h(\theta,x_i)[y_j])}\right) + \mu\|\theta\|_2^2$$

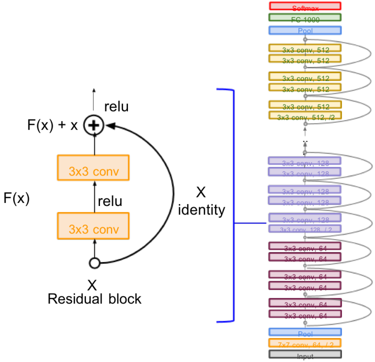
where $h(\theta, x_i)$ is output from network, and (x_i, y_i) are data points.

	Cifar-10	Cifar-100
# num_class	10	100
# number per class (training set)	5,000	500
# number per class (testing set)	1,000	100
# Total parametes of VGG-16	15,253,578	15,299,748
# Total parameters of ResNet-18	11,173,962	11,220,132

Table: A description of datasets used in the neural network experiments

ResNet Architecture

- Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Cited by **114474 since 2015** at Google scholar
- Stack residual blocks. Every residual block has two 3x3 conv layers.
- Make networks from shallow to deep.
- Fancy network architecture. Many Applications.
- High-computationally-cost !
- ResNet-50 on ImageNet, **29 hours using 8 Tesla P100 GPUs**



自然语言处理

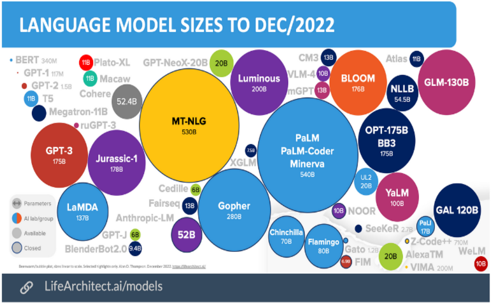
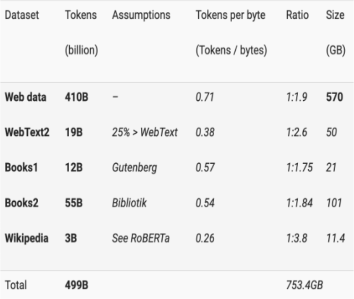


Chart. Major AI language models 2018-2022, GPT-3 on the left in red.



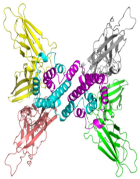
Dataset	Tokens (billion)	Assumptions	Tokens per byte (Tokens / bytes)	Ratio	Size (GB)
Web data	**410B**	–	*0.71*	*1:1.9*	**570**
WebText2	**19B**	*25% > WebText*	*0.38*	*1:2.6*	*50*
Books1	**12B**	*Gutenberg*	*0.57*	*1:1.75*	*21*
Books2	**55B**	*Bibliotik*	*0.54*	*1:1.84*	*101*
Wikipedia	**3B**	*See RoBERTa*	*0.26*	*1:3.8*	*11.4*
Total	**499B**			*753.4GB*	

Table. GPT-3 Datasets. Disclosed in **bold**. Determined in *italics*.

ChatGPT (2022/12)训练代价

- 硬件代价：超级计算机 10,000 GPUs 和 285,000 CPU 核，约10亿美元
- 人员费用：首席科学家Ilya Sutskever 190万美元/年(2016年)，世界顶级Ph.D团队，120人，第一年人员费用超过2亿美元
- 数据收集时间12-18个月，训练时间9-12个月

AI For Science



J. Jumper *et al.* Nature (2021)

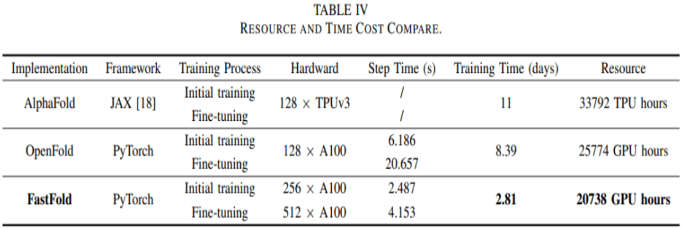
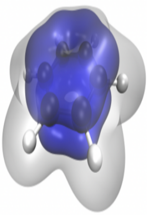
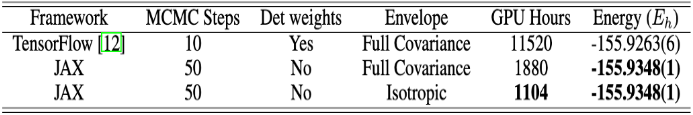


TABLE IV
RESOURCE AND TIME COST COMPARE.

Implementation	Framework	Training Process	Hardward	Step Time (s)	Training Time (days)	Resource
AlphaFold	JAX [18]	Initial training	128 × TPUv3	/	11	33792 TPU hours
		Fine-tuning		/		
OpenFold	PyTorch	Initial training	128 × A100	6.186	8.39	25774 GPU hours
		Fine-tuning		20.657		
FastFold	PyTorch	Initial training	256 × A100	2.487	**2.81**	**20738 GPU hours**
		Fine-tuning	512 × A100	4.153		





Framework	MCMC Steps	Det weights	Envelope	GPU Hours	Energy (E_h)
TensorFlow [12]	10	Yes	Full Covariance	11520	-155.9263(6)
JAX	50	No	Full Covariance	1880	**-155.9348(1)**
JAX	50	No	Isotropic	**1104**	**-155.9348(1)**

Table 2: Speed and accuracy comparison for different training runs of the FermiNet on bicyclobutane. All runs were for 200,000 iterations, with the same number of determinants and hidden units as in [12]. Number of GPU hours for TensorFlow assumes 30 days of training on 16 GPUs. Lower energies are strictly better as FermiNet provides an upper bound to the exact energy.

Outline

1. Problem Description
2. Stochastic Gradient Methods
3. Convergence Analysis
4. Variance Reduction
5. Natural Gradient Method

机器学习模型

- $x \in \mathcal{X}$为数据；$y \in \mathcal{Y}$为标签，其中$\mathcal{X}$为数据所在的空间，$\mathcal{Y}$为标签的所在的空间，并且x与y服从某个未知的分布π，也即$(x, y) \sim \pi$. 数据集为$\mathcal{S} = \{(x_i, y_i)\}_{i=1}^N$.
- $h_\theta(x)$为得到的模型，其中θ为需要优化的变量. 我们通过训练参数θ来得到合适的模型$h_\theta(x)$.
- 损失函数
  - 平方损失函数: $\ell(y, h_\theta(x)) = \|h_\theta(x) - y\|^2$.
  - 交叉熵函数: $\ell(y, h_\theta(x)) = y^{\mathrm{T}} \log(h_\theta(x)) = \sum_{j=1}^C y_j \log(h_\theta(x))_j$.
  - 链接损失函数(Hinge loss): $\ell(y, h_\theta(x)) = \max(0, 1 - h_\theta(x)^{\mathrm{T}} y)$;
  - 指数损失函数(Exponential loss): $\ell(y, h_\theta(x)) = \exp(-h_\theta(x)^{\mathrm{T}} y)$;
  - 对数损失函数(Logisticloss): $\ell(y, h_\theta(x)) = \log(1 + \exp(-h_\theta(x)^{\mathrm{T}} y))$;
  - KL 散度(KL divergence): $\ell(y, h_\theta(x)) = \sum_{j=1}^C y_j \log h_\theta(x_j) - \sum_{j=1}^C y_j \log y_j$.

优化模型

- 期望风险(Expected risk):

$$\min_{\theta} \quad f_{\pi}(\theta) = R[h_{\theta}] \overset{\text{def}}{=\!=} \mathop{\mathbb{E}}_{(x,y)\sim\pi}[\ell(y, h_{\theta}(x))]. \tag{1}$$

- 经验风险(Empirical risk): 令$f_i(\theta) = \ell(y_i, h_{\theta}(x_i))$,

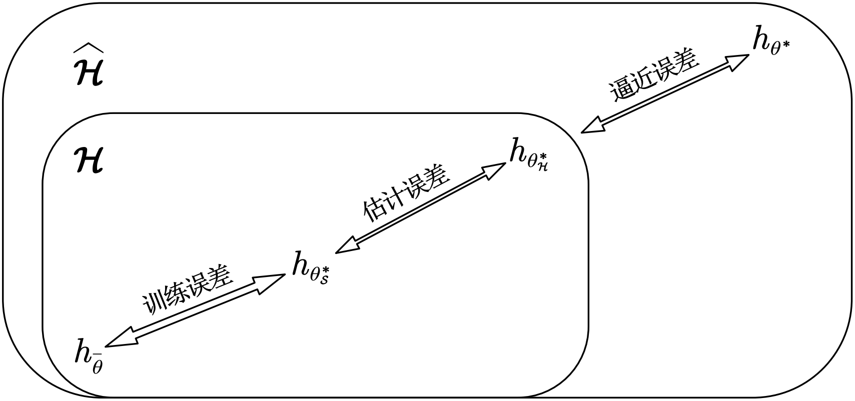
$$\min_{\theta} \quad f_{\mathcal{S}}(\theta) = \hat{R}_{\mathcal{S}}[h_{\theta}] \overset{\text{def}}{=\!=} \frac{1}{N}\sum_{i=1}^{N} f_i(\theta). \tag{2}$$

- $\mathop{\mathbb{E}}_{(x,y)\sim\pi}$ 表示关于(x, y)的分布求期望:

$$\begin{aligned} \mathop{\mathbb{E}}_{(x_i,y_i)\sim\pi}[\hat{R}_{\mathcal{S}}[h_{\theta}]] &= \mathop{\mathbb{E}}_{(x_i,y_i)\sim\pi}\left[\frac{1}{N}\sum_{i=1}^{N}\ell(y_i, h_{\theta}(x_i))\right] \\ &= \mathop{\mathbb{E}}_{(x,y)\sim\pi}[\ell(y, h_{\theta}(x))] = R[h_{\theta}]. \end{aligned}$$

误差分解

- 假设真实解满足$h_{\theta^*} \in \widehat{\mathcal{H}}$. 然而现实情况下时只能取近似的假设空间$\mathcal{H} \subseteq \widehat{\mathcal{H}}$，比如$\mathcal{H}$是多项式、分段线性函数或者神经网络等组成的函数空间. 在不引起歧义的情况下，混淆使用h 和h_θ所属的空间，即$\theta \in \mathcal{H}$表示$h_\theta \in \mathcal{H}$.



误差分解

- 设θ^*为最优模型对应的最优参数，对应的模型为h_{θ^*},即:

$$\theta^* = \operatorname*{argmin}_{\theta} R[h_\theta], \text{ s.t. } \theta \in \hat{\mathcal{H}}. \tag{3}$$

- 设$\theta^*_{\mathcal{H}}$为在假设空间$\mathcal{H}$中的最小化期望风险模型对应的最优参数，对应的模型为$h_{\theta^*_{\mathcal{H}}}$,即：

$$\theta^*_{\mathcal{H}} = \operatorname*{argmin}_{\theta} R[h_\theta], \text{ s.t. } \theta \in \mathcal{H}. \tag{4}$$

- 设$\theta^*_{\mathcal{S}}$为属于假设空间$\mathcal{H}$并且在样本数据集$\mathcal{S}$上经验风险模型对应的最优参数，对应的模型为$h_{\theta^*_{\mathcal{S}}}$,即:

$$\theta^*_{\mathcal{S}} = \operatorname*{argmin}_{\theta} \hat{R}_{\mathcal{S}}[h_\theta], \text{ s.t. } \theta \in \mathcal{H}. \tag{5}$$

- 设$\bar{\theta}$为(5)的近似解，对应的模型为$h_{\bar{\theta}}$,即:

$$\bar{\theta} \approx \operatorname*{argmin}_{\theta} \hat{R}_{\mathcal{S}}[h_\theta], \text{ s.t. } \theta \in \mathcal{H}. \tag{6}$$

误差估计

- 相应的实际模型与最优模型的期望风险的误差分解为：

$$R[h_{\theta^*}] - R[h_{\bar{\theta}}] = \underbrace{R[h_{\theta^*}] - R[h_{\theta^*_{\mathcal{H}}}]}_{\text{逼近误差}} + \underbrace{R[h_{\theta^*_{\mathcal{H}}}] - R[h_{\theta^*_{\mathcal{S}}}]}_{\text{泛化误差}} + \underbrace{R[h_{\theta^*_{\mathcal{S}}}] - R[h_{\bar{\theta}}]}_{\text{优化误差}}.$$

- 逼近误差与假设空间$\mathcal{H}$的表达能力有关
- 泛化误差与样本和假设空间有关．通常情况下，样本量越大，求解得到的$h_{\theta^*_{\mathcal{S}}}$与$h_{\theta^*_{\mathcal{H}}}$越接近．
- 优化误差(训练误差)与许多因素都有关系，比如模型的选择，优化算法的设计，包括初始点的选择等．

Hoeffding Inequality

Let $X_1, X_2, \cdots$ be a sequence of i.i.d. random variables and assume that for all i, $E(X_i) = \mu$ and $\mathcal{P}(a \leq X_i \leq b) = 1$. Then for any $\epsilon > 0$

$$\mathcal{P}\left(\left|\frac{1}{n}\sum_{i=1}^{n} X_i - \mu\right| \geq \epsilon\right) \leq 2\exp\left(-\frac{2n\epsilon^2}{(b-a)^2}\right) \tag{7}$$

- The Hoeffding Inequality describes the asymptotic property that sampling mean convergences to expectation.
- Azuma-Hoeffding inequality is a martingale version. Let $X_1, X_2, \cdots$ be a martingale difference sequence with $|X_i| \leq B$ for all $i = 1, 2, ...$ Then

$$\mathcal{P}(\sum_{i=1}^{n} X_i \geq t) \leq \exp\left(-\frac{2t^2}{nB^2}\right)$$

$$\mathcal{P}(\sum_{i=1}^{n} X_i \leq t) \leq \exp\left(-\frac{2t^2}{nB^2}\right)$$

误差估计

设假设空间$\mathcal{H}$为有限的，损失函数满足$0 \le \ell(h(x_i), y_i) \le 1$, $\forall x_i, y_i,\ h \in \mathcal{H}$. 则对于$0 < \delta < 1$,则有以概率$1-\delta$,有

$$|R[h] - \hat{R}[h]| \le \sqrt{\frac{\ln|\mathcal{H}| + \ln(\frac{2}{\delta})}{2N}}.$$

证明: 根据$\hat{R}[h]$定义及$\mathbb{E}[\hat{R}[h]] = R[h]$，由霍夫丁不等式有

$$P(|\hat{R}[h] - R[h]| \ge \varepsilon) \le 2e^{-2N\varepsilon^2}.$$

所以对于有限的假设空间$\mathcal{H}$,有

$$P\left(\bigcup_{h\in\mathcal{H}} \{|\hat{R}[h] - R[h]| \ge \varepsilon\}\right) \le 2|\mathcal{H}|e^{-2N\varepsilon^2}.$$

如果希望$\mathcal{P}\left(\bigcup_{h\in\mathcal{H}} \{|\hat{R}[h] - R[h]| \ge \varepsilon\}\right) \le \delta$,则需要样本量满足

$$N = \frac{1}{2\varepsilon^2}\ln\left(\frac{2|\mathcal{H}|}{\delta}\right) = \mathcal{O}\left(\frac{\ln|\mathcal{H}| + \ln(\delta^{-1})}{\varepsilon^2}\right). \tag{8}$$

泛化误差估计

$$R[h_{\mathcal{S}}^*] - R[h_{\mathcal{H}}^*] \le 2\sqrt{\frac{\ln|\mathcal{H}| + \ln(\frac{2}{\delta})}{2N}}.$$

证明：由于

$$R[h_{\mathcal{S}}^*] - R[h_{\mathcal{H}}^*] = \underbrace{R(h_{\mathcal{S}}^*) - \hat{R}(h_{\mathcal{S}}^*)}_{(1)} + \underbrace{\hat{R}(h_{\mathcal{S}}^*) - \hat{R}(h_{\mathcal{H}}^*)}_{(2)} + \underbrace{\hat{R}(h_{\mathcal{H}}^*) - R(h_{\mathcal{H}}^*)}_{(3)}.$$

对于(1)式和(3)式，其绝对值均小于$\sup|R[h] - \hat{R}[h]|, \forall h \in \mathcal{H}$.对于(2)式，根据$\hat{R}[h]$的定义我们可以知道其非正，由此可得：

$$|R[h_{\mathcal{S}}^*] - R[h_{\mathcal{H}}^*]| \le 2\sup|R[h] - \hat{R}[h]|, \ \forall h \in \mathcal{H}.$$

再根据前一页可知在大概率下有下式成立:

$$|R[h_{\mathcal{S}}^*] - R[h_{\mathcal{H}}^*]| \le 2\sqrt{\frac{\ln|\mathcal{H}| + \ln(\frac{2}{\delta})}{2N}}.$$

在$\mathcal{H}$为常数的情况下，通过增大样本数量来极小化经验误差求解得到的$h_{\mathcal{S}}^*$的期望误差与其经验误差相差不大.

泛化误差

- What if $|\mathcal{H}| = \infty$? This bound doesn't work
- For a two label classification problem, with a probability $1-\delta$, we have

$$\sup_{h\in\mathcal{H}} |\hat{R}_n[h] - R[h]| \leq O\left(\sqrt{\frac{VC[\mathcal{H}]\log(\frac{n}{VC[\mathcal{H}]}) + \log(\frac{1}{\delta}))}{n}}\right) \tag{9}$$

  where $VC[\mathcal{H}]$ is a VC dimension of $\mathcal{H}$.

- Finite VC dimension is sufficient and necessary condition of empirical risk concentration for two label classification.
- 拉德马赫复杂度

Outline

1. Problem Description
2. Stochastic Gradient Methods
3. Convergence Analysis
4. Variance Reduction
5. Natural Gradient Method

梯度下降算法

考虑如下随机优化问题：

$$\min_{x\in\mathbb{R}^n} \quad f(x) \stackrel{\text{def}}{=\!=} \frac{1}{N}\sum_{i=1}^{N} f_i(x), \tag{10}$$

- 假设每一个$f_i(x)$是凸的、可微的．可以运用梯度下降算法

$$x^{k+1} = x^k - \alpha_k \nabla f(x^k).$$

- 梯度必须计算出所有的$\nabla f_i(x^k)$然后将它们相加:

$$\nabla f(x^k) = \frac{1}{N}\sum_{i=1}^{N} \nabla f_i(x^k).$$

- 然而机器学习中采集到的样本量巨大，因此计算$\nabla f(x^k)$需要非常大的计算量．使用传统的梯度法求解机器学习问题并不是一个很好的做法．

随机梯度下降算法(SGD)

- SGD的基本迭代格式为

$$x^{k+1} = x^k - \alpha_k \nabla f_{s_k}(x^k), \tag{11}$$

其中s_k是从$\{1, 2, \cdots, N\}$中随机等可能地抽取的一个样本

- 步长α_k在机器学习中被称为学习率(learning rate).
- 随机梯度的条件期望恰好是全梯度，即

$$\mathbb{E}_{s_k}[\nabla f_{s_k}(x^k) | x^k] = \nabla f(x^k).$$

- 常用的形式是小批量（mini-batch）随机梯度法. 随机选择元素个数很少的集合$\mathcal{I}_k \subset \{1, 2, \cdots, N\}$，然后执行迭代格式

$$x^{k+1} = x^k - \frac{\alpha_k}{|\mathcal{I}_k|} \sum_{s \in \mathcal{I}_k} \nabla f_s(x^k),$$

- 如果$f_i(x)$不可微，可以考虑随机次梯度算法:

$$x^{k+1} = x^k - \alpha_k g^k$$

动量方法

- 为了克服随机梯度下降法收敛速度慢的缺陷，提出了动量方法（momentum），其思想是在算法迭代时一定程度上保留之前更新的方向，同时利用当前计算的梯度调整最终的更新方向.
- 动量方法的具体迭代格式如下：

$$v^{k+1} = \mu_k v^k - \alpha_k \nabla f_{s_k}(x^k), \tag{12}$$

$$x^{k+1} = x^k + v^{k+1}. \tag{13}$$

  在计算当前点的随机梯度$\nabla f_{s_i}(x^k)$后，将其和上一步更新方向v^k做线性组合来得到新的更新方向v^{k+1}.
- 当$\mu_k = 0$时该方法退化成随机梯度下降法. 参数μ_k的范围是$[0,1)$，通常取$\mu_k \geq 0.5$，其含义为迭代点带有较大惯性，每次迭代会在原始迭代方向的基础上做一个小的修正.
- 在普通的梯度法中，每一步迭代只用到了当前点的梯度估计，动量方法的更新方向还使用了之前的梯度信息.
- 当许多连续的梯度指向相同的方向时，步长就会很大，这从直观上看也是非常合理的.

动量方法

图1比较了梯度法和动量方法的表现．可以看到普通梯度法生成的点列会在椭圆的短轴方向上来回移动，而动量方法生成的点列更快收敛到了最小值点．

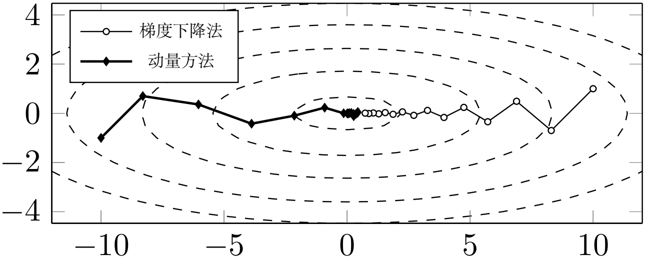


Figure: 动量方法在海瑟矩阵病态条件下的表现

Nesterov加速算法

- 假设$f(x)$为光滑的凸函数．针对凸问题的Nesterov加速算法为

$$\begin{aligned} y^{k+1} &= x^k + \mu_k(x^k - x^{k-1}) \\ x^{k+1} &= y^k - \alpha_k \nabla f(y^k) \end{aligned}$$

- 针对光滑问题的Nesterov加速算法迭代的随机版本为

$$y^{k+1} = x^k + \mu_k(x^k - x^{k-1}), \tag{14}$$

$$x^{k+1} = y^{k+1} - \alpha_k \nabla f_{s_k}(y^{k+1}), \tag{15}$$

  其中$\mu_k = \frac{k-1}{k+2}$，步长α_k是一个固定值或者由线搜索确定．

- 可以看出，二者的唯一区别为随即版本将全梯度$\nabla f(y^k)$替换为随机梯度$\nabla f_{s_k}(y^{k+1})$．

Nesterov加速算法与动量方法的联系

- 若在第k步迭代引入速度变量$v^k = x^k - x^{k-1}$，再合并原始Nesterov加速算法的两步迭代可以得到

$$x^{k+1} = x^k + \mu_k(x^k - x^{k-1}) - \alpha_k \nabla f_k(x^k + \mu_k(x^k - x^{k-1})).$$

- 定义有关v^{k+1}的迭代式

$$v^{k+1} = \mu_k v^k - \alpha_k \nabla f_k(x^k + \mu_k v^k),$$

- 于是得到关于x^k和v^k的等价迭代：

$$\begin{aligned} v^{k+1} &= \mu_k v^k - \alpha_k \nabla f_{s_k}(x^k + \mu_k v^k), \\ x^{k+1} &= x^k + v^{k+1}. \end{aligned}$$

- 二者的主要差别在梯度的计算上．Nesterov加速算法先对点施加速度的作用，再求梯度，可以理解为对标准动量方法做了校正．

AdaGrad

- 令$g^k = \nabla f_{s_k}(x^k)$，引入向量

$$G^k = \sum_{i=1}^{k} g^i \odot g^i.$$

G^k的每个分量是梯度在该分量处的累积平方和．当G^k的某分量较大时，该分量变化比较剧烈，因此应采用小步长，反之亦然．

- 因此AdaGrad的迭代格式为

$$x^{k+1} = x^k - \frac{\alpha}{\sqrt{G^k + \varepsilon \mathbf{1}_n}} \odot g^k, \tag{16}$$

$$G^{k+1} = G^k + g^{k+1} \odot g^{k+1}, \tag{17}$$

这里$\frac{\alpha}{\sqrt{G^k+\varepsilon\mathbf{1}_n}}$中的除法和求根运算都是对向量每个分量分别操作的（下同），α为初始步长，引入$\varepsilon\mathbf{1}_n$这一项是为了防止除零运算．

AdaGrad

- 可以看到AdaGrad的步长大致反比于历史梯度累计值的算术平方根，所以梯度较大时步长下降很快，反之则下降较慢，这样做的效果是在参数空间更平缓的方向上，前后两次迭代的距离较大.
- 在凸优化问题中AdaGrad有比较好的理论性质，但实际应用中也发现在训练深度神经网络模型时，从训练开始就积累梯度平方会导致步长过早或过多减小.
- 如果在AdaGrad中使用真实梯度$\nabla f(x^k)$，那么AdaGrad也可以看成是一种介于一阶和二阶的优化算法.
- 考虑$f(x)$在点x^k处的二阶泰勒展开：

$$f(x) \approx f(x^k) + \nabla f(x^k)^\top (x - x^k) + \frac{1}{2}(x - x^k)^\top B^k (x - x^k),$$

AdaGrad是使用一个对角矩阵来作为B^k:

$$B^k = \frac{1}{\alpha}\mathrm{Diag}(\sqrt{G^k + \varepsilon \mathbf{1}_n})$$

RMSProp

- RMSProp（root mean square propagation）是对AdaGrad的一个改进，该方法在非凸问题上可能表现更好．AdaGrad会累加之前所有的梯度分量平方，这就导致步长是单调递减的，因此在训练后期步长会非常小，计算的开销也较大．
- RMSProp提出只需使用离当前迭代点比较近的项，同时引入衰减参数ρ．具体地，令

$$M^{k+1} = \rho M^k + (1-\rho) g^{k+1} \odot g^{k+1},$$

再对其每个分量分别求根，就得到均方根(root mean square)

$$R^k = \sqrt{M^k + \varepsilon \mathbf{1}_n}, \tag{18}$$

最后将均方根的倒数作为每个分量步长的修正．

RMSProp

- RMSProp迭代格式为：

$$x^{k+1} = x^k - \frac{\alpha}{\sqrt{M^k + \varepsilon \mathbf{1}_n}} \odot g^k, \tag{19}$$

$$M^{k+1} = \rho M^k + (1-\rho) g^{k+1} \odot g^{k+1}. \tag{20}$$

  引入参数ε同样是为了防止分母为0的情况发生．一般取$\rho = 0.9$, $\alpha = 0.001$．
- 可以看到RMSProp 和AdaGrad 的唯一区别是将G^k替换成了M^k．

Adam

- Adam选择了一个动量项进行更新：

$$S^k = \rho_1 S^{k-1} + (1-\rho_1) g^k.$$

- 类似RMSProp，Adam也会记录梯度的二阶矩：

$$M^k = \rho_2 M^{k-1} + (1-\rho_2) g^k \odot g^k.$$

- 与原始动量方法和RMSProp的区别是，由于S^k和M^k本身带有偏差，Adam在更新前先对其进行修正：

$$\hat{S}^k = \frac{S^k}{1-\rho_1^k}, \quad \hat{M}^k = \frac{M^k}{1-\rho_2^k},$$

  这里ρ_1^k, ρ_2^k分别表示ρ_1, ρ_2的k次方.

- Adam最终使用修正后的一阶矩和二阶矩进行迭代点的更新.

$$x^{k+1} = x^k - \frac{\alpha}{\sqrt{\hat{M}^k + \varepsilon \mathbf{1}_n}} \odot \hat{S}^k.$$

深度学习训练

- 数据准备
- 优化模型，网络架构
- 梯度计算：前向/后向传播
- 初始化
- 步长/学习率等参数调试
- 训练与测试，交叉检验
- 梯度消失，梯度爆炸
- 网络架构与算法的适配调整

Outline

1. Problem Description
2. Stochastic Gradient Methods
3. Convergence Analysis
4. Variance Reduction
5. Natural Gradient Method

随机变量收敛性与概率不等式

考虑随机变量序列$\{X_n\}_{n=1}^{\infty}$

- 几乎必然收敛(概率1收敛): $P(\lim_{n\to\infty} X_n = X) = 1$
- 依概率收敛: 对于任意$\epsilon > 0$, $\lim_{n\to\infty} P(|X_n - X| \geq \epsilon) = 0$
- 依分布收敛: 对于所有的α, $P(X_n \leq \alpha) \to P(X \leq \alpha)$
- 强大数定理
- 中心极限定理
- 马尔可夫不等式，切比雪夫不等式
- Hoeffding 不等式
- Azuma-Hoeffding 不等式

收敛性：凸函数

随机次梯度算法: $x^{k+1} = x^k - \alpha_k g^k, \quad g^k \in \partial f_{s_k}(x^k)$

- 每个$f_i(x)$是闭凸函数，存在次梯度且无偏，即$\mathbb{E}[g^k|x^k] \in \partial f(x^k)$
- 随机次梯度二阶矩是一致有界的，即存在M，对任意的$x \in \mathbb{R}^n$以及随机下标s_k，有

$$\mathbb{E}_{s_k}[\|g^k\|^2] \le M^2 < +\infty, \quad g^k \in \partial f_{s_k}(x^k);$$

- $\{x^k\}$处处有界，即$\|x^k - x^*\| \le R, \forall K$，其中$x^*$是问题的最优解.

引理

在上述假设下，令$\{\alpha_k\}$是任一正步长序列，$\{x^k\}$是由随机次梯度法产生的序列，那么对所有的$K \ge 1$，有

$$\sum_{k=1}^{K} \alpha_k \mathbb{E}[f(x^k) - f(x^*)] \le \frac{1}{2}\mathbb{E}[\|x^1 - x^*\|^2] + \frac{1}{2}\sum_{k=1}^{K} \alpha_k^2 M^2. \tag{21}$$

引理的证明

- 令$\bar{g}^k = \mathbb{E}[g^k|x^k]$, $\xi^k = g^k - \bar{g}^k$.
- 由随机次梯度法的性质，

$$\bar{g}^k = \mathbb{E}[g^k|x^k] \in \partial f(x^k),$$

- 由次梯度的性质，

$$\langle \bar{g}^k, x^* - x^k \rangle \le f(x^*) - f(x^k).$$

可以推导得

$$\begin{aligned}
& \|x^{k+1} - x^*\|^2 \\
= & \|x^k - \alpha_k g^k - x^*\|^2 \\
= & \|x^k - x^*\|^2 + 2\alpha_k \langle g^k, x^* - x^k \rangle + \alpha_k^2 \|g^k\|^2 \\
= & \|x^k - x^*\|^2 + 2\alpha_k \langle \bar{g}^k, x^* - x^k \rangle + \alpha_k^2 \|g^k\|^2 + 2\alpha_k \langle \xi^k, x^* - x^k \rangle \\
\le & \|x^k - x^*\|^2 + 2\alpha_k (f(x^*) - f(x^k)) + \alpha_k^2 \|g^k\|^2 + 2\alpha_k \langle \xi^k, x^* - x^k \rangle .
\end{aligned} \tag{22}$$

- 注意到$\mathbb{E}[\xi^k | x^k] = 0$，所以

$$\mathbb{E}[\langle \xi^k, x^* - x^k \rangle] = \mathbb{E}[\mathbb{E}[\langle \xi^k, x^* - x^k \rangle | x_k]] = 0.$$

- 对不等式(22)两端求期望就得到

$$\alpha_k \mathbb{E}[f(x^k) - f(x^*)] \le \frac{1}{2}\mathbb{E}[\|x^k - x^*\|^2] - \frac{1}{2}\mathbb{E}[\|x^{k+1} - x^*\|^2] + \frac{\alpha_k^2}{2} M^2. \quad (23)$$

- 两边对k求和即得证.

随机次梯度算法的收敛性1

$$\sum_{k=1}^{K}\alpha_k\mathbb{E}[f(x^k)-f(x^*)]\le\frac{1}{2}\mathbb{E}[\|x^1-x^*\|^2]+\frac{1}{2}\sum_{k=1}^{K}\alpha_k^2M^2.$$

我们很容易得到随机次梯度算法在收缩步长下的收敛性.

定理 (随机次梯度算法的收敛性1)

在收敛性假设的条件下，令$A_K=\sum_{i=1}^{K}\alpha_i$，定义$\bar{x}_K=\frac{1}{A_K}\sum_{k=1}^{K}\alpha_k x^k$，则

$$\mathbb{E}[f(\bar{x}_K)-f(x^*)]\le\frac{R^2+\sum\limits_{k=1}^{K}\alpha_k^2M^2}{2\sum\limits_{k=1}^{K}\alpha_k}. \tag{24}$$

定理的证明

- 由$f(x)$的凸性以及引理1得到

$$\begin{aligned}
&A_k\mathbb{E}[f(\bar{x}_K)-f(x^*)]\\
\leq&\sum_{k=1}^{K}\alpha_k\mathbb{E}[f(x^k)-f(x^*)]\\
\leq&\frac{1}{2}\mathbb{E}[\|x^1-x^*\|^2]+\frac{1}{2}\sum_{k=1}^{K}\alpha_k^2M^2\\
=&\frac{R^2+\sum_{k=1}^{K}\alpha_k^2M^2}{2}
\end{aligned}$$

- 不等式两边同除以A_K得到

$$\mathbb{E}[f(\bar{x}_K)-f(x^*)]\leq\frac{R^2+\sum\limits_{k=1}^{K}\alpha_k^2M^2}{2A_K}.$$

随机次梯度算法的收敛性1

- 从定理1可以看到，当

$$\sum_{k=1}^{\infty} \alpha_k = +\infty, \quad \frac{\sum_{k=1}^{K} \alpha_k^2}{\sum_{k=1}^{K} \alpha_k} \to 0$$

  时，随机次梯度算法收敛.

- 对一个固定的步长α，不等式(24)右侧有一个不随K递减的常数，因此固定步长随机次梯度算法在函数值取期望意义下是不收敛的，它仅仅能找到一个次优解：

$$\mathbb{E}[f(\bar{x}_K) - f(x^*)] \le \frac{R^2}{2K\alpha} + \frac{\alpha M^2}{2}.$$

  特别地，对于给定的迭代次数K，选取固定步长$\alpha = \frac{R}{M\sqrt{K}}$，可以达到$\mathcal{O}(1/\sqrt{K})$的精度，即

$$\mathbb{E}[f(\bar{x}_K) - f(x^*)] \le \frac{RM}{\sqrt{K}}.$$

随机次梯度算法的收敛性2

在步长不增的情况下，我们可以得到直接平均意义下的收敛性.

定理 (随机次梯度算法的收敛性2)

在收敛性假设的条件下，令$\{\alpha_k\}$是一个不增的正步长序列，$\bar{x}_K = \frac{1}{K}\sum_{k=1}^{K} x^k$，则

$$\mathbb{E}[f(\bar{x}_K) - f(x^*)] \leq \frac{R^2}{2K\alpha_K} + \frac{1}{2K}\sum_{k=1}^{K} \alpha_k M^2. \tag{25}$$

定理的证明

- 对(23)式两边同除α_k，就有

$$\mathbb{E}[f(x^k)-f(x^*)] \le \frac{1}{2\alpha_k}\mathbb{E}[\|x^k-x^*\|_2^2] - \frac{1}{2\alpha_k}\mathbb{E}[\|x^{k+1}-x^*\|_2^2] + \frac{\alpha_k}{2}M^2.$$

- 再对k求和，并且利用$f(x)$的凸性和α_k的单调性得

$$\begin{aligned}
\mathbb{E}[f(\bar{x}_K)-f(x^*)] &\le \frac{1}{K}\sum_{k=1}^{K}\mathbb{E}[f(x^k)-f(x^*)] \\
&\le \frac{1}{2K}\left(\frac{1}{\alpha_1}\mathbb{E}[\|x^1-x^*\|^2] + \sum_{k=1}^{K}\alpha_k M^2 + \right. \\
&\left. \sum_{k=2}^{K}\left(\frac{1}{\alpha_k}-\frac{1}{\alpha_{k-1}}\right)\mathbb{E}[\|x^k-x^*\|^2\right) \\
&\le \frac{R^2}{2K\alpha_K} + \frac{1}{2K}\sum_{k=1}^{K}\alpha_k M^2.
\end{aligned}$$

随机次梯度算法的收敛性2

- 注意该定理和定理1 的不同之处在于$\bar{x}_K$的定义.
- 通过选取$\mathcal{O}(1/\sqrt{k})$阶数的步长，我们可以得到目标函数的收敛速度为$\mathcal{O}(1/\sqrt{k})$：

推论

在收敛性假设的条件下，令$\alpha_k = \frac{R}{M\sqrt{k}}$，则

$$\mathbb{E}[f(\bar{x}_K) - f(x^*)] \leq \frac{3RM}{2\sqrt{K}}. \tag{26}$$

其中$\bar{x}_K$的定义和定理2 相同.

推论的证明

- 注意到
$$\sum_{k=1}^{K}\frac{1}{\sqrt{k}} \le \int_0^K \frac{1}{\sqrt{t}}\mathrm{d}t = 2\sqrt{K}.$$
- 将$\alpha_k = \frac{R}{M\sqrt{k}}$代入式(25)就得到
$$\mathbb{E}[f(\bar{x}_K) - f(x^*)] \le \frac{R^2}{2K\frac{R}{M\sqrt{K}}} + \frac{RM}{2K}2\sqrt{K} = \frac{3RM}{2\sqrt{K}}.$$
- 我们可以发现随机次梯度算法和非随机次梯度算法具有相同的收敛速度——$\mathcal{O}(1/\sqrt{k})$.
- 随机次梯度算法每步的计算代价远小于非随机次梯度，这一定程度上解释了为什么随机算法在一些问题中的表现要远远好于非随机算法.

随机次梯度算法的收敛性3

下面主要讨论随机次梯度算法在依概率意义下的收敛性和收敛速度.

定理

选择上述推论中的步长α_k，使得$\mathbb{E}[f(\bar{x}_K)-f(x^*)]\to 0$，那么我们有依概率收敛$f(\bar{x}_K)-f(x^*)\xrightarrow{P}0\ (K\to\infty)$，即对任意的$\varepsilon>0$，都有

$$\lim_{K\to\infty} P(f(\bar{x}_K)-f(x^*)\geq\varepsilon)=0. \tag{27}$$

- 由马尔可夫不等式立即得到

$$P(f(\bar{x}_K)-f(x^*)\geq\varepsilon)\leq\frac{1}{\varepsilon}\mathbb{E}[f(\bar{x}_K)-f(x^*)]\to 0.$$

随机变量的收敛性

- 称随机变量序列$\{X_n\}_{n=1}^{\infty}$几乎必然收敛到X，如果
$$P\left(\lim_{n\to\infty} X_n = X\right) = 1;$$
- 称随机变量序列$\{X_n\}_{n=1}^{\infty}$依概率收敛到X，如果
$$\lim_{n\to\infty} P(|X_n - X| \geq \varepsilon) = 0.$$

定理 (随机次梯度算法的收敛性3)

在收敛性假设的条件下，进一步假设对于所有的随机次梯度g，有$\|g\| \leq M$. 那么对任意的$\varepsilon > 0$，

$$f(\bar{x}_K) - f(x^*) \leq \frac{R^2}{2K\alpha_K} + \frac{1}{2K}\sum_{k=1}^{K}\alpha_k M^2 + \frac{RM}{\sqrt{K}}\varepsilon \tag{28}$$

以大于等于$1 - e^{-\frac{1}{2}\varepsilon^2}$的概率成立，其中步长列$\{\alpha_k\}$是单调不增序列，$\bar{x}_K$的定义和定理2 中的定义相同.

鞅差序列

定义

设$\{X_n\}_{n=1}^{\infty}$与$\{Z_n\}_{n=1}^{\infty}$为$(\Omega, \mathcal{F}, P)$上的随机过程．如果$\forall n \in \mathbb{N}_+$，

1. $\mathbb{E}[|X_n|] < +\infty$；
2. X_n属于$Z_1, \ldots, Z_n$生产的σ-代数；
3. $\mathbb{E}[X_{n+1}|Z_1, \ldots, Z_n] = 0$．

则称$\{X_n\}_{n=1}^{\infty}$为关于$\{Z_n\}_{n=1}^{\infty}$的鞅差序列．特别地，
若$\{Z_n\}_{n=1}^{\infty} = \{X_n\}_{n=1}^{\infty}$，则称$\{X_n\}_{n=1}^{\infty}$为鞅差序列．

- 鞅差序列的例子如：设$\{Z_n\}_{n=1}^{\infty}$为一维简单随机游走，$\{X_n\}_{n=1}^{\infty}$定义为$X_n = Z_n - Z_{n-1}$（补充定义$Z_0 = 0$）．
- 由一维简单随机游走的性质，X_n以$1/2$的概率为1，以$1/2$的概率为-1．因此$\mathbb{E}[|X_n|] = 1 < +\infty$；
- X_n由Z_{n-1}及Z_n完全决定，因此属于$Z_1, \ldots, Z_n$生成的σ-代数；
- 由一维简单随机游走的性质，$X_{n+1} = Z_{n+1} - Z_n$独立于$Z_1, \ldots, Z_n$，所以$\mathbb{E}[X_{n+1}|Z_1, \ldots, Z_n] = \mathbb{E}[X_{n+1}] = 0$.

Azuma-Hoeffding不等式

下面给出Azuma-Hoeffding不等式，即鞅版本的Hoeffding不等式：

引理

设$\{X_n\}_{n=1}^{\infty}$为鞅差序列，且$\forall n \in \mathbb{N}_+, |X_n| \le B$. 则$\forall t > 0$，

$$P\left(\sum_{i=1}^{n} X_i \ge t\right) \le \exp\left(-\frac{2t^2}{nB^2}\right);$$

$$P\left(\sum_{i=1}^{n} X_i \le -t\right) \le \exp\left(-\frac{2t^2}{nB^2}\right).$$

定理的证明

- 令$\bar{g}^k = \mathbb{E}[g^k | x^k]$, $\xi^k = g^k - \bar{g}^k$．由(22)式的推导过程我们已经得到

$$f(x^k) - f(x^*) \le \frac{1}{2\alpha_k}\|x^k - x^*\|^2 - \frac{1}{2\alpha_k}\|x^{k+1} - x^*\|^2 + \frac{\alpha_k}{2}\|g^k\|^2 + \langle \xi^k, x^* - x^k \rangle.$$

- 利用$f(x)$的凸性与α_k的单调性有

$$\begin{aligned} f(\bar{x}_K) - f(x^*) &\le \frac{1}{K}\sum_{k=1}^{K} f(x^k) - f(x^*) \\ &\le \frac{R^2}{2K\alpha_K} + \frac{1}{2K}\sum_{k=1}^{K}\alpha_k\|g^k\|^2 + \frac{1}{K}\sum_{k=1}^{K}\langle \xi^k, x^* - x^k \rangle \\ &\le \frac{R^2}{2K\alpha_K} + \frac{1}{2K}\sum_{k=1}^{K}\alpha_k M^2 + \frac{1}{K}\sum_{k=1}^{K}\langle \xi^k, x^* - x^k \rangle. \end{aligned}$$

- 令

$$\omega = \frac{R^2}{2K\alpha_K} + \frac{1}{2K}\sum_{k=1}^{K}\alpha_k M^2$$

得到

$$P(f(\bar{x}_K) - f(x^*) - \omega \geq t) \leq P\left(\frac{1}{K}\sum_{k=1}^{K}\langle \xi^k, x^* - x^k\rangle \geq t\right). \tag{29}$$

- 设$Z^k = (x^1, x^2, \cdots, x^{k+1})$. 因为

$$\mathbb{E}[\xi^k | Z^{k-1}] = \mathbb{E}[\xi^k | x^k] = 0, \quad \mathbb{E}[x^k | Z^{k-1}] = x^k,$$

我们知道$\langle \xi^k, x^* - x^k\rangle$是一个鞅差序列.

- 同时由

$$\|\xi^k\|_2 = \|g^k - \overline{g}^k\|_2 \leq 2M$$

推出

$$|\langle \xi^k, x^* - x^k \rangle| \leq \|\xi^k\|\|x^* - x^k\|_2 \leq 2MR$$

即$\langle \xi_k, x^* - x_k \rangle$有界.

- 由Azuma-Hoeffding不等式得到

$$P\left(\frac{1}{K}\sum_{k=1}^{K}\langle \xi^k, x^* - x^k \rangle \geq t\right) \leq \exp\left(-\frac{Kt^2}{2M^2R^2}\right)$$

- 将$t = \frac{MR\varepsilon}{\sqrt{K}}$代入，有

$$P\left(\frac{1}{K}\sum_{k=1}^{K}\langle \xi^k, x^* - x^k \rangle \geq \frac{MR\varepsilon}{\sqrt{K}}\right) \leq \exp\left(-\frac{\varepsilon^2}{2}\right).$$

结合(29)式，定理得证.

随机次梯度算法的收敛性3

- 如果取$\alpha_k = \frac{R}{\sqrt{kM}}$，并令$\delta = e^{-\frac{1}{2}\varepsilon^2}$，就有

$$P\left(f(\bar{x}_K) - f(x^*) \le \frac{3RM}{2\sqrt{K}} + \frac{RM\sqrt{2\ln 1/\delta}}{\sqrt{K}}\right) \ge 1 - \delta. \tag{30}$$

- 可以看到除一个很小的概率外，函数值以$\mathcal{O}\left(\frac{1}{\sqrt{K}}\right)$的速度收敛.

随机梯度算法的收敛性：强凸函数

1. $f(x)$是可微函数，每个$f_i(x)$梯度存在；
2. $f(x)$是梯度利普希茨连续的，相应常数为L；
3. $f(x)$是强凸函数，强凸参数为μ；
4. 随机梯度二阶矩是一致有界的，即存在M，对任意的$x \in \mathbb{R}^n$以及随机下标s^k，有

$$\mathbb{E}_{s_k}[\|\nabla f_{s_k}(x)\|^2] \le M^2 < +\infty.$$

定理 (随机梯度算法的收敛性)

在收敛性假设的条件下，定义$\Delta_k = \|x^k - x^*\|$. 对固定的步长$\alpha_k = \alpha$，$0 < \alpha < \frac{1}{2\mu}$，有

$$\mathbb{E}[f(x^{K+1}) - f(x^*)] \le \frac{L}{2}\mathbb{E}[\Delta_{K+1}^2] \le \frac{L}{2}\left[(1-2\alpha\mu)^K\Delta_1^2 + \frac{\alpha M^2}{2\mu}\right]. \qquad (31)$$

定理的证明

- 根据随机梯度算法的更新公式，

$$\begin{aligned}\Delta_{k+1}^2 &= \|x^{k+1} - x^*\|^2 = \|x^k - \alpha_k \nabla f_{s_k}(x^k) - x^*\|^2 \\ &= \|x^k - x^*\|^2 - 2\alpha_k \langle \nabla f_{s_k}(x^k), x^k - x^* \rangle + \alpha_k^2 \|\nabla f_{s_k}(x^k)\|^2 \\ &= \Delta_k^2 - 2\alpha_k \langle \nabla f_{s_k}(x^k), x^k - x^* \rangle + \alpha_k^2 \|\nabla f_{s_k}(x^k)\|^2,\end{aligned}$$

- 由条件期望的性质$\mathbb{E}[X] = \mathbb{E}[\mathbb{E}[X|Y]]$，有

$$\begin{aligned}&\mathbb{E}_{s_1,s_2,\cdots,s_k}[\langle \nabla f_{s_k}(x^k), x^k - x^* \rangle] \\ =& \mathbb{E}_{s_1,s_2,\cdots,s_{k-1}}[\mathbb{E}_{s_k}[\langle \nabla f_{s_k}(x^k), x^k - x^* \rangle | s_1, \cdots, s_{k-1}]] \\ =& \mathbb{E}_{s_1,s_2,\cdots,s_{k-1}}[\langle \mathbb{E}_{s_k}[\nabla f_{s_k}(x_k) | s_1, s_2, \cdots, s_{k-1}], x^k - x^* \rangle] \\ =& \mathbb{E}_{s_1,s_2,\cdots,s_{k-1}}[\langle \nabla f(x^k), x^k - x^* \rangle] \\ =& \mathbb{E}_{s_1,s_2,\cdots,s_k}[\langle \nabla f(x^k), x^k - x^* \rangle].\end{aligned}$$

- 根据强凸函数的单调性，

$$\left\langle \nabla f(x^k), x^k - x^* \right\rangle = \left\langle \nabla f(x^k) - \nabla f(x^*), x^k - x^* \right\rangle \geq \mu \|x^k - x^*\|^2.$$

- 由随机梯度二阶矩的一致有界性，

$$\mathbb{E}_{s_1,s_2,\cdots,s_k}[\Delta_{k+1}^2] \leq (1-2\alpha\mu)\mathbb{E}_{s_1,s_2,\cdots,s_k}[\Delta_k^2] + \alpha^2 M^2. \tag{32}$$

- 对k做归纳，就得到

$$\mathbb{E}_{s_1,s_2,\cdots,s_K}[\Delta_{K+1}^2] \leq (1-2\alpha\mu)^K \Delta_1^2 + [1-(1-2\alpha\mu)^K]\frac{\alpha M^2}{2\mu}.$$

由$0 < 2\alpha\mu < 1$可知

$$\mathbb{E}_{s_1,s_2,\cdots,s_K}[\Delta_{K+1}^2] \leq (1-2\alpha\mu)^K \Delta_1^2 + \frac{\alpha M^2}{2\mu}. \tag{33}$$

- 利用梯度L-利普希茨连续函数的二次上界，可以得到

$$f(x^{K+1})-f(x^*)\leq\left\langle\nabla f(x^*),x^{K+1}-x^*\right\rangle+\frac{L}{2}\|x^{K+1}-x^*\|^2.$$

- 利用$\nabla f(x^*)=0$并对上式左右两边取期望可得

$$\mathbb{E}[f(x^{K+1})-f(x^*)]\leq\frac{L}{2}\mathbb{E}[\Delta_{K+1}^2]\leq\frac{L}{2}\left[(1-2\alpha\mu)^K\Delta_1^2+\frac{\alpha M^2}{2\mu}\right].$$

随机梯度算法的收敛性

下面的定理表明，如果设置递减的步长，收敛阶可以达到$\mathcal{O}(1/K)$.

定理

随机梯度算法的收敛速度 在上述定理的结果中，在收敛性假设的条件下，取递减的步长

$$\alpha_k = \frac{\beta}{k+\gamma},$$

其中$\beta > \frac{1}{2\mu}, \gamma > 0$，使得$\alpha_1 \leq \frac{1}{2\mu}$，那么对于任意的$k \geq 1$，都有

$$\mathbb{E}[f(x^k) - f(x^*)] \leq \frac{L}{2}\mathbb{E}[\Delta_k^2] \leq \frac{L}{2}\frac{v}{\gamma+k}, \tag{34}$$

这里

$$v = \max\left\{\frac{\beta^2 M^2}{2\beta\mu - 1}, (\gamma+1)\Delta_1^2\right\}.$$

定理的证明

- 之前的定理已经证明了

$$\mathbb{E}_{s_1,s_2,\cdots,s_k}[\Delta_{k+1}^2] \le (1-2\alpha_k\mu)\mathbb{E}_{s_1,s_2,\cdots,s_k}[\Delta_k^2] + \alpha_k^2 M^2.$$

- 用数学归纳法证明(34)式．由v的定义知$k=1$时(34)式成立．
- 现假设该式对k成立，定义$\hat{k}=\gamma+k$，则$\alpha_k=\beta/\hat{k}$．由归纳假设，

$$\begin{aligned}\mathbb{E}[\Delta_{k+1}^2] &\le \left(1-\frac{2\beta\mu}{\hat{k}}\right)\frac{v}{\hat{k}}+\frac{\beta^2M^2}{\hat{k}^2}\\ &= \frac{\hat{k}-1}{\hat{k}^2}v-\frac{2\beta\mu-1}{\hat{k}^2}v+\frac{\beta^2M^2}{\hat{k}^2}\\ &\le \frac{v}{\hat{k}+1}\end{aligned}$$

最后一个不等式用到了v的定义．所以(34)式对$k+1$也成立．

随机梯度算法的收敛性

上述定理表明对于强凸函数，随机梯度下降法的收敛速度可以达到$\mathcal{O}(1/K)$. 对于一般的凸函数随机梯度算法也有一定的收敛性，为此我们在下表比较随机算法和普通算法的复杂度.

Table: 梯度下降法的算法复杂度

	f凸(次梯度算法)	f可微强凸	f可微强凸且L-光滑
随机算法	$\mathcal{O}\left(\frac{1}{\varepsilon^2}\right)$	$\mathcal{O}\left(\frac{1}{\varepsilon}\right)$	$\mathcal{O}\left(\frac{1}{\varepsilon}\right)$
普通算法	$\mathcal{O}\left(\frac{N}{\varepsilon^2}\right)$	$\mathcal{O}\left(\frac{N}{\varepsilon}\right)$	$\mathcal{O}\left(N\ln\left(\frac{1}{\varepsilon}\right)\right)$

Outline

1. Problem Description
2. Stochastic Gradient Methods
3. Convergence Analysis
4. Variance Reduction
5. Natural Gradient Method

梯度下降法与随机梯度下降法的比较

下面分析梯度下降法与随机梯度下降法的主要区别：

- 在强凸性假设下，对梯度下降法有

$$
\begin{aligned}
\Delta_{k+1}^2 &= \|x^{k+1} - x^*\|^2 = \|x^k - \alpha\nabla f(x^k) - x^*\|^2 \\
&= \Delta_k^2 - 2\alpha\langle\nabla f(x^k), x^k - x^*\rangle + \alpha^2\|\nabla f(x^k)\|^2 \\
&\le (1 - 2\alpha\mu)\Delta_k^2 + \alpha^2\|\nabla f(x^k)\|_2^2 \qquad (\mu\text{-强凸}) \\
&\le (1 - 2\alpha\mu + \alpha^2L^2)\Delta_k^2. \qquad (L\text{-光滑})
\end{aligned} \tag{35}
$$

梯度下降法与随机梯度下降法的比较

- 对随机梯度下降法，利用条件期望的性质有

$$\begin{aligned}\mathbb{E}[\Delta_{k+1}^2] &= \mathbb{E}[\|x^{k+1}-x^*\|_2^2] = \mathbb{E}[\|x^k-\alpha\nabla f_{s_k}(x^k)-x^*\|]^2 \\ &= \mathbb{E}[\Delta_k^2]-2\alpha\mathbb{E}[\langle\nabla f_{s_k}(x^k), x^k-x^*\rangle]+\alpha^2\mathbb{E}[\|\nabla f_{s_k}(x^k)\|^2] \\ &= \mathbb{E}[\Delta_k^2]-2\alpha\mathbb{E}[\langle\nabla f(x^k), x^k-x^*\rangle]+\alpha^2\mathbb{E}[\|\nabla f_{s_k}(x^k)\|^2] \\ &\le (1-2\alpha\mu)\mathbb{E}[\Delta_k^2]+\alpha^2\mathbb{E}[\|\nabla f_{s_k}(x^k)\|^2] \qquad (\mu\text{-强凸}) \\ &= (1-2\alpha\mu)\mathbb{E}[\Delta_k^2]+\alpha^2\mathbb{E}[\|\nabla f_{s_k}(x^k)-\nabla f(x^k)+\nabla f(x^k)\|^2] \\ &\le \underbrace{(1-2\alpha\mu+\alpha^2L^2)\mathbb{E}[\Delta_k^2]}_{A}+\underbrace{\alpha^2\mathbb{E}[\|\nabla f_{s_k}(x^k)-\nabla f(x^k)\|^2]}_{B}.\end{aligned} \tag{36}$$

- 可以看到两种算法的主要差别就在B项上，也就是梯度估计的某种方差．它导致了随机梯度算法只能有$\mathcal{O}(1/k)$的收敛速度．

方差减小技术

- 在许多机器学习的应用中，随机梯度算法的收敛速度更快一些.
- 这主要是因为许多应用对解的精度要求不太高，而在开始部分方差较小，即有$B \ll A$，那么我们会观察到近似Q-线性收敛速度；
- 而随着迭代步数增多，方差增大，最终的收敛速度为$\mathcal{O}(1/k)$.
- 为了能获得比较快的渐进收敛速度，我们的主要目标即减少方差项B. 下面介绍三种减小方差的算法：
  - SAG (stochastic average gradient)
  - SAGA
  - SVRG (stochastic variance reduced gradient)

SAG算法

- 当迭代接近收敛时，上一步的随机梯度也是当前迭代点处梯度的一个很好的估计．随机平均梯度法(SAG)就是基于这一想法．
- 在迭代中，SAG算法记录所有之前计算过的随机梯度，再与当前新计算的随机梯度求平均，最终作为下一步的梯度估计．
- 具体来说，SAG算法在内存中开辟了存储N个随机梯度的空间

  $$[g_1^k, g_2^k, \cdots, g_N^k],$$

  分别用于记录和第i个样本相关的最新的随机梯度．在第k步更新时，若抽取的样本点下标为s_k，则计算随机梯度后将$g_{s_k}^k$的值更新为当前的随机梯度值，而其他未抽取到的下标对应的g_i^k保持不变．每次SAG 算法更新使用的梯度方向是所有g_i^k的平均值．

SAG算法

- SAG算法的迭代格式为

$$x^{k+1} = x^k - \frac{\alpha_k}{N} \sum_{i=1}^{N} g_i^k$$

其中g_i^k的更新方式为

$$g_i^k = \begin{cases} \nabla f_{s_k}(x^k), & i = s_k, \\ g_i^{k-1}, & \text{其他}, \end{cases} \tag{37}$$

- 每次迭代只有一个g_i^k发生了改变．因此SAG迭代公式还可以写成

$$x^{k+1} = x^k - \alpha_k \left(\frac{1}{N}(\nabla f_{s_k}(x^k) - g_{s_k}^{k-1}) + \frac{1}{N} \sum_{i=1}^{N} g_i^{k-1} \right), \tag{38}$$

- $\{g_i^k\}$的初值可简单地取为0或中心化的随机梯度向量，
- SAG算法每次使用的随机梯度的条件期望并不是真实梯度$\nabla f(x^k)$，但随着迭代进行，随机梯度的期望和真实梯度的偏差会越来越小．

SAGA算法

- SAGA算法的迭代方式为

$$x^{k+1} = x^k - \alpha_k \left(\nabla f_{s_k}(x^k) - g_{s_k}^{k-1} + \frac{1}{N} \sum_{i=1}^{N} g_i^{k-1} \right). \tag{39}$$

- 对比(38)式可以发现，SAGA算法去掉了$\nabla f_{s_k}(x^k) - g_{s_k}^{k-1}$前面的系数$1/N$. 可以证明每次迭代使用的梯度方向都是无偏的，即

$$\mathbb{E}\left[\nabla f_{s_k}(x^k) - g_{s_k}^{k-1} + \frac{1}{N} \sum_{i=1}^{N} g_i^{k-1} \;\middle|\; x^k \right] = \nabla f(x^k).$$

SAGA算法的收敛性

SAGA算法同样有Q-线性收敛速度：

定理 (SAGA算法的收敛性)

在强凸性收敛性假设的条件下，取固定步长$\alpha_k = \frac{1}{2(\mu N+L)}$．定义$\Delta_k = \|x^k - x^*\|$，则对任意的$k \geq 1$有

$$\mathbb{E}[\Delta_k^2] \leq \left(1 - \frac{\mu}{2(\mu N + L)}\right)^k \left(\Delta_1^2 + \frac{N(f(x^1) - f(x^*))}{\mu N + L}\right). \tag{40}$$

如果强凸的参数μ是未知的，也可以取$\alpha = \frac{1}{3L}$，有类似的收敛结果．

SVRG算法

- 与SAG算法和SAGA算法不同，SVRG算法通过周期性缓存全梯度的方法来减小方差.
- 具体做法是在随机梯度下降方法中，每经过m次迭代就设置一个检查点，计算一次全梯度，在之后的m次迭代中，将这个全梯度作为参考点来达到减小方差的目的.
- 令$\tilde{x}^j$是第j个检查点，则我们需要计算点$\tilde{x}^j$处的全梯度

$$\nabla f(\tilde{x}^j) = \frac{1}{N}\sum_{i=1}^{N} \nabla f_i(\tilde{x}^j),$$

在之后的迭代中使用方向v^k作为更新方向：

$$v^k = \nabla f_{s_k}(x^k) - (\nabla f_{s_k}(\tilde{x}^j) - \nabla f(\tilde{x}^j)), \tag{41}$$

其中$s_k \in \{1,2,\cdots,N\}$是随机选取的一个样本.

SVRG算法

- 注意到给定$s_1, s_2, \cdots, s_{k-1}$时$x^k, \tilde{x}^j$均为定值，由$v^k$的表达式可知

$$\begin{aligned}
&\mathbb{E}[v^k|s_1, s_2, \cdots, s_{k-1}] \\
=&\mathbb{E}[\nabla f_{s_k}(x^k)|x^k] - \mathbb{E}[\nabla f_{s_k}(\tilde{x}^j) - \nabla f(\tilde{x}^j)|s_1, s_2, \cdots, s_{k-1}] \\
=&\nabla f(x^k) - 0 = \nabla f(x^k),
\end{aligned}$$

- 公式(41)有简单的直观理解：我们希望用$\nabla f_{s_k}(\tilde{x}^j)$去估计$\nabla f(\tilde{x}^j)$，那么$\nabla f_{s_k}(\tilde{x}^j) - \nabla f(\tilde{x}^j)$就可以看作梯度估计的误差，所以在每一步随机梯度迭代用该项来对$\nabla f_{s_k}(x^k)$做一个校正．

SVRG算法

- 假设
$$\|\nabla f_i(x) - \nabla f_i(y)\| \le L\|x-y\|, \quad i = 1,2,\cdots,N.$$
- 令$y=\tilde{x}^j$，x^*为$f(x)$的最小值点，$\Delta_k = \|x^k - x^*\|$，则
$$\begin{aligned}
&\mathbb{E}\left[\|v^k\|^2\right] = \mathbb{E}\left[\|\nabla f_{s_k}(x^k) - (\nabla f_{s_k}(y) - \nabla f(y))\|^2\right] \\
=& \mathbb{E}\left[\|\nabla f_{s_k}(x^k) - \nabla f_{s_k}(y) + \nabla f(y) + \nabla f_{s_k}(x^*) - \nabla f_{s_k}(x^*)\|^2\right] \\
\le& 2\mathbb{E}\left[\|\nabla f_{s_k}(x^k) - \nabla f_{s_k}(x^*)\|^2\right] + 2\mathbb{E}\left[\|\nabla f_{s_k}(y) - \nabla f(y) - \nabla f_{s_k}(x^*)\|^2\right] \\
\le& 2L^2\mathbb{E}\left[\Delta_k^2\right] + 2\mathbb{E}\left[\|\nabla f_{s_k}(y) - \nabla f_{s_k}(x^*)\|^2\right] \\
\le& 2L^2\mathbb{E}\left[\Delta_k^2\right] + 2L^2\mathbb{E}\left[\|y - x^*\|^2\right].
\end{aligned} \tag{42}$$
- 其中第一个不等式是因为$\|a+b\|^2 \le 2\|a\|^2 + 2\|b\|^2$，第二个不等式使用了有关二阶矩的不等式
$$\mathbb{E}[\|\xi - \mathbb{E}\xi\|^2] \le \mathbb{E}[\|\xi\|^2].$$

SVRG算法的收敛性

下面给出SVRG算法的收敛性．这里的收敛性是针对参考点序列$\{\tilde{x}^j\}$而言的．

定理 (SVRG算法的收敛性)

设m为利用每个$\tilde{x}^j$更新的次数．设每个$f_i(x)$可微，且梯度L-利普希茨连续；函数$f(x)$强凸，强凸参数为μ．取步长$\alpha \in \left(0, \frac{1}{2L}\right]$，并且$m$充分大使得

$$\rho = \frac{1}{\mu\alpha(1-2L\alpha)m} + \frac{2L\alpha}{1-2L\alpha} < 1, \tag{43}$$

则*SVRG*算法对于参考点$\tilde{x}^j$在函数值期望的意义下有*Q*-线性收敛速度：

$$\mathbb{E}f(\tilde{x}^j) - f(x^*) \le \rho\mathbb{E}[f(\tilde{x}^{j-1}) - f(x^*)]. \tag{44}$$

定理的证明

- 定义$\Delta_k = \|x^k - x^*\|$.
- 对于内层循环，

$$\begin{aligned}\mathbb{E}[\Delta_{k+1}^2] = [\|x^{k+1} - x^*\|^2] &= \mathbb{E}[\|x^k - \alpha v^k - x^*\|^2] \\ &= \mathbb{E}[\Delta_k^2] - 2\alpha\mathbb{E}[\langle v^k, x^k - x^*\rangle] + \alpha^2\mathbb{E}[\|v^k\|^2] \\ &= \mathbb{E}[\Delta_k^2] - 2\alpha\mathbb{E}[\langle \nabla f(x^k), x^k - x^*\rangle] + \alpha^2\mathbb{E}[\|v^k\|^2] \\ &\le \mathbb{E}[\Delta_k^2] - 2\alpha\mathbb{E}[(f(x^k) - f(x^*))] + \alpha^2\mathbb{E}[\|v^k\|^2].\end{aligned}$$

- 构造辅助函数

$$\phi_i(x) = f_i(x) - f_i(x^*) - \nabla f_i(x^*)(x - x^*),$$

注意到$\phi_i(x)$也是凸函数且梯度L-利普希茨连续，因此有

$$\frac{1}{2L}\|\nabla\phi_i(x)\|^2 \le \phi_i(x) - \phi_i(x^*)$$

- 展开$\phi_i(x)$与$\nabla\phi_i(x)$的表达式可得

$$\|\nabla f_i(x)-\nabla f_i(x^*)\|^2\le 2L[f_i(x)-f_i(x^*)-\nabla f_i(x^*)^\top(x-x^*)].$$

- 对i从1到N进行求和，注意$\nabla f(x^*)=0$：

$$\frac{1}{N}\sum_{i=1}^{N}\|\nabla f_i(x)-\nabla f_i(x^*)\|^2\le 2L[f(x)-f(x^*)],\quad \forall x. \tag{45}$$

- 利用(42)式的推导过程可得

$$\mathbb{E}[\|v^k\|^2]\le 2\mathbb{E}[\|\nabla f_{s_k}(x^k)-\nabla f_{s_k}(x^*)\|^2]+2\mathbb{E}[\|\nabla f_{s_k}(\tilde{x}^{j-1})-\nabla f_{s_k}(x^*)\|^2].$$

对上式右侧第一项，有

$$\begin{aligned}&\mathbb{E}[\|\nabla f_{s_k}(x^k)-\nabla f_{s_k}(x^*)\|^2]\\=&\,\mathbb{E}[\mathbb{E}[\|\nabla f_{s_k}(x^k)-\nabla f_{s_k}(x^*)\|^2|s_1,s_2,\cdots,s_{k-1}]]\\=&\,\mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N}\|\nabla f_i(x^k)-\nabla f_i(x^*)\|^2\right]\le 2L\mathbb{E}[f(x^k)-f(x^*)],\end{aligned}$$

- 类似地，对右侧第二项，有

$$\mathbb{E}[\|\nabla f_{s_k}(\tilde{x}^{j-1}) - \nabla f_{s_k}(x^*)\|^2] \le 2L\mathbb{E}[f(\tilde{x}^{j-1}) - f(x^*)].$$

- 最终可得对$\mathbb{E}[\|v^k\|^2]$的估计：

$$\mathbb{E}[\|v^k\|^2] \le 4L(\mathbb{E}[f(x^k) - f(x^*)] + \mathbb{E}[f(\tilde{x}^{j-1}) - f(x^*)]).$$

- 将$\mathbb{E}[\|v^k\|^2]$的上界代入对$\mathbb{E}[\Delta_{k+1}^2]$的估计，就有

$$\begin{aligned}\mathbb{E}[\Delta_{k+1}^2] &\le \mathbb{E}[\Delta_k^2] - 2\alpha\mathbb{E}[f(x^k) - f(x^*)] + \alpha^2\mathbb{E}[\|v^k\|^2] \\ &\le \mathbb{E}[\Delta_k^2] - 2\alpha(1 - 2\alpha L)\mathbb{E}[f(x^k) - f(x^*)] \\ &+ 4L\alpha^2\mathbb{E}[f(\tilde{x}^{j-1}) - f(x^*)].\end{aligned}$$

- 对k从1到m求和，并且注意到$x^1 = \tilde{x}^{j-1}$就可以得到

$$\begin{aligned}
&\mathbb{E}[\Delta_{m+1}^2] + 2\alpha(1-2\alpha L)\sum_{k=1}^{m}\mathbb{E}[f(x^k) - f(x^*)] \\
\le\ &\mathbb{E}[\|\tilde{x}^{j-1} - x^*\|^2] + 4L\alpha^2 m\mathbb{E}[f(\tilde{x}^{j-1}) - f(x^*)] \\
\le\ &\frac{2}{\mu}\mathbb{E}[f(\tilde{x}^{j-1}) - f(x^*)] + 4L\alpha^2 m\mathbb{E}[f(\tilde{x}^{j-1}) - f(x^*)],
\end{aligned}$$

- 注意到$\tilde{x}^j = \frac{1}{m}\sum_{k=1}^m x^k$，所以

$$\begin{aligned}
&\mathbb{E}[f(\tilde{x}^j) - f(x^*)] \\
\le\ &\frac{1}{m}\sum_{k=1}^{m}\mathbb{E}[f(x^k) - f(x^*)] \\
\le\ &\frac{1}{2\alpha(1-2\alpha L)m}\left(\frac{2}{\mu} + 4mL\alpha^2\right)\mathbb{E}[f(\tilde{x}^{j-1}) - f(x^*)] \\
=\ &\rho\mathbb{E}[f(\tilde{x}^{j-1}) - f(x^*)].
\end{aligned}$$

Outline

1. Problem Description
2. Stochastic Gradient Methods
3. Convergence Analysis
4. Variance Reduction
5. Natural Gradient Method

常见记号约定

记号	含义
$\mathcal{S}$	数据标签集$\{x_i, y_i\}_{i=1}^{N}$
$\mathcal{S}_x$	数据集$\{x_i\}_{i=1}^{N}$
$\mathcal{S}_y$	标签集$\{y_i\}_{i=1}^{N}$
$\mathcal{Q}_{x,y}$	真实数据与标签的分布函数
$\mathcal{Q}_x$	真实数据的分布函数
$\mathcal{Q}_{y\|x}$	给定数据x后标签y的分布函数
$q(x, y)$	$\mathcal{Q}_{x,y}$的概率密度函数
$q(x)$	$\mathcal{Q}_x$的概率密度函数
$q(y\|x)$	$\mathcal{Q}_{y\|x}$的概率密度函数
$P_{x,y}(\theta)$	实际训练学习到的分布函数
$P_{y\|x}(\theta)$	给定参数θ与数据x后标签y的分布函数
$p(x, y\|\theta)$	$P_{x,y}(\theta)$的概率密度函数
$p(y\|x, \theta)$	$P_{y\|x}(\theta)$的概率密度函数
$R_{y\|z}$	给定神经网络输出z后标签y的分布函数
$r_{y\|z}$	$R_{y\|z}$的概率密度函数

概率统计模型

- 选择用KL散度来刻画学习到的分布函数$P_{x,y}(\theta)$与真实数据与标签的分布函数$Q_{x,y}$的距离

$$\begin{aligned}\mathsf{KL}(Q_{x,y}\|P_{x,y}(\theta)) &:= \int q(x,y)\log\frac{q(x,y)}{p(x,y|\theta)}dxdy\\ &= \int q(x,y)\log\frac{q(y|x)q(x)}{p(y|x,\theta)q(x)}dxdy\\ &= \int q(x)\int q(y|x)\log\frac{q(y|x)}{p(y|x,\theta)}dxdy\\ &= \mathbb{E}_{Q_x}[\mathsf{KL}(Q_{y|x}\|P_{y|x}(\theta))].\end{aligned}$$

- 实际情况用数据集的分布近似真正数据的分布

$$\mathbb{E}_{\hat{Q}_x}[\mathsf{KL}(\hat{Q}_{y|x}\|P_{y|x})] \propto -\frac{1}{|\mathcal{S}_x|}\sum_{(x,y)\in\mathcal{S}}\log(p(y|x,\theta)),$$

其中$\propto$表示在θ固定的情况并且省略常数的情况下左右两端关于θ成固定比例.

优化问题

- 优化问题：

$$\min_{\theta} \quad -\frac{1}{|\mathcal{S}_x|}\sum_{(x,y)\in\mathcal{S}} \log(p(y|x,\theta)). \tag{46}$$

- 设神经网络的输出为$h_\theta(x)$，并且将$P_{y|x}(\theta)$表示为神经网络输出$h_\theta(x)$与另一个概率分布输出$R_{y|h_\theta(x)}$的复合函数，即$P_{y|x}(\theta) = R_{y|h_\theta(x)}$.设$R_{y|h_\theta(x)}$的概率密度函数为$r(y|h_\theta(x))$，则对于任意的损失函数$\ell(y, h_\theta(x))$，可以定义$r(y|h_\theta(x)) \propto \exp(-\ell(y, h_\theta(x)))$.

- 如果
  设$|\mathcal{S}_x| = N$，$\ell(y, h_\theta(x)) \propto -\log(r(y|h_\theta(x))) = -\log(p(y|x,\theta))$，则与之前的优化模型一致．极小化经验风险等价于求解关于概率密度函数$p(y|x,\theta)$关于参数θ的极大似然估计问题．可以证明若$R_{y|h_\theta(x)}$为高维独立高斯分布，则其对应的损失函数为平方误差损失函数$\ell(y, h_\theta(x)) = \|h_\theta(x) - y\|^2$.

费希信息矩阵

- 对于概率分布函数$P_{x,y}(\theta)$，其费希信息矩阵定义为：

$$\mathbf{F} = \mathbb{E}_{P_{x,y}(\theta)}\left[\nabla \log p(x,y|\theta) \nabla \log p(x,y|\theta)^{\mathrm{T}}\right], \tag{47}$$

  其中∇为对θ求梯度.

- 根据等式

$$\nabla \log p(x,y|\theta) = \nabla \log p(y|x,\theta) + \nabla \log q(x) = \nabla \log p(y|x,\theta)$$

  所以FIM也可以写成

$$\mathbf{F} = \mathbb{E}_{Q_x}\left[\mathbb{E}_{P_{y|x}(\theta)}\left[\nabla \log p(y|x,\theta) \nabla \log p(y|x,\theta)^{\mathrm{T}}\right]\right].$$

- 定义函数$\log p(y|x,\theta)$关于θ的海瑟矩阵为$\mathbf{H}_{\log p(y|x,\theta)}$，则有：

$$\mathbb{E}_{P_{x,y}(\theta)}\left[\mathbf{H}_{\log p(y|x,\theta)}\right] = -\mathbf{F}. \tag{48}$$

自然梯度方向

- 对于任意损失函数$\nabla f(\theta)$梯度方向具有下面的性质：

$$\frac{-\nabla f(\theta)}{\|\nabla f(\theta)\|} = \lim_{\varepsilon \to 0} \frac{1}{\varepsilon} \operatorname*{argmin}_{d} f(\theta + d), \quad \text{s.t. } \|d\| \leq \varepsilon. \tag{49}$$

- 对于*KL*散度约束，我们有：

$$-\sqrt{2}\frac{\mathbf{F}^{-1}\nabla f}{\|\nabla f\|_{\mathbf{F}^{-1}}} = \lim_{\varepsilon \to 0} \frac{1}{\varepsilon} \operatorname*{argmin}_{\mathrm{KL}[P_{x,y}(\theta)\|P_{x,y}(\theta+d)]\leq\varepsilon^2} f(\theta + d). \tag{50}$$

- 经验费希信息矩阵$\hat{\mathbf{F}}$定义为

$$\begin{aligned} \hat{\mathbf{F}}(\theta) &= \mathbb{E}_{\hat{Q}_{x,y}}\left[\nabla \log p(y|x,\theta) \nabla \log p(y|x,\theta)^{\mathrm{T}}\right] \\ &= \frac{1}{N}\sum_{i=1}^{N} \nabla \log p(y_i|x_i,\theta) \nabla \log p(y_i|x_i,\theta)^{\mathrm{T}}. \end{aligned} \tag{51}$$

自然梯度法

Algorithm 1 自然梯度法

```
输入：目标函数$f$，初始参数$\theta^0, k = 0$.
while 未达到收敛准则 do
    计算梯度$\nabla_\theta f(\theta^k)$.
    计算费希信息矩阵$\mathbf{F}_k$(或者经验费希矩阵$\hat{\mathbf{F}}_k$).
    计算自然梯度方向$\tilde{\nabla}_\theta f(\theta^k) = \mathbf{F}_k^{-1} \nabla f(\theta)$(或者$\hat{\mathbf{F}}_k^{-1} \nabla f(\theta)$).
    更新参数$\theta^{k+1} = \theta^k - \alpha_k \tilde{\nabla}_\theta f(\theta^k)$，其中$\alpha_k$为第$k$步的步长.
    $k = k + 1$.
end while
```

KFAC

- 考虑ℓ层前馈全连接神经网络，设s_i为第i层经过权重矩阵作用后的输出，a_i为经过第i层激活函数ϕ_i的输出：

$$s_i = W_i \bar{a}_{i-1}, \quad a_i = \phi(s_i), \tag{52}$$

其中$i \in \{1, \cdots, \ell\}, \phi_i$为第$i$层的激活函数, W_i为权重矩阵.

- 定义神经网络的参数为$\theta = [\mathrm{vec}(W^{(1)})^{\mathrm{T}}, \mathrm{vec}(W^{(2)})^{\mathrm{T}}, \cdots, \mathrm{vec}(W^{(l)})^{\mathrm{T}}]$.
设$g^{(l)} = \frac{\partial \ell(y,\hat{y})}{\partial z^{(l)}} \cdot (a^{(l-1)})^{\mathrm{T}}$.

$$\mathcal{D}W^{(i)} := \frac{\partial \ell(y, \hat{y})}{\partial W^{(i)}} = g^{(i)} (\bar{a}^{(i-1)})^{\mathrm{T}}. \tag{53}$$

因此有$\mathcal{D}\theta = [\mathrm{vec}(\mathcal{D}W^{(1)})^{\mathrm{T}}, \mathrm{vec}(\mathcal{D}W^{(2)})^{\mathrm{T}}, \cdots, \mathrm{vec}(\mathcal{D}W^{(l)})^{\mathrm{T}}]^{\mathrm{T}}$.

- 由此可知费希信息矩阵的表达式为：

$$\mathbf{F} = \mathbb{E}[\mathcal{D}\theta \mathcal{D}\theta^{\mathrm{T}}] = [\mathbf{F}_{i,j}].$$

Kronecker product

- $A \otimes B$ denotes the Kronecker product between A and B:

$$A \otimes B \equiv \begin{bmatrix} [A]_{1,1}B & \cdots & [A]_{1,n}B \\ \vdots & \ddots & \vdots \\ [A]_{m,1}B & \cdots & [A]_{m,n}B \end{bmatrix}.$$

- $\mathrm{vec}(uv^\top) = v \otimes u$.
- $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$.
- $(B^\top \otimes A)\,\mathrm{vec}(X) = \mathrm{vec}(AXB)$
- $\mathrm{vec}(G_i A_i^\top) = (A_i \otimes G_i)\,\mathrm{vec}(I)$.
- $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$ for any A, B, C, D with correct sizes.

KFAC

- 近似：

$$\begin{aligned}\mathbf{F}_{i,j} &= \mathbb{E}[\mathrm{vec}(\mathcal{D}W^{(i)})\mathrm{vec}(\mathcal{D}W^{(j)})^{\mathrm{T}}] \\ &= \mathbb{E}[\bar{a}^{(i-1)}(\bar{a}^{(j-1)})^{\mathrm{T}} \otimes g^{(i)}(g^{(j)})^{\mathrm{T}}] \\ &\approx \mathbb{E}[\bar{a}^{(i-1)}(\bar{a}^{(j-1)})^{\mathrm{T}}] \otimes \mathbb{E}[g^{(i)}(g^{(j)})^{\mathrm{T}}] \\ &= \bar{A}_{i-1,j-1} \otimes G_{i,j} := \tilde{\mathbf{F}}_{i,j},\end{aligned}$$

其中$\tilde{\mathbf{F}}_{i,j}$是$\mathbf{F}_{i,j}$的近似，$\bar{A}_{i,j} := \mathbb{E}[\bar{a}^{(i)}(\bar{a}^{(j)})^{\mathrm{T}}], G_{i,j} := \mathbb{E}[g^{(i)}(g^{(j)})^{\mathrm{T}}]$.第三个关系式是用到了交换的性质.

- 由此得到第k步θ^{k+1}的更新方式为

$$\begin{aligned}\theta_i^{k+1} &= \theta_i^k - \alpha_k \hat{\mathbf{F}}_{ii}^{-1} g_k \\ &= \theta_i^k - \alpha_k (\bar{A}_{i-1,i-1} + \sqrt{\lambda} I)^{-1} \otimes (G_{i,i} + \sqrt{\lambda} I)^{-1} g_k\end{aligned}$$

其中g_k为第k步目标函数的梯度，α_k为第k步的步长

Randomized Numerical Linear Algebra

`http://bicmr.pku.edu.cn/~wenzw/bigdata2023.html`

Acknowledgement: this slides is based on Prof. Petros Drineas and Prof. Michael W. Mahoney's, Prof. Gunnar Martinsson lecture notes

Outline

1. Randomized Numerical Linear Algebra (RandNLA)
2. Approximating Matrix Multiplication
3. Approximate SVD
4. Random Sampling for SVD
5. Single View Algorithm For Matrix Approximation

Why RandNLA?

Randomization and sampling allow us to design provably accurate algorithms for problems that are:

- **Massive**
  (matrices so large that can not be stored at all, or can only be stored in slow memory devices)
- **Computationally expensive or NP-hard**
  (combinatorial optimization problems such as the Column Subset Selection Problem)

RandNLA: sampling rows/columns

Randomized algorithms

- By (carefully) sampling rows/columns of a matrix, we can construct new, smaller matrices that are close to the original matrix (w.r.t. matrix norms) with high probability.

$$\begin{pmatrix} \\ \quad A \quad \\ \\ \end{pmatrix} \begin{pmatrix} \\ \quad B \quad \\ \\ \end{pmatrix} \approx \begin{pmatrix} \\ C \\ \\ \end{pmatrix} \begin{pmatrix} \quad R \quad \end{pmatrix}$$

- By preprocessing the matrix using random projections, we can sample rows/columns much less carefully (uniformly at random) and still get nice bounds with high probability.

RandNLA: sampling rows/columns

Matrix perturbation theory

- The resulting smaller matrices behave similarly (in terms of singular values and singular vectors) to the original matrices thanks to the norm bounds.

Structural results that “decouple” the “randomized” part from the “matrix perturbation” part are important in the analyses of such algorithms.

Interplay

- Applications in BIG DATA: (Data Mining, Information Retrieval, Machine Learning, Bioinformatics, etc.)
- Numerical Linear Algebra: Matrix computations and linear algebra (ie., perturbation theory)
- Theoretical Computer Science: Randomized and approximation algorithms

Issues

- Selecting good columns that “capture the structure” of the top principal components
    - Combinatorial optimization problem; hard even for small matrices.
    - Often called the Column Subset Selection Problem (CSSP).
    - Not clear that such columns even exist.

The two issues:

- Fast approximation to the top k singular vectors of a matrix, and
- Selecting columns that capture the structure of the top k singular vectors

are connected and can be tackled using the same framework

Outline

1. Randomized Numerical Linear Algebra (RandNLA)
2. Approximating Matrix Multiplication
3. Approximate SVD
4. Random Sampling for SVD
5. Single View Algorithm For Matrix Approximation

Approximating Matrix Multiplication

Problem Statement

Given an m-by-n matrix A and an n-by-p matrix B, approximate the product AB, Or equvialently, Approximate the sum of n rank-one matrices

$$AB = \sum_{i=1}^{n} \underbrace{\left(A^{(i)} \right) \left(\quad B_{(i)} \quad \right)}_{\in \mathbb{R}^{m \times p}}$$

- $A^{(i)}$ the i-th column of A
- $B_{(i)}$ the i-th row of B
- Each term in the summation is a rank-one matrix

A sampling approach

$$AB = \sum_{i=1}^{n} \underbrace{\left(A^{(i)} \right) \left(\quad B_{(i)} \quad \right)}_{\in \mathbb{R}^{m \times p}}$$

Algorithm

- Fix a set of probabilities p_i, $i = 1, \ldots, n$, summing up to 1.
- For $t = 1, \ldots, c$,
  set $j_t = i$, where $P(j_t = i) = p_i$.
  (Pick c terms of the sum, with replacement, with respect to the p_i.)
- Approximate the product AB by summing the c terms, after scaling.

Generate Discrete Distributions

Consider a discrete random variable with possible values $c_1 < \ldots < c_n$. The probability attached to c_i is p_i. Let

$$q_0 = 0, \quad q_i = \sum_{j=1}^{i} p_j.$$

They are the cumulative probabilities associated with c_i, i.e., $q_i = F(c_i)$.

To sample this distribution

- generate a uniform U
- find $K \in \{1, \ldots, n\}$ such that $q_{K-1} < U < q_K$
- set $X = c_K$

With/without replacement

- Sampling with replacement:
  Each data unit in the population is allowed to appear in the sample more than once.
  It is easy to analyze mathematically.
- Sampling without replacement:
  Each data unit in the population is allowed to appear in the sample no more than once.

A sampling approach

$$
\begin{aligned}
AB &= \sum_{i=1}^{n} \underbrace{\left(A^{(i)} \right) \left(\quad B_{(i)} \quad \right)}_{\in \mathbb{R}^{m \times p}} \\
&\approx \frac{1}{c} \sum_{t=1}^{c} \frac{1}{p_{j_t}} \underbrace{\left(A^{(j_t)} \right) \left(\quad B_{(j_t)} \quad \right)}_{\in \mathbb{R}^{m \times p}}
\end{aligned}
$$

Keeping the terms $j_1, j_2, \ldots, j_c$

The algorithm (matrix notation)

$$\overset{m \times n}{\begin{pmatrix} & & \\ & A & \\ & & \end{pmatrix}} \overset{n \times p}{\begin{pmatrix} & & \\ & B & \\ & & \end{pmatrix}} \approx \overset{m \times c}{\begin{pmatrix} & & \\ & C & \\ & & \end{pmatrix}} \overset{c \times p}{\begin{pmatrix} & R & \end{pmatrix}}$$

Algorithm:

- Pick c columns of A to form an m-by-c matrix C and the corresponding c rows of B to form a c-by-p matrix R.
- Approximate AB by CR.

Note

- We pick the columns and rows with non-uniform probabilities.
- We scale the columns (rows) prior to including them in $C(R)$.

The algorithm (matrix notation)

$$\overset{m \times n}{\begin{pmatrix} \\ & A & \\ \\ \end{pmatrix}} \overset{n \times p}{\begin{pmatrix} \\ & B & \\ \\ \end{pmatrix}} \approx \overset{m \times c}{\begin{pmatrix} \\ & C & \\ \\ \end{pmatrix}} \overset{c \times p}{\begin{pmatrix} & R & \end{pmatrix}}$$

Algorithm:

- Create C and R by performing c i.i.d. trials, with replacement.
- For $t = 1, \ldots, c$, pick a column $A^{(j_t)}$ and a row $B_{(j_t)}$ with probability

$$\mathbf{P}(j_t = i) = \frac{\|A^{(i)}\|_2 \|B_{(i)}\|_2}{\sum_{j=1}^{n} \|A^{(j)}\|_2 \|B_{(j)}\|_2}$$

- Include $A^{(j_t)}/(cp_{j_t})^{1/2}$ as a column of C, and $B_{(j_t)}/(cp_{j_t})^{1/2}$ as a row of R

The algorithm (matrix notation)

- Let S be an n-by-c matrix whose t-th column (for $t = 1, \ldots, c$) has a single non-zero entry, namely

  $$S_{j_t t} = \frac{1}{\sqrt{c p_{j_t}}}$$

  Clearly:

  $$AB \approx CR = (AS)(S^T B)$$

  Note: S is sparse (has exactly c non-zero elements, one per column).
- It is easy to implement this particular sampling in two passes.

A bound for the Frobenius norm

For the above algorithm,

$$\mathbf{E}[\|AB - CR\|_F] = \mathbf{E}[\|AB - ASS^T B\|_F] \leq \frac{1}{c}\|A\|_F\|B\|_F$$

- The expectation of CR (element-wise) is AB (unbiased estimator), regardless of the sampling probabilities.
- Our particular choice of sampling probabilities minimizes the variance of the estimator (w.r.t. the Frobenius norm of the error AB-CR).
- prove using elementary manipulations of expectation
- Measure concentration follows from a martingale argument.
- The above bound also implies an upper bound for the spectral norm of the error $AB - CR$.

Proofs

Let $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{p \times p}$, $1 \leq c \leq n$, and $p_i \geq 0$, $\sum_i p_i = 1$. Then

$$\mathbf{E}[(CR)_{ij}] = (AB)_{ij}, \quad \mathrm{Var}[(CR)_{ij}] = \frac{1}{c} \sum_{i=1}^{n} \frac{A_{ik}^2 B_{kj}^2}{p_k} - \frac{1}{c}(AB)_{ij}^2$$

- Define $X_t = \left(\frac{A^{(i_t)} B_{(i_t)}}{c p_{i_t}} \right)_{ij} = \frac{A_{i i_t} B_{i_t j}}{c p_{i_t}}$. Then

$$\mathbf{E}[X_t] = \sum_{k=1}^{n} p_k \frac{A_{ik} B_{kj}}{c p_k} = \frac{1}{c}(AB)_{ij} \text{ and } \mathbf{E}[X_t^2] = \sum_{k=1}^{n} \frac{A_{ik}^2 B_{kj}^2}{c^2 p_k}$$

- $\mathbf{E}[(CR)_{ij}] = \sum_{t=1}^{c} \mathbf{E}[X_t] = (AB)_{ij}$

$$\mathrm{Var}[X_t] = E[X_t^2] - E[X_t]^2 = \sum_{k=1}^{n} \frac{A_{ik}^2 B_{kj}^2}{c^2 p_k} - \frac{1}{c^2}(AB)_{ij}^2$$

Proofs

Lemma:

$$\mathbf{E}[\|AB - CR\|_F^2] = \sum_{k=1}^{n} \frac{|A^{(k)}|^2 |B_{(k)}|^2}{cp_k} - \frac{1}{c}\|AB\|_F^2$$

Proof:

$$\begin{aligned}
\mathbf{E}[\|AB - CR\|_F^2] &= \sum_{i=1}^{n}\sum_{j=1}^{p} \mathbf{E}[(AB - CR)_{ij}^2] = \sum_{i=1}^{n}\sum_{j=1}^{p} \mathrm{Var}[(CR)_{ij}] \\
&= \frac{1}{c}\sum_{k=1}^{n} \frac{1}{p_k}\left(\sum_i A_{ik}^2\right)\left(\sum_i B_{kj}^2\right) - \frac{1}{c}\|AB\|_F^2 \\
&= \frac{1}{c}\sum_{k=1}^{n} \frac{1}{p_k}|A^{(k)}|^2 |B_{(k)}|^2 - \frac{1}{c}\|AB\|_F^2
\end{aligned}$$

Proofs

- Find p_k to minimize $\mathbf{E}[\|AB - CR\|_F^2]$:

$$\min_{\sum_{k=1}^n p_k = 1} f(p_1, \ldots, p_n) = \sum_{k=1}^n \frac{1}{p_k} |A^{(k)}|^2 |B_{(k)}|^2$$

- Introduce $L = f(p_1, \ldots, p_n) + \lambda(\sum_{k=1}^n p_k - 1)$ and solve $\frac{\partial L}{\partial p_i} = 0$
- It gives $p_k = \frac{|A^{(k)}||B_{(k)}|}{\sum_{k'=1}^n |A^{(k')}||B_{(k')}|}$. Then

$$\begin{aligned} \mathbf{E}[\|AB - CR\|_F^2] &= \frac{1}{c}\left(\sum_{k=1}^n |A^{(k)}||B_{(k)}|\right)^2 - \frac{1}{c}\|AB\|_F^2 \\ &\le \frac{1}{c}\|A\|_F^2 \|B\|_F^2 \end{aligned}$$

Special case: $B = A^T$

If $B = A^T$, then the sampling probabilities are

$$\mathbf{P}(j_t = i) = \frac{\|A^{(i)}\|_2^2}{\|A\|_F^2}$$

Also, $R = C^T$, and the error bounds are:

$$\mathbf{E}[\|AA^T - CC^T\|_F] = \mathbf{E}[\|AA^T - ASS^TA^T\|_F] \leq \frac{1}{c}\|A\|_F^2$$

Special case: $B = A^T$

A better spectral norm bound via matrix Chernoff/Bernstein inequalities:

Assumptions:

- Spectral norm of A is one (not important, just normalization)
- Frobenius norm of A is at least 0.2 (not important, simplifies bounds).
- Important: Set

$$c = \Omega\left(\frac{\|A\|_F^2}{\epsilon^2}\ln\left(\frac{\|A\|_F^2}{\epsilon^2\sqrt{\delta}}\right)\right)$$

Then: for any $0 < \epsilon < 1$ with probability at least $1 - \delta$

$$\mathbf{E}[\|AA^T - CC^T\|_F] = \mathbf{E}[\|AA^T - ASS^TA^T\|_F] \leq \epsilon$$

Outline

1. Randomized Numerical Linear Algebra (RandNLA)
2. Approximating Matrix Multiplication
3. Approximate SVD
4. Random Sampling for SVD
5. Single View Algorithm For Matrix Approximation

Low-Rank Matrix Approximation

Problem Statement:
Given: mxn matrix A, and $0 < k < \min(m,n) = n$.
Goal: Compute a rank-k approximation to A.

- Fast low-rank matrix approximation is key to efficiency of superfast direct solvers for integral equations and many large sparse linear systems.
- Indispensable tool in mining large data sets.
- Randomized algorithms compute accurate truncated SVD.
- Minimum work and communication/Exceptionally high success rate.

Low-rank Approximation

seek to compute a rank-k approximation with $k \ll n$

$$\overset{m \times n}{\begin{pmatrix} & & \\ & & \\ & A & \\ & & \\ & & \end{pmatrix}} \approx \overset{m \times k}{\begin{pmatrix} & & \\ & & \\ & U_k & \\ & & \\ & & \end{pmatrix}} \overset{k \times n}{\begin{pmatrix} & & \\ & X & \\ & & \end{pmatrix}}$$

- Eigenvectors corresponding to leading eigenvalues.
- Singular Value Decomposition (SVD) / Principal Component Analysis (PCA).
- Spanning columns or rows.

The problem being addressed is ubiquitous in applications.

Review of existing methods: dense matrix

For a dense $n \times n$ matrix that fits in RAM, excellent algorithms are already part of LAPACK (and incorporated into Matlab, Mathematica, etc).

- Double precision accuracy.
- Very stable.
- $O(n^3)$ asymptotic complexity. Reasonably small constants.
- Require extensive random access to the matrix.
- When the target rank k is much smaller than n, there also exist $O(n^2k)$ methods with similar characteristics (the well-known Golub-Businger method, RRQR by Gu and Eisentstat, etc).
- For small matrices, the state-of-the-art is quite satisfactory. (By "small" we mean something like $n \leq 10000$ on today's computers.)

Review of existing methods: structured matrix

If the matrix is large, but can rapidly be applied to a vector (if it is sparse, or sparse in Fourier space, or amenable to the FMM, etc.), so called Krylov subspace methods often yield excellent accuracy and speed.

Lanczos-based methods:

1. From $v \in \mathbb{R}^n$, computes orthonormal basis V for

$$\mathcal{K}(A, v) = span\left\{v, Av, A^2v, \cdots, A^{k-1}v\right\}$$

2. Rayleigh-Ritz: $eig(V^TAV) \Rightarrow$ Ritz pairs $\approx$ eigenpairs
3. If "not converged", update v and go to Step 1.

Strength and weakness:

- Most efficient in terms of the number of Av (or SpMv)
- Fast and reliable for computing "not too many" eigenpairs
- Lower concurrency and unable to be warm-started

“New” challenges in algorithmic design

The existing state-of-the-art methods of numerical linear algebra that we have very briefly outlined were designed for an environment where the matrix fits in RAM and the key to performance was to minimize the number of floating point operations required. Currently, communication is becoming the real bottleneck:

- While clock speed is hardly improving at all anymore, the cost of a flop keeps going down rapidly. (Multi-core processors, GPUs, cloud computing, etc.)
- The cost of slow storage (hard drives, flash memory, etc.) is also going down rapidly.
- Communication costs are decreasing, but not rapidly. Moving data from a hard-drive. Moving data between nodes of a parallel machine. (Or cloud computer ...) The amount of fast cache memory close to a processor is not improving much. (In fact, it could be said to be shrinking — GPUs, multi-core, etc.)
- “Deluge of data”. Driven by ever cheaper storage and acquisition techniques. Web search, data mining in archives of documents

Linear Time SVD Algorithm

- Input: m-by-n matrix A, $1 \leq k \leq c \leq n$, $\{p_i\}_{i=1}^n$ such that $p_i \geq 0$ and $\sum_i p_i = 1$
- Sampling:
  - For $t = 1$ to c
    pick $i_t \in \{1, \ldots, n\}$ with $\mathcal{P}(i_t = \alpha) = p_\alpha$.
    Set $C^{(t)} = \frac{A^{(i_t)}}{\sqrt{cp_{i_t}}}$
- Compute $C^T C$ and its eigenvalue decomposition, say $C^T C = \sum_{t=1}^c \sigma_t(C)^2 y_t y_t^T$
- Compute $h_t = \frac{Cy_t}{\sigma_t(C)}$ for $t = 1, \ldots, k$.
  (Note: $A = U\Sigma V^T$ and $C = H\Sigma_C Y^T \Longrightarrow H = CY\Sigma_C^{-1}$)
- Return H_k where $H_k^{(t)} = h_t$ and $\sigma_t(C)$ for $t = 1, \ldots, k$

The left singular vectors of C are with high probability approximations to the left singular vectors of A

Extract approximate SVD

- Given A. Let X be an approximation of the left singular vectors of A corresponding to k largest singular values

```
method = 2;    % = 1 or 2
Y = (X'*A)';     % Y = A'*X;
switch method
    case 1;
        [V,S,W] = svd(Y,0);
        U = X*W;
    case 2;
        [V,R] = qr(Y,0);
        [W,S,Z] = svd(R');
        U = X*W; V = V*Z;
 end
```

- The pair (U, S, V) is an approximation of the k-dominant SVD

Main theoretical results

Let H_k be constructed the linear Time SVD

$$\mathbf{E}[\|A - H_k H_k^T A\|_F^2] \leq \|A - A_k\|_F^2 + \epsilon \|A\|_F^2$$

- Exact SVD of $A = U\Sigma V^T$, $A_k = U_k \Sigma_k V_k^T = U_k U_k^T A = A V_k V_k^T$.
  $\min_{\text{rank}(B) \leq k} \|A - B\|_2 = \|A - A_k\|_2 = \sigma_{k+1}(A)$
  $\min_{\text{rank}(B) \leq k} \|A - B\|_F^2 = \|A - A_k\|_F^2 = \sum_{t=k+1}^{r} \sigma_t^2(A)$
- perturbation theory of matrices

  $$\max_{1 \leq t \leq n} |\sigma_t(A+E) - \sigma_t(A)| \leq \|E\|_2, \quad \sum_{k=1}^{n} (\sigma_k(A+E) - \sigma_k(A))^2 \leq \|E\|_F^2$$

  the latter is known as Hoffman-Wielandt inequality
- Exact SVD of $C = H \Sigma_C Y^T$

Proofs

Lemma:

$$
\begin{aligned}
\|A - H_k H_k^T A\|_F^2 &\leq \|A - A_k\|_F^2 + 2\sqrt{k}\|AA^T - CC^T\|_F \\
\|A - H_k H_k^T A\|_2^2 &\leq \|A - A_k\|_2^2 + 2\|AA^T - CC^T\|_2
\end{aligned}
$$

Proof of the first inequality

- $\|X\|_F^2 = \mathrm{Tr}(X^T X)$ and $\mathrm{Tr}(X+Y) = \mathrm{Tr}(X) + \mathrm{Tr}(Y)$

$$
\begin{aligned}
\|A - H_k H_k^T A\|_F^2 &= \mathrm{Tr}((A - H_k H_k^T A)^T (A - H_k H_k^T A)) \\
&= \mathrm{Tr}(A^T A) - \mathrm{Tr}(A^T H_k H_k^T A) = \|A\|_F^2 - \|A^T H_k\|_F^2
\end{aligned}
$$

- Using Cauchy-Schwartz inequality:

$$
\left| \|A^T H_k\|_F^2 - \sum_{t=1}^{k} \sigma_t^2(C) \right| \leq \sqrt{k} \left(\sum_{t=1}^{k} (|A^T h_t|^2 - \sigma_t^2(C))^2 \right)^{1/2}
$$

$$
\begin{aligned}
&= \sqrt{k} \left(\sum_{t=1}^{k} (|A^T h_t|^2 - |C^T h_t|^2)^2 \right)^{1/2} = \sqrt{k} \left(\sum_{t=1}^{k} ((h_t)^T (AA^T - CC^T) h_t)^2 \right)^{1/} \\
&\leq \sqrt{k}\|AA^T - CC^T\|_F
\end{aligned}
$$

Proofs

- by Hoffman-Wielandt inequality

$$\left|\sum_{t=1}^{k}\sigma_t^2(C)-\sum_{t=1}^{k}\sigma_t^2(A)\right| \le \sqrt{k}\left(\sum_{t=1}^{k}(\sigma_t^2(C)-\sigma_t^2(A))^2\right)^{1/2}$$

$$= \sqrt{k}\left(\sum_{t=1}^{k}(\sigma_t(CC^T)-\sigma_t(AA^T))^2\right)^{1/2}$$

$$\le \sqrt{k}\left(\sum_{t=1}^{m}(\sigma_t(CC^T)-\sigma_t(AA^T))^2\right)^{1/2} \le \sqrt{k}\|CC^T-AA^T\|_F$$

- Therefore

$$\left|\|A^TH_k\|_F^2-\sum_{t=1}^{k}\sigma_t^2(A)\right| \le 2\sqrt{k}\|AA^T-CC^T\|_F$$

Proofs

- matrix approximation gives
$$\mathbf{E}[\|AB - CR\|_F^2] \;\leq\; \frac{1}{c}\|A\|_F^2\|B\|_F^2$$
which yields
$$2\sqrt{k}\mathbf{E}[\|AA^T - CC^T\|_F] \;\leq\; \left(\frac{4k}{c}\right)^{1/2}\|A\|_F^2$$
- $$\|A^T H_k\|_F^2 \geq \sum_{t=1}^{k}\sigma_t^2(A) - 2\sqrt{k}\|AA^T - CC^T\|_F$$
- If $c \geq 4k/\epsilon^2$, then
$$\begin{aligned}\mathbf{E}[\|A - H_k H_k^T A\|_F^2] &\leq \|A\|_F^2 - \sum_{t=1}^{k}\sigma_t^2(A) + 2\sqrt{k}\mathbf{E}[\|AA^T - CC^T\|_F] \\ &\leq \|A - A_k\|_F^2 + \epsilon\|A\|_F^2\end{aligned}$$

Outline

1. Randomized Numerical Linear Algebra (RandNLA)

2. Approximating Matrix Multiplication

3. Approximate SVD

4. Random Sampling for SVD

5. Single View Algorithm For Matrix Approximation

Range finding problem

Given an $m \times n$ matrix A and an integer $k < \min(m, n)$, find an orthonormal $m \times k$ matrix Q such that

$$A \approx QQ^T A$$

Solving the primitive problem via randomized sampling — intuition

- Draw random vectors $r_1, r_2, \ldots, r_k \in \mathbb{R}^n$.
- Form “sample” vectors $y_1 = Ar_1, y_2 = Ar_2, \ldots, y_k = Ar_k \in \mathbb{R}_m$.
- Form orthonormal vectors $q_1, q_2, \ldots, q_k \in \mathbb{R}^m$ such that

$$\mathrm{span}\{q_1, q_2, \ldots, q_k\} = \mathrm{span}\{y_1, y_2, \ldots, y_k\}$$

Almost always correct if A has exact rank k

Low-Rank Approximation: Randomized Sampling

Algorithm RandSam0

Input: mxn matrix A, int k, p.

- Draw a random $n \times (k+p)$ matrix Ω
- Compute $QR = A\Omega$
- and SVD: $Q^T A = \hat{U}\hat{\Sigma}\hat{V}^T$
- Truncate SVD: $\hat{U}_k\hat{\Sigma}_k\hat{V}_k^T$

Output: $B = (Q\hat{U}_k)\hat{\Sigma}_k\hat{V}_k^T$

- Easy to implement.
- Very efficient computation.
- Minimum communication.

error for Gaussian test matrices

Ref: Halko, Martinsson, Tropp, 2009 & 2011; Martinsson, Rokhlin, Tygert (2006)

- Let A denote an $m \times n$ matrix with singular values $\{\sigma_j\}_{j=1}^{\min(m,n)}$
- Let k denote a target rank and let p denote an over-sampling parameter.
- Let Ω denote an $n \times (k+p)$ Gaussian matrix.
- Let Q denote the $m \times (k+p)$ matrix $Q = \mathtt{orth}(A\Omega)$.

If $p \geq 4$, then

$$\|A - QQ^*A\|_2 \leq \left(1 + 6\sqrt{(k+p)p\log p}\right)\sigma_{k+1} + 3\sqrt{k+p}\left(\sum_{j>k}\sigma_j^2\right)^{1/2}$$

except with probability at most $3p^{-p}$.

Improved Randomized Sampling

Algorithm RandSam1

Input: mxn matrix A, int k, p, c.

- Draw a random $n \times (k+p+c)$ matrix Ω
- Compute $QR = A\Omega$
- and SVD: $Q^T A = \hat{U}\hat{\Sigma}\hat{V}^T$
- Truncate SVD: $\hat{U}_k\hat{\Sigma}_k\hat{V}_k^T$

Output: $B = (Q\hat{U}_k)\hat{\Sigma}_k\hat{V}_k^T$

- Only change from RandSam0: p becomes p + c
- Smallest modification of any algorithm.
- c allows a drastically different error bound, controls accuracy.
- p remains in control of failure chance.

Randomized Power Method

Algorithm RandSam2

Input: mxn matrix A, int k, p, c, q

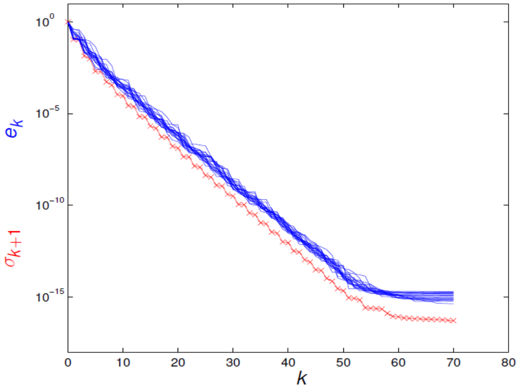
- Draw a random $n \times (k+p+c)$ matrix Ω
- Compute $QR = (AA^T)^q A\Omega$
- and SVD: $Q^T A = \hat{U}\hat{\Sigma}\hat{V}^T$
- Truncate SVD: $\hat{U}_k\hat{\Sigma}_k\hat{V}_k^T$

Output: $B = (Q\hat{U}_k)\hat{\Sigma}_k\hat{V}_k^T$

- QR needs done carefully for numerical accuracy.
- Algorithm is old one when q = 0; but q = 1 far more accurate.
- Should converge faster when singular values do not decay very fast.

Example 1

We consider a 1000×1000 matrix A whose singular values are shown below:



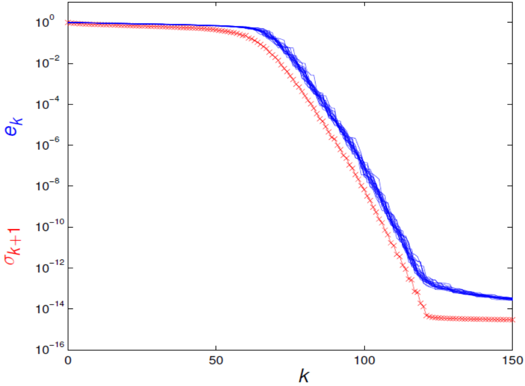
The red line indicates the singular values σ_{k+1} of **A**. These indicate the theoretically minimal approximation error.

The blue lines indicate the actual errors e_k incurred by 20 instantiations of the proposed method.

A is a discrete approximation of a certain compact integral operator normalized so that $\|A\| = 1$. Curiously, the nature of A is in a strong sense irrelevant: the error distribution depends only on $\{\sigma_j\}_{j=1}^{\min(m,n)}$.

Example 2

We consider a 1000×1000 matrix A whose singular values are shown below:



The red line indicates the singular values σ_{k+1} of **A**. These indicate the theoretically minimal approximation error.

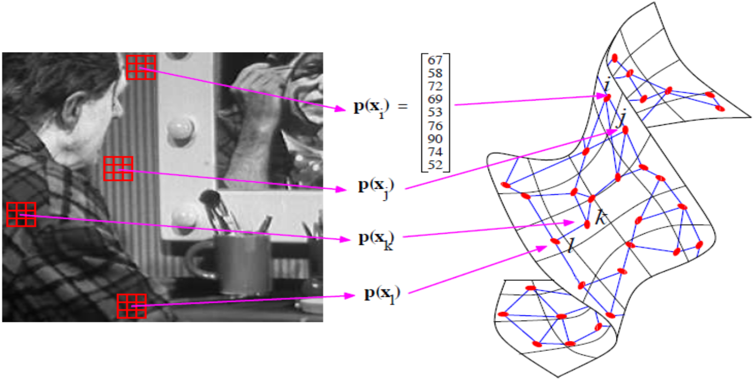
The blue lines indicate the actual errors e_k incurred by 20 instantiations of the proposed method.

A is a discrete approximation of a certain compact integral operator normalized so that $\|A\| = 1$. Curiously, the nature of A is in a strong sense irrelevant: the error distribution depends only on $\{\sigma_j\}_{j=1}^{\min(m,n)}$.

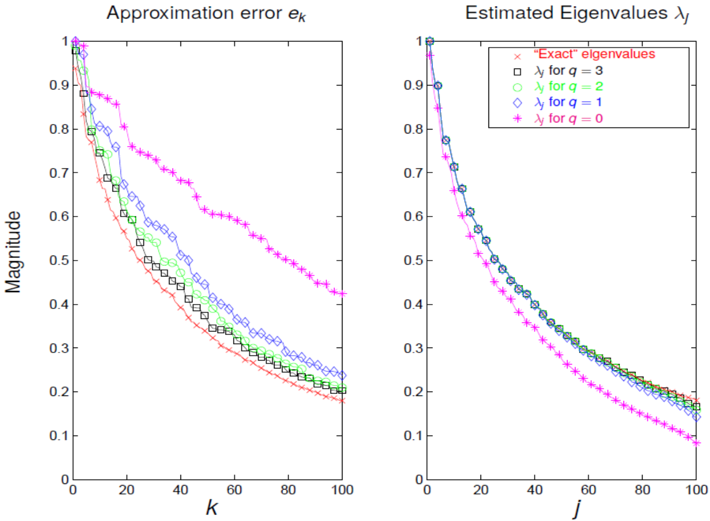
Example 3

The matrix A being analyzed is a 9025×9025 matrix arising in a diffusion geometry approach to image processing.

To be precise, A is a graph Laplacian on the manifold of 3×3 patches.



$\mathbf{p}(\mathbf{x}_i) = \begin{bmatrix} 67 \\ 58 \\ 72 \\ 69 \\ 53 \\ 76 \\ 90 \\ 74 \\ 52 \end{bmatrix}$
$\mathbf{p}(\mathbf{x}_j)$
$\mathbf{p}(\mathbf{x}_k)$
$\mathbf{p}(\mathbf{x}_l)$
i
j
k
l



The pink lines illustrates the performance of the basic random sampling scheme. The errors for q = 0 are huge, and the estimated eigenvalues are much too small.

Outline

1. Randomized Numerical Linear Algebra (RandNLA)
2. Approximating Matrix Multiplication
3. Approximate SVD
4. Random Sampling for SVD
5. Single View Algorithm For Matrix Approximation

Low-rank reconstruction

Given $A \in \mathbb{R}^{m \times n}$ and a target rank r. Select k and ℓ. Given random matrices $\Omega \in \mathbb{R}^{n \times k}$ and $\Psi \in \mathbb{R}^{\ell \times m}$. Compute

$$Y = A\Omega, \qquad W = \Psi A,$$

Then an approximation $\hat{A}$ is computed:

- Form an orthogonal-triangular factorization $Y = QR$ where $Q \in \mathbb{R}^{m \times k}$.
- Solve a least-squares problem to obtain $X = (\Psi Q)^{\dagger} W \in \mathbb{R}^{k \times n}$
- Construct the rank-k approximation $\hat{A} = QX$

Suppose $k = 2r + 1$ and $\ell = 4r + 2$, then

$$\mathbf{E}\|A - \hat{A}\|_F \leq 2 \min_{\mathrm{rank}(Z) \leq r} \|A - Z\|_F$$

Linear update of A

- Suppose that A is sent as a sequence of additive updates:

  $$A = H_1 + H_2 + H_3 + \cdots$$

  Then one compute

  $$Y \leftarrow Y + H\Omega, \quad W = W + \Psi H$$

- Suppose that A is sent as a sequence of additive updates:

  $$A = \theta A + \eta H$$

  Then one compute

  $$Y \leftarrow \theta Y + \eta H\Omega, \quad W = \theta W + \eta \Psi H$$

Intuition

- Suppose

  $$A \approx QQ^*A.$$

  We want to form the rank-k approximation $Q(Q^*A)$, but we cannot compute the factor Q^*A without revisiting the target matrix A.

- Note

  $$W = \Psi(QQ^*A) + \Psi(A - QQ^*A) \approx (\Psi Q)(Q^*A)$$

- The construction of X:

  $$X = (\Psi Q)^{\dagger} W \approx Q^*A$$

- Hence

  $$\hat{A} = QX \approx QQ^*A \approx A$$

Projection onto a Convex Set.

Let C be a closed and convex set. Define the projection:

$$\Pi_C(M) = \arg\min_X \quad \|X - M\|_F^2, \text{ s.t. } X \in C$$

- Suppose $A \in C$. Let $\hat{A}_{in}$ be an initial approximation of A,

$$\|A - \Pi_C(\hat{A}_{in})\|_F \leq \|A - \hat{A}_{in}\|_F$$

- Conjugate Symmetric Approximation

$$H^n = \{X \in \mathcal{C}^{n\times n} | X = X^*\}$$

The projection

$$\Pi_{H^n}(M) = \frac{1}{2}(M + M^*)$$

Conjugate Symmetric Approximation.

Let $A \in H^n$. Let $\hat{A} = QX$.

- $$\Pi_{H^n}(\hat{A}) = \frac{1}{2}(QX + X^*Q^*) = \frac{1}{2}[Q, X^*] \begin{pmatrix} 0 & I \\ I & 0 \end{pmatrix} [Q, X^*]^*$$
- Let $[Q, X^*] = U[T_1, T_2]$. Then
$$S = \frac{1}{2}(T_1T_2^* + T_2T_1^*)$$
- Construct
$$\hat{A}_{sym} = USU^*$$

PSD Approximation

Let A be positive semidefinite (PSD). Let $\hat{A} = QX$.

- Form eigenvalue decomposition

$$S = VDV^*$$

- Compute

$$\hat{A}_{sym} = (UV)D(UV)^*$$

- Construct

$$\hat{A}_+ = \Pi_{H_+^n}(\hat{A}) = (UV)D_+(UV)^*$$

Phase Retrieval

`http://bicmr.pku.edu.cn/~wenzw/bigdata2020.html`

Acknowledgement: this slides is based on Prof. Emmanuel Candès 's lecture notes

Outline

1. Introduction

2. Classical Phase Retrieval

3. PhaseLift

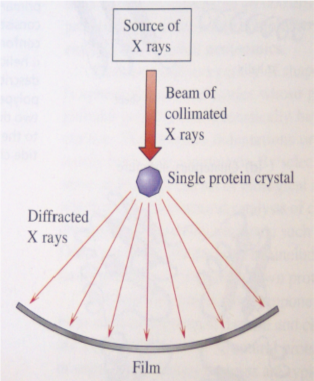
4. PhaseCut

5. Wirtinger Flows

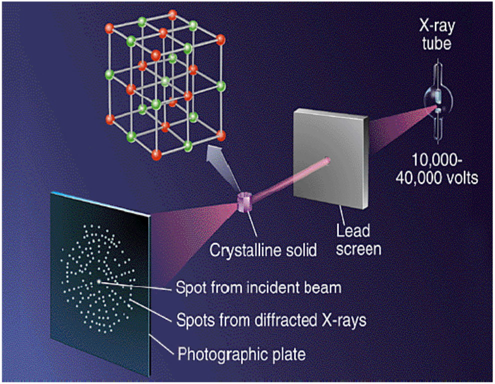
6. Gauss-Newton Method

X-ray crystallography

Method for determining atomic structure within a crystal



principle

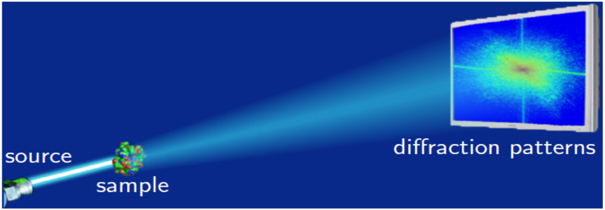


typical setup

10 Nobel Prizes in X-ray crystallography, and counting...

Missing phase problem

Detectors record intensities of diffracted rays $\Longrightarrow$ phaseless data only!



Fraunhofer diffraction $\Longrightarrow$ intensity of electrical $\approx$ Fourier transform

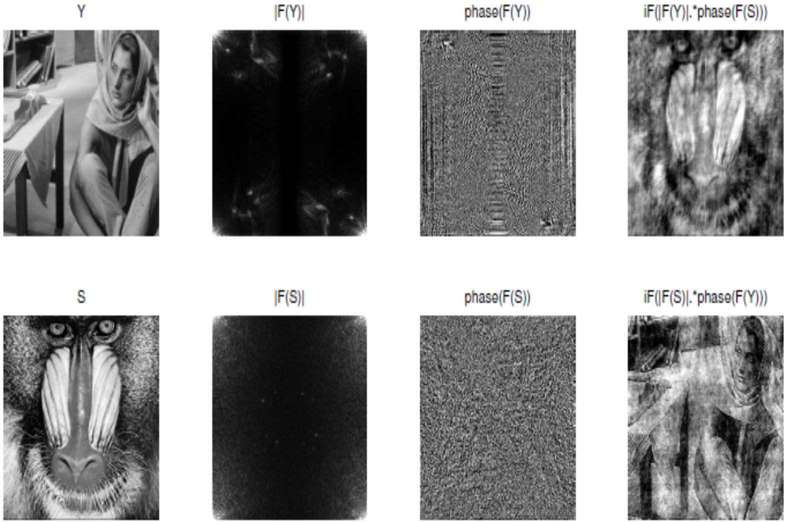
$$|\hat{x}(f_1, f_2)|^2 = \left| \int x(t_1, t_2) e^{-i2\pi(f_1 t_1 + f_2 t_2)} dt_1 dt_2 \right|$$

Electrical field $\hat{x} = |\hat{x}| e^{i\phi}$ with intensity $|\hat{x}|^2$

Phase retrieval problem (inversion)

How can we recover the phase (or signal $x(t_1, t_2)$) from $|\hat{x}(f_1, f_2)|$

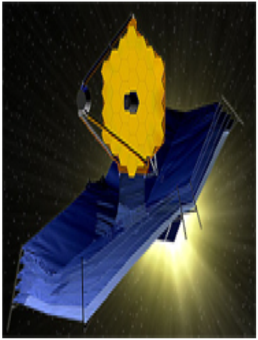
Phase and magnitude



Phase carries more information than magnitude

Other applications of phase retrieval

Hubble telescope



James Webb space telescope

Outline

1. Introduction
2. Classical Phase Retrieval
3. PhaseLift
4. PhaseCut
5. Wirtinger Flows
6. Gauss-Newton Method

Classical Phase Retrieval

Feasibility problem

$$\text{find } x \in S \cap \mathcal{M} \text{ or } \text{ find } x \in S_+ \cap \mathcal{M}$$

- given Fourier magnitudes:

$$\mathcal{M} := \{x(r) \mid |\hat{x}(\omega)| = b(\omega)\}$$

  where $\hat{x}(\omega) = \mathcal{F}(x(r))$, $\mathcal{F}$: Fourier transform
- given support estimate:

$$S := \{x(r) \mid x(r) = 0 \text{ for } r \notin D\}$$

  or

$$S_+ := \{x(r) \mid x(r) \geq 0 \text{ and } x(r) = 0 \text{ if } r \notin D\}$$

Error Reduction

Alternating projection:

$$x^{k+1} = \mathcal{P}_{\mathcal{S}}\mathcal{P}_{\mathcal{M}}(x^k)$$

- projection to S:

$$\mathcal{P}_S(x) = \begin{cases} x(r), & \text{if } r \in D, \\ 0, & \text{otherwise,} \end{cases}$$

- projection to $\mathcal{M}$:

$$\mathcal{P}_{\mathcal{M}}(x) = \mathcal{F}^*(\hat{y}), \text{ where } \hat{y} = \begin{cases} b(\omega)\frac{\hat{x}(\omega)}{|\hat{x}(\omega)|}, & \text{if } \hat{x}(\omega) \neq 0, \\ b(\omega), & \text{otherwise,} \end{cases}$$

Summary of projection algorithms

- Basic input-output (BIO)

$$x^{k+1} = (\mathcal{P}_{\mathcal{S}}\mathcal{P}_{\mathcal{M}} + I - \mathcal{P}_{\mathcal{M}})\,(x^k)$$

- Hybrid input-output (HIO)

$$x^{k+1} = ((1+\beta)\mathcal{P}_{\mathcal{S}}\mathcal{P}_{\mathcal{M}} + I - \mathcal{P}_{\mathcal{S}} - \beta\mathcal{P}_{\mathcal{M}})\,(x^k)$$

- Hybrid projection reflection (HPR)

$$x^{k+1} = \left((1+\beta)\mathcal{P}_{\mathcal{S}_+}\mathcal{P}_{\mathcal{M}} + I - \mathcal{P}_{\mathcal{S}_+} - \beta\mathcal{P}_{\mathcal{M}}\right)(x^k)$$

- Relaxed averaged alternating reflection (RAAR)

$$x^{k+1} = \left(2\beta\mathcal{P}_{\mathcal{S}_+}\mathcal{P}_{\mathcal{M}} + \beta I - \beta\mathcal{P}_{\mathcal{S}_+} + (1-2\beta)\mathcal{P}_{\mathcal{M}}\right)(x^k)$$

- Difference map (DF)

$$x^{k+1} = (I + \beta(\mathcal{P}_{\mathcal{S}}((1-\gamma_2)\mathcal{P}_{\mathcal{M}} - \gamma_2 I) + \mathcal{P}_{\mathcal{M}}((1-\gamma_1)\mathcal{P}_{\mathcal{S}} - \gamma_1 I)))\,(x^k)$$

ADMM

Consider problem

$$\text{find } x \text{ and } y, \text{ such that } x = y, \ x \in \mathcal{X} \text{ and } y \in \mathcal{Y}$$

- $\mathcal{X}$ is either $\mathcal{S}$ or $\mathcal{S}_+$, and $\mathcal{Y}$ is $\mathcal{M}$.
- Augmented Lagrangian function

$$\mathcal{L}(x, y, \lambda) := \lambda^\top (x - y) + \frac{1}{2}\|x - y\|^2$$

- ADMM:

$$\begin{aligned} x^{k+1} &= \arg\min_{x \in \mathcal{X}} \ \mathcal{L}(x, y^k, \lambda^k), \\ y^{k+1} &= \arg\min_{y \in \mathcal{Y}} \mathcal{L}(x^{k+1}, y, \lambda^k), \\ \lambda^{k+1} &= \lambda^k + \beta(x^{k+1} - y^{k+1}), \end{aligned}$$

ADMM

- ADMM

$$\begin{aligned} x^{k+1} &= \mathcal{P}_{\mathcal{X}}(y^k - \lambda^k), \\ y^{k+1} &= \mathcal{P}_{\mathcal{Y}}(x^{k+1} + \lambda^k), \\ \lambda^{k+1} &= \lambda^k + \beta(x^{k+1} - y^{k+1}), \end{aligned}$$

- ADMM is equivalent to HIO or HPR
  - if $\mathcal{P}_{\mathcal{X}}(x+y) = \mathcal{P}_{\mathcal{X}}(x) + \mathcal{P}_{\mathcal{X}}(y)$

    $$x^{k+2} + \lambda^{k+1} = [(1+\beta)\mathcal{P}_{\mathcal{X}}\mathcal{P}_{\mathcal{Y}} + (I - \mathcal{P}_{\mathcal{X}}) - \beta\mathcal{P}_{\mathcal{Y}}](x^{k+1} + \lambda^k)$$

    Hybrid input-output (HIO)

    $$x^{k+1} = ((1+\beta)\mathcal{P}_{\mathcal{S}}\mathcal{P}_{\mathcal{M}} + I - \mathcal{P}_{\mathcal{S}} - \beta\mathcal{P}_{\mathcal{M}})\,(x^k)$$

  - if $\beta = 1$

ADMM

- ADMM: updating Lagrange Multiplier twice

$$\begin{aligned} x^{k+1} &:= \mathcal{P}_{\mathcal{X}}(y^k - \pi^k), \\ \pi^{k+1} &:= \pi^k + \beta(x^{k+1} - y^k) = -(I - \beta\mathcal{P}_{\mathcal{X}})(y^k - \pi^k), \\ y^{k+1} &:= \mathcal{P}_{\mathcal{Y}}(x^{k+1} + \lambda^k), \\ \lambda^{k+1} &:= \lambda^k + \nu(x^{k+1} - y^{k+1}) = (I - \nu\mathcal{P}_{\mathcal{Y}})(x^{k+1} + \lambda^k), \end{aligned}$$

- ADMM is equivalent to ER if $\beta = \nu = 1$

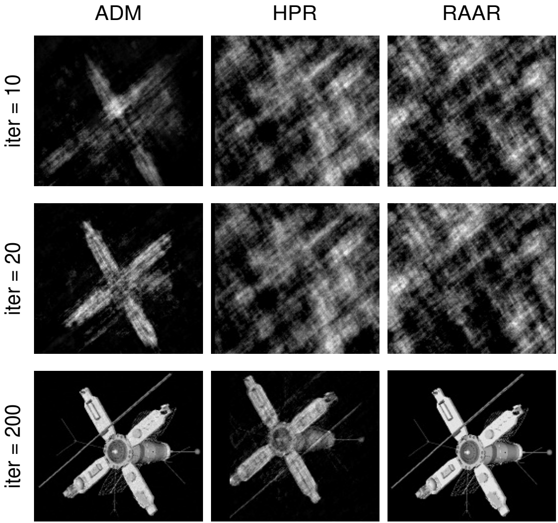
$$x^{k+1} := \mathcal{P}_{\mathcal{X}}(y^k) \text{ and } y^{k+1} := \mathcal{P}_{\mathcal{Y}}(x^{k+1}).$$

- ADMM is equivalent to BIO if $\beta = \nu = 1$

$$x^{k+1} + \lambda^k = \left(\mathcal{P}_{\mathcal{X}}\mathcal{P}_{\mathcal{Y}} + I - \mathcal{P}_{\mathcal{Y}}\right)\left(x^k + \lambda^{k-1}\right)$$

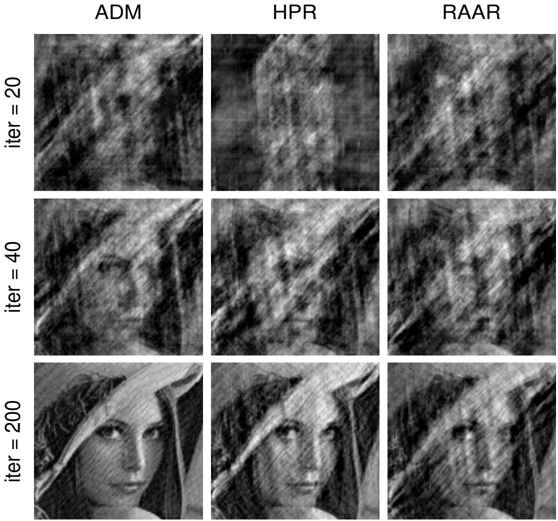
Numerical comparison

The parameter β in HPR and RAAR was updated dynamically with $\beta_0 = 0.95$. For ADMM, $\beta = 0.5$.



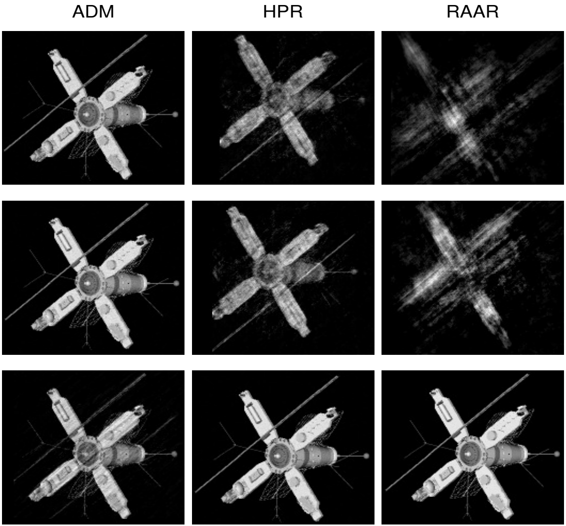
Numerical comparison

The parameter β in HPR and RAAR was updated dynamically with $\beta_0 = 0.95$. For ADMM, $\beta = 0.5$.



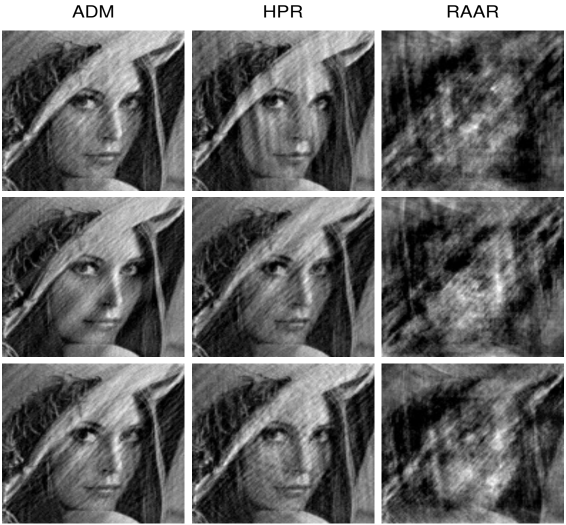
Numerical comparison

The parameter β was fixed at 0.6, 0.8 and 0.95 for the first, second and third rows respectively.



Numerical comparison

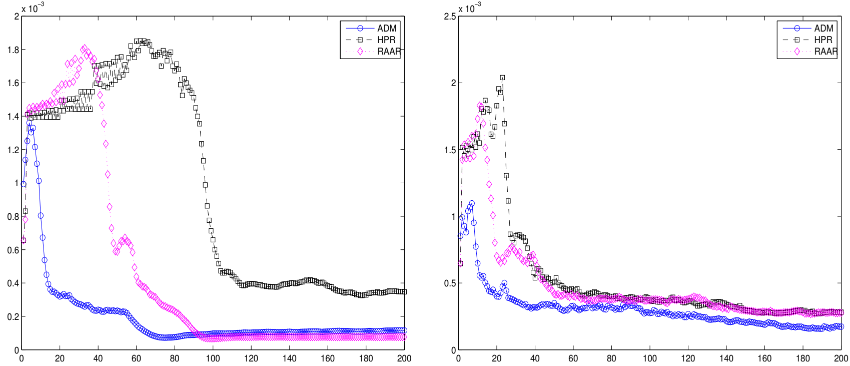
The parameter β was fixed at 0.6, 0.8 and 0.95 for the first, second and third rows respectively.



Numerical Results

Convergence behavior:

$$\text{err}^k = \frac{\|\mathcal{P}_{\mathcal{X}}(\mathcal{P}_{\mathcal{Y}}(x^k)) - \mathcal{P}_{\mathcal{Y}}(x^k)\|_F}{\|m\|_F}$$



(a) Satellite (b) lena

Outline

1. Introduction
2. Classical Phase Retrieval
3. **PhaseLift**
4. PhaseCut
5. Wirtinger Flows
6. Gauss-Newton Method

Discrete mathematical model

- Phaseless measurements about $x_0 \in \mathcal{C}^n$

$$b_k = |\langle a_k, x_0 \rangle|^2, \quad k \in \{1, \ldots, m\}$$

- Phase retrieval is feasibility problem

$$\begin{aligned} \text{find} \quad & x \\ \text{s.t.} \quad & |\langle a_k, x_0 \rangle|^2 = b_k, k = 1, \ldots, m \end{aligned}$$

Solving quadratic equations is NP-complete in general

NP-complete stone problem

Given weights $w_i \in \mathbb{R}$, $i = 1, \ldots, n$, is there an assignment $x_i = \pm 1$ such that

$$\sum_{i=1}^{n} w_i x_i = 0?$$

Formulation as a quadratic system

$$\begin{aligned} |x_i|^2 &= 1, \quad i = 1, \ldots, n \\ \left| \sum_{i=1}^{n} w_i x_i \right|^2 &= 0 \end{aligned}$$

PhaseLift (C., Eldar, Strohmer, Voroninski, 2011)

Lifting: $X = xx^*$

$$b_k = |\langle a_k, x_0 \rangle|^2 = a_k^* x x^* a_k = \langle a_k a_k^*, X \rangle$$

Turns quadratic measurements into linear measurements $b = \mathcal{A}(X)$ about xx^*

Phase retrieval problem

$$\begin{array}{ll} \text{find} & X \\ \text{s.t.} & \mathcal{A}(X) = b \\ & X \succeq 0, \operatorname{rank}(X) = 1 \end{array}$$

PhaseLift

$$\begin{array}{ll} \text{find} & X \\ \text{s.t.} & \mathcal{A}(X) = b \\ & X \succeq 0 \end{array}$$

Connections: relaxation of quadratically constrained QP's

- Shor (87) [Lower bounds on nonconvex quadratic optimization problems]
- Goemans and Williamson (95) [MAX-CUT]
- Chai, Moscoso, Papanicolaou (11)

Exact generalized phase retrieval via SDP

Phase retrieval problem

$$\begin{array}{ll} \text{find} & x \\ \text{s.t.} & b_k = |\langle a_k, x_0 \rangle|^2 \end{array}$$

PhaseLift

$$\begin{array}{ll} \text{find} & \mathrm{tr}(X) \\ \text{s.t.} & \mathcal{A}(X) = b, \quad X \succeq 0 \end{array}$$

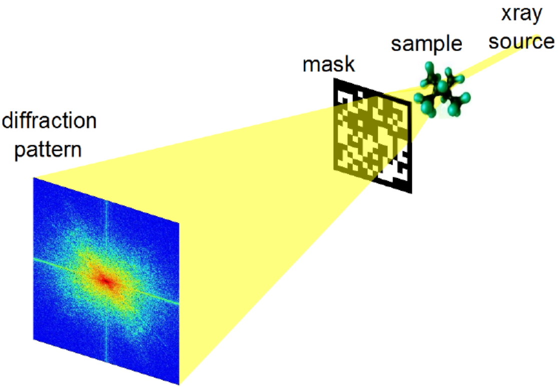
Theorem (C. and Li ('12); C., Strohmer and Voroninski ('11))

- a_k independently and uniformly sampled on unit sphere
- $m \gtrsim n$

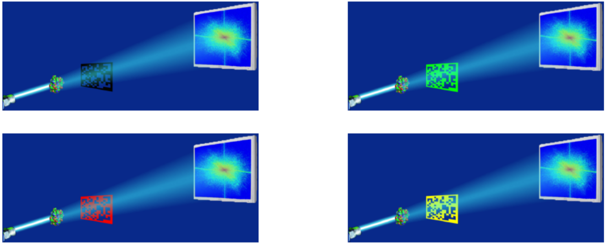
Then with prob. $1 - O(e^{-\gamma m})$, only feasible point is xx^*

$$\{X : \mathcal{A}(X) = b, \text{ and } X \succeq 0\} = \{xx^*\}$$

Extensions to physical setups



Coded diffraction

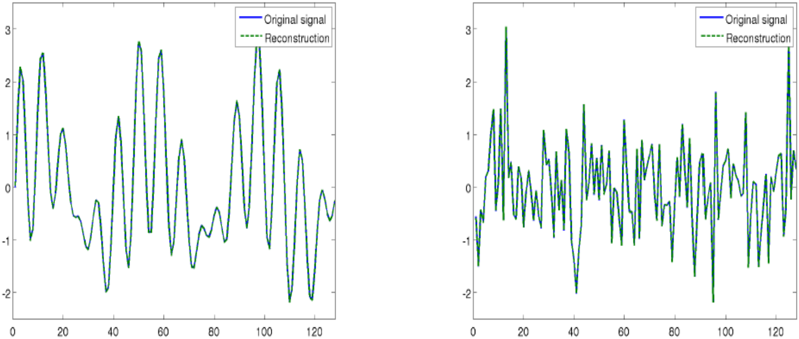


Collect diffraction patterns of modulated samples

$$|\mathcal{F}(w[t]x[t])|^2 \qquad w \in \mathcal{W}$$

Makes problem well-posed (for some choices of $\mathcal{W}$)

Exact recovery

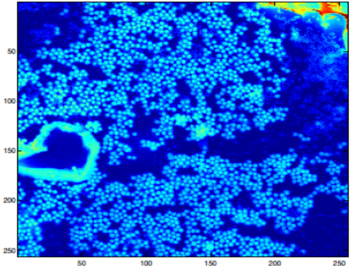


(a) Smooth signal (real part)

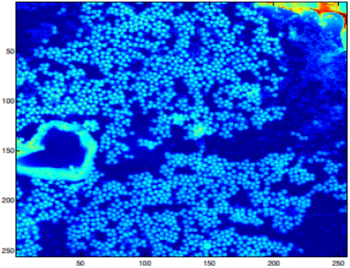
(b) Random signal (real part)

Figure: Recovery from 6 random binary masks

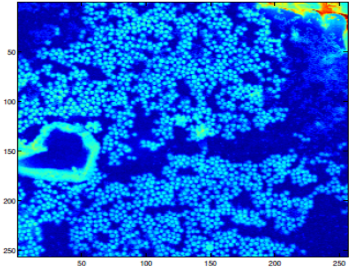
Numerical results: noiseless 2D images



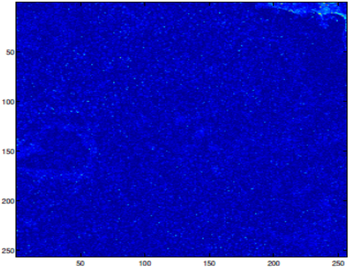
original image



3 Gaussian masks



8 binary masks



error with 8 binary masks

Outline

1. Introduction
2. Classical Phase Retrieval
3. PhaseLift
4. **PhaseCut**
5. Wirtinger Flows
6. Gauss-Newton Method

PhaseCut

- Given $A \in \mathcal{C}^{m\times n}$ and $b \in \mathbb{R}^m$

$$\text{find } x, \ \text{s.t. } |Ax| = b.$$

(Candes et al. 2011b, Alexandre d'Aspremont 2013)

- An equivalent model

$$\begin{aligned} \min_{x\in\mathcal{C}^n, y\in\mathbb{R}^m} \quad & \frac{1}{2}\|Ax-y\|_2^2 \\ \text{s.t.} \quad & |y| = b. \end{aligned}$$

PhaseCut

- Reformulation:

$$\begin{aligned} \min_{x\in\mathbb{C}^n, u\in\mathbb{C}^m} \quad & \frac{1}{2}\|Ax - \mathrm{diag}(b)u\|_2^2 \\ s.t. \quad & |u_i| = 1, , i = 1, \ldots, m. \end{aligned}$$

- Given u, the signal variable is $x = A^\dagger \mathrm{diag}(b)u$. Then

$$\begin{aligned} \min_{u\in\mathbb{C}^m} \quad & u^* M u \\ s.t. \quad & |u_i| = 1, i = 1, \ldots, m, \end{aligned}$$

  where $M = \mathrm{diag}(b)(I - AA^\dagger)\mathrm{diag}(b)$ is positive semidefinite.
- The MAXCUT problem

$$\begin{aligned} \min_{U\in S_m} \quad & Tr(UM) \\ s.t. \quad & U_{ii} = 1,\ i = 1, \cdots, m,\ U \succeq 0. \end{aligned}$$

Outline

1. Introduction
2. Classical Phase Retrieval
3. PhaseLift
4. PhaseCut
5. **Wirtinger Flows**
6. Gauss-Newton Method

Phase retrieval by non-convex optimization

Solve the equations: $y_r = |\langle a_r, x\rangle|^2, \quad r = 1, 2, ..., m.$

- **Gaussian model:**

$$a_r \in \mathbb{C}^n \overset{i.i.d.}{\sim} \mathcal{N}(0, I/2) + i\mathcal{N}(0, I/2).$$

- **Coded Diffraction model:**

$$y_r = \left|\sum_{t=0}^{n-1} x[t]\bar{d}_l(t)e^{-i2\pi kt/n}\right|^2, \quad r = (l, k), \quad 0 \le k \le n-1, \quad 1 \le l \le L.$$

Nonlinear least square problem:

$$\min_{z\in\mathbb{C}^n} \quad f(z) = \frac{1}{4m}\sum_{k=1}^{m}(y_k - |\langle a_k, z\rangle|^2)^2$$

- Pro: operates over vectors and not matrices
- Con: f is nonconvex, many local minima

Wirtinger flow: C., Li and Soltanolkotabi ('14)

Strategies:

- Start from a sufficiently accurate initialization
- Make use of Wirtinger derivative

$$\begin{aligned} f(z) &= \frac{1}{4m}\sum_{k=1}^{m}(y_k - |\langle a_k, z\rangle|^2)^2 \\ \nabla f(z) &= \frac{1}{m}\sum_{k=1}^{m}(|\langle a_k, z\rangle|^2 - y_k)(a_k a_k^*)z \end{aligned}$$

- Careful iterations to avoid local minima

Algorithm: Gaussian model

- **Spectral Initialization:**
  1. Input measurements $\{a_r\}$ and observation $\{y_r\}(r = 1, 2, ..., m)$.
  2. Calculate z_0 to be the leading eigenvector of $Y = \frac{1}{m} \sum_{r=1}^{m} y_r a_r a_r^*$.
  3. Normalize z_0 such that $\|z_0\|^2 = n \frac{\sum_r y_r}{\sum_r \|a_r\|^2}$.
- **Iteration via Wirtinger derivatives:** for $\tau = 0, 1, \ldots$

$$z_{\tau+1} = z_\tau - \frac{\mu_{\tau+1}}{\|z_0\|^2} \nabla f(z_\tau)$$

Convergence property: Gaussian model

distance (up to global phase)

$$\mathbf{dist}(z, x) = \arg \min_{\pi \in [0,2\pi]} \|z - e^{i\phi} x\|$$

Theorem

Convergence for Gaussian model (C. Li and Soltanolkotabi ('14))

- *number of samples* $m \gtrsim n \log n$
- *Step size* $\mu \leq c/n (c > 0)$

Then with probability at least $1 - 10e^{-\gamma n} - 8/n^2 - me^{-1.5n}$*, we have dist*$(z_0, x) \leq \frac{1}{8}\|x\|$ *and after* τ *iteration*

$$\mathbf{dist}(z_\tau, x) \leq \frac{1}{8}(1 - \frac{\mu}{4})^{\tau/2}\|x\|.$$

Here γ *is a positive constant.*

Numerical results: 1D signals

Consider the following two kinds of signals:

- **Random low-pass signals:**

  $$x[t] = \sum_{k=-(M/2-1)}^{M/2} (X_k + iY_k) e^{2\pi i (k-1)(t-1)/n},$$

  with M=n/8 and X_k and Y_k are i.i.d. $\mathcal{N}(0,1)$.

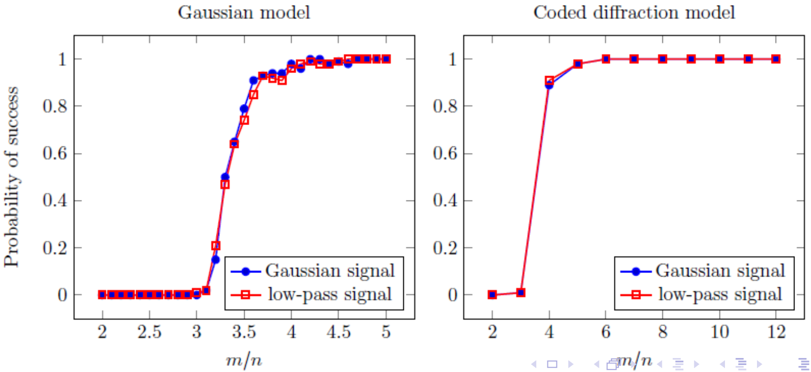
- **Random Guassian signals:** where $x \in \mathbb{C}^n$ is a random complex Gaussian vector with i.i.d. entries of the form

  $$X[t] = X + iY,$$

  with X and Y distributed as $\mathcal{N}(0,1/2)$.

Success rate

- Set $n = 128$.
- Apply 50 iterations of the power method as initialization.
- Set $\mu_\tau = \min(1 - e^{-\tau/\tau_0}, 0.2)$, where $\tau_0 \approx 330$.
- Stop after 2500 iterations, and declare a trial successful if the relative error of the reconstruction $dist(\hat{x}, x)/\|x\|$ falls below 10^{-5}.
- The empirical probability of success is an average over 100 trials.



Numerical results: natural images

- View RGB image as $n_1 \times n_2 \times 3$ array, and run the WF algorithm separately on each color band.
- Apply 50 iterations of the power method as initialization.
- Set the step length parameter $\mu_\tau = min(1 - exp(-\tau/\tau_0), 0.4)$, where $\tau_0 \approx 330$. Stop after 300 iterations.
- One FFT unit is the amount of time it takes to perform a single FFT on an image of the same size.

Numerical results: natural images

Figure: Milky way Galaxy. Image size is 1080×1920 pixels; timing is 1318.1 sec or 41900 FFT units. The relative error is 9.3×10^{-16}.

Recall the main theorems

Theorem

Convergence for Gaussian model (C. Li and Soltanolkotabi ('14))

- *number of samples* $m \gtrsim n \log n$
- *Step size* $\mu \leq c/n (c > 0)$

Then with probability at least $1 - 10e^{-\gamma n} - 8/n^2 - me^{-1.5n}$*, we have dist*$(z_0, x) \leq \frac{1}{8}\|x\|$ *and after* τ *iteration*

$$\textbf{dist}(z_\tau, x) \leq \frac{1}{8}(1 - \frac{\mu}{4})^{\tau/2}\|x\|.$$

Here γ *is a positive constant.*

Regularity condition

Definition

Definition We say that the function f satisfies the regularity condition or $RC(\alpha, \beta, \epsilon)$ if for all vectors $z \in E(\epsilon)$ we have

$$Re\left(\langle \nabla f(z), z - xe^{i\phi(z)}\rangle\right) \geq \frac{1}{\alpha} dist^2(z, x) + \frac{1}{\beta} \|\nabla f(z)\|^2.$$

- $\phi(z) := \arg\min_{\phi \in [0, 2\pi]} \|z - e^{i\phi} x\|$.
- $dist(z, x) := \|z - e^{i\phi(z)} x\|$.
- $E(\epsilon) := \{z \in \mathbb{C}^n : dist(z, x) \leq \epsilon\}$.

Proof of convergence

Lemma 1

Assume that f obeys $RC((\alpha, \beta, \epsilon))$ for all $z \in E(\epsilon)$. Furthermore, suppose $z_0 \in E(\epsilon)$, and assume $0 < \mu \leq 2/\beta$. Consider the following update

$$z_{\tau+1} = z_\tau - \mu \nabla f(z_\tau).$$

Then for all τ we have $z_\tau \in E(\epsilon)$ and

$$dist^2(z_\tau, x) \leq \left(1 - \frac{2\mu}{\alpha}\right)^\tau dist^2(z_0, x).$$

Proof of convergence

Proof.

We prove that if $z \in E(\epsilon)$ then for all $0 < \mu \leq 2/\beta$

$$z_+ = z - \mu \nabla f(z)$$

obeys

$$dist^2(z_+, x) \leq \left(1 - \frac{2\mu}{\alpha}\right) dist^2(z, x).$$

Then the lemma holds by inductively applying the equation above.

Proof of convergence

Simple algebraic manipulations together with the regularity condition give

$$
\begin{aligned}
& \left\| z_+ - xe^{i\phi(z)} \right\|^2 \\
= \quad & \left\| z - xe^{i\phi(z)} - \mu \nabla f(z) \right\|^2 \\
= \quad & \left\| z - xe^{i\phi(z)} \right\|^2 - 2\mu Re \left(\langle \nabla f(z), z - xe^{i\phi(z)} \rangle \right) + \mu^2 \left\| \nabla f(z) \right\|^2 \\
\leq \quad & \left\| z - xe^{i\phi(z)} \right\|^2 - 2\mu \left(\frac{1}{\alpha} \left\| z - xe^{i\phi(z)} \right\|^2 + \frac{1}{\beta} \left\| \nabla f(z) \right\|^2 \right) \\
& + \mu^2 \left\| \nabla f(z) \right\|^2 \\
= \quad & \left(1 - \frac{2\mu}{\alpha} \right) \left\| z - xe^{i\phi(z)} \right\|^2 + \mu \left(\mu - \frac{2}{\beta} \right) \left\| \nabla f(z) \right\|^2 \\
\leq \quad & \left(1 - \frac{2\mu}{\alpha} \right) \left\| z - xe^{i\phi(z)} \right\|^2 ,
\end{aligned}
$$

which concludes the proof.

Proof of regularity condition

We will make use of the following lemma:

Lemma 2

1. x is a solution obeying $\|x\| = 1$, and is independent from the sampling vectors;
2. $m \geq c(\delta) n \log n$ in Gaussian model or $L \geq c(\delta) \log^3 n$ in CD model.

Then,

$$\left\| \nabla^2 f(x) - \mathbb{E} \nabla^2 f(x) \right\| \leq \delta$$

holds with pabability at least $1 - 10e^{-\gamma n} - 8/n^2$ and $1 - (2L+1)/n^3$ for the Gaussian and CD model, respectively.

- The concentration of the Hessian matrix at the optimizers.

Proof of regularity condition

Based on the lemma above with $\delta = 0.01$, we prove the regularity condition by establishing the local curvature condition and the local smoothness condition.

Local curvature condition

We say that the function f satisfies the local curvature condition or $LCC(\alpha, \epsilon, \delta)$ if for all vectors $z \in E(\epsilon)$,

$$Re\left(\langle \nabla f(z), z - xe^{i\phi(z)} \rangle\right) \geq \left(\frac{1}{\alpha} + \frac{1-\delta}{4}\right) dist^2(z, x) + \frac{1}{10m} \sum_{r=1}^{m} \Big| a_r^*(z - xe^{i\phi(z)}$$

The LCC condition states that the function curves sufficiently upwards along most directions near the curve of global optimizers.

For the CD model, LCC holds with $\alpha \geq 30$ and $\epsilon = \frac{1}{8\sqrt{n}}$;

For the Gaussian model, LCC holds with $\alpha \geq 8$ and $\epsilon = \frac{1}{8}$.

Proof of regularity condition

Local smoothness condition

We say that the function f satisfies the local smoothness condition or $LSC(\beta, \epsilon, \delta)$ if for all vectors $z \in E(\epsilon)$ we have

$$\|\nabla f(z)\|^2 \leq \beta \left(\frac{(1-\delta)}{4} dist^2(z,x) + \frac{1}{10m} \sum_{r=1}^{m} \left| a_r^*(z - xe^{i\phi(z)}) \right|^4 \right).$$

The LSC condition states that the gradient of the function is well behaved near the curve of global optimizers. Using $\delta = 0.01$, LSC holds with $\beta \geq 550 + 3n$

$$\beta \geq 550 \quad for \quad \epsilon = 1/(8\sqrt{n}),$$
$$\beta \geq 550 + 3n \quad for \quad \epsilon = 1/8.$$

Proof of regularity condition

In conclusion, when $\delta = 0.01$, for the Gaussian model, the regularity condition holds with

$$\alpha \geq 8, \beta \geq 550 + 3n, \ \mathit{and} \ \epsilon = 1/8.$$

while for the CD model, the regularity condition holds with

$$\alpha \geq 30, \beta \geq 550, \ \mathit{and} \ \epsilon = 1/(8\sqrt{n}),$$

Therefore, for the Gaussian model, linear convergence holds if the initial points satisfies $dist(z_0, x) \leq 1/8$; for the CD model, linear convergence holds if $dist(z_0, x) \leq 1/(8\sqrt{n})$.

Proof of initialization

Recall the initialization algorithm:

1. Input measurements $\{a_r\}$ and observation $\{y_r\}(r = 1, 2, ..., m)$.
2. Calculate z_0 to be the leading eigenvector of $Y = \frac{1}{m}\sum_{r=1}^{m} y_r a_r a_r^*$.
3. Normalize z_0 such that $\|z_0\|^2 = n\frac{\sum_r y_r}{\sum_r \|a_r\|^2}$.

Ideas:

$$\mathbb{E}\left[\frac{1}{m}\sum_{r=1}^{m} y_r a_r a_r^*\right] = I + 2xx^*,$$

and any leading eigenvector of $I + 2xx^*$ is of the form λx. Therefore, by the strong law of large number, the initialization step would recover the direction of x perfectly as long as there are enough samples.

Proof of initialization

In the detailed proof, we will use the following lemma:

Lemma 3

In the setup of Lemma 2,

$$\left\| I - \frac{1}{m} \sum_{r=1}^{m} a_r a_r^* \right\| \le \delta,$$

holds with probability at least $1 - 2e^{-\gamma m}$ for the Gaussian model and $1 - 1/n^2$ for the CD model. On this event,

$$(1-\delta)\|h\|^2 \le \frac{1}{m} \sum_{r=1}^{m} |a_r^* h|^2 \le (1+\delta)\|h\|^2$$

holds for all $h \in \mathbb{C}^n$.

Proof of initialization

Detailed proof:
Lemma 2 gives

$$\|Y - (xx^* + \|x\|^2 I)\| \leq \epsilon := 0.001.$$

Let $\tilde{z}_0$ be the unit eigenvector corresponding to the top eigenvalue λ_0 of Y, then

$$|\lambda_0 - (|\tilde{z}_0 x|^2 + 1)| = |\tilde{z}_0^*(Y - (xx^* + I))\tilde{z}_0| \leq \|Y - (xx^* + I)\| \leq \epsilon.$$

Therefore, $|\tilde{z}_0^* x|^2 \geq \lambda_0 - 1 - \epsilon$. Meanwhile, since λ_0 is the top eigenvalue of Y, and $\|x\| = 1$, we have

$$\lambda_0 \geq x^* Y x = x^*(Y - (I + x^* x))x + 2 \geq 2 - \epsilon.$$

Combining the above two inequalities together, we have

$$|\tilde{z}_0^* x|^2 \geq 1 - 2\epsilon \ \Rightarrow \ dist^2(\tilde{z}_0, x) \leq 2 - 2\sqrt{1 - 2\epsilon} \leq \frac{1}{256} \ \Rightarrow \ dist(\tilde{z}_0, x) \leq \frac{1}{16}.$$

Proof of initialization

Now consider the normalization. Recall that $z_0 = \left(\sqrt{\frac{1}{m} \sum_{r=1}^{m} |a_r^* x|^2} \right) \tilde{z}_0$.

By Lemma 3, with high probability we have

$$||\|z_0\| - 1| \leq \left| \|z_0\|^2 - 1 \right| = \left| \frac{1}{m} \sum_{r=1}^{m} |a_r^* x|^2 - 1 \right| \leq \delta < \frac{1}{16}.$$

Therefore, we have

$$dist(z_0, x) \leq \|z_0 - \tilde{z}_0\| + dist(\tilde{z}_0, x) \leq ||\|z_0\| - 1| + dist(\tilde{z}_0, x) \leq \frac{1}{8}.$$

Outline

1. Introduction
2. Classical Phase Retrieval
3. PhaseLift
4. PhaseCut
5. Wirtinger Flows
6. Gauss-Newton Method

Nonlinear least square problem

$$\min_{z\in\mathbb{C}^n} \quad f(z) = \frac{1}{4m}\sum_{k=1}^{m}(y_k - |\langle a_k, z\rangle|^2)^2$$

Using Wirtinger derivative:

$$\begin{aligned}
\mathbf{z} \quad &:= \quad \begin{bmatrix} z \\ \bar{z} \end{bmatrix};\\
g(z) \quad &:= \quad \nabla_c f(z) = \frac{1}{m}\sum_{r=1}^{m}\left(|a_r^{\mathrm{T}}z|^2 - y_r\right)\begin{bmatrix} (a_r a_r^{\mathrm{T}})z \\ (\bar{a}_r a_r^{\top})\bar{z} \end{bmatrix};\\
J(z) \quad &:= \quad \frac{1}{\sqrt{m}}\sum_{r=1}^{m}\begin{bmatrix} |a_1^* z|a_1, & |a_2^* z|a_2, & \cdots, & |a_m^* z|a_m \\ |a_1^* z|\bar{a}_1, & |a_2^* z|\bar{a}_2, & \cdots, & |a_m^* z|\bar{a}_m \end{bmatrix}^{\mathrm{T}};\\
\Psi(z) \quad &:= \quad J(z)^{\mathrm{T}}J(z) = \frac{1}{m}\sum_{r=1}^{m}\begin{bmatrix} |a_r^{\mathrm{T}}z|^2 a_r a_r^{\mathrm{T}} & (a_r^{\mathrm{T}}z)^2 a_r a_r^{\top} \\ (a_r^{\overline{\mathrm{T}}}z)^2 \bar{a}_r a_r^{\mathrm{T}} & |a_r^{\mathrm{T}}z|^2 \bar{a}_r a_r^{\top} \end{bmatrix}.
\end{aligned}$$

The Modified LM method for Phase Retrieval

Levenberg-Marquardt Iteration:

$$\mathbf{z}_{k+1} = \mathbf{z}_k - \left(\Psi(z_k) + \mu_k I\right)^{-1} g(z_k)$$

Algorithm

1. **Input:** Measurements $\{a_r\}$, observations $\{y_r\}$. Set $\epsilon \geq 0$.
2. Construct z_0 using the spectral initialization algorithms.
3. **While** $\|g(z_k)\| \geq \epsilon$ **do**
   - Compute s_k by solving equation

     $$\Psi_{z_k}^{\mu_k} s_k = \left(\Psi(z_k) + \mu_k I\right) s_k = -g(z_k).$$

     until

     $$\left\|\Psi_{z_k}^{\mu_k} s_k + g(z_k)\right\| \leq \eta_k \|g(z_k)\|.$$

   - Set $\mathbf{z}_{k+1} = \mathbf{z}_k + s_k$ and $k := k + 1$.
3. **Output:** z_k.

Convergence of the Gaussian Model

Theorem

If the measurements follow the Gaussian model, the LM equation is solved accurately ($\eta_k = 0$ for all k), and the following conditions hold:

- $m \geq cn \log n$, *where* c *is sufficiently large;*
- *If* $f(z_k) \geq \frac{\|z_k\|^2}{900n}$, *let* $\mu_k = 70000n\sqrt{nf(z_k)}$; *if else, let* $\mu_k = \sqrt{f(z_k)}$.

Then, with probability at least $1 - 15e^{-\gamma n} - 8/n^2 - me^{-1.5n}$, *we have* $dist(z_0, x) \leq (1/8)\|x\|$, *and*

$$\mathbf{dist}(z_{k+1}, x) \leq c_1 dist(z_k, x),$$

Meanwhile, once $f(z_s) < \frac{\|z_s\|^2}{900n}$, *for any* $k \geq s$ *we have*

$$dist(z_{k+1}, x) < c_2 dist(z_k, x)^2.$$

Convergence of the Gaussian Model

In the theorem above,

$$c_1 := \begin{cases} \left(1 - \frac{||x||}{4\mu_k}\right), & \text{if } f(z_k) \geq \frac{1}{900n}\|z_k\|^2; \\ \frac{4.28+5.56\sqrt{n}}{9.89\sqrt{n}}, & \text{otherwise.} \end{cases}$$

and

$$c_2 = \frac{4.28 + 5.56\sqrt{n}}{\|x\|}.$$

Key to proof

Lower bound of GN matrix's second smallest eigenvalue

For any $y,\ z \in \mathbb{C}^n$, $Im(y^*z) = 0$, we have:

$$\mathbf{y}^*\Psi(z)\mathbf{y} \geq \|y\|^2\|z\|^2,$$

holds with high probability.

$$Im(y^*z) = 0 \quad \Rightarrow \quad \|(\Psi_z^\mu)^{-1}\mathbf{y}\| \leq \frac{2}{\|z\|^2 + \mu}\|\mathbf{y}\|.$$

Key to proof

Local error bound property

$$\frac{1}{4}\mathbf{dist}(z,x)^2 \leq f(z) \leq 8.04\mathbf{dist}(z,x)^2 + 6.06n\mathbf{dist}(z,x)^4,$$

holds for any z satisfying $\mathbf{dist}(z,x) \leq \frac{1}{8}$.

Regularity condition

$$\mu(z)\mathbf{h}^* \left(\Psi_z^\mu\right)^{-1} g(\mathbf{z}) \geq \frac{1}{16}\|\mathbf{h}\|^2 + \frac{1}{64100n\|h\|}\|g(\mathbf{z})\|^2$$

holds for any $z = x + h$, $\|h\| \leq \frac{1}{8}$, and $f(z) \geq \frac{\|z\|^2}{900n}$.

Convergence for the inexact LM method

Theorem

Convergence of the inexact LM method for the Gaussian model:

- $m \gtrsim n \log n$;
- μ_k *takes the same value as in the exact LM method for the Gaussian model;*
- $\eta_k \le \frac{(1-c_1)\mu_k}{25.55 n \|z_k\|}$ *if* $f(z_k) \ge \frac{\|z_k\|^2}{900n}$; *otherwise* $\eta_k \le \frac{(4.33\sqrt{n}-4.28)\mu_k \|g_k\|}{372.54 n^2 \|z_k\|^3}$.

Then, with probability at least $1 - 15e^{-\gamma n} - 8/n^2 - me^{-1.5n}$, *we have* $dist(z_0, x) \le \frac{1}{8}\|x\|$, *and*

$$
\begin{aligned}
dist(z_{k+1}, x) &\le \frac{1+c_1}{2} dist(z_k, x), &\quad \text{for all } k = 0, 1, \ldots \\
dist(z_{k+1}, x) &\le \frac{9.89\sqrt{n} + c_2\|x\|}{2\|x\|} dist(z_k, x)^2, &\quad \text{for all } f(z_k) < \frac{\|z_k\|^2}{900n}.
\end{aligned}
$$

Here c_1 *and* c_2 *take the same values as in the exact algorithm for the Gaussian model.*

Solving the LM Equation: PCG

Solve

$$(\Psi_k + \mu_k I)u = g_k$$

by Pre-conditioned Conjugate Gradient Method:

$$M^{-1}(\Psi_k + \mu_k I)u = M^{-1}g_k, \qquad M = \Phi_k + \mu_k I.$$

$$\Phi(z) := \begin{bmatrix} zz^* & 2zz^T \\ 2\bar{z}z^* & \bar{z}z^T \end{bmatrix} + \|z\|^2 I_{2n}$$

- **small condition number**
- **Easy to inverse:** $M = (\mu_k + \|z_k\|^2)I + M_1$, where M_1 is rank-2 matrix.

Solving the LM Equation: PCG

- small condition number.

Lemma

Consider solving the equation $(\Phi_z^\mu)^{-1}\Psi_z^\mu s = (\Phi_z^\mu)^{-1} g(\mathbf{z})$ *by the CG method from* $s_0 := -(\Phi_z^\mu)^{-1} g(\mathbf{z})$. *Let* s_* *be the solution of the system. Define* $V := \{\times : \times = [x^*, x^T]^*, x \in \mathbb{C}^n\}$. *Then,* V *is an invariant subspace of* $(\Phi_z^\mu)^{-1}\Psi_z^\mu$, *and* $s_0, s_* \in V$. *Meanwhile, choosing* $\mu_k = Kn\sqrt{f(z)}$, *then the eigenvalues of* $(\Phi_z^\mu)^{-1}\Psi_z^\mu$ *on* V *satisfy:*

$$1 - \frac{57}{K\sqrt{n}} \leq \lambda \leq 1 + \frac{57}{K\sqrt{n}}.$$

Solving the LM Equation: PCG

- Easy to inverse.

Calculate by Sherman-Morrison-Woodbury theorem:

$$(\Phi_z^\mu)^{-1} = aI_{2n} + b\begin{bmatrix} z \\ \bar{z} \end{bmatrix}[z^*, z^T] + c\begin{bmatrix} z \\ -\bar{z} \end{bmatrix}[z^*, -z^T]$$

where

$$a = \frac{1}{\|z\|^2 + \mu}, \quad b = -\frac{3}{2(\|z\|^2 + \mu)(4\|z\|^2 + \mu)}, \quad c = \frac{1}{2(\|z\|^2 + \mu)\mu}.$$

Dynamic Programming: MDP

`http://bicmr.pku.edu.cn/~wenzw/bigdata2022.html`

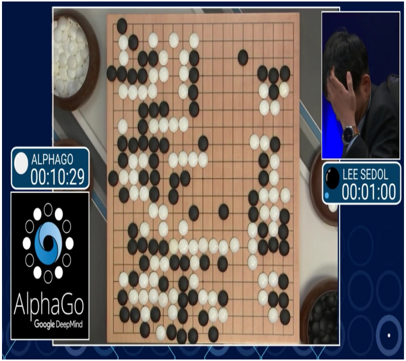
Acknowledgement: this slides is based on OpenAI Spinning Up and and Prof. Shipra Agrawal's lecture notes

Outline

1. Introduction
2. Dynamic Programming
3. Markov Decision Process (MDP)
4. Bellman Equation
5. Example: Airfare Pricing
6. Iterative algorithms (discounted reward case)
    - Value Iteration
    - Q-value iteration
    - Policy iteration

What Can RL Do?

RL methods have recently enjoyed a wide variety of successes. For example, it's been used to teach computers to control robots in simulation



What Can RL Do?

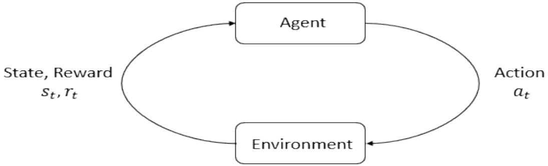
It's also famously been used to create breakthrough AIs for sophisticated strategy games, most notably ‘Go‘ and ‘Dota‘, taught computers to ‘play Atari games‘ from raw pixels, and trained simulated robots ‘to follow human instructions‘.

- Go: `https://deepmind.com/research/alphago`
- Dota: `https://blog.openai.com/openai-five`
- play Atari games: `https://deepmind.com/research/dqn/`
- to follow human instructions: `https://blog.openai.com/deep-reinforcement-learning-from-human-preferences`

RL: agent and environment

- **environment**: the world that the agent lives in and interacts with. At every step of interaction, the agent sees a (possibly partial) observation of the state of the world, and then decides on an action to take. The environment changes when the agent acts on it, but may also change on its own.

- **agent**: perceives a **reward** signal from the environment, a number that tells it how good or bad the current world state is. The goal of the agent is to maximize its cumulative reward, called **return**. Reinforcement learning methods are ways that the agent can learn behaviors to achieve its goal.



Outline

1. Introduction
2. Dynamic Programming
3. Markov Decision Process (MDP)
4. Bellman Equation
5. Example: Airfare Pricing
6. Iterative algorithms (discounted reward case)
   - Value Iteration
   - Q-value iteration
   - Policy iteration

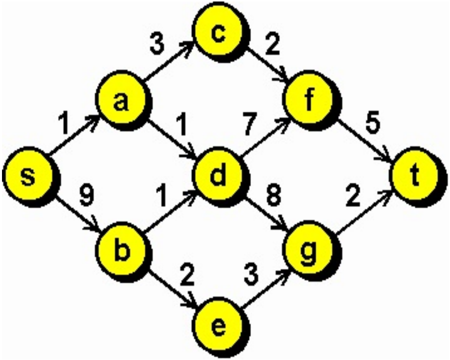
Dynamic Programming

- Basically, we want to solve a big problem that is hard
- We can first solve a few smaller but similar problems, if those can be solved, then the solution to the big problem will be easy to get
- To solve each of those smaller problems, we use the same idea, we first solve a few even smaller problems.
- Continue doing it, we will eventually encounter a problem we know how to solve

Dynamic programming has the same feature, the difference is that at each step, there might be some optimization involved.

Shortest Path Problem

You have a graph, you want to find the shortest path from s to t



Here we use d_{ij} to denote the distance between node i and node j

DP formulation for the shortest path problem

Let V_i denote the shortest distance between i to t.

- Eventually, we want to compute V_s
- It is hard to directly compute V_i in general
- However, we can just look one step

We know if the first step is to move from i to j, the shortest distance we can get must be $d_{ij} + V_j$.

- To minimize the total distance, we want to choose j to minimize $d_{ij} + V_j$

To write into a math formula, we get

$$V_i = \min_j \{d_{ij} + V_j\}$$

DP for shortest path problem

We call this the recursion formula

$$V_i = \min_j \{d_{ij} + V_j\} \quad \text{for all } i$$

We also know if we are already at our destination, then the distance is 0. i.e.,

$$V_t = 0$$

The above two equations are the DP formulation for this problem

Solve the DP

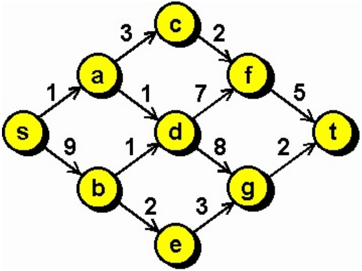
Given the formula, how to solve the DP?

$$V_i = \min_j \{d_{ij} + V_j\} \quad \text{for all } i, \quad V_t = 0$$

We use backward induction.

- From the last node (which we know the value), we solve the values of V's backwardly.

Example

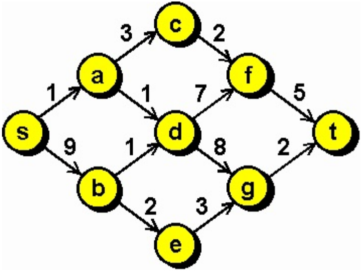


- We have $V_t = 0$. Then we have

$$V_f = \min_{(f,j) \text{ is a path}} \{d_{fj} + V_j\}$$

- Here, we only have one path, thus $V_f = 5 + V_t = 5$
- Similarly, $V_g = 2$

Example Continued 1



- We have $V_t = 0, V_f = 5$ and $V_g = 2$
- Now consider c, d, e. For c and e there is only one path

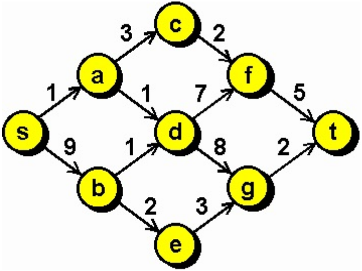
$$V_c = d_{cf} + V_f = 7, \quad V_e = d_{eg} + V_g = 5$$

- For d, we have

$$V_g = \min_{(d,j) \text{ is a path}} \{d_{dj} + V_j\} = \min\{d_{df} + V_f, d_{dg} + V_g\} = 10$$

- The optimal way to choose at d is go to g

Example Continued 2



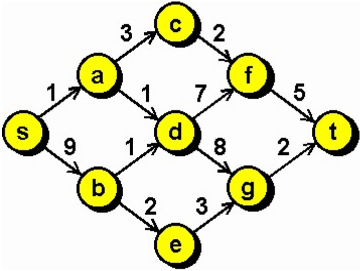
- We got $V_c = 7, V_d = 10$ and $V_e = 5$. Now we compute V_a and V_b

$$V_a = \min\{d_{ac} + V_c, d_{ad} + V_d\} = \min\{3 + 7, 1 + 10\} = 10$$

$$V_b = \min\{d_{bd} + V_d, d_{be} + V_e\} = \min\{1 + 10, 2 + 5\} = 7$$

and the optimal path to go at a is to choose c, and the optimal path to go at b is to choose e.

Example Continued 3



- Finally, we have

$$V_s = \min\{d_{sa} + V_a, d_{sb} + V_b\} = \min\{1 + 10, 9 + 7\} = 11$$

  and the optimal path to go at s is to choose a

- Therefore, we found the optimal path is 11, and by connecting the optimal path, we get

$$s \to a \to c \to f \to t$$

Summary of the example

In the example, we saw that we have those V_i's, indicating the shortest length to go from i to t.

- We call this V the value function

We also have those nodes $s, a, b, \ldots, g, t$

- We call them the states of the problem
- The value function is a function of the state

And the recursion formula

$$V_i = \min_j \{d_{ij} + V_j\} \quad \text{for all } i$$

connects the value function at different states. It is known as the Bellman equation

Stochastic DP

In some cases, when you choose action a at x, the next state is not certain (e.g., you decide a price, but the demand is random).

- There will be $p(x, y, a)$ probability you move from x to y if you choose action $a \in A(x)$

Then the recursion formula becomes:

$$V(x) = \min_{a \in A(x)} \{r(x, a) + \sum_y p(x, y, a) V(y)\}$$

or if we choose to use the expectation notation:

$$V(x) = \min_{a \in A(x)} \{r(x, a) + \mathbf{E} V(x, a))\}$$

Example: Stochastic Shortest Path Problem

Stochastic setting:

- One no longer controls which exact node to jump to next
- Instead one can choose between different actions $a \in \mathcal{A}$
- Each action a is associated with a set of transition probabilities $p(j|i;a)$ for all $i, j \in \mathcal{S}$.
- The arc length may be random w_{ija}

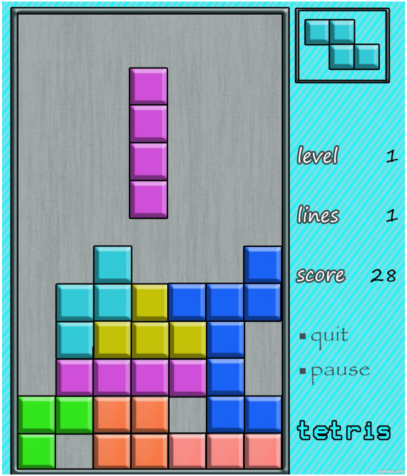
Objective:

- One needs to decide on the action for every possible current node. In other words, one wants to find a policy or strategy that maps from $\mathcal{S}$ to A.

Bellman Equation for Stochastic SSP:

$$V(i) = \min_a \sum_{j \in S} p(j|i;a)(w_{ija} + V(j)), \quad i \in \mathcal{S}$$

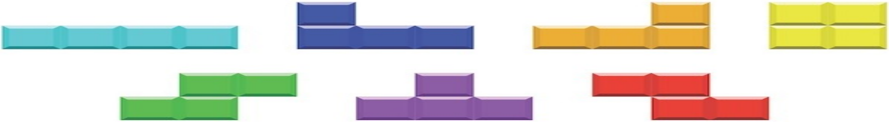
Tetris



- Height: 12
- Width: 7
- Rotate and move the falling shape
- Gravity related to current height
- Score when eliminating an entire level
- Game over when reaching the ceiling

DP Model of Tetris

- State: The current board, the current falling tile, predictions of future tiles
- Termination state: when the tiles reach the ceiling, the game is over with no more future reward
- Action: Rotation and shift
- System Dynamics: The next board is deterministically determined by the current board and the player's placement of the current tile. The future tiles are generated randomly.
- Uncertainty: Randomness in future tiles
- Transitional cost g: If a level is cleared by the current action, score 1; otherwise score 0.
- Objective: Expectation of total score.



Interesting facts about Tetris

- First released in 1984 by Alexey Pajitnov from the Soviet Union
- Has been proved to be NP-complete.
- Game will be over with probability 1.
- For a 12×7 board, the number of possible states $\approx 2^{12\times 7} \approx 10^{25}$
- Highest score achieved by human $\approx$ 1 million
- Highest score achieved by algorithm $\approx$ 35 million (average performance)

Outline

1. Introduction
2. Dynamic Programming
3. Markov Decision Process (MDP)
4. Bellman Equation
5. Example: Airfare Pricing
6. Iterative algorithms (discounted reward case)
    - Value Iteration
    - Q-value iteration
    - Policy iteration

States and Observations

- **state**: s is a complete description of the state of the world. There is no information about the world which is hidden from the state. An **observation** o is a partial description of a state, which may omit information.
- In deep RL, we almost always represent states and observations by a "real-valued vector, matrix, or higher-order tensor". For instance, a visual observation could be represented by the RGB matrix of its pixel values; the state of a robot might be represented by its joint angles and velocities.
- When the agent is able to observe the complete state of the environment, we say that the environment is **fully observed**. When the agent can only see a partial observation, we say that the environment is **partially observed**.
- We often write that the action is conditioned on the state, when in practice, the action is conditioned on the observation because the agent does not have access to the state.

Action Spaces

- Different environments allow different kinds of actions. The set of all valid actions in a given environment is often called the **action space**. Some environments, like Atari and Go, have **discrete action spaces**, where only a finite number of moves are available to the agent. Other environments, like where the agent controls a robot in a physical world, have **continuous action spaces**. In continuous spaces, actions are real-valued vectors.
- This distinction has some quite-profound consequences for methods in deep RL. Some families of algorithms can only be directly applied in one case, and would have to be substantially reworked for the other.

Markov decision processes

- The defining property of MDPs is the Markov property which says that the future is independent of the past given the current state. This essentially means that the state in this model captures all the information from the past that is relevant in determining the future states and rewards.
- A Markov Decision Process (MDP) is specified by a tuple (S, s_1, A, P, R, H), where S is the set of states, s_1 is the starting state, A is the set of actions. The process proceeds in discrete rounds $t = 1, 2, \cdots, H$, starting in the initial state s_1. In every round, t the agent observes the current state $s_t \in S$, takes an action $a_t \in A$, and observes a feedback in form of a reward signal $r_{t+1} \in \mathbb{R}$. The agent then observes transition to the next state $s_{t+1} \in S$.

Formal definition

- The probability of transitioning to a particular state depends only on current state and action, and not on any other aspect of the history. The matrix $P \in [0,1]^{S \times A \times S}$ specifies these probabilities. That is,

$$\begin{aligned} Pr(s_{t+1} = s' \mid history\, till\, time\, t) &= Pr(s_{t+1} = s' \mid s_t = s, a_t = a) \\ &= P(s, a, s') \end{aligned}$$

- The reward distribution depends only on the current state and action. So, that the expected reward at time t is a function of current state and action. A matrix R specifies these rewards.

$$\mathbb{E}[r_{t+1} \mid history\, till\, time\, t] = \mathbb{E}[r_{t+1} \mid s_t = s, a_t = a] = R(s, a)$$

- Let $R(s, a, s')$ be the expected (or deterministic) reward when action a is taken in state s and transition to state s' is observed. Then, we can obtain the same model as above by defining

$$R(s, a) = \mathbb{E}[r_{t+1} \mid s_t = s, a_t = a] = \mathbb{E}_{s' \sim P(s,a)}[R(s, a, s')]$$

Policy

- A policy specifies what action to take at any time step. A history dependent policy at time t is a mapping from history till time t to an action. A Markovian policy is a mapping from state space to action π: $S \rightarrow A$. Due to Markovian property of the MDP, it suffices to consider Markovian policies (in the sense that for any history dependent policy same performance can be achieved by a Markovian policy). Therefore, in this text, policy refers to a Markovian policy.
- A deterministic policy π: $S \rightarrow A$ is mapping from any given state to an action. A randomized policy $\pi : S \rightarrow \Delta^A$ is a mapping from any given state to a distribution over actions. Following a policy π_t at time t means that if the current state $s_t = s$, the agent takes action $a_t = \pi_t(s)$ (or $a_t \sim \pi(s)$ for randomized policy). Following a stationary policy π means that $\pi_t = \pi$ for all rounds $t = 1, 2, \ldots$

Policy

- Any stationary policy π defines a Markov chain, or rather a 'Markov reward process' (MRP), that is, a Markov chain with reward associated with every transition.
- The transition probability vector and reward for this MRP in state s is given by $\Pr\left(s'|s\right) = P_s^\pi, \mathbb{E}\left[r_t|s\right] = r_s^\pi$, where P^π is an $S \times S$ matrix, and r^π is an S-dimensional vector defined as:

$$P_{s,s'}^\pi = \mathbb{E}_{a \sim \pi(s)}\left[P\left(s, a, s'\right)\right], \forall s, s' \in S$$

$$r_s^\pi = \mathbb{E}_{a \in \pi(s)}[R(s, a)]$$

- The stationary distribution (if exists) of this Markov chain when starting from state s_1 is also referred to as the stationary distribution of the policy π, denoted by d^π:

$$d^\pi(s) = \lim_{t \to \infty} \Pr\left(s_t = s|s_1, \pi\right)$$

Goals, finite horizon MDP

- The tradeoffs between immediate reward vs. future rewards of the sequential decisions and the need for planning ahead is captured by the goal of the Markov Decision Process. At a high level, the goal is to maximize some form of cumulative reward. Some popular forms are total reward, average reward, or discounted sum of rewards.

finite horizon MDP

- actions are taken for $t = 1, \ldots, H$ where H is a finite horizon. The total (discounted) reward criterion is simply to maximize the expected total (discounted) rewards in an episode of length H. (In reinforcement learning context, when this goal is used, the MDP is often referred to as an episodic MDP.) For discount $0 \leq \gamma \leq 1$, the goal is to maximize

$$\mathbb{E}\left[\sum_{t=1}^{H} \gamma^{t-1} r_t | s_1\right]$$

Infinite horizon MDP

- Expected total discounted reward criteria: The most popular form of cumulative reward is expected discounted sum of rewards. This is an asymptotic weighted sum of rewards, where with time the weights decrease by a factor of $\gamma < 1$. This essentially means that the immediate returns more valuable than those far in the future.

$$\lim_{T\to\infty} \mathbb{E}\left[\sum_{t=1}^{T} \gamma^{t-1} r_t | s_1\right]$$

Infinite horizon MDP

- Expected total reward criteria: Here, the goal is to maximize

$$\lim_{T \to \infty} \mathbb{E}\left[\sum_{t=1}^{T} r_t | s_1\right]$$

The limit may not always exist or be bounded. We are only interested in cases where above exists and is finite. This requires restrictions on reward and/or transition models. Interesting cases include the case where there is an undesirable state, the reward after reaching that state is 0. For example, end of a computer game. The goal would be to maximize the time to reach this state. (A minimization version of this model is where there is a cost associated with each state and the game is to minimize the time to reach winning state, called the shortest path problem).

Infinite horizon MDP

- Expected average reward criteria: Maximize

$$\lim_{T\to\infty} \mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} r_t | s_1\right]$$

  Intuitively, the performance in a few initial rounds does not matter here, what we are looking for is a good asymptotic performance. This limit may not always exist. Assuming bounded rewards and finite state spaces, it exists under some further conditions on policy used.

Gain of the MDP

Gain (roughly the ‘expected value objective’ or formal goal) of an MDP when starting in state s_1 is defined as (when supremum exists):

- episodic MDP:

$$J\left(s_1\right) = \sup_{\{\pi_t\}} \mathbb{E}\left[\sum_{t=1}^{H} \gamma^{t-1} r_t | s_1\right]$$

- Infinite horizon expected total reward:

$$J\left(s_1\right) = \sup_{\{\pi_t\}} \lim_{T\to\infty} \mathbb{E}\left[\sum_{t=1}^{T} r_t | s_1\right]$$

- Infinite horizon discounted sum of rewards:

$$J\left(s_1\right) = \sup_{\{\pi_t\}} \lim_{T\to\infty} \mathbb{E}\left[\sum_{t=1}^{T} \gamma^{t-1} r_t | s_1\right]$$

Gain of the MDP

- infinite horizon average reward:

$$J\left(s_1\right) = \sup_{\{\pi_t\}} \lim_{T\to\infty} \mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} r_t | s_1\right]$$

Here, expectation is taken with respect to state transition and reward distribution, supremum is taken over all possible sequence of policies for the given MDP. It is also useful to define gain ρ^{π} of a stationary policy π, which is the expected (total/total discounted/average) reward when policy π is used in all time steps. For example, for infinite average reward:

$$J^{\pi}\left(s_1\right) = \lim_{T\to\infty} \mathbb{E}\left[\frac{1}{T}\sum_{t=1}^{T} r_t | s_1\right]$$

where $a_t = \pi\left(s_t\right), t = 1, \ldots, T$

Optimal policy

- Optimal policy is defined as the one that maximizes the gain of the MDP.
- Due to the structure of MDP it is not difficult to show that it is sufficient to consider Markovian policies. Henceforth, we consider only Markovian policies.
- For infinite horizon MDP with average/discounted reward criteria, a further observation that comes in handy is that such a MDP always has a stationary optimal policy, whenever optimal policy exists. That is, there always exists a fixed policy so that taking actions specified by that policy at all time steps maximizes average/discounted/total reward.
- The agent does not need to change policies with time. This insight reduces the question of finding the best sequential decision making strategy to the question of finding the best stationary policy.

Outline

1. Introduction
2. Dynamic Programming
3. Markov Decision Process (MDP)
4. **Bellman Equation**
5. Example: Airfare Pricing
6. Iterative algorithms (discounted reward case)
    - Value Iteration
    - Q-value iteration
    - Policy iteration

Value Functions

It's often useful to know the **value** of a state, or state-action pair. By value, we mean the expected return if you start in that state or state-action pair, and then act according to a particular policy forever after. **Value functions** are used, one way or another, in almost every RL algorithm.

- The **On-Policy Value Function** $V^{\pi}(s)$, which gives the expected return if you start in state s and always act according to policy π:

$$V^{\pi}(s) = \lim_{T\to\infty} \mathbb{E}\left[\sum_{t=1}^{T} \gamma^{t-1} r_t | s_1 = s\right]$$

- The **On-Policy Action-Value Function** $Q^{\pi}(s, a)$, which gives the expected return if you start in state s, take an arbitrary action a (which may not have come from the policy), and then forever after act according to policy π:

$$Q^{\pi}(s, a) = \lim_{T\to\infty} \mathbb{E}\left[\sum_{t=1}^{T} \gamma^{t-1} r_t | s_1 = s, a_1 = a\right]$$

Value Functions

- The **Optimal Value Function** $V^*(s)$, which gives the expected return if you start in state s and always act according to the *optimal* policy in the environment:

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

- The **Optimal Action-Value Function**, $Q^*(s,a)$, which gives the expected return if you start in state s, take an arbitrary action a, and then forever after act according to the *optimal* policy in the environment:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

Value Functions

- When we talk about value functions, if we do not make reference to time-dependence, we only mean expected **infinite-horizon discounted return.** Value functions for finite-horizon undiscounted return would need to accept time as an argument. Can you think about why? Hint: what happens when time's up?
- There are two key connections between the value function and the action-value function that come up pretty often:

$$V^{\pi}(s) = \mathop{\mathbf{E}}_{a\sim\pi}\left[Q^{\pi}(s,a)\right],$$

and

$$V^{*}(s) = \max_{a} Q^{*}(s,a).$$

These relations follow pretty directly from the definitions just given: can you prove them?

The Optimal Q-Function and the Optimal Action

- There is an important connection between the optimal action-value function $Q^*(s, a)$ and the action selected by the optimal policy. By definition, $Q^*(s, a)$ gives the expected return for starting in state s, taking (arbitrary) action a, and then acting according to the optimal policy forever after.

- The optimal policy in s will select whichever action maximizes the expected return from starting in s. As a result, if we have Q^*, we can directly obtain the optimal action, $a^*(s)$, via

$$a^*(s) = \arg\max_a Q^*(s, a).$$

- Note: there may be multiple actions which maximize $Q^*(s, a)$, in which case, all of them are optimal, and the optimal policy may randomly select any of them. But there is always an optimal policy which deterministically selects an action.

Bellman Equations

- All four of the value functions obey special self-consistency equations called **Bellman equations.** The basic idea is: *The value of your starting point is the reward you expect to get from being there, plus the value of wherever you land next.*
- The Bellman equations for the on-policy value functions are

$$
\begin{aligned}
V^{\pi}(s) &= \underset{\substack{a\sim\pi \\ s'\sim P}}{\mathbf{E}} \left[R(s,a,s') + \gamma V^{\pi}(s')\right], \\
Q^{\pi}(s,a) &= \underset{s'\sim P}{\mathbf{E}} \left[R(s,a,s') + \gamma \underset{a'\sim\pi}{\mathbf{E}} \left[Q^{\pi}(s',a')\right]\right],
\end{aligned}
$$

where $s' \sim P$ is shorthand for $s' \sim P(\cdot|s,a)$, indicating that the next state s' is sampled from the environment's transition rules; $a \sim \pi$ is shorthand for $a \sim \pi(\cdot|s)$; and $a' \sim \pi$ is shorthand for $a' \sim \pi(\cdot|s')$.

Proof of Bellman equations

Proof. $V^\pi = R^\pi + \gamma P^\pi V^\pi$:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \ldots | s_1 = s\right] \\ &= \mathbb{E}\left[r_1 | s_1 = s\right] + \gamma \mathbb{E}\left[\mathbb{E}\left[r_2 + \gamma r_3 + \gamma^2 r_4 + \ldots | s_2\right] | s_1 = s\right] \end{aligned}$$

The first term here is simply the expected reward in state s when action is given by $\pi(s)$. The second term is γ times the value function at $s_2 \sim P(s, \pi(s), \cdot)$

$$\begin{aligned} V^\pi(s) &= \mathbb{E}\left[R\left(s, \pi(s), s_1\right) + \gamma V^\pi\left(s_2\right) | s_1 = s\right] \\ &= R(s, \pi(s)) + \gamma \sum_{s_2 \in S} P\left(s, \pi(s), s_2\right) V^\pi\left(s_2\right) \\ &= R^\pi(s) + \gamma\left[P^\pi V^\pi\right](s) \end{aligned}$$

Bellman Optimal Equations

- The Bellman equations for the optimal value functions are

$$V^*(s) = \max_a \mathop{\mathbf{E}}_{s' \sim P} \left[R(s,a) + \gamma V^*(s') \right],$$

$$Q^*(s,a) = \mathop{\mathbf{E}}_{s' \sim P} \left[R(s,a) + \gamma \max_{a'} Q^*(s',a') \right].$$

  The crucial difference between the Bellman equations for the on-policy value functions and the optimal value functions, is the absence or presence of the $\max$ over actions. Its inclusion reflects the fact that whenever the agent gets to choose its action, in order to act optimally, it has to pick whichever action leads to the highest value.

- The term “Bellman backup” comes up quite frequently in the RL literature. The Bellman backup for a state, or state-action pair, is the right-hand side of the Bellman equation: the reward-plus-next-value.

Proof of Bellman optimality equations

Proof. for all s, from the theorem ensuring stationary optimal policy:

$$\begin{aligned} V^*(s) = \max_\pi V^\pi(s) &= \max_\pi \mathbb{E}_{a\sim\pi(s),s'\sim P(s,a)} \left[R\left(s,a,s'\right) + \gamma V^\pi\left(s'\right)\right] \\ &\leq \max_a R(s,a) + \gamma \sum_{s'} P\left(s,a,s'\right) \max_\pi V^\pi\left(s'\right) \\ &= \max_a R(s,a) + \gamma \sum_{s'} P\left(s,a,s'\right) V^*\left(s'\right) \end{aligned}$$

Now, if the above inequality is strict then the value of state s can be improved by using a (possibly non-stationary) policy that uses action $\arg\max_a R(s,a)$ in the first step. This is a contradiction to the definition $V^*(s)$. Therefore,

$$V^*(s) = \max_a R(s,a) + \gamma \sum_{s'} P\left(s,a,s'\right) V^*\left(s'\right)$$

Bellman optimality equations

- Technically, above only shows that V^* satisfies the Bellman equations.
- Theorem 6.2.2 (c) in Puterman [1994] shows that V^* is in fact unique solution of above equations.
- Therefore, satisfying these equations is sufficient to guarantee optimality, so that it is not difficult to see that the deterministic (stationary) policy

$$\pi^*(s) = \arg\max_a R(s,a) + \gamma \sum_{s'} P\left(s,a,s'\right) V^*\left(s'\right)$$

is optimal (see Puterman [1994] Theorem 6.2.7 for formal proof).

Linear programming

Linear programming

The fixed point for above Bellman optimality equations can be found by formulating a linear program. It amounts to :

$$\begin{aligned} \min_{\mathbf{v}\in\mathbb{R}^S} \quad & \sum_s w_s \mathbf{v}_s \\ \text{s.t.} \quad & \mathbf{v}_s \geq R(s,a) + \gamma P(s,a)^\top \mathbf{v} \quad \forall a, s \end{aligned}$$

Proof. V^* clearly satisfies the constraints of the above LP. Next, we show that $\mathbf{v} = V^*$ minimizes the obj. fun. The constraint implies that

$$\mathbf{v}_s \geq R\left(s, \pi^*(s)\right) + \gamma P\left(s, \pi^*(s)\right)^\top \mathbf{v}, \forall s$$

(Above is written assuming π^* is deterministic, which is in fact true in the infinite horizon discounted reward case.) Or,

$$\left(I - \gamma P^{\pi^*}\right)\mathbf{v} \geq R^{\pi^*}$$

Proof

Because $\gamma < 1$, $(I - \gamma P^{\pi})^{-1}$ exists for all π, and for any $u \geq 0$

$$(I - \gamma P^{\pi})^{-1} u = \left(I + \gamma P^{\pi} + \gamma^2 (P^{\pi})^2 + \cdots\right) u \geq u$$

Therefore, from above

$$\left(I - \gamma P^{\pi^*}\right)^{-1} \left(\left(I - \gamma P^{\pi^*}\right) \mathbf{v} - R^{\pi^*}\right) \geq 0$$

Or,

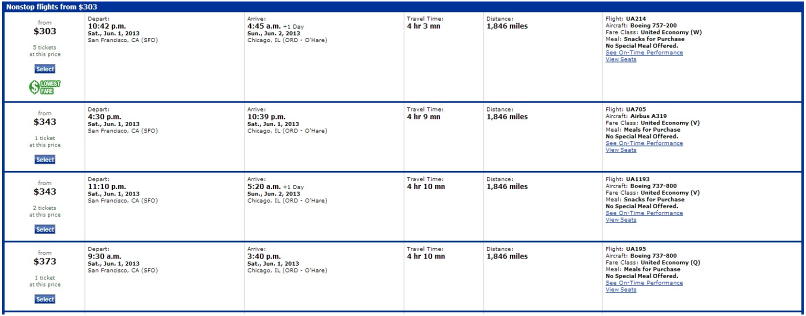
$$\mathbf{v} \geq \left(I - \gamma P^{\pi^*}\right)^{-1} R^{\pi^*} = V^*$$

Therefore, $\mathbf{w}^\top \mathbf{v}$ for $w > 0$ is minimized by $\mathbf{v} = V^*$.

Outline

1. Introduction
2. Dynamic Programming
3. Markov Decision Process (MDP)
4. Bellman Equation
5. **Example: Airfare Pricing**
6. Iterative algorithms (discounted reward case)
    - Value Iteration
    - Q-value iteration
    - Policy iteration

An Example in Revenue Management: Airfare Pricing



Nonstop flights from $303

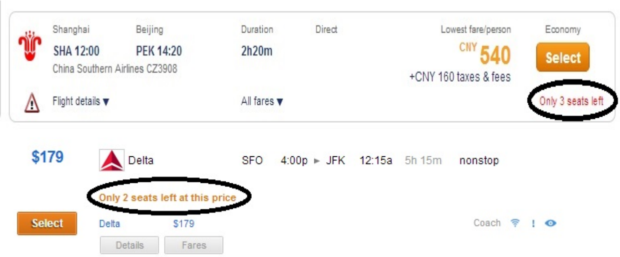
from **$303** 5 tickets at this price	Depart: **10:42 p.m.** **Sat., Jun. 1, 2013** San Francisco, CA (SFO)	Arrive: **4:45 a.m.** +1 Day **Sun., Jun. 2, 2013** Chicago, IL (ORD - O'Hare)	Travel Time: **4 hr 3 mn**	Distance: **1,846 miles**	Flight: **UA214** Aircraft: **Boeing 757-200** Fare Class: **United Economy (W)** Meal: **Snacks for Purchase** **No Special Meal Offered.** See On-Time Performance View Seats
from **$343** 1 ticket at this price	Depart: **4:30 p.m.** **Sat., Jun. 1, 2013** San Francisco, CA (SFO)	Arrive: **10:39 p.m.** **Sat., Jun. 1, 2013** Chicago, IL (ORD - O'Hare)	Travel Time: **4 hr 9 mn**	Distance: **1,846 miles**	Flight: **UA705** Aircraft: **Airbus A319** Fare Class: **United Economy (V)** Meal: **Meals for Purchase** **No Special Meal Offered.** See On-Time Performance View Seats
from **$343** 2 tickets at this price	Depart: **11:10 p.m.** **Sat., Jun. 1, 2013** San Francisco, CA (SFO)	Arrive: **5:20 a.m.** +1 Day **Sun., Jun. 2, 2013** Chicago, IL (ORD - O'Hare)	Travel Time: **4 hr 10 mn**	Distance: **1,846 miles**	Flight: **UA1193** Aircraft: **Boeing 737-800** Fare Class: **United Economy (V)** Meal: **Snacks for Purchase** **No Special Meal Offered.** See On-Time Performance View Seats
from **$373** 1 ticket at this price	Depart: **9:30 a.m.** **Sat., Jun. 1, 2013** San Francisco, CA (SFO)	Arrive: **3:40 p.m.** **Sat., Jun. 1, 2013** Chicago, IL (ORD - O'Hare)	Travel Time: **4 hr 10 mn**	Distance: **1,846 miles**	Flight: **UA195** Aircraft: **Boeing 737-800** Fare Class: **United Economy (Q)** Meal: **Meals for Purchase** **No Special Meal Offered.** See On-Time Performance View Seats

The price corresponding to each fare class rarely changes (this is determined by other department), however, the RM department determines when to close low fare classes

- From the passenger's point of view, when the RM system closes a class, the fare increases
- Closing fare class achieves dynamic pricing

Fare classes

And when you make booking, you will frequently see messages like



Only 3 seats left

Only 2 seats left at this price

This is real. It means there are only that number of tickets at that fare class (there is one more sale that will trigger the next protection level)

- You can try to buy one ticket with only one remaining, and see what happens

Dynamic Arrival of Consumers

Assumptions

- There are T periods in total indexed forward (the first period is 1 and the last period is T)
- There are C inventory at the beginning
- Customers belong to n classes, with $p_1 > p_2 > ... > p_n$
- In each period, there is a probability λ_i that a class i customer arrives
- Each period is small enough so that there is at most one arrival in each period

Decisions

- When at period t and when you have x inventory remaining, which fare class should you accept (if such a customer comes)
- Instead of finding a single optimal price or reservation level, we now seek for a decision rule, i.e., a mapping from (t, x) to $\{I | I \subset \{1, ..., n\}\}$.

Dynamic Arrival - a T-stage DP problem

- State: Inventory level x_k for stages $k = 1, ..., T$
- Action: Let $u^{(k)} \in \{0, 1\}^n$ to be the decision variable at period k

  $$u_i^{(k)} = \begin{cases} 1 & \text{accept class i customer} \\ 0 & \text{reject class i customer} \end{cases}$$

  decision vector $u^{(k)}$ at stage k, where $u_i^{(k)}$ decides whether to accept the ith class
- Random disturbance: Let $w_k, k \in \{0, ..., T\}$ denotes the type of new arrival during the kth stage (type 0 means no arrival). Then $P(w_k = i) = \lambda_i$ for $k = 1, \ldots, T$ and $P(w_k = 0) = 1 - \sum_{i=1}^{n} \lambda_i$

Value Function: A Rigorous Definition

- State transition cost:

$$g_k(x_k, u^{(k)}, w_k) = u^{(k)}_{w_k} p_{w_k}$$

  where we take $p_0 = 0$. Clearly, $\mathbf{E}[g_k(x_k, u^{(k)}, w_k) | x_k] = \sum_{i=1}^n u_i^{(k)} p_i \lambda_i$

- State transition dynamics

$$x_{k+1} = \begin{cases} x_k - 1 & \text{if } u^{(k)}_{w_k} w_k \neq 0 \text{ (with probability } \sum_{i=1}^n u_i^{(k)} \lambda_i) \\ x_k & \text{otherwise (with probability } 1 - \sum_{i=1}^n u_i^{(k)} \lambda_i) \end{cases}$$

- The overall revenue is

$$\max_{\mu_1, \ldots, \mu_T} \mathbf{E}\left[\sum_{k=0}^{T} g_k(x_k, \mu_k(x_k), w_k)\right]$$

  subject to the $\mu_k : x \to \{u\}$ for all k

A Dynamic Programming Model

- Let $V_t(x)$ denote the optimal revenue one can earn (by using the optimal policy onward) starting at time period t with inventory x

$$V_t(x) = \max_{\mu_t,\ldots,\mu_T} \mathbf{E}\left[\sum_{k=t}^{T} g_k(x_k, \mu_k(x_k), w_k) | x_t = x\right]$$

- We call $V_t(x)$ the value function (a function of stage t and state x)
- Suppose that we know the optimal pricing strategy from time $t+1$ for all possible inventory levels x.
- More specifically, suppose that we know $V_{t+1}(x)$ for all possible state x. Now let us find the best decisions at time t.

Bellman's Equation for Dynamic Arrival Model

We just proved the Bellman's equation. In the airfare model, Bellman's equation is

$$V_t(x) = \max_u \left\{ \sum_{i=1}^n \lambda_i (p_i u_i + u_i V_{t+1}(x-1)) + (1 - \sum_{i=1}^n \lambda_i u_i) V_{t+1}(x) \right\}$$

with $V_{T+1}(x) = 0$ for all x and $V_t(0) = 0$ for all t

We can rewrite this as

$$V_t(x) = V_{t+1}(x) + \max_u \left\{ \sum_{i=1}^n \lambda_i u_i (p_i + V_{t+1}(x-1) - V_{t+1}(x)) \right\}$$

For every (t, x), we have an equality and an unknown. The Bellman equation bears a unique solution.

Dynamic Programming Analysis

$$V_t(x) = V_{t+1}(x) + \max_u \left\{ \sum_{i=1}^{n} \lambda_i u_i (p_i - \Delta V_{t+1}(x)) \right\}$$

Therefore the optimal decision at time t with inventory x should be

$$u_i^* = \begin{cases} 1 & p_i \geq \Delta V_{t+1}(x) \\ 0 & p_i < \Delta V_{t+1}(x) \end{cases}$$

This is also called bid-price control policy

- The bid-price is $\Delta V_{t+1}(x)$
- If the customer pays more than the bid-price, then accept
- Otherwise reject

Dynamic Programming Analysis

Of course, to implement this strategy, we need to know $\Delta V_{t+1}(x)$

- We can compute all the values of $V_{t+1}(x)$ backwards
- Computational complexity is $O(nCT)$
- With those, we can have a whole table of $V_{t+1}(x)$. And we can execute based on that

Proposition (Properties of the Bid-prices)

For any x and t, i) $\Delta V_t(x+1) \leq \Delta V_t(x)$, ii) $\Delta V_{t+1}(x) \leq \Delta V_t(x)$

Intuitions:

- Fixed t, the value of the inventory has decreasing marginal returns
- The more time one has, the more valuable an inventory worth
- Proof by induction using the DP formula

Outline

1. Introduction
2. Dynamic Programming
3. Markov Decision Process (MDP)
4. Bellman Equation
5. Example: Airfare Pricing
6. Iterative algorithms (discounted reward case)
    - Value Iteration
    - Q-value iteration
    - Policy iteration

Value Iteration

- Indirect method that finds optimal value function (value vector v in above), not explicit policy.

Pseudocode

- Start with an arbitrary initialization v^0. Specify $\epsilon > 0$
- **Repeat** for $k = 1, 2, \ldots$ **until** $\left\|\mathbf{v}^k(s) - \mathbf{v}^{k-1}(s)\right\|_\infty \leq \epsilon \frac{(1-\gamma)}{2\gamma}$:
    - for every $s \in S$, improve the value vector as:

$$\mathbf{v}^k(s) = \max_{a \in A} R(s,a) + \gamma \sum_{s'} P\left(s,a,s'\right) \mathbf{v}^{k-1}\left(s'\right) \tag{1}$$

- Compute optimal policy as

$$\pi(s) \in \arg\max_a R(s,a) + \gamma P(s,a)^\top \mathbf{v}^k \tag{2}$$

Bellman operator

- It is useful to represent the iterative step (1) using operator $\mathbf{L} : \mathbb{R}^S \to \mathbb{R}^S$.

$$\mathbf{L}V(s) := \max_{a \in A} R(s,a) + \gamma \sum_{s'} P\left(s,a,s'\right) V\left(s'\right)$$

$$\mathbf{L}^\pi V(s) := \mathbb{E}_{a \in \pi(s)} \left[R(s,a) + \gamma \sum_{s'} P\left(s,a,s'\right) V\left(s'\right) \right] \tag{3}$$

- Then, (1) is same as

$$\mathbf{v}^k = \mathbf{L}\mathbf{v}^{k-1} \tag{4}$$

- For any policy π, if V^π denotes its value function, then, by Bellman equations:

$$V^* = \mathbf{L}V^*, V^\pi = \mathbf{L}^\pi V^\pi \tag{5}$$

Bellman operator

Below is a useful ‘contraction’ property of the Bellman operator, which underlies the convergence properties of all DP based iterative algorithms.

Lemma 6

The operator $\mathbf{L}(\cdot)$ and $\mathbf{L}^{\pi}(\cdot)$ defined by (3) are contraction mappings, i.e.,

$$\|\mathbf{L}v - \mathbf{L}u\|_{\infty} \leq \gamma \|v - u\|_{\infty}$$

$$\|\mathbf{L}^{\pi}v - \mathbf{L}^{\pi}u\|_{\infty} \leq \gamma \|v - u\|_{\infty}$$

Proof of contraction

Proof. First assume $\mathbf{L}v(s) \geq \mathbf{L}u(s)$.
Let $a_s^* = \arg\max_{a \in A} R(s,a) + \gamma \sum_{s'} P(s,a,s')\, v(s')$

$$\begin{aligned} 0 &\leq \mathbf{L}v(s) - \mathbf{L}u(s) \\ &\leq R(s,a_s^*) + \gamma \sum_{s'} P(s,a_s^*,s')\, v(s') - R(s,a_s^*) - \gamma \sum_{s'} P(s,a_s^*,s')\, u(s') \\ &= \gamma P(s,a_s^*)^\top (v-u) \\ &\leq \gamma \|v-u\|_\infty \end{aligned}$$

Repeating a symmetric argument for the case $\mathbf{L}v(s) \geq \mathbf{L}u(s)$ gives the lemma statement. Similar proof holds for $\mathbf{L}^\pi$.

Convergence

Theorem 7 (Theorem 6.3.3, Section 6.3.2 in Puterman [1994])

The convergence rate of the above algorithm is linear at rate γ. Specifically,

$$\left\|\mathbf{v}^k - V^*\right\|_\infty \leq \frac{\gamma^k}{1-\gamma}\left\|\mathbf{v}^1 - \mathbf{v}^0\right\|_\infty$$

Further, let π^k be the policy given by (2) using v^k. Then,

$$\left\|V^{\pi^k} - V^*\right\|_\infty \leq \frac{2\gamma^k}{1-\gamma}\left\|v^1 - v^0\right\|_\infty$$

Proof of Convergence

Proof. By Bellman equations $V^* = \mathbf{L}V^*$

$$\begin{aligned}
\left\|V^* - \mathbf{v}^k\right\|_\infty &= \left\|\mathbf{L}V^* - v^k\right\|_\infty \\
&\leq \left\|\mathbf{L}V^* - \mathbf{L}v^k\right\|_\infty + \left\|\mathbf{L}v^k - v^k\right\|_\infty \\
&= \left\|\mathbf{L}V^* - \mathbf{L}v^k\right\|_\infty + \left\|\mathbf{L}v^k - \mathbf{L}v^{k-1}\right\|_\infty \\
&\leq \gamma \left\|V^* - v^k\right\| + \gamma \left\|v^k - v^{k-1}\right\| \\
&\leq \gamma \left\|V^* - v^k\right\| + \gamma^k \left\|v^1 - v^0\right\| \\
\left\|V^* - v^k\right\|_\infty &\leq \frac{\gamma^k}{1-\gamma} \left\|v^1 - v^0\right\|
\end{aligned}$$

Let $\pi = \pi^k$ be the policy at the end of k iterations. Then, $V^\pi = \mathbf{L}^\pi V^\pi$ by Bellman equations. Further, by definition of $\pi = \pi^k$,

$$\mathbf{L}^\pi v^k(s) = \max_a R(s,a) + \gamma \sum_{s'} P\left(s,a,s'\right) v^k\left(s'\right) = \mathbf{L}v^k(s)$$

Proof of Convergence

Therefore,

$$\begin{aligned}
\left\|V^{\pi}-v^{k}\right\|_{\infty} &= \left\|\mathbf{L}^{\pi}V^{\pi}-v^{k}\right\|_{\infty} \\
&\leq \left\|\mathbf{L}^{\pi}V^{\pi}-\mathbf{L}^{\pi}v^{k}\right\|_{\infty}+\left\|\mathbf{L}^{\pi}v^{k}-v^{k}\right\|_{\infty} \\
&= \left\|\mathbf{L}^{\pi}V^{\pi}-\mathbf{L}^{\pi}v^{k}\right\|_{\infty}+\left\|\mathbf{L}v^{k}-\mathbf{L}v^{k-1}\right\|_{\infty} \\
&\leq \gamma\left\|V^{\pi}-v^{k}\right\|+\gamma\left\|v^{k}-v^{k-1}\right\| \\
\left\|V^{\pi}-v^{k}\right\|_{\infty} &\leq \frac{\gamma}{1-\gamma}\left\|v^{k}-v^{k-1}\right\| \\
&\leq \frac{\gamma^{k}}{1-\gamma}\left\|v^{1}-v^{0}\right\|
\end{aligned}$$

Adding the two results above:

$$\left\|V^{\pi}-V^{*}\right\|_{\infty} \leq \frac{2\gamma^{k}}{1-\gamma}\left\|v^{1}-v^{0}\right\|_{\infty}$$

Convergence

- In average reward case, the algorithm is similar, but the Bellman operator used to update the values is now $\mathbf{L}V(s) = \max_a r_{s,a} + P(s,a)^\top V$. Also, here $\mathbf{v}^k$ will converge to $\mathbf{v}^* + c\mathbf{e}$ for some constant c. Therefore, the stopping condition used is instead $sp\left(v^k - v^{k-1}\right) \leq \epsilon$ where $\mathrm{sp}(v) := \max_s v_s - \min_s v_s$. That is, span is used instead of L_∞ norm. Further since there is no discount $(\gamma = 1)$, a condition on the transition matrix is required to prove convergence. Let

$$\gamma := \max_{s,s',a,a'} 1 - \sum_{j \in S} \min \left\{ P(s,a,j), P\left(s',a',j\right) \right\}$$

- Then, linear convergence with rate γ is guaranteed if $\gamma < 1$. This condition ensures that the Bellman operator in this case: is still a contraction. For more details, refer to Section 8.5.2 in Puterman [1994].

Q-value iteration

- $Q^*(s,a)$: expected utility on taking action a in state s, and thereafter acting optimally. Then,$V^*(s) = \max_a Q^*(s,a)$. Therefore, Bellman equations can be written as,

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P\left(s,a,s'\right) \left(\max_{a'} Q^*\left(s',a'\right) \right)$$

Based on above a Q-value-iteration algorithm can be derived:

Pseudocode

- Start with an arbitrary initialization $\mathbf{Q}^0 \in \mathbb{R}^{S \times A}$.
- In every iteration k, improve the Q-value vector as:

$$\mathbf{Q}^k(s,a) = R(s,a) + \gamma \mathbb{E}_{s'} \left[\max_{a'} Q^{k-1}\left(s',a'\right) | s,a \right], \forall s,a$$

- Stop if $\left\| Q^k - Q^{k-1} \right\|_\infty$ is small.

Policy iteration

Start with an arbitrary initialization of policy π^1. The k-th policy iteration has two steps:

- **Policy evaluation**: Find $\mathbf{v}^k$ by solving $\mathbf{v}^k = \mathbf{L}^{\pi^k}\mathbf{v}^k$, i.e.,

$$\mathbf{v}^k(s) = \mathbb{E}_{a\sim\pi(s)}\left[R\left(s,a,s'\right) + \gamma\sum_{s'} P\left(s,a,s'\right)\mathbf{v}^k\left(s'\right)\right], \forall s$$

- **Policy improvement**: Find π^{k+1} such that $\mathbf{L}^{\pi^{k+1}}\mathbf{v}^k = \mathbf{L}\mathbf{v}^k$, i.e.,

$$\pi^{k+1}(s) = \arg\max_a R(s,a) + \gamma\mathbb{E}_{s'}\left[\mathbf{v}^k\left(s'\right)|s,a\right], \forall s$$

TD-learning and Q-learning

http://bicmr.pku.edu.cn/~wenzw/bigdata2020.html

Acknowledgement: this slides is based on Prof. Shipra Agrawal's lecture notes

Outline

1. TD-learning
    - TD(0)-learning
    - TD(λ)
2. Q-learning (tabular)
    - The Q-learning method
    - Stochastic Approximation method
3. Q-learning with function approximation
4. Deep Q-learning Networks (DQN)

TD-learning

- TD-learning is essentially approximate version of policy evaluation using samples. Adding policy improvement gives an approximate version of policy iteration.

- Since $V^{\pi}(s)$ is defined as the expectation of the random return when the process is started from the given state s, an obvious way of estimating this value is to compute an average over multiple independent realizations started from the given state. This is an instance of the so-called Monte-Carlo method.

- Unfortunately, the variance of the observed returns can be high. The Monte-Carlo technique is further difficult to apply if the system is not accessible through a simulator but rather estimation happens while actually interacting with the system.

TD(0)-learning

- Policy evaluation is about estimating $V^{\pi}(\cdot)$, which by Bellman equations is equivalent to finding a stationary point of

$$V^{\pi}(s) = \mathbb{E}_{a \sim \pi(s)} \left[R(s,a) + \gamma \sum_{s'} P(s,a,s') \, V^{\pi}(s') \right], \forall s$$

- However, we need to estimate this using only observations r_t, s_{t+1} on playing some action at a_t current state s_t.
- Let the current estimate of $V(s)$ is $\hat{V}(s)$. Let on taking action $a_t = \pi(s_t)$ in the current state s_t, s_{t+1} is the observed (sample) next state. The predicted value function for the next state s_{t+1} is $\hat{V}(s_{t+1})$, giving another prediction of value function at state s_t $r_t + \gamma \hat{V}(s_{t+1})$. Note that

$$\mathbb{E}\left[r_t + \hat{V}(s_{t+1}) \,|\, s_t, \hat{V} \right] = \mathbb{E}_{a \sim \pi(s_t)} \left[R(s_t,a) + \sum_{s'} P(s_t,a,s') \, \hat{V}(s') \,|\, s_t, \hat{V} \right]$$

TD(0)-learning

- From Bellman equations, we are looking for $\hat{V}$ such that

$$\hat{V}(s_t) \approx r_t + \hat{V}(s_{t+1})$$

- The TD method performs the following update to the value function estimate at s_t, moving it towards the new estimate:

$$\hat{V}(s_t) \leftarrow (1 - \alpha_t)\hat{V}(s_t) + \alpha_t \left(r_t + \gamma \hat{V}(s_{t+1})\right)$$

- Let δ_t be the following gap:

$$\delta_t := r_t + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t)$$

  referred to as temporal difference, i.e., the difference between current estimate, and one-lookahead estimate. Then, the above also be written as:

$$\hat{V}(s_t) \leftarrow \hat{V}(s_t) + \alpha_t \delta_t \tag{1}$$

SGD interpretation: a general principal

- Let z_s be a random variable independent to θ.
- Consider:

$$\min_{\theta} \quad (x_\theta - \mathbf{E}_s[z_s])^2$$

- It is equivalent to

$$\begin{aligned} & \min_{\theta} \quad (x_\theta - \mathbf{E}_s[z_s])^2 - (\mathbf{E}_s[z_s])^2 + \mathbf{E}_s[z_s^2] \\ \Longleftrightarrow \quad & \min_{\theta} \quad x_\theta^2 + (\mathbf{E}_s[z_s])^2 - 2x_\theta \mathbf{E}_s[z_s] - (\mathbf{E}_s[z_s])^2 + \mathbf{E}_s[z_s^2] \\ \Longleftrightarrow \quad & \min_{\theta} \quad x_\theta^2 - 2x_\theta \mathbf{E}_s[z_s] + \mathbf{E}_s[z_s^2] \\ \Longleftrightarrow \quad & \min_{\theta} \quad \mathbf{E}_s[x_\theta - z_s]^2 \end{aligned}$$

Expectation is taken out. It enables us to perform sample on z_s.

SGD interpretation

- Recall the Bellman equation:

$$V^\pi(s) = \mathop{\mathbf{E}}_{\substack{a\sim\pi \\ s'\sim P}} \left[R(s,a,s') + \gamma V^\pi(s')\right],$$

- we introduce a target V^π_{targ} and approximate:

$$V^\pi(s) \approx \mathop{\mathbf{E}}_{\substack{a\sim\pi \\ s'\sim P}} \left[R(s,a,s') + \gamma V^\pi_{targ}(s')\right]$$

- construct a least-squares problem:

$$\begin{aligned} &\min_{V^\pi(s)} \quad \left(V^\pi(s) - \mathop{\mathbf{E}}_{\substack{a\sim\pi \\ s'\sim P}} \left[R(s,a,s') + \gamma V^\pi_{targ}(s')\right]\right)^2 \\ \iff &\min_{V^\pi(s)} \quad \mathop{\mathbf{E}}_{\substack{a\sim\pi \\ s'\sim P}} \left[V^\pi(s) - \left[R(s,a,s') + \gamma V^\pi_{targ}(s')\right]\right]^2 \end{aligned}$$

SGD interpretation

- use SGD to solve:

$$\min_{V^\pi(s)} \quad \frac{1}{2} \mathop{\mathbf{E}}_{\substack{a\sim\pi \\ s'\sim P}} \left[V^\pi(s) - [R(s,a,s') + \gamma V^\pi_{targ}(s')] \right]^2$$

- The gradient with respect to $V^\pi(s)$ is

$$\mathop{\mathbf{E}}_{\substack{a\sim\pi \\ s'\sim P}} \left[V^\pi(s) - [R(s,a,s') + \gamma V^\pi_{targ}(s')] \right].$$

  Take one sample:

$$V^\pi(s_t) - [R(s_t,a,s_{t+1}) + \gamma V^\pi_{targ}(s_{t+1})] = -\delta_t$$

- Hence, one step of SGD is

$$V^\pi(s_t) \leftarrow V^\pi(s_t) + \alpha\delta_t$$

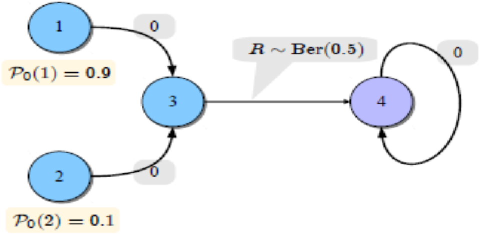
Tabular TD(0) method for policy evaluation

Algorithm 1 Tabular TD(0) method for policy evaluation

```
Initialization: Given a starting state distribution $D_0$, policy $\pi$, the method evaluates $V^\pi(s)$ for all states $s$.
Initialize $\hat{V}$ as an empty list/array for storing the value estimates.
repeat
    Set $t = 1$, $s_1 \sim D_0$. Choose step sizes $\alpha_1, \alpha_2, \ldots$.
    Perform TD($0$) updates over an episode:
    repeat
        Take action at $a_t \sim \pi(s_t)$. Observe reward $r_t$, and new state $s_{t+1}$.
        $\delta_t := r_t + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t)$
        Update $\hat{V}(s_t) \leftarrow \hat{V}(s_t) + \alpha_t \delta_t$
        $t = t + 1$
    until episode terminates
until change in $\hat{V}$ over consecutive episodes is small
```

Monte Carlo method

Why use only 1-step lookahead to construct target z? Why not lookahead entire trajectory (in problems where there is a terminal state, also referred to as episodic MDPs)?



[Szepesvari, 1999] In this example, all transitions are deterministic. The reward is zero, except when transitioning from state 3 to state 4, when it is given by a Bernoulli random variable with parameter 0.5. State 4 is a terminal state. When the process reaches the terminal state, it is reset to start at state 1 or 2. The probability of starting at state 1 is 0.9, while the probability of starting at state 2 is 0.1.

Monte Carlo method

- The resulting method is referred to as Monte Carlo method, here for z a sample trajectory starting at s_t is used

  $$z = \sum_{n=0}^{\infty} \gamma^n r_{t+n} =: \mathcal{R}_t$$

  so that

  $$\begin{aligned}\delta_t &= z - \hat{V}(s_t) = \mathcal{R}_t - \hat{V}(s_t)\\ \hat{V}(s_t) &= (1-\alpha)\hat{V}(s_t) + \alpha\mathcal{R}_t\end{aligned}$$

TD(0) or Monte-Carlo?

$\textit{This example is taken from page } 22-23, \textit{ Szepesvari } [1999]$

- First, let us consider an example when $TD(0)$ converges faster. Consider the above undiscounted episodic MRP shown on the above figure.
- The initial states are either 1 or 2. With high probability the process starts at state 1, while the process starts at state 2 less frequently.
- Consider now how $TD(0)$ will behave at state 2. By the time state 2 is visited the kth time, on the average state 3 has already been visited 10 k times.
- Assume that $\alpha_t = 1/(t+1)$ (the TD updates with this step size reduce to averaging of target observations). At state 1 and 2, the target is $\hat{V}(3)$ (since immediate reward is 0 and transition probability to state 3 is 1).

TD(0) or Monte-Carlo?

- Therefore, whenever state 2 is visited the $TD(0)$ sets its value as the average of estimates $\hat{V}^t(3)$ over the time steps t when state 1 was visited (similarly for state 2). At state 3 the $TD(0)$ update reduces to averaging the Bernoulli rewards incurred upon leaving state 3. At the kth visit of state 2, $\mathrm{Var}(\hat{V}(3)) \simeq 1/(10k)$ Clearly, $\mathbb{E}[\hat{V}(3)] = 0.5$. Thus, the target of the update of state 2 will be an estimate of the true value of state 2 with accuracy increasing with k.
- Now, consider the Monte-Carlo method. The Monte-Carlo method ignores the estimate of the value of state 3 and uses the Bernoulli rewards directly. In particular, $\mathrm{Var}\left(\mathcal{R}_t | s_t = 2\right) = 0.25$, i.e., the variance of the target does not change with time.
- On this example, this makes the Monte-Carlo method slower to converge, showing that sometimes bootstrapping might indeed help.

TD(0) or Monte-Carlo?

- To see an example when bootstrapping is not helpful, imagine that the problem is modified so that the reward associated with the transition from state 3 to state 4 is made deterministically equal to one.
- In this case, the Monte-Carlo method becomes faster since $\mathcal{R}_t = 1$ is the true target value, while for the value of state 2 to get close to its true value, $TD(0)$ has to wait until the estimate of the value at state 3 becomes close to its true value. This slows down the convergence of $TD(0)$.
- In fact, one can imagine a longer chain of states, where state $i+1$ follows state i, for $i \in 1, \ldots, N$ and the only time a nonzero reward is incurred is when transitioning from state $N-1$ to state N.
- In this example, the rate of convergence of the Monte-Carlo method is not impacted by the value of N, while $TD(0)$ would get slower with N increasing.

TD(λ)

- TD(λ) is a "middle-ground" between $TD(0)$ and Monte-Carlo evaluation.
- Here, the algorithm considers ℓ-step predictions:

$$z_t^\ell = \sum_{n=0}^{\ell} \gamma^n r_{t+n} + \gamma^{\ell+1} \hat{V}\left(s_{t+\ell+1}\right)$$

with temporal difference:

$$\begin{aligned}
\delta_t^\ell &= z - \hat{V}\left(s_t\right) \\
&= \sum_{n=0}^{\ell} \gamma^n r_{t+n} + \gamma^{\ell+1} \hat{V}\left(s_{t+\ell+1}\right) - \hat{V}\left(s_t\right) \\
&= \sum_{n=0}^{\ell} \gamma^n \left(r_{t+n} + \gamma \hat{V}\left(s_{t+n+1}\right) - \hat{V}\left(s_{t+n}\right)\right) \\
&= \sum_{n=0}^{\ell} \gamma^n \delta_{t+n}
\end{aligned}$$

TD(λ)

- In TD(λ) method, a mixture of ℓ-step predictions is used, with weight $(1-\lambda)\lambda^\ell$ for $\ell \geq 0$. Therefore, $\lambda = 0$ gives $TD(0)$, and $\lambda \to 1$ gives Monte-Carlo method. $\lambda > 1$ gives a multi-step method. To summarize, the $TD(\lambda)$ update is given as:

$$\hat{V}(s_t) \leftarrow \hat{V}(s_t) + \alpha_t \sum_{\ell=0}^{\infty} (1-\lambda)\lambda^\ell \delta_t^\ell = \hat{V}(s_t) + \alpha_t \sum_{n=0}^{\infty} \lambda^n \gamma^n \delta_{t+n}$$

Policy improvement with TD-learning

TD-learning allows evaluating a policy. For using TD-learning for finding optimal policy, we need to be able to improve the policy. Recall policy iteration effectively requires evaluating Q-value of a policy, where $Q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s'} P^{\pi}\left(s,a,s'\right) V^{\pi}\left(s'\right)$. With simple modification, TD-learning can be used to estimate Q-value of a policy. There, the updates would be replaced by:
$\delta_t := r_t + \gamma \hat{Q}\left(s_{t+1}, \pi\left(s_{t+1}\right)\right) - \hat{Q}\left(s_t, a_t\right)$
Update $\hat{Q}\left(s_t, a_t\right) \leftarrow \hat{Q}\left(s_t, a_t\right) + \alpha_t \delta_t$
Then, the scheme for policy improvement is similar to policy iteration. Repeat the following until convergence to some policy:

- Use TD-learning to evaluate the policy π^k. The method outputs $\hat{Q}^{\pi^k}(s,a), \forall s, a$
- Compute new 'improved policy' π^{k+1} as $\pi^{k+1}(s) \leftarrow \arg\max_a \hat{Q}^{\pi^k}(s,a)$.

Outline

1. TD-learning
    - TD(0)-learning
    - TD(λ)
2. Q-learning (tabular)
    - The Q-learning method
    - Stochastic Approximation method
3. Q-learning with function approximation
4. Deep Q-learning Networks (DQN)

Q-learning (tabular)

- Q-learning is a sample based version of Q-value iteration. This method attempts to directly find optimal Q-values, instead of computing Q-values of a given policy.
- Recall Q-value iteration: for all s, a update,

  $$Q_{k+1}(s,a) \leftarrow R(s,a) + \gamma \sum_{s'} P\left(s,a,s'\right)\left(\max_{a'} Q_k\left(s',a'\right)\right)$$

  Q-learning approximates these updates using sample observations, similar to TD-learning.
- In steps $t = 1, 2, \ldots$ of an episode, the algorithm observes reward r_t and next state $s_{t+1} \sim P\left(\cdot, s_t, a_t\right)$ for some action a_t. It updates the Q-estimates for pair (s_t, a_t) as follows:

  $$Q_{k+1}\left(s_t,a_t\right) = (1-\alpha)Q_k\left(s_t,a_t\right) + \alpha \underbrace{\left(r_t + \gamma \max_{a'} Q_k\left(s_{t+1},a'\right)\right)}_{\text{target}}$$

SGD Interpretation

- The Bellman optimal equation is

$$Q^*(s,a) = \mathop{\mathbf{E}}_{s' \sim P}\left[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right]$$

- We introduce a target $Q_{targ}(s,a)$ and approximate:

$$Q(s,a) \approx \mathop{\mathbf{E}}_{s' \sim P}\left[R(s,a,s') + \gamma \max_{a'} Q_{targ}(s',a')\right]$$

- Construct a least square problem:

$$\begin{aligned} & \min \quad \left(Q(s,a) - \mathop{\mathbf{E}}_{s' \sim P}\left[R(s,a,s') + \gamma \max_{a'} Q_{targ}(s',a')\right]\right)^2 \\ \Longleftrightarrow \quad & \min \quad \mathop{\mathbf{E}}_{s' \sim P}\left[Q(s,a) - [R(s,a,s') + \gamma \max_{a'} Q_{targ}(s',a')]\right]^2 \end{aligned}$$

SGD Interpretation

- use SGD to solve the least square problem:

$$\min \quad \frac{1}{2} \mathop{\mathbf{E}}_{s' \sim P} \left[Q(s,a) - [R(s,a,s') + \gamma \max_{a'} Q_{targ}(s',a')] \right]^2$$

- One sample of the gradient is

$$Q(s,a) - [R(s,a,s_{t+1}) + \gamma \max_{a'} Q_{targ}(s_{t+1},a')] = -\delta_t.$$

- One SGD step is

$$Q(s,a) \leftarrow Q(s,a) + \alpha_t \delta_t.$$

Q-learning (tabular)

Algorithm 2 Tabular Q-learning method

```
Initialization: Given a starting state distribution $D_0$.
Initialize $\hat{Q}$ as an empty list/array for storing the $Q$-value estimates.
repeat
    Set $t = 1$, $s_1 \sim D_0$. Choose step sizes $\alpha_1, \alpha_2, \ldots$.
    Perform $Q$-learning updates over an episode:
    repeat
        Take action at $a_t$. Observe reward $r_t$, and new state $s_{t+1}$.
        $\delta_t := \left(r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a')\right) - \hat{Q}(s_t, a_t)$
        Update $\hat{Q}(s_t, a_t) \leftarrow \hat{Q}(s_t, a_t) + \alpha_t \delta_t$
        $t = t + 1$
    until episode terminates
until change in $\hat{Q}$ over consecutive episodes is small
```

How to select actions, the issue of exploration

- The convergence results (discussed below) for Q-learning will say that if all actions and states are infinitely sampled, learning rate is small, but does not decrease too quickly, then Q-learning converges. (Does not matter how you select actions, as long as they are infinitely sampled).
- One option is to select actions greedily according to the current estimate $\max_a Q_k(s, a)$. But, this will reinforce past errors, and may fail to sample and estimate Q-values for actions which have higher error levels. We may get stuck at a subset of (suboptimal) actions.
- Therefore, exploration is required. The ϵ-greedy approach (i.e., with ϵ probability pick an action uniformly at random instead of greedy choice) can ensure infinite sampling of every action, but can be very inefficient.

How to select actions, the issue of exploration

- The same issue occurs in TD-learning based policy improvement methods. The choice of action is specified as the greedy policy according to the previous episode estimates. Without exploration this may not ensure that all actions and states are infinitely sampled.
- One option is to replace policy improvement step by greedy choice. That is, the policy improvement step will now compute the new 'improved policy' π^{k+1} as the randomized policy:

$$\pi^{k+1}(s) = \begin{cases} a_k^* := \arg\max_a \hat{Q}^{\pi^k}(s,a), & \text{with probability } 1-\epsilon+\frac{\epsilon}{|A|} \\ a, & \text{with probability } \frac{\epsilon}{|A|}, a \neq a_k^* \end{cases}$$

- Then, in policy evaluation, this 'randomized policy' must be used.
  7 : $\delta_t := r_t + \gamma \mathbb{E}_{a\sim\pi^{k+1}(s_{t+1})}\left[\hat{Q}\left(s_{t+1},a\right)\right] - \hat{Q}\left(s_t,a_t\right)$
  8 : Update $\hat{Q}\left(s_t,a_t\right) \leftarrow \hat{Q}\left(s_t,a_t\right) + \alpha_t\delta_t$

Convergence theorem

Theorem 1 (Watkins and Dayan [1992])

Given bounded rewards $|r_t| \leq R$, learning rates $0 \leq \alpha_t < 1$, and

$$\sum_{i=1}^{\infty} \alpha_{n^i(s,a)} = \infty, \sum_{i=1}^{\infty} \left(\alpha_{n^i(s,a)}\right)^2 < \infty$$

then $\hat{Q}^t(s,a) \to Q(s,a)$ as $t \to \infty$ for all s, a with probability 1. Here, $n^i(s,a)$ is the index of the ith time the action a is tried in state s, and $\hat{Q}^t(s,a)$ is the estimate $\hat{Q}$ in round t.

- If all actions and states are infinitely sampled, learning rate is small, but does not decrease too quickly, then Q-learning converges. (Does not matter how you select actions, as long as they are infinitely sampled).
- The proof of this and many similar results in RL algorithms follow the analysis of a more general online learning/optimization method - the stochastic approximation method.

Stochastic Approximation method

- The stochastic approximation (SA) algorithm essentially solves a system of (nonlinear) equations of the form

$$h(\theta) = 0$$

  for unknown $h(\cdot)$, based on noisy measurements of $h(\theta)$.

- More specifically, consider a (continuous) function $\mathbb{R}^d \to \mathbb{R}^d$, with $d \geq 1$, which depends on a set of parameters $\theta \in \mathbb{R}^d$. Suppose that $h(\theta)$ is unknown. However, for any θ we can measure $Z = h(\theta) + \omega$, where ω is some 0-mean noise. The classical SA algorithm (Robbins and Monro [1951]) is of the form

$$\begin{aligned} \theta_{n+1} &= \theta_n + \alpha_n Z_n \\ &= \theta_n + \alpha_n \left(h\left(\theta_n\right) + \omega_n \right), \quad n \geq 0 \end{aligned}$$

- Since ω_n is 0-mean noise, the stationary points of the above algorithm coincide with the solutions of $h(\theta) = 0$.

Asynchronous version

- More relevant to the RL methods discussed here is the asynchronous version of the SA method. In the asynchronous version of SA method, we may observe only one coordinate (say i^{th}) of $Z_n = h\left(\theta_n\right) + \omega_n$ at a time step, and we use that to update ith component of our parameter estimate:

$$\theta_{n+1}[i] = \theta_n[i] + \alpha_n Z_n[i]$$

- The convergence for this method will be proven similarly to the synchronous version, under the assumption that every coordinate is sampled infinitely often.

Outline

1. TD-learning
   - TD(0)-learning
   - TD(λ)
2. Q-learning (tabular)
   - The Q-learning method
   - Stochastic Approximation method
3. Q-learning with function approximation
4. Deep Q-learning Networks (DQN)

Q-learning with function approximation

- The tabular Q-learning does not scale with increase in the size of state space. In most real applications, there are too many states to keep visit, and keep track of.
- For scalability, we want to generalize, i.e., use what we have learned about already visited (relatively small number of) states, and generalize it to new, similar states.
- A fundamental idea is to use 'function approximation', i.e., use a lower dimensional feature representation of the state- action pair s, a and learn a parametric approximation $Q_\theta(s, a)$.

Q-learning with function approximation

- For example, the function $Q_\theta(s,a)$ can simply be a linear function in θ and features $Q_\theta(s,a) = \theta_0 f_0(s,a) + \theta_1 f_1(s,a) + \ldots + \theta_n f_n(s,a)$, or a deep neural net. Given parameter θ, the Q-function can be computed for unseen s, a. Instead of learning the $|S| \times |A|$ dimensional Q-table, the Q-learning algorithm will learn the parameter θ. Here, on observing sample transition to s' from s on playing action a, instead of updating the estimate of $Q(s,a)$ in the Q-table, the algorithm updates the estimate of θ.
- Intuitively, we are trying to find a θ such that for every s, a the Bellman equation,

$$Q_\theta(s,a) = \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[R\left(s,a,s'\right) + \gamma \max_{a'} Q_\theta\left(s',a'\right) \right]$$

can be approximated well for all s, a.

SGD Interpretation

- Similarly, we obtain a least square problem:

$$\min_{\theta} \quad \frac{1}{2} \mathop{\mathbf{E}}_{s' \sim P} \left[Q_\theta(s,a) - [R(s,a,s') + \gamma \max_{a'} Q_{\theta_{targ}}(s',a')] \right]^2$$

$$\min_{\theta} \quad \ell_\theta(s,a) = \mathop{\mathbf{E}}_{s' \sim P} \left[\ell_\theta\left(s,a,s'\right) \right]$$

- One sample of the gradient is

$$\begin{aligned} & \nabla_\theta \ell_\theta(s,a,s') \\ = & \left(Q_\theta(s,a) - [R(s,a,s') + \gamma \max_{a'} Q_{targ}(s_{t+1},a')] \right) \nabla_\theta Q_\theta(s,a) \\ = & -\delta_t \nabla_\theta Q_\theta(s,a). \end{aligned}$$

- One SGD step is

$$Q(s,a) \leftarrow Q(s,a) + \alpha_t \delta_t \nabla_\theta Q_\theta(s,a).$$

Q-learning Algorithm overview

Start with initial state $s = s_0$. In iteration $k = 1, 2, \ldots$,

- Take an action a.
- Observe reward r, transition to state $s' \sim P(\cdot|s, a)$.
- $\theta_{k+1} \leftarrow \theta_k - \alpha_k \nabla_{\theta_k} \ell_{\theta_k}(s, a, s')$, where

$$\nabla_\theta \ell_{\theta_k}\left(s, a, s'\right) = -\delta_t \nabla_\theta Q_{\theta_k}(s, a)$$

$$\delta_t = r + \gamma \max_{a'} Q_{\theta_k}\left(s', a'\right) - Q_{\theta_k}(s, a)$$

- $s \leftarrow s'$,

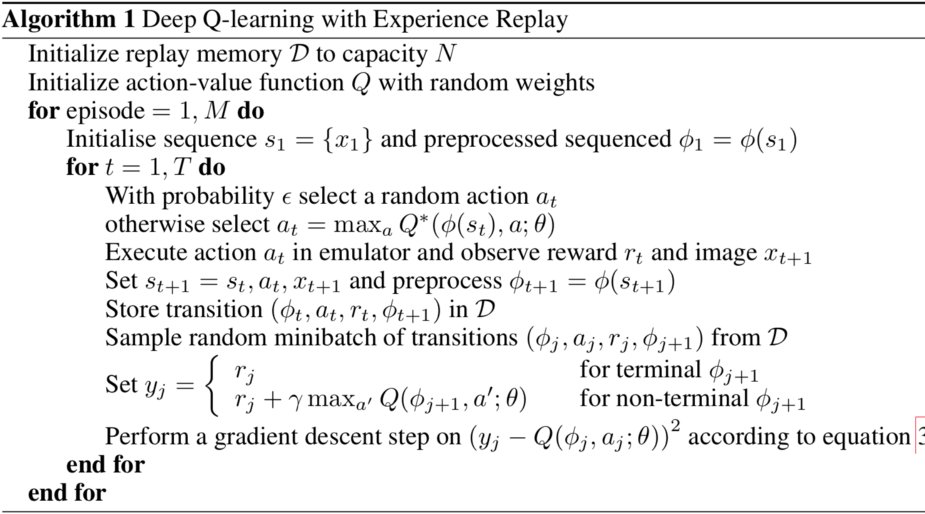
If s' reached at some point is a terminal state, s is reset to starting state.

Outline

1. TD-learning
    - TD(0)-learning
    - TD(λ)
2. Q-learning (tabular)
    - The Q-learning method
    - Stochastic Approximation method
3. Q-learning with function approximation
4. Deep Q-learning Networks (DQN)

Deep Q-Networks — Algorithm

DQN: θ is a deep neural network



Algorithm 1 Deep Q-learning with Experience Replay

```
Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize action-value function $Q$ with random weights
for episode $= 1, M$ do
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ do
        With probability $\epsilon$ select a random action $a_t$
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation
    end for
end for
```

Deep Deterministic Policy Gradient

- Deep Deterministic Policy Gradient (DDPG) is an algorithm which concurrently learns a Q-function and a policy. It uses off-policy data and the Bellman equation to learn the Q-function, and uses the Q-function to learn the policy.

- This approach is closely connected to Q-learning, and is motivated the same way: if you know the optimal action-value function $Q^*(s,a)$, then in any given state, the optimal action $a^*(s)$ can be found by solving

$$a^*(s) = \arg\max_a Q^*(s,a).$$

- DDPG interleaves learning an approximator to $Q^*(s,a)$ with learning an approximator to $a^*(s)$, and it does so in a way which is specifically adapted for environments with continuous action spaces. But what does it mean that DDPG is adapted *specifically* for environments with continuous action spaces? It relates to how we compute the max over actions in $\max_a Q^*(s,a)$.

- When there are a finite number of discrete actions, the max poses no problem, because we can just compute the Q-values for each action separately and directly compare them. (This also immediately gives us the action which maximizes the Q-value.) But when the action space is continuous, we can't exhaustively evaluate the space, and solving the optimization problem is highly non-trivial. Using a normal optimization algorithm would make calculating $\max_a Q^*(s, a)$ a painfully expensive subroutine. And since it would need to be run every time the agent wants to take an action in the environment, this is unacceptable.
- Because the action space is continuous, the function $Q^*(s, a)$ is presumed to be differentiable with respect to the action argument. This allows us to set up an efficient, gradient-based learning rule for a policy $\mu(s)$ which exploits that fact. Then, instead of running an expensive optimization subroutine each time we wish to compute $\max_a Q(s, a)$, we can approximate it with $\max_a Q(s, a) \approx Q(s, \mu(s))$.

The Q-Learning Side of DDPG

- First, let's recap the Bellman equation describing the optimal action-value function, $Q^*(s,a)$. It's given by

$$Q^*(s,a) = \underset{s'\sim P}{\mathrm{E}}\left[r(s,a) + \gamma \max_{a'} Q^*(s',a')\right]$$

  where $s' \sim P$ is shorthand for saying that the next state, s', is sampled by the environment from a distribution $P(\cdot|s,a)$.

- This Bellman equation is the starting point for learning an approximator to $Q^*(s,a)$. Suppose the approximator is a neural network $Q_\phi(s,a)$, with parameters ϕ, and that we have collected a set $\mathcal{D}$ of transitions (s,a,r,s',d) (where d indicates whether state s' is terminal). We can set up a **mean-squared Bellman error (MSBE)** function, which tells us roughly how closely Q_ϕ comes to satisfying the Bellman equation:

$$L(\phi,\mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}}\left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d)\max_{a'} Q_\phi(s',a')\right)\right)^2\right]$$

- Here, in evaluating $(1-d)$, we've used: “True" to 1 and “False” to zero. Thus, when “d==True”—which is to say, when s' is a terminal state—the Q-function should show that the agent gets no additional rewards after the current state.

- Q-learning algorithms for function approximators, such as DQN (and all its variants) and DDPG, are largely based on minimizing this MSBE loss function. There are two main tricks employed by all of them which are worth describing, and then a specific detail for DDPG.

- **Trick One: Replay Buffers**. All standard algorithms for training a deep neural network to approximate $Q^*(s, a)$ make use of an experience replay buffer. This is the set $\mathcal{D}$ of previous experiences. In order for the algorithm to have stable behavior, the replay buffer should be large enough to contain a wide range of experiences, but it may not always be good to keep everything. If you only use the very-most recent data, you will overfit to that and things will break; if you use too much experience, you may slow down your learning. This may take some tuning to get right.

- **Trick Two: Target Networks.** Q-learning algorithms make use of **target networks.** The term

$$r + \gamma(1 - d) \max_{a'} Q_{\phi}(s', a')$$

is called the **target**, because when we minimize the MSBE loss, we are trying to make the Q-function be more like this target. Problematically, the target depends on the same parameters we are trying to train: ϕ. This makes MSBE minimization unstable. The solution is to use a set of parameters which comes close to ϕ, but with a time delay—that is to say, a second network, called the target network, which lags the first. The parameters of the target network are denoted ϕ_{targ}.

- In DQN-based algorithms, the target network is just copied over from the main network every some-fixed-number of steps. In DDPG-style algorithms, the target network is updated once per main network update by polyak averaging:

$$\phi_{\text{targ}} \leftarrow \rho \phi_{\text{targ}} + (1-\rho)\phi$$

  where ρ is between 0 and 1 (usually close to 1).

- **DDPG Detail: Calculating the Max Over Actions in the Target.** As mentioned earlier: computing the maximum over actions in the target is a challenge in continuous action spaces. DDPG deals with this by using a **target policy network** to compute an action which approximately maximizes $Q_{\phi_{\text{targ}}}$. The target policy network is found the same way as the target Q-function: by polyak averaging the policy parameters over the course of training.

- Putting it all together, Q-learning in DDPG is performed by minimizing the following MSBE loss with stochastic gradient descent:

$$L(\phi, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}}\left[\left(Q_{\phi}(s,a) - \left(r + \gamma(1-d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))\right)\right)^2\right],$$

  where $\mu_{\theta_{\text{targ}}}$ is the target policy.

The Policy Learning Side of DDPG

- Policy learning in DDPG is fairly simple. We want to learn a deterministic policy $\mu_\theta(s)$ which gives the action that maximizes $Q_\phi(s,a)$. Because the action space is continuous, and we assume the Q-function is differentiable with respect to action, we can just perform gradient ascent (with respect to policy parameters only) to solve

$$\max_{\theta} \mathop{\mathrm{E}}_{s \sim \mathcal{D}} \left[Q_\phi(s, \mu_\theta(s)) \right].$$

- Note that the Q-function parameters are treated as constants here.

Pseudocode: DDPG

Algorithm 3 Deep Deterministic Policy Gradient

```
Input: initial policy parameters $\theta$, Q-function parameters $\phi$, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
    Observe state $s$ and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
    Execute $a$ in the environment
    Observe next state $s'$, reward $r$, and done signal $d$ to indicate whether $s'$ is terminal
    Store $(s, a, r, s', d)$ in replay buffer $\mathcal{D}$
    If $s'$ is terminal, reset environment state.
    if it's time to update then
        for however many updates do
            Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
            Compute targets $y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$
            Update Q-function by one step of gradient descent using $\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} (Q_\phi(s, a) - y(r, s', d))^2$
            Update policy by one step of gradient ascent using $\nabla_\theta \frac{1}{|B|} \sum_{s\in B} Q_\phi(s, \mu_\theta(s))$
            Update target networks with

                $\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$
                $\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$

        end for
    end if
until convergence
```

Twin Delayed DDPG (TD3)

- A common failure mode for DDPG is that the learned Q-function begins to dramatically overestimate Q-values, which then leads to the policy breaking, because it exploits the errors in the Q-function.
- **Trick One: Clipped Double-Q Learning.** TD3 learns *two* Q-functions instead of one (hence "twin"), and uses the smaller of the two Q-values to form the targets in the Bellman error loss functions.
- **Trick Two: "Delayed" Policy Updates.** TD3 updates the policy (and target networks) less frequently than the Q-function. The paper recommends one policy update for every two Q-function updates.
- **Trick Three: Target Policy Smoothing.** TD3 adds noise to the target action, to make it harder for the policy to exploit Q-function errors by smoothing out Q along changes in action.

Key Equations: target policy smoothing

TD3 concurrently learns two Q-functions, Q_{ϕ_1} and Q_{ϕ_2}, by mean square Bellman error minimization, in almost the same way that DDPG learns its single Q-function.

- **target policy smoothing.** Actions used to form the Q-learning target are based on the target policy, $\mu_{\theta_{\text{targ}}}$, but with clipped noise added on each dimension of the action. After adding the clipped noise, the target action is then clipped to lie in the valid action range (all valid actions, a, satisfy $a_{Low} \leq a \leq a_{High}$). The target actions are thus:

$$a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right), \qquad \epsilon \sim \mathcal{N}(0, \sigma)$$

- Target policy smoothing essentially serves as a regularizer for the algorithm. It addresses a particular failure mode that can happen in DDPG: if the Q-function approximator develops an incorrect sharp peak for some actions, the policy will quickly exploit that peak and then have brittle or incorrect behavior. This can be averted by smoothing out the Q-function over similar actions, which target policy smoothing is designed to do.

clipped double-Q learning

- Both Q-functions use a single target, calculated using whichever of the two Q-functions gives a smaller target value:

  $$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{i,\text{targ}}}(s', a'(s')),$$

  and then both are learned by regressing to this target:

  $$L(\phi_1, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathbf{E}} \left[\Bigg(Q_{\phi_1}(s, a) - y(r, s', d) \Bigg)^2 \right],$$
  $$L(\phi_2, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathbf{E}} \left[\Bigg(Q_{\phi_2}(s, a) - y(r, s', d) \Bigg)^2 \right].$$

- Using the smaller Q-value for the target, and regressing towards that, helps fend off overestimation in the Q-function.

- the policy is learned just by maximizing Q_{ϕ_1}:

$$\max_{\theta} \mathop{\mathrm{E}}_{s \sim \mathcal{D}} \left[Q_{\phi_1}(s, \mu_\theta(s)) \right],$$

  which is pretty much unchanged from DDPG. However, in TD3, the policy is updated less frequently than the Q-functions are. This helps damp the volatility that normally arises in DDPG because of how a policy update changes the target.

- **Exploration vs. Exploitation**: TD3 trains a deterministic policy in an off-policy way. Because the policy is deterministic, if the agent were to explore on-policy, in the beginning it would probably not try a wide enough variety of actions to find useful learning signals. To make TD3 policies explore better, we add noise to their actions at training time, typically uncorrelated mean-zero Gaussian noise. To facilitate getting higher-quality training data, you may reduce the scale of the noise over the course of training.

Pseudocode: TD3

Algorithm 4 Twin Delayed DDPG

```
Input: initial policy $\theta$, Q-function $\phi_1$, $\phi_2$, empty replay buffer $\mathcal{D}$. Set $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
repeat
    Observe state $s$ and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
    Execute $a$ in the environment
    Observe next state $s'$, reward $r$, and done signal $d$ to indicate whether $s'$ is terminal
    Store $(s, a, r, s', d)$ in replay buffer $\mathcal{D}$
    If $s'$ is terminal, reset environment state.
    if it's time to update then
        for $j$ in range(however many updates) do
            Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
            Compute target actions $a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right), \quad \epsilon \sim \mathcal{N}(0, \sigma)$
            Compute targets $y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a'(s'))$
            Update Q-functions by one gradient step: $\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} \left(Q_{\phi,i}(s, a) - y(r, s', d)\right)^2$ for $i = 1, 2$
            if $j \mod \texttt{policy\_delay} = 0$ then
                Update policy by one step of gradient ascent using $\nabla_\theta \frac{1}{|B|} \sum_{s\in B} Q_{\phi,1}(s, \mu_\theta(s))$
                Update target networks with

                $$\phi_{\text{targ},i} \leftarrow \rho\phi_{\text{targ},i} + (1 - \rho)\phi_i \text{ for } i = 1, 2, \quad \theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$

            end if
        end for
    end if
until convergence
```

Policy Gradient Methods

http://bicmr.pku.edu.cn/~wenzw/bigdata2023.html

Acknowledgement: this slides is based on Prof. Shipra Agrawal's lecture notes

Outline

1. Policy gradient methods
   - Finite horizon MDP
2. Actor-critic methods
3. TRPO and PPO
4. MCTS and AlphaGo Zero

Policy gradient methods

- In Q-learning function approximation was used to approximate Q-function, and policy was a greedy policy based on estimated Q-function. In policy gradient methods, we approximate a stochastic policy directly using a parametric function approximator.
- More formally, given an MDP (S, A, s_1, R, P), let $\pi_\theta : S \to \Delta^A$ denote a randomized policy parameterized by parameter vector $\theta \in \mathbb{R}^d$ For a scalable formulation, we want $d << |S|$.
- For example, the policy π_θ might be represented by a neural network whose input is a representation of the state, whose output is action selection probabilities, and whose weights form the policy parameters θ. (The architecture for such a [deep] neural network is similar to that for a multi-label classifier, with input being a state, and labels being different actions. The network should be trained to predict the probability of different actions given an input state).

Policy gradient methods

- For simplicity, assume that π_θ is differentiable with respect to θ, i.e. $\frac{\partial \pi_\theta(s,a)}{\partial \theta}$ exists. This is true for example, if a neural network with differentiable activation functions is used to define π_θ. Let $\rho\left(\pi_\theta\right)$ denote the gain of policy π_θ.This may be defined as long term average reward, long term discounted reward or total reward in an episode or finite horizon. Therefore, solving for optimal policy reduces to the problem of solving

$$\max_\theta \rho\left(\pi_\theta\right)$$

- In order to use stochastic gradient descent algorithm for finding a stationary point of the above problem, we need to compute (an unbiased) estimate of gradient of $\rho\left(\pi_\theta\right)$ with respect to θ.

Finite horizon MDP

- Here performance measure to optimize is total expected reward over a finite horizon H.

$$\rho(\pi) = \mathbb{E}\left[\sum_{t=1}^{H} \gamma^{t-1} r_t | \pi, s_1\right]$$

- Let $\pi(s, a)$ denote the probability of action a in state s for randomized policy π. Let $D^{\pi}(\tau)$ denote the probability distribution of a trajectory (state-action sequence) $\tau = (s_1, a_1, s_2, \ldots, a_{H-1}, s_H)$ of states on starting from state s_1 and following policy π. That is,

$$D^{\pi}(\tau) := \prod_{i=1}^{H-1} \pi\left(s_i, a_i\right) P\left(s_i, a_i, s_{i+1}\right)$$

Finite horizon MDP

Theorem 2

For finite horizon MDP (S, A, s_1, P, R, H), let $R(\tau)$ be the total reward for an sample trajectory τ, on following π_θ for H steps, starting from state s_1. Then,

$$\nabla_\theta \rho\left(\pi_\theta\right) = \mathbb{E}_\tau\left[R(\tau) \nabla_\theta \log\left(D^{\pi_\theta}(\tau)\right)\right] = \mathbb{E}_\tau\left[R(\tau) \sum_{t=1}^{H-1} \nabla_\theta \log\left(\pi_\theta\left(s_t, a_t\right)\right)\right]$$

Proof. Let $R(\tau)$ be expected total reward for an entire sample trajectory τ, on following π_θ for H steps, starting from states_1. That is, given a sample trajectory $\tau = (s_1, a_1, s_2, \ldots, a_{H-1}, s_H)$ from distribution D^{π_θ},

$$R(\tau) := \sum_{t=1}^{H-1} \gamma^{t-1} R\left(s_t, a_t\right)$$

Proof of Theorem 2

Then,

$$\rho\left(\pi_\theta\right) = \mathbb{E}_{\tau \sim D^{\pi_\theta}}[R(\tau)]$$

Now, (the calculations below implicitly assume finite state and action space, so that the distribution $D(\tau)$ has a finite support)

$$\begin{aligned}
\frac{\partial \rho\left(\pi_\theta\right)}{\partial \theta} &= \frac{\partial}{\partial \theta} \mathbb{E}_{\tau \sim D^{\pi_\theta}}[R(\tau)] \\
&= \frac{\partial}{\partial \theta} \sum_{\tau : D^{\pi_\theta}(\tau) > 0} D^{\pi_\theta}(\tau) R(\tau) \\
&= \sum_{\tau : D^{\pi_\theta}(\tau) > 0} D^{\pi_\theta}(\tau) \frac{\partial}{\partial \theta} \log\left(D^{\pi_\theta}(\tau)\right) R(\tau) \\
&= \mathbb{E}_{\tau \sim D^{\pi_\theta}}\left[\frac{\partial}{\partial \theta} \log\left(D^{\pi_\theta}(\tau)\right) R(\tau)\right]
\end{aligned}$$

Proof of Theorem 2

Further, for a given sample trajectory τ^i.

$$\begin{aligned}
\nabla_\theta \log\left(D^{\pi_\theta}\left(\tau^i\right)\right) &= \sum_{t=1}^{H-1} \nabla_\theta \log\left(\pi_\theta\left(s_t^i, a_t^i\right)\right) + \nabla_\theta \log P\left(s_t^i, a_t^i, s_{t+1}^i\right) \\
&= \sum_{t=1}^{H-1} \nabla_\theta \log\left(\pi_\theta\left(s_t^i, a_t^i\right)\right)
\end{aligned}$$

Finite horizon MDP

- The gradient representation given by above theorem is extremely useful, as given a sample trajectory this can be computed only using the policy parameter, and does not require knowledge of the transition model $P(\cdot,\cdot,\cdot)$! This does seem to require knowledge of reward model, but that can be handled by replacing $R\left(\tau^i\right)$ by $\hat{R}\left(\tau^i\right) = r_1 + \gamma r_2 + \ldots, \gamma^{H-2} r_{H-1}$, the total of sample rewards observed in this trajectory.
- Since, given a trajectory τ, the quantity $D^{\pi_\theta}(\tau)$ is determined, and $\mathbb{E}[\hat{R}(\tau)|\tau] = R(\tau)$

$$\begin{aligned}
\nabla_\theta \rho\left(\pi_\theta\right) &= \mathbb{E}_\tau\left[R(\tau)\nabla_\theta \log\left(D^{\pi_\theta}(\tau)\right)\right] \\
&= \mathbb{E}_\tau\left[\hat{R}(\tau)\nabla_\theta \log\left(D^{\pi_\theta}(\tau)\right)\right] \\
&= \mathbb{E}_\tau\left[\hat{R}(\tau)\sum_{t=1}^{H-1} \nabla_\theta \log\left(\pi_\theta\left(s_t, a_t\right)\right)\right]
\end{aligned}$$

Unbiased estimator of gradient from samples

- From above, given sample trajectories τ^i, $i = 1, \ldots, m$, an unbiased estimator for gradient $\nabla_\theta \rho\left(\pi_\theta\right)$ is given as:

$$\begin{aligned}\hat{\mathbf{g}} &= \frac{1}{m}\sum_{i=1}^{m}\hat{R}\left(\tau^i\right)\nabla_\theta \log\left(D^{\pi_\theta}\left(\tau^i\right)\right)\\ &= \frac{1}{m}\sum_{i=1}^{m}\hat{R}\left(\tau^i\right)\sum_{t=1}^{H-1}\nabla_\theta \log\left(\pi_\theta\left(s_t^i, a_t^i\right)\right)\end{aligned}$$

Baseline

- Note that for any constant b (or b that is conditionally independent of sampling from π_θ given θ), we have:

$$\mathbb{E}_\tau \left[b \frac{\partial}{\partial \theta_j} \log \left(D^{\pi_\theta} (\tau) \right) | \theta, s_1 \right] = b \int_\tau \frac{\partial}{\partial \theta_j} \left(D^{\pi_\theta} (\tau) \right) = b \frac{\partial}{\partial \theta_j} \int_\tau D^{\pi_\theta} (\tau) = 0$$

- Therefore, choosing any 'baseline' b, following is also an unbiased estimator of the $\nabla_\theta \rho \left(\pi_\theta \right)$:

$$\hat{\mathbf{g}} = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H-1} \left(\hat{R} \left(\tau^i \right) - b \right) \nabla_\theta \log \left(\pi_\theta \left(s_t^i, a_t^i \right) \right)$$

Baseline

- Or, more generally, one could even use a state and time dependent baseline $b_t\left(s_t^i\right)$ conditionally is independent of sampling from π_θ given s_t^i, θ, to get estimator:

$$\hat{\mathbf{g}} = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=1}^{H-1} \left(\hat{R}\left(\tau^i\right) - b_t\left(s_t^i\right)\right) \nabla_\theta \log\left(\pi_\theta\left(s_t^i, a_t^i\right)\right) \tag{1}$$

- Below we show this is unbiased. The expectations below are over trajectories $(s_1, a_1, \ldots, a_{H-1}, s_H)$, where $a_t \sim \pi\left(s_t, \cdot\right)$, given s_t. For any fixed θ, t, the baseline $b_t\left(s_t\right)|s_t$ needs to be deterministic or independent of $a_t|s_t$. For simplicity we assume it is deterministic.

Baseline

$$
\begin{aligned}
& \mathbb{E}_\tau\left[\sum_{t=1}^{H-1} b_t\left(s_t\right) \frac{\partial}{\partial \theta_j} \log \left(\pi_\theta\left(s_t, a_t\right)\right) \mid \theta, s_1\right] \\
= \; & \mathbb{E}\left[\sum_{t=1}^{H-1} \mathbb{E}\left[b_t\left(s_t\right) \frac{\partial}{\partial \theta_j} \log \left(\pi_\theta\left(s_t, a_t\right)\right) \mid s_t\right] \mid \theta, s_1\right] \\
= \; & \mathbb{E}\left[\sum_{t=1}^{H-1} b_t\left(s_t\right) \mathbb{E}\left[\frac{\partial}{\partial \theta_j} \log \left(\pi_\theta\left(s_t, a_t\right)\right) \mid s_t\right] \mid \theta, s_1\right] \\
= \; & \mathbb{E}\left[\sum_{t=1}^{H-1} b_t\left(s_t\right) \sum_a \pi_\theta\left(s_t, a\right) \frac{\partial}{\partial \theta_j} \log \left(\pi_\theta\left(s_t, a\right)\right) \mid \theta, s_1\right] \\
= \; & \mathbb{E}\left[\sum_{t=1}^{H-1} b_t\left(s_t\right) \sum_a \frac{\partial}{\partial \theta_j} \pi_\theta\left(s_t, a\right) \mid \theta, s_1\right]
\end{aligned}
$$

Baseline

$$
\begin{aligned}
&= \mathbb{E}\left[\sum_{t=1}^{H-1} b_t\left(s_t\right) \frac{\partial}{\partial \theta_j} \sum_a \pi_\theta\left(s_t, a\right) \mid \theta, s_1\right] \\
&= \mathbb{E}\left[\sum_{t=1}^{H-1} b_t\left(s_t\right) \frac{\partial}{\partial \theta_j} 1 \mid \theta, s_1\right] \\
&= 0
\end{aligned}
$$

- An example of such state dependent baseline $b_t(s)$, given s and θ, is $V_{H-t}^{\pi_\theta}(s)$, i.e., the value of policy π_θ, starting from state s at time t. We will see later that such a baseline is useful in reducing the variance of gradient estimates.

Vanilla policy gradient algorithm

Initialize policy parameter θ, and baseline.
In each iteration,

- Execute current policy π_θ to obtain several sample trajectories τ^i, $i = 1, \dots, m$.
- Use these sample trajectories and chosen baseline to compute the gradient estimator $\hat{\mathbf{g}}$ as in (1)
- Update $\theta \leftarrow \theta + \alpha \hat{g}$
- Update baseline as required.

Above is essentially same as the REINFORCE algorithm introduced by [Williams, 1988, 1992].

Softmax policies

- Consider policy set parameterized by $\theta \in \mathbb{R}^d$ such that given $s \in S$, probability of picking action $a \in A$ is given by:

  $$\pi_\theta(s,a) = \frac{e^{\theta^\top \phi_{sa}}}{\sum_{a' \in A} e^{\theta^\top \phi_{sa'}}}$$

  where each ϕ_{sa} is an d-dimensional feature vector characterizing state-action pair s, a. This is a popular form of policy space called softmax policies. Here,

  $$\nabla_\theta \log\left(\pi_\theta(s,a)\right) = \phi_{sa} - \left(\sum_{a' \in A} \phi_{sa'} \pi_\theta\left(s, a'\right)\right) = \phi_{sa} - \mathbb{E}_{a' \sim \pi(s)}\left[\phi_{sa'}\right]$$

Gaussian policy for continuous action spaces

- In continuous action spaces, it is natural to use Gaussian policies. Given state s, the probability of action a is given as:

  $$\pi_\theta(s,a) = \mathcal{N}\left(\phi(s)^T\theta, \sigma^2\right)$$

  for some constant σ. Here $\phi(s)$ is a feature representation of s. Then,

  $$\nabla_\theta \log\left(\pi_\theta(s,a)\right) = \nabla_\theta \frac{-\left(a-\theta^\top\phi(s)\right)^2}{2\sigma^2} = \frac{\left(\theta^\top\phi(s)-a\right)}{\sigma^2}\phi(s)$$

Outline

1. Policy gradient methods
    - Finite horizon MDP
2. Actor-critic methods
3. TRPO and PPO
4. MCTS and AlphaGo Zero

Actor-critic methods

- Actor-only methods (vanilla policy gradient) work with a parameterized family of policies.
- The gradient of the performance, with respect to the actor parameters, is directly estimated by simulation, and the parameters are updated in a direction of improvement.
- A possible drawback of such methods is that the gradient estimators may have a large variance.
- As the policy changes, a new gradient is estimated independently of past estimates (by sampling trajectories).
- There is no "learning", in the sense of accumulation and consolidation of older information.

Actor-critic methods

- Critic-only methods (e.g., Q-learning, TD-learning) rely exclusively on value function approximation and aim at learning an approximate solution to the Bellman equation, which will then hopefully prescribe a near-optimal policy.
- Such methods are indirect in the sense that they do not try to optimize directly over a policy space.
- A method of this type may succeed in constructing a "good" approximation of the value function, yet lack reliable guarantees in terms of near-optimality of the resulting policy.

Actor-critic methods

- Actor-critic methods aim at combining the strong points of actor-only and critic-only methods, by incorporating value function approximation in the policy gradient methods.
- We already saw the potential of using value function approximation for picking baseline for variance reduction.
- Another more obvious place to incorporate Q-value approximation is for approximating Q-function in the policy gradient expression. Recall, by policy gradient theorem:

$$\nabla_\theta \rho\left(\pi_\theta\right) = \sum_s d^{\pi_\theta}(s) \mathbb{E}_{a \sim \pi(s)}\left[\left(Q^{\pi_\theta}(s, a) - b^{\pi_\theta}(s)\right) \nabla_\theta \log\left(\pi_\theta(s, a)\right)\right]$$

for any baseline $b^{\pi_\theta}(\cdot)$.

Theorem 1 (Sutton et al. [1999])

If function f_ω is compatible with policy parametrization θ in the sense that for every s, a,

$$\nabla_\omega f_\omega(s,a) = \frac{1}{\pi_\theta(s,a)} \nabla_\theta \pi_\theta(s,a) = \nabla_\theta \log\left(\pi_\theta(s,a)\right)$$

And, further we are given parameter ω which is a stationary point of the following least squares problem:

$$\min_\omega \mathbb{E}_{s\sim d^{\pi_\theta}, a\sim\pi_\theta(s,)} \left[\left(Q^{\pi_\theta}(s,a) - b(s;\theta) - f_\omega(s,a)\right)^2 \right]$$

where $b(\cdot;\theta)$ is any baseline, which may depend on the current policy π_θ. Then,

$$\nabla_\theta \rho\left(\pi_\theta\right) = \mathbb{E}_{s\sim d^{\pi_\theta}} \mathbb{E}_{a\in\pi_\theta(s)} \left[f_\omega(s,a) \nabla_\theta \log\left(\pi_\theta(s,a)\right) \right]$$

That is, function approximation f_ω can be used in place of Q-function to obtain gradient with respect to θ.

Proof of Policy gradient theorem

(Here, we abuse the notation and use $\mathbb{E}_{s\sim d^{\pi_\theta}}[x]$ as a shorthand for $\sum_s d^{\pi_\theta}(s)x$. This is not technically correct in the discounted case since in that case $d^{\pi_\theta}(s) = \mathbb{E}_{s_1}\left[\sum_{t=1}^{\infty} \Pr\left(s_t = s; \pi, s_1\right)\gamma^{t-1}\right]$, which is not a distribution. In fact in discounted case, $(1-\gamma)d^{\pi_\theta}$ is a distribution.)

Proof. Given θ, for stationary point ω of the least squares problem:

$$\mathbb{E}_{s\sim d^{\pi}\theta, a\sim\pi_\theta(s,\cdot)}\left[\left(Q^{\pi_\theta}(s,a) - b(s;\theta) - f_\omega(s,a)\right)\nabla_\omega f_\omega(s,a)\right] = 0$$

Substituting the compatibility condition:

$$\mathbb{E}_{s\sim d^{\pi_\theta}, a\sim\pi_\theta(s,\cdot)}\left[\left(Q^{\pi_\theta}(s,a) - b(s;\theta) - f_\omega(s,a)\right)\nabla_\theta \pi_\theta(s,a)\frac{1}{\pi_\theta(s,a)}\right] = 0$$

Proof of Policy gradient theorem

Or,

$$\sum_s d^{\pi_\theta}(s) \sum_a \nabla_\theta \pi_\theta(s,a) \left(Q^{\pi_\theta}(s,a) - b(s;\theta) - f_\omega(s,a)\right) = 0$$

Since $b(s;\theta) \sum_a \nabla_\theta \pi_\theta(s,a) = 0$

$$\sum_s d^{\pi_\theta}(s) \sum_a \nabla_\theta \pi_\theta(s,a) \left(Q^{\pi_\theta}(s,a) - f_\omega(s,a)\right) = 0$$

using this with the policy gradient theorem, we get

$$\nabla_\theta \rho\left(\pi_\theta\right) = \sum_s d^{\pi_\theta}(s) \sum_a \nabla_\theta \pi_\theta(s,a) f_\omega(s,a)$$

Example: softmax policy

- Consider policy set parameterized by θ such that given $s \in S$, probability of picking action $a \in A$ is given by:

$$\pi_\theta(s,a) = \frac{e^{\theta^\top \phi_{sa}}}{\sum_{a' \in A} e^{\theta^\top \phi_{sa'}}}$$

where each ϕ_{sa} is an ℓ-dimensional feature vector characterizing state-action pair s, a. This is a popular form of parameterization. Here,

$$\nabla_\theta \pi_\theta(s,a) = \phi_{sa}\pi_\theta(s,a) - \left(\sum_{a' \in A} \phi_{sa'}\pi_\theta\left(s,a'\right)\right)\pi_\theta(s,a)$$

Example: softmax policy

- Meeting the compatibility condition in Theorem 1 requires that

$$\nabla_\omega f_\omega(s,a) = \frac{1}{\pi_\theta(s,a)} \nabla_\theta \pi_\theta(s,a) = \phi_{sa} - \sum_{a' \in A} \phi_{sa'} \pi_\theta\left(s,a'\right)$$

- A natural form of $f_\omega(s,a)$ satisfying this condition is:

$$f_\omega(s,a) = \omega^\top \left(\phi_{sa} - \sum_{b \in A} \phi_{sb} \pi_\theta(s,b) \right)$$

- Thus f_ω must be linear in the same features as the policy, except normalized to be mean zero for each state. In this sense it is better to think of f_ω as an approximation of the advantage function, $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$, rather than Q^π.

Example: Gaussian policy for continuous action spaces

- In continuous action spaces, it is natural to use Gaussian policy. Given state s, the probability of action a is given as:

  $$\pi_\theta(s,a) = \mathcal{N}\left(\phi(s)^T\theta, \sigma^2\right)$$

  for some constant σ. Here $\phi(s)$ is a feature representation of s. Then, compatibility condition for $f_\omega(s,a)$:

$$\nabla_\omega f_\omega(s,a) = \nabla_\theta \log\left(\pi_\theta(s,a)\right) = \nabla_\theta \frac{-\left(a-\theta^\top\phi(s)\right)^2}{2\sigma^2} = \frac{\left(\theta^\top\phi(s)-a\right)}{\sigma^2}\phi(s)$$

- For f_ω to satisfy this, it must be linear in ω, e.g., $f_\omega(s,a) = \frac{\left(\theta^\top\phi(s)-a\right)}{\sigma^2}\phi(s)^\top\omega$

Policy iteration algorithm with function approximation

Let $f_\omega(\cdot,\cdot)$ be such that $\nabla f_\omega(s,a) = \nabla_\theta \log \pi_\theta(s,a)$ for all ω, θ, s, a.
Initialize θ_1, $\pi_1 := \pi_{\theta_1}$. Pick step sizes $\alpha_1, \alpha_2, \ldots,$.
In iteration $k = 1, 2, 3, \ldots,$

- Policy evaluation: Find $w_k = w$ such that

$$\mathbb{E}_{s\sim d^{\pi_k}} \mathbb{E}_{a\sim\pi_k(s)} \left[\left(Q^{\pi_k}(s,a) - b_k(s) - f_\omega(s,a) \right) \nabla_\theta \log\left(\pi_k(s,a)\right) \right] = 0$$

  (Here, d^{π_k} is not normalized to 1, and sums to $1/(1-\gamma)$.)

- Policy improvement:

$$\theta_{k+1} \leftarrow \theta_k + \alpha_k \mathbb{E}_{s\sim d^{\pi_k}} \mathbb{E}_{a\sim\pi_k(s)} \left[f_\omega(s,a) \nabla_\theta \log\left(\pi_k(s,a)\right) \right]$$

A similar algorithm appears in Konda and Tsitsiklis [1999].

Convergence Guarantees

Following version of convergence guarantees were provided by Sutton et al. [1999] for infinite horizon MDPs (average or discounted).

Theorem 2 (Sutton et al. [1999])

Given $\alpha_1, \alpha_2, \ldots$, such that

$$\lim_{T\to\infty} \sum_{k=1}^{T} \alpha_k = \infty, \lim_{T\to\infty} \sum_{k=1}^{T} \alpha_k^2 < \infty$$

and $\max_{\theta,s,a,i,j} \frac{\partial^2 \pi_\theta(s,a)}{\partial\theta_i\theta_j} < \infty$. Then, for $\theta_1, \theta_2, \ldots$, obtained by the above algorithm,

$$\lim_{k\to\infty} \nabla_\theta \rho\left(\theta\right)\big|_{\theta_k} = 0$$

Pseudocode: Vanilla Policy Gradient Algorithm

Algorithm 1 Vanilla Policy Gradient Algorithm

```
Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$
for $k = 0, 1, 2, \dots$ do
    Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
    Compute rewards-to-go $\hat{R}_t$.
    Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function $V_{\phi_k}$.
    Estimate policy gradient as
        $$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta (a_t|s_t)|_{\theta_k} \hat{A}_t.$$
    Compute policy update, either using standard gradient ascent,
        $$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k,$$
    or via another gradient ascent algorithm like Adam.
    Fit value function by regression on mean-squared error:
        $$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left( V_\phi(s_t) - \hat{R}_t \right)^2,$$
    typically via some gradient descent algorithm.
end for
```

Outline

1. Policy gradient methods
    - Finite horizon MDP

2. Actor-critic methods

3. TRPO and PPO

4. MCTS and AlphaGo Zero

Trust Region Policy Optimization (John Schulman, etc)

- Assume start-state distribution d_0 is independent with policy
- Total expected discounted reward with policy π

$$\eta(\pi) = E_\pi[\sum_{t=0}^{\infty} \gamma^t r(s_t)]$$

- Advantage function: $A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$
- Between any two different policy $\widetilde{\pi}$ and π

$$\begin{aligned}
\eta(\widetilde{\pi}) &= \eta(\pi) + E_{\widetilde{\pi}}[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t)] \\
&= \eta(\pi) + \sum_{t=0}^{\infty} \sum_s P(s_t = s|\widetilde{\pi}) \sum_a \widetilde{\pi}(a|s) \gamma^t A_\pi(s,a) \\
&= \eta(\pi) + \sum_s \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\widetilde{\pi}) \sum_a \widetilde{\pi}(a|s) A_\pi(s,a) \\
&= \eta(\pi) + \sum_s d_{\widetilde{\pi}}(s) \sum_a \widetilde{\pi}(a|s) A_\pi(s,a).
\end{aligned}$$

Trust Region Policy Optimization

- Find new policy $\widetilde{\pi}$ to maximize $\eta(\widetilde{\pi}) - \eta(\pi)$ for given π, that is

$$\max_{\widetilde{\pi}} \eta(\pi) + \sum_s d_{\widetilde{\pi}}(s) \sum_a \widetilde{\pi}(a|s) A_\pi(s,a)$$

- For simpleness, maximize the approximator

$$L_\pi(\widetilde{\pi}) = \eta(\pi) + \sum_s d_\pi(s) \sum_a \widetilde{\pi}(a|s) A_\pi(s,a)$$

- Parameterize the policy $\pi(a|s) := \pi_\theta(a|s)$

$$L_{\pi_{\theta_{old}}}(\pi_\theta) = \eta(\pi_{\theta_{old}}) + \sum_s d_{\pi_{\theta_{old}}}(s) \sum_a \pi_\theta(a|s) A_{\pi_{\theta_{old}}}(s,a)$$

Why $L_{\pi_{\theta_{old}}}(\pi_\theta)$?

- A sufficiently small step $\theta_{old} \to \theta$ improves $L_{\pi_{\theta_{old}}}(\pi_\theta)$ also improves η

$$\begin{aligned} L_{\pi_{\theta_{old}}}(\pi_{\theta_{old}}) =& \eta(\pi_{\theta_{old}}), \\ \nabla_\theta L_{\pi_{\theta_{old}}}(\pi_\theta)|_{\theta=\theta_{old}} =& \nabla_\theta \eta(\pi_\theta)|_{\theta=\theta_{old}}. \end{aligned}$$

- Lower bounds on the improvement of η

$$\eta(\pi_{\theta_{new}}) \geq L_{\pi_{\theta_{old}}}(\pi_{\theta_{new}}) - \frac{2\epsilon\gamma}{(1-\gamma)^2}\alpha^2$$

where

$$\begin{aligned} \epsilon =& \max_s |E_{a\sim\pi_{\theta_{new}}} A_{\pi_{\theta_{old}}}(s,a)| \\ \alpha =& D_{TV}^{max}(\pi_{\theta_{old}}||\pi_{\theta_{new}}) = \max_s D_{TV}(\pi_{\theta_{old}}(\cdot|s)||\pi_{\theta_{new}}(\cdot|s)) \end{aligned}$$

Lower bound

- TV divergence between two distribution p, q (discrete case)

$$D_{TV}(p\|q) = \frac{1}{2}\sum_{X} |p(X) - q(X)|$$

- KL divergence between two distribution p, q (discrete case)

$$D_{KL}(p\|q) = \sum_{X} p(X) \log \frac{p(X)}{q(X)}$$

- $(D_{TV}(p||q))^2 \leq D_{KL}(p||q)$ (Pollard(2000),Ch.3)
- Thus obtain a lower bound

$$\eta(\pi_{\theta_{new}}) \geq L_{\pi_{\theta_{old}}}(\pi_{\theta_{new}}) - \frac{2\epsilon\gamma}{(1-\gamma)^2}\alpha$$

where

$$\alpha = D_{KL}^{max}(\pi_{\theta_{old}}||\pi_{\theta_{new}}) := \max_{s} D_{KL}(\pi_{\theta_{old}}(\cdot|s)||\pi_{\theta_{new}}(\cdot|s))$$

Practical algorithm

- The penalty coefficient $\frac{2\epsilon\gamma}{(1-\gamma)^2}$ is large in practice, which yields small update

- Take a constraint on the KL divergence, i.e., a trust region constraint:

$$\begin{aligned} \max_{\theta} \quad & L_{\pi_{\theta_{old}}}(\pi_\theta) \\ \text{s.t.} \quad & D_{KL}^{max}(\pi_{\theta_{old}} || \pi_\theta) \leq \delta \end{aligned}$$

- A heuristic approximation

$$\begin{aligned} \max_{\theta} \quad & L_{\pi_{\theta_{old}}}(\pi_\theta) \\ \text{s.t.} \quad & \overline{D}_{KL}^{\rho_{\pi_{\theta_{old}}}}(\pi_{\theta_{old}} || \pi_\theta) \leq \delta \end{aligned}$$

where

$$\overline{D}_{KL}^{\rho_{\pi_{\theta_{old}}}}(\pi_{\theta_{old}} || \pi_\theta) = E_{\pi_{\theta_{old}}}(D_{KL}(\pi_{\theta_{old}}(\cdot|s) || \pi_\theta(\cdot|s)))$$

TRPO

- The objective and constraint are both zero when $\theta = \theta_k$. Furthermore, the gradient of the constraint with respect to θ is zero when $\theta = \theta_k$.

- The theoretical TRPO update isn't the easiest to work with, so TRPO makes some approximations to get an answer quickly. We Taylor expand the objective and constraint to leading order around θ_k:

$$L_{\theta_k}(\theta) \approx g^T(\theta - \theta_k)$$
$$\bar{D}_{KL}(\theta_k||\theta) \approx \frac{1}{2}(\theta - \theta_k)^T H(\theta - \theta_k)$$

resulting in an approximate optimization problem,

$$\theta_{k+1} = \arg\max_{\theta} g^T(\theta - \theta_k)$$
$$\text{s.t. } \frac{1}{2}(\theta - \theta_k)^T H(\theta - \theta_k) \le \delta.$$

TRPO

- By happy coincidence, the gradient g of the surrogate advantage function with respect to θ, evaluated at $\theta = \theta_k$, is exactly equal to the policy gradient, $\nabla_\theta J(\pi_\theta)$
- This approximate problem can be analytically solved by the methods of Lagrangian duality, yielding the solution:

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g.$$

- TRPO adds a modification to this update rule: a backtracking line search,

$$\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g,$$

where $\alpha \in (0,1)$ is the backtracking coefficient, and j is the smallest nonnegative integer such that $\pi_{\theta_{k+1}}$ satisfies the KL constraint and produces a positive surrogate advantage.

TRPO

- computing and storing the matrix inverse, H^{-1}, is painfully expensive when dealing with neural network policies with thousands or millions of parameters. TRPO sidesteps the issue by using the 'conjugate gradient' algorithm to solve $Hx = g$ for $x = H^{-1}g$, requiring only a function which can compute the matrix-vector product Hx instead of computing and storing the whole matrix H directly.

$$Hx = \nabla_\theta \left(\left(\nabla_\theta \bar{D}_{KL}(\theta_k || \theta)\right)^T x \right)$$

Pseudocode: TRPO

Algorithm 2 Trust Region Policy Optimization

```
Input: initial policy $\theta_0$, initial value function $\phi_0$, KL-divergence limit $\delta$, backtracking coefficient $\alpha$
for $k = 0, 1, 2, \ldots$ do
    Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
    Compute rewards-to-go $\hat{R}_t$.
    Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function $V_{\phi_k}$.
    Estimate policy gradient as
        $$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t.$$
    Use the conjugate gradient algorithm to compute $\hat{x}_k \approx \hat{H}_k^{-1} \hat{g}_k$, where $\hat{H}_k$ is the Hessian of the sample average KL-divergence.
    Update the policy by backtracking line search with $\theta_{k+1} = \theta_k + \alpha^j \sqrt{\frac{2\delta}{\hat{x}_k^T \hat{H}_k \hat{x}_k}} \hat{x}_k$, where $j$ is the smallest value which improves the sample loss and satisfies the sample KL-divergence constraint.
    Fit value function by regression on mean-squared error:
        $$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left( V_\phi(s_t) - \hat{R}_t \right)^2,$$
    typically via some gradient descent algorithm.
end for
```

Proximal Policy Optimization (PPO)

- PPO is motivated by the same question as TRPO: how can we take the biggest possible improvement step on a policy using the data we currently have, without stepping so far that we accidentally cause performance collapse? Where TRPO tries to solve this problem with a complex second-order method, PPO is a family of first-order methods that use a few other tricks to keep new policies close to old.

- **PPO-Penalty** approximately solves a KL-constrained update like TRPO, but penalizes the KL-divergence in the objective function instead of making it a hard constraint, and automatically adjusts the penalty coefficient over the course of training so that it's scaled appropriately.

- **PPO-Clip** doesn't have a KL-divergence term in the objective and doesn't have a constraint at all. Instead relies on specialized clipping in the objective function to remove incentives for the new policy to get far from the old policy.

Key Equations

- PPO-clip updates policies via

  $$\theta_{k+1} = \arg\max_{\theta} \underset{s,a\sim\pi_{\theta_k}}{\mathrm{E}} \left[L(s,a,\theta_k,\theta) \right],$$

  typically taking multiple steps of (usually minibatch) SGD to maximize the objective. Here L is given by

  $$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a), \;\; \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon \right) A^{\pi_{\theta_k}}(s,a) \right),$$

  in which ϵ is a (small) hyperparameter which roughly says how far away the new policy is allowed to go from the old.
- `https://openai.com/research/openai-baselines-ppo`

- This is a pretty complex expression, and it's hard to tell at first glance what it's doing, or how it helps keep the new policy close to the old policy. As it turns out, there's a considerably simplified version of this objective which is a bit easier to grapple with

  $$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a), \;\; g(\epsilon, A^{\pi_{\theta_k}}(s,a))\right),$$

  where

  $$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0. \end{cases}$$

- To figure out what intuition to take away from this, let's look at a single state-action pair (s,a), and think of cases.

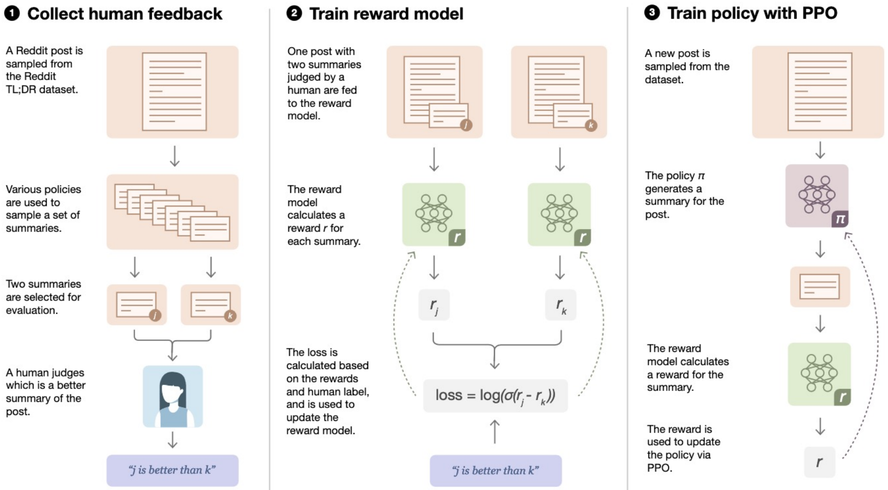
Pseudocode: PPO

Algorithm 3 PPO-Clip

```
Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$
for $k = 0, 1, 2, \ldots$ do
    Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
    Compute rewards-to-go $\hat{R}_t$.
    Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function $V_{\phi_k}$.
    Update the policy by maximizing the PPO-Clip objective:
        $\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left( \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$
    typically via stochastic gradient ascent with Adam.
    Fit value function by regression on mean-squared error:
        $\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left( V_\phi(s_t) - \hat{R}_t \right)^2,$
    typically via some gradient descent algorithm.
end for
```

Reinforcement Learning from Human Feedback

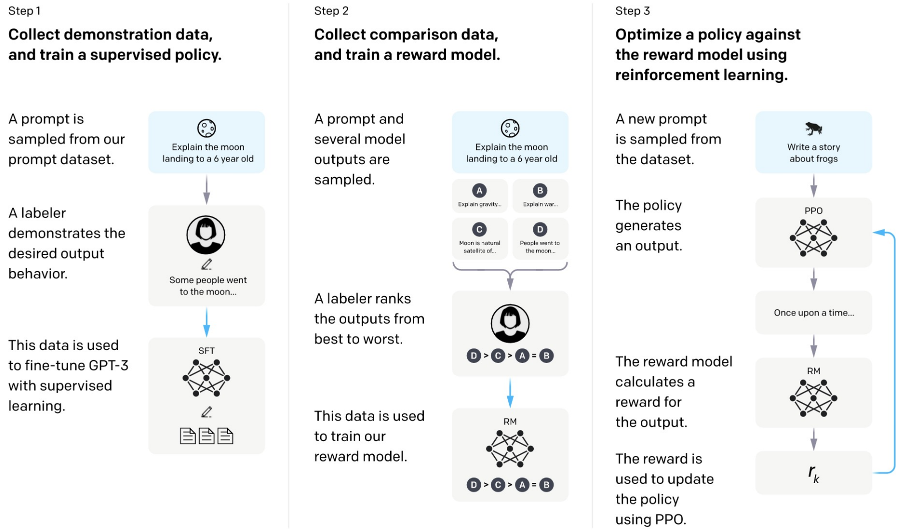
RLHF: Nisan Stiennon, etc, Learning to summarize with human feedback, NeurIPS 2020



❶ Collect human feedback
A Reddit post is sampled from the Reddit TL;DR dataset.
Various policies are used to sample a set of summaries.
Two summaries are selected for evaluation.
j
k
A human judges which is a better summary of the post.
"j is better than k"
❷ Train reward model
One post with two summaries judged by a human are fed to the reward model.
j
k
The reward model calculates a reward r for each summary.
r
r
r_j
r_k
The loss is calculated based on the rewards and human label, and is used to update the reward model.
$\text{loss} = \log(\sigma(r_j - r_k))$
"j is better than k"
❸ Train policy with PPO
A new post is sampled from the dataset.
The policy π generates a summary for the post.
π
The reward model calculates a reward for the summary.
r
The reward is used to update the policy via PPO.
r

InstructGPT

provide detailed, accurate, and instructive responses to user queries.



Step 1

Collect demonstration data, and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3 with supervised learning.

Step 2

Collect comparison data, and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.

Step 3

Optimize a policy against the reward model using reinforcement learning.

A new prompt is sampled from the dataset.

The policy generates an output.

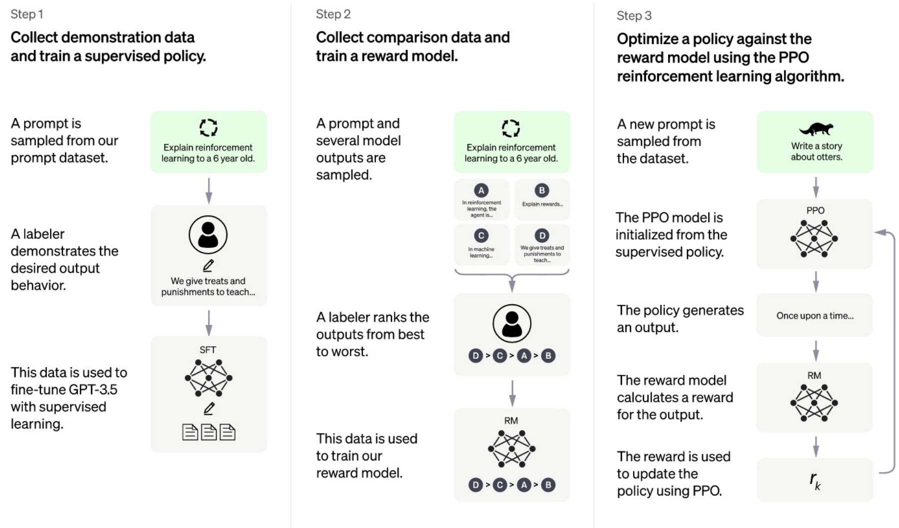
The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.

Explain the moon landing to a 6 year old
Some people went to the moon...
SFT
Explain the moon landing to a 6 year old
A Explain gravity...
B Explain war...
C Moon is natural satellite of...
D People went to the moon...
D > C > A = B
RM
D > C > A = B
Write a story about frogs
PPO
Once upon a time...
RM
r_k

ChatGPT

generate human-like text based on the input it's given, and it can carry out a wide-ranging conversation on various topics.



Step 1

Collect demonstration data and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3.5 with supervised learning.

Step 2

Collect comparison data and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.

Step 3

Optimize a policy against the reward model using the PPO reinforcement learning algorithm.

A new prompt is sampled from the dataset.

The PPO model is initialized from the supervised policy.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.

Explain reinforcement learning to a 6 year old.
We give treats and punishments to teach...
SFT
Explain reinforcement learning to a 6 year old.
A
In reinforcement learning, the agent is...
B
Explain rewards...
C
In machine learning...
D
We give treats and punishments to teach...
D > C > A > B
RM
D > C > A > B
Write a story about otters.
PPO
Once upon a time...
RM
r_k

Outline

1. Policy gradient methods
   - Finite horizon MDP
2. Actor-critic methods
3. TRPO and PPO
4. MCTS and AlphaGo Zero

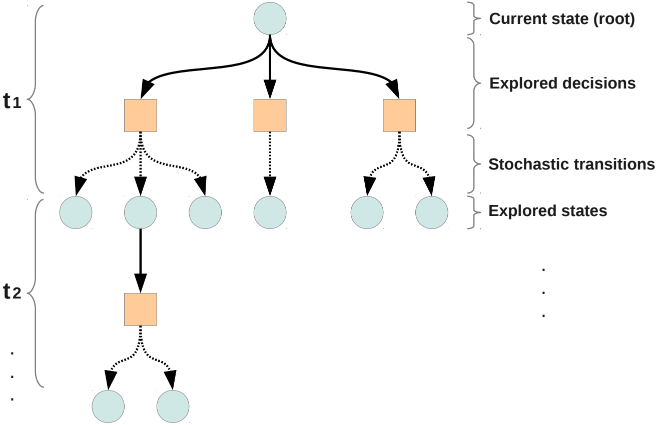
Monte-Carlo Tree Search (MCTS)

- MCTS is a recent algorithm for *sequential decision making*
- MCTS is a versatile algorithm (it does not require knowledge about the problem)
- especially, does not require any knowledge about the Bellman value function
- stable on high dimensional problems
- it outperforms all other algorithms on some problems (difficult games like Go, general game playing, . . .)

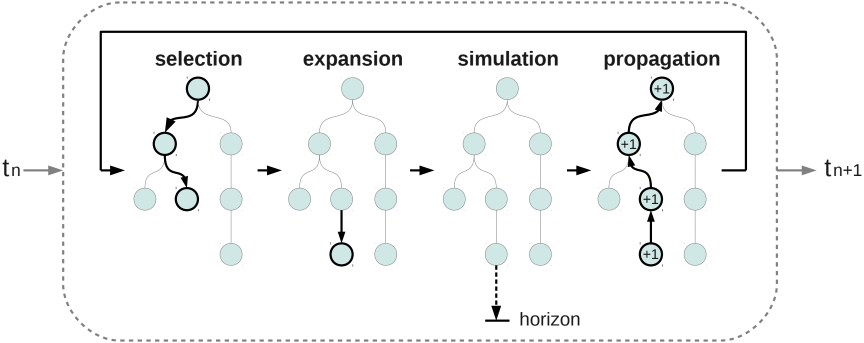
MCTS

Problems are represented as a tree structure:

- blue circles = states
- plain edges + red squares = decisions
- dashed edges = stochastic transitions between two states



Main steps of MCTS

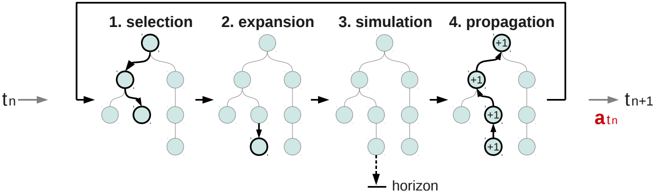


Main steps of MCTS

Starting from an initial state:

1. **Selection**: select the state we want to expand from
2. **Expansion**: One (or more) child nodes are added to expand the tree, according to the available actions.
3. **Simulation**: A simulation is run from the new node(s) according to the default policy (pick actions randomly) to produce an outcome.
4. **Back-propagation** of some information:
   - $N(s, a)$: number of times decision a has been simulated in s
   - $N(s)$: number of time s has been visited in simulations
   - $Q(s, a)$: mean reward of simulations where a was chosen in s

Main steps of MCTS

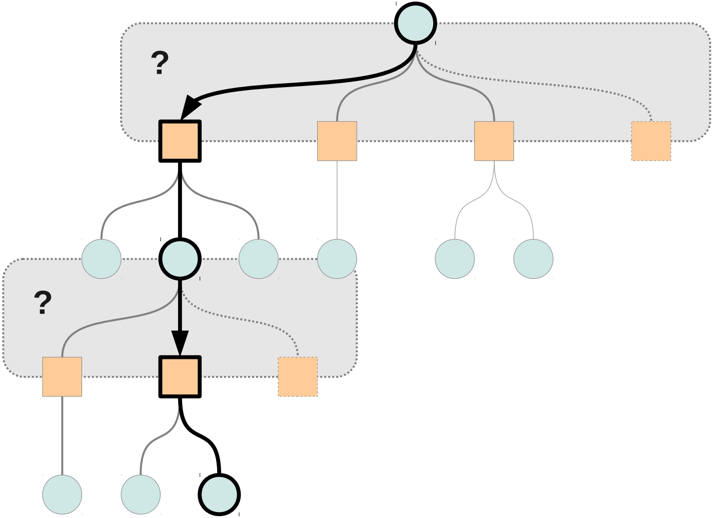


The selected decision

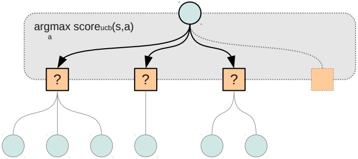
a_{t_n} = the most visited decision from the current state (root node)

Selection step

How to select the state to expand ?



How to select the state to expand ?



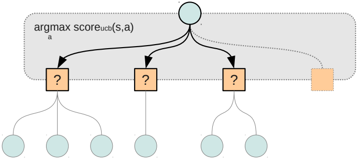
The *selection* phase is driven by Upper Confidence Bound (UCB):

$$\text{score}_{\text{ucb}}(s,a) = \underbrace{Q(s,a)}_{1} + \underbrace{\sqrt{\frac{\log(2+N(s))}{2+N(s,a)}}}_{2}$$

① mean reward of simulations including action a in state s

② the uncertainty on this estimation of the action’s value

How to select the state to expand ?



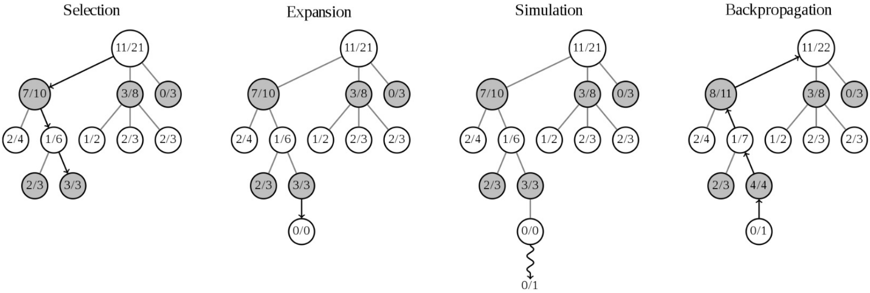
The *selection* phase is driven by Upper Confidence Bound (UCB):

$$\text{score}_{\text{ucb}}(s,a) = \underbrace{Q(s,a)}_{1} + \underbrace{\sqrt{\frac{\log(2+N(s))}{2+N(s,a)}}}_{2}$$

The selected action:

$$a^\star = \arg\max_a \ \text{score}_{\text{ucb}}(s,a)$$

Example: Back-propagation



AlphaZero

2018/12/7, AlphaZero at “Science”. It demonstrates learning chess, shogi and go, tabula rasa without any domain-specific human knowledge or data, only using self-play. The evaluation is performed against strongest programs available.

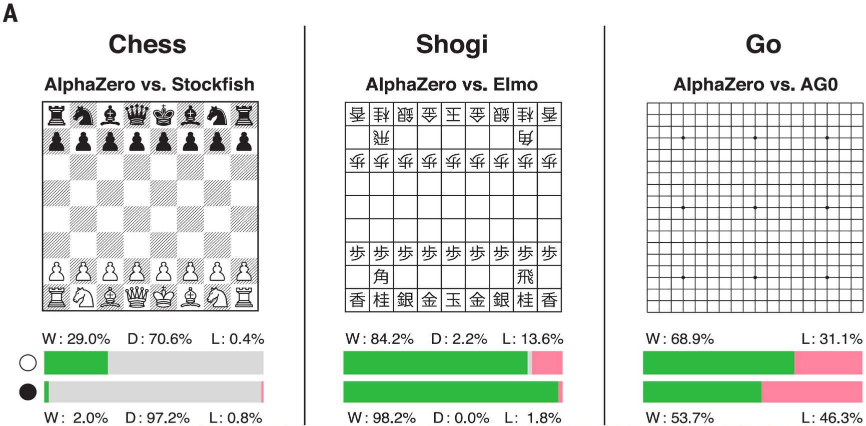


Figure 2 of the paper "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play" by David Silver et al.

AlphaZero

- AlphaZero uses a neural network predicting $(p(s), v(s)) = f(s, \theta)$ for a given state s
  - $p(s)$ is a vector of move probabilities
  - $v(s)$ is expected outcome of the game in range $[-1, 1]$.
- Unlike the standard MCTS, Alpha Go Zero does not use a default policy to perform a rollout in order to achieve an estimate of the value of a state.
- By a sequence of simulated self-play games, the search can improve the estimate of p and v, and can be considered a powerful policy evaluation operator given a network f predicting policy p and value estimate v, MCTS produces a more accurate policy π and better value estimate w for a given state:

$$(\pi(s), w(s)) \leftarrow \text{MCTS}(p(s), v(s), f) \text{ for } (p(s), v(s)) = f(s, \theta).$$

AlphaZero: MCTS

MCTS keeps a tree of currently explored states from a fixed root state. Each node corresponds to a game state and to every non-root node we got by performing an action a from the parent state. Each state-action pair (s, a) stores the following set of statistics:

- visit count $N(s, a)$
- total action-value $W(s, a)$
- mean action-value $Q(s, a) = W(s, a)/N(s, a)$, which is not stored explicitly
- prior probability $P(s, a)$ of selecting action a in state s

AlphaZero: UCB

- Each simulation starts in the root node and finishes in a leaf node s_L. In a state s_t, an action is selected using a variant of PUCT algorithm as

  $$a_t = \arg\max_a \quad Q(s_t, a) + U(s_t, a)$$

  where

  $$U(s, a) = C(s)P(s, a)\frac{\sqrt{N(s)}}{1 + N(s, a)},$$

  with $C(s) = \log\left(\frac{1+N(s)+c_{base}}{c_{base}}\right) + c_{init}$.
- In the Alphazero paper, $c_{init} = 1.25$ and $c_{base} = 19652$.

AlphaZero

When reaching a leaf node s_L,

- evaluate it by the network, generating (p, v)
- add all its children with $N = W = 0$ and the prior probability p,
- in the backward pass for all $t \leq L$, we update the statistics in nodes by performing
  - $N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$, and
  - $W(s_t, a_t) \leftarrow W(s_t, a_t) \pm v$, depending on the player on turn.

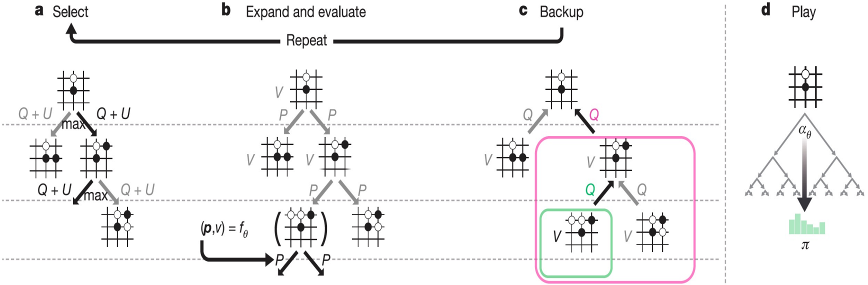


Figure 2 of the paper "Mastering the game of Go without human knowledge" by David Silver et al.

AlphaZero

- The MCTS runs usually several hundreds simulations in a single tree. The result is a distribution proportional to exponentiated visit counts $N(s_{root}, a)^{1/\tau}$ using a temperature ($\tau = 1$ is mostly used), together with the predicted value function.
- The next move is chosen as either:
  - proportional to visit counts $N(s_{root}, \cdot)^{1/\tau}$

    $$\pi_{root}(a) \sim N(s_{root}, \cdot)^{1/\tau}$$

  - deterministically as the most visited action

    $$\pi_{root} = \arg\max_a N(s_{root}, a)$$

- During self-play, the stochastic policy is used for the first 30 moves of the game, while the deterministic is used for the rest of the moves. (This does not affect the internal MCTS search, there we always sample according to PUCT rule.)

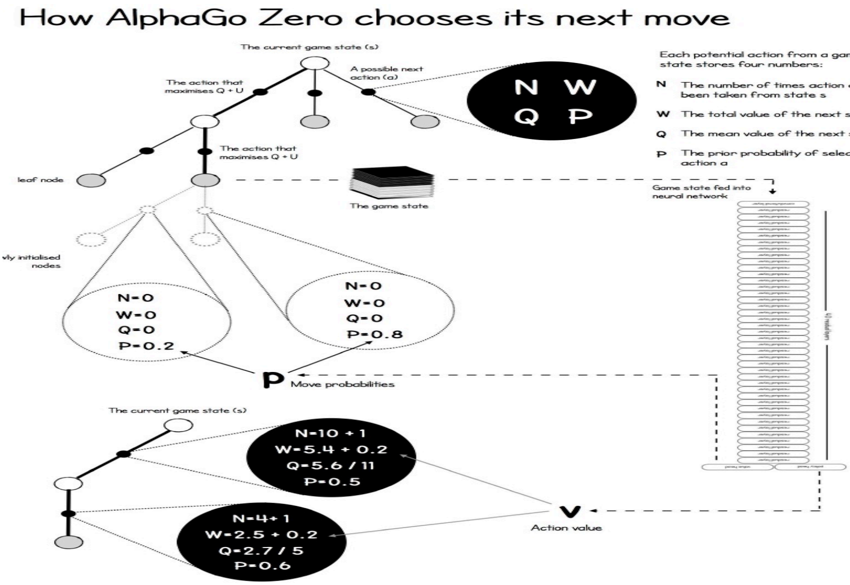
AlphaZero: Loss function

- The network is trained from self-play games.
- A game is played by repeatedly running MCTS from a state s_t and choosing a move $a_t \sim \pi_t$, until a terminal position s_T is encountered, which is then scored according to game rules as $z \in \{-1, 0, 1\}$.
- Finally, the network parameters are trained to minimize

$$L = (z - v)^2 + \pi^\top \log p + c\|\theta\|^2$$

  - a mean squared error between the predicted outcome v and the simulated outcome z
  - a crossentropy/KL divergence for the action distribution, i.e., the similarity of the policy vector p and the search probabilities π
  - L2 regularization

AlphaGo Zero

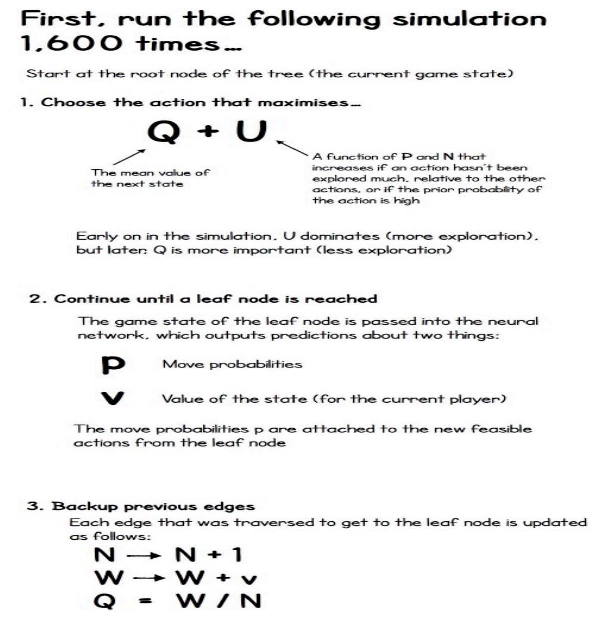


How AlphaGo Zero chooses its next move

Each potential action from a game state stores four numbers:

N	The number of times action a has been taken from state s
W	The total value of the next state
Q	The mean value of the next state
P	The prior probability of selecting action a

AlphaGo Zero



First, run the following simulation 1,600 times…

Start at the root node of the tree (the current game state)

1. Choose the action that maximises…

$$Q + U$$

The mean value of the next state

A function of P and N that increases if an action hasn't been explored much, relative to the other actions, or if the prior probability of the action is high

Early on in the simulation, U dominates (more exploration), but later, Q is more important (less exploration)

2. Continue until a leaf node is reached

The game state of the leaf node is passed into the neural network, which outputs predictions about two things:

p	Move probabilities
v	Value of the state (for the current player)

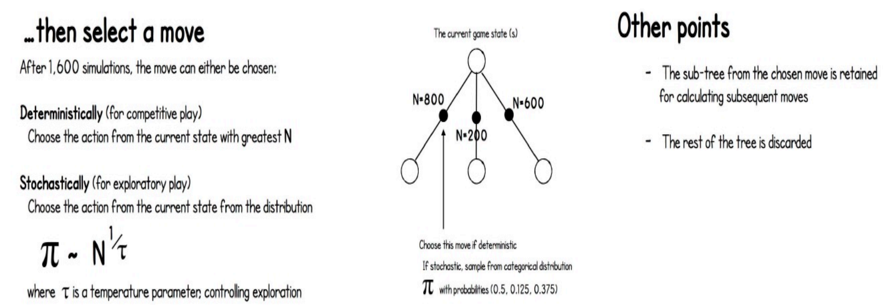
The move probabilities p are attached to the new feasible actions from the leaf node

3. Backup previous edges

Each edge that was traversed to get to the leaf node is updated as follows:

$$N \longrightarrow N + 1$$
$$W \longrightarrow W + v$$
$$Q = W / N$$

AlphaGo Zero



...then select a move

After 1,600 simulations, the move can either be chosen:

Deterministically (for competitive play)
Choose the action from the current state with greatest N

Stochastically (for exploratory play)
Choose the action from the current state from the distribution

$$\pi \sim N^{1/\tau}$$

where τ is a temperature parameter, controlling exploration

The current game state (s)
N=800
N=200
N=600

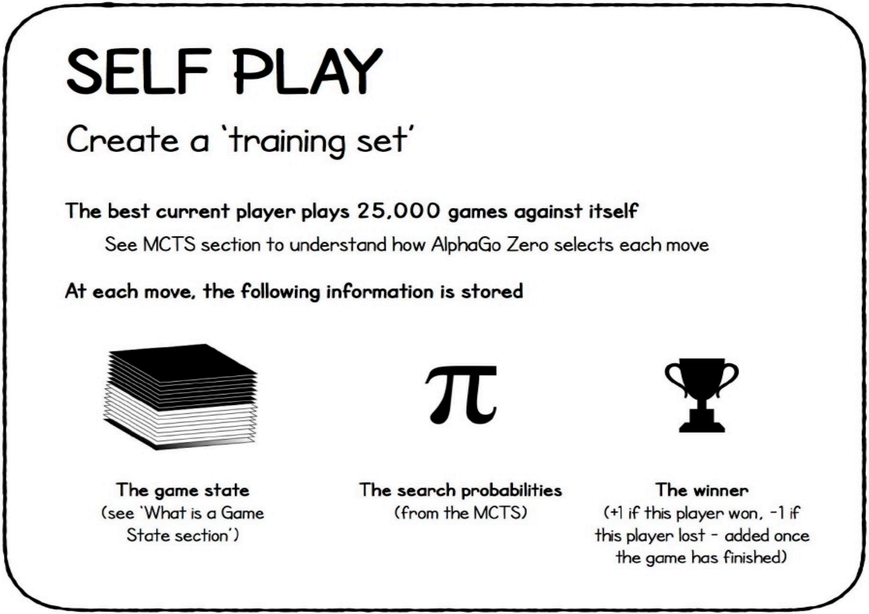
Choose this move if deterministic

If stochastic, sample from categorical distribution π with probabilities (0.5, 0.125, 0.375)

Other points

- The sub-tree from the chosen move is retained for calculating subsequent moves
- The rest of the tree is discarded

AlphaGo Zero



SELF PLAY

Create a 'training set'

The best current player plays 25,000 games against itself

See MCTS section to understand how AlphaGo Zero selects each move

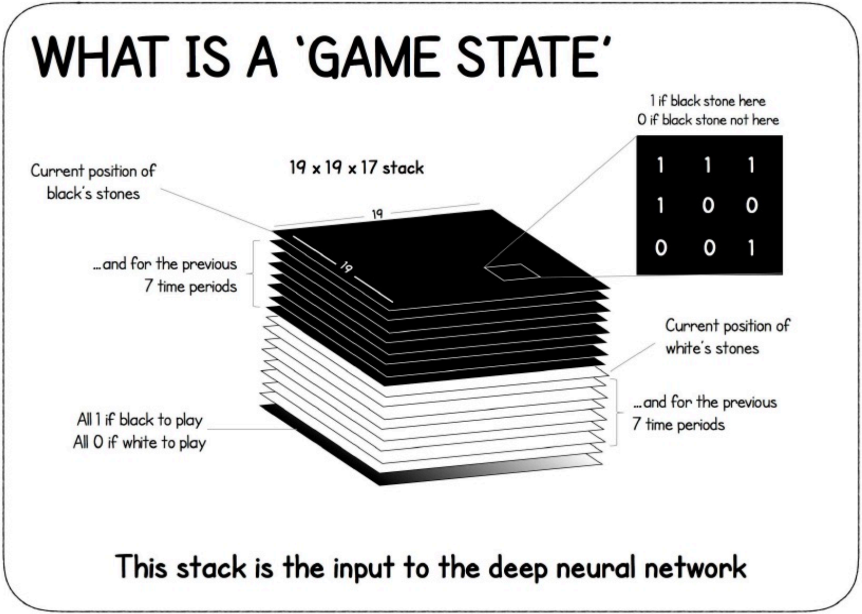
At each move, the following information is stored

The game state
(see 'What is a Game State section')

The search probabilities
(from the MCTS)

The winner
(+1 if this player won, -1 if this player lost - added once the game has finished)

AlphaGo Zero

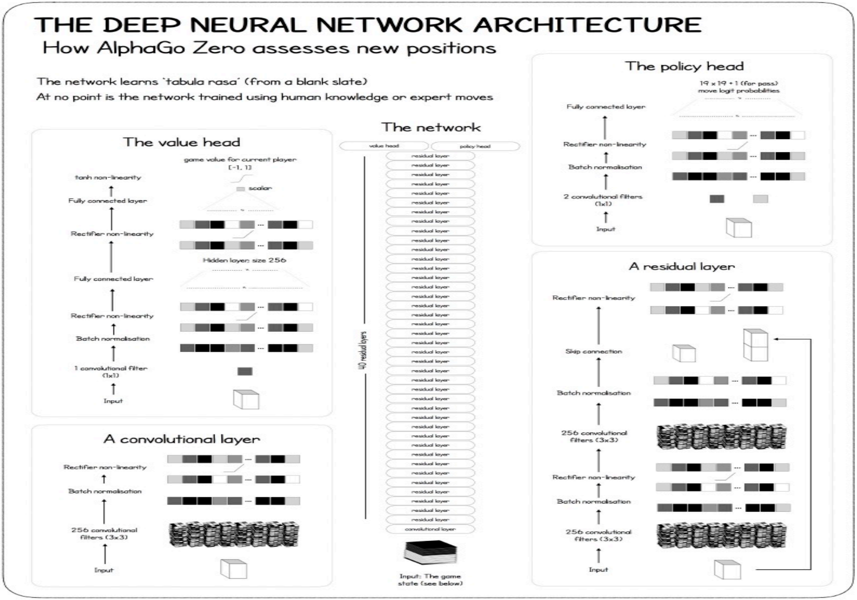


WHAT IS A 'GAME STATE'

1 if black stone here
0 if black stone not here
19 x 19 x 17 stack
Current position of black's stones
19
19
1 1 1
1 0 0
0 0 1
...and for the previous 7 time periods
Current position of white's stones
...and for the previous 7 time periods
All 1 if black to play
All 0 if white to play

This stack is the input to the deep neural network

AlphaGo Zero



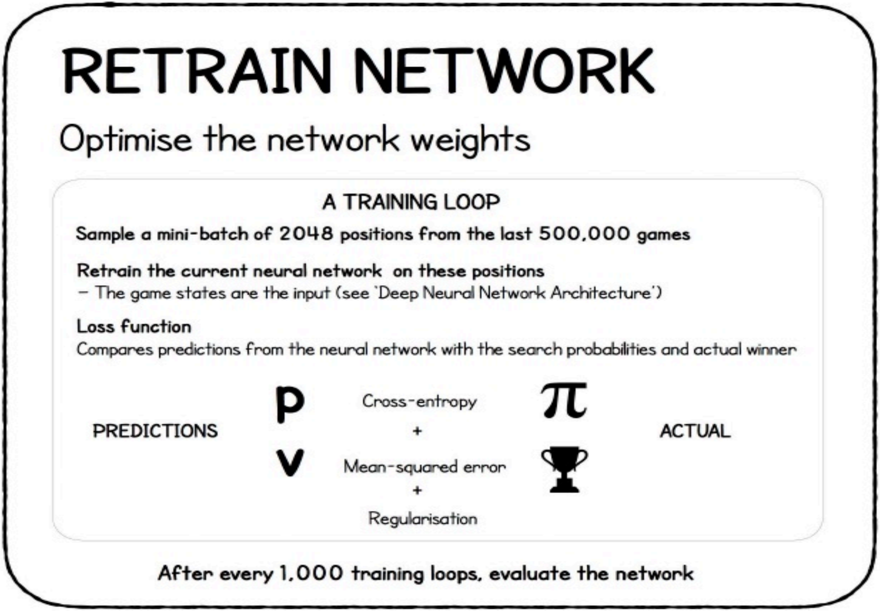
THE DEEP NEURAL NETWORK ARCHITECTURE

How AlphaGo Zero assesses new positions

The network learns 'tabula rasa' (from a blank slate)

At no point is the network trained using human knowledge or expert moves

AlphaGo Zero



RETRAIN NETWORK

Optimise the network weights

A TRAINING LOOP

Sample a mini-batch of 2048 positions from the last 500,000 games

Retrain the current neural network on these positions

– The game states are the input (see 'Deep Neural Network Architecture')

Loss function

Compares predictions from the neural network with the search probabilities and actual winner

PREDICTIONS

$\mathbf{p}$ Cross-entropy π

+

$\mathbf{v}$ Mean-squared error 🏆

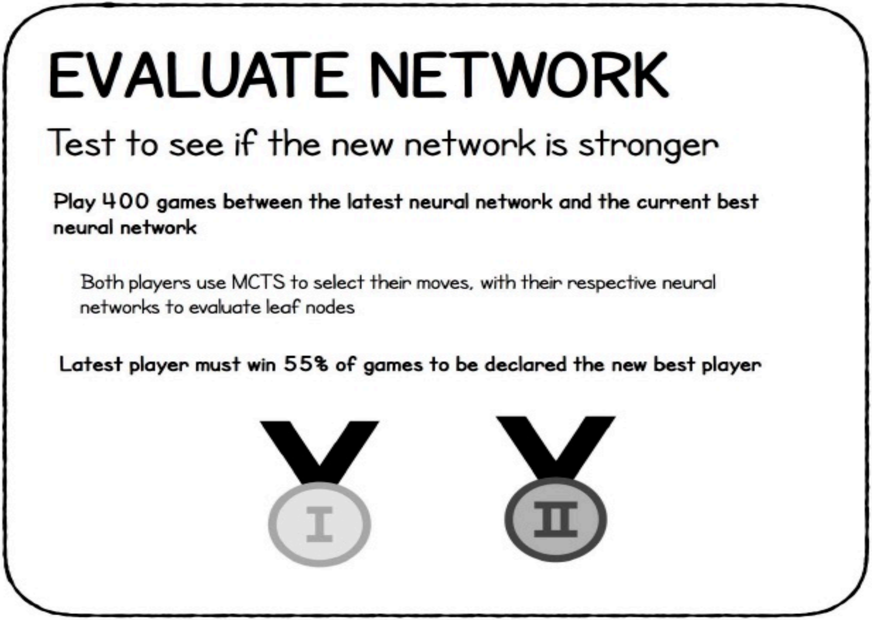
+

Regularisation

ACTUAL

After every 1,000 training loops, evaluate the network

AlphaGo Zero



EVALUATE NETWORK

Test to see if the new network is stronger

Play 400 games between the latest neural network and the current best neural network

Both players use MCTS to select their moves, with their respective neural networks to evaluate leaf nodes

Latest player must win 55% of games to be declared the new best player