# Final Review

2020年11月25日 8:01

30% first half
70% second half

History of AI
Supervised Learning Framework
Gradient Descent Analysis
Linear Regression
Ridge Lasso Compressed Sensing
SVM
Generalization Theory

## Neural Network Optimization

- similar with kernel method but the kernel changed
- filtering matrix $Z = I_{r,i} x_i$ and $H = Z^T Z$

updating rule:

- $\mathrm{W}(t+1) = W(t) - \eta Z(f(x) - y)$
- $f_{W(t+1)}(X) = f_{W(t)}(X) - \eta Z^T Z(f(x) - y)$

The implication and limitation (need $H(t) \approx H(0)$) of the analysis

- $w_r$ changes but $Z, H$ does not change too much
- when learning rate very slow and network super wide, $\|w_r(0)\|$ Gaussian and $\|w_r(t) - w_r(0)\|$ relatively small by linear convergence

## PCA/SVD

- How to do dimension reduction (JL Lemma? does not utilize the feature of data)
- PCA different interpretation: maximal variance, minimal reconstruction error
- related to SVD: $X = U\Sigma V^T, XX^T = U^T \Sigma^2 U$
- Power method find the most significant eigenvector, remove it and iterate
- Decoupling: reduce the degree to which each program module relies on each of the other modules
- Microservices,data processing as pipeline, asynchronous communication
- not coupled with physical servers, concurrency(inside a micro), database

## LSH

c-approximate R-near neighbour
definition: a family $H$ $(R, cR, P_1, P_2)$-sensitive

- $\Pr_{h \in H}[h(p) = h(q)] \geq P_1, \forall \|p - q\| \leq R$
- $\Pr_{h \in H}[h(p) = h(q)] \leq P_2, \forall \|p - q\| \geq cR$

LSH library of $\ell_2$: $h_{r,b} = \left\lfloor \frac{\langle r, x \rangle + b}{w} \right\rfloor$ (intuition: project all points to a line and split it into buckets)

- L functions from $H$, each from $R^d$ to $R^k$. $\rho = \frac{\log\frac{1}{p_1}}{\log\frac{1}{p_2}}$, $L = n^\rho$, $k = \log_{\frac{1}{p_2}} n$

Metric Learning

- searching in orginial $R^d$ space may not be the best choice, find $f: R^d \to R^k$
- NCA: $p_{i,j} = \frac{e^{-\left\|f(x_i)-f(x_j)\right\|^2}}{\sum_{k \neq i} e^{-\left\|f(x_i)-f(x_k)\right\|^2}}$
  optimize $f(A) = \sum_i P_i$ , where $P_i = \sum_{j \in C_i} p_{i,j}$
- LMNN: Triplet loss $\max(0, \left\|f(x) - f(x^+)\right\| - \left\|f(x) - f(x^-)\right\| + r)$
  Pull positive points together and push the negative points far away
- Spreading vectors: use regularizer to push points away

## Decision Tree

- good explanation but hard switch paths and easy to overfit

Boolean function analysis

- Fourier basis: $\chi_S(x) = \prod_{i \in S} x_i$ for $S \subset [n]$
- dot product defined on uniform distribution $D$ on $\{-1,1\}^n$
- $f_\alpha(x)$: all Fourier basis starting with $\alpha$ summed together (a natural function in Decision trees)

Approximate any $s$ leaf node decision tree by $\frac{s^2}{\epsilon}$ sparse $\log(\frac{s}{\epsilon})$ degree boolean function

- proof: cut → L1(f)<s → L0(f) < $\frac{s^2}{\epsilon}$

KM algorithm

- If $E[f_\alpha^2] \geq \theta^2$, which means the branch is still promising, further explore $\alpha 0$, $\alpha 1$

LMN algorithm

- take $m$ samples and use them to estimate $\hat{f}_S$: $\frac{1}{m}\sum_{i=1}^m f(x_i)\chi_S(x_i)$
- Compressed Sensing: $\hat{f}_S(x)$ sparse input x, $\chi_S$ basis as matrix A.

Gini Index

- Pick splitting variable!
- Gini = 1- $\sum_{i=1}^n p_i^2$, i means in the i-th class
- 0 means all in one class and 1 means random
- The Gini index for a random variable is the weighted summation

Random Forest

- Bagging: samples $n$ times with replacement, use the data to train a tree $T_b$, repeat B times
- Feature Bagging: Each tree takes random subset of features, force to use all features
- Boosting: combine weak learners to form a strong one
- Repeat T times with updating data samples!

Adaboost

- Intuition: sample more data on hard cases (with wrong answer by weak learner $h_t$)
- Training error $\leq \prod_t 2\sqrt{\epsilon_t(1-\epsilon_t)}$
- proof: compute final distribution → training error = $\prod_t Z_t$ → compute $Z_t$
- Generalization: Small train loss margin implies small test loss + adaboost train loss margin→0
- Extension to regression: use coordinate descent since dim is too large
- Gradient boosting: fit a weak model for data $(x_i, y_i - F(x_i))$
- $F(x_i) = F(x_i) + h(x_i)$ equals to(a weak estimation of) $F(x_i) = F(x_i) - \frac{\partial L}{\partial F(x_i)}$

## Robust ML

Adversarial attack

- Projected gradient descent: $\max_{\delta \in \Delta} L(x + \delta, y; \theta)$
- $\delta = P_\Delta(\delta + \nabla_\delta L(x + \delta, y; \theta))$

- FGSM: use $\ell_\infty$ ball as $\Delta$, essentially clipping the gradient

- Danskin's Theorem: we can optimize through the max just by finding its maximizing value

Robust features:
- get a robust model and its feature extract function generate $x_r$ from random initialization such that $g(x_r) \approx g(x)$
- intuitively, robust features are similar but non-robust features are independent, so training on robust features gives robust performance

Randomized Smoothing (proof for $\ell_2$, Gaussian)
- get smoothed classifier $g$ by base classifier $f$, greedy fill based on $\frac{\Pr_{x\ ball}(x)}{\Pr_{x+\delta\ ball}(x)}$

## Hyperparameter Tuning

Bayesian Optimization
- prior, sample (balance exploitation and exploration), update prior
- hard to parallelize

Gradient Descent
- continuous
- Compute $\nabla_\eta f(w_0, \eta)$ (How to store long chain of backpropagation?)
- naive idea: store parameters (memory inefficient)
- revert back idea: store all gradient (memory inefficient)
- **improvement:** use momemtum to store info for all gradients (cons: finite precision)
- the momentum $v_t$ stores the compressed info of gradient $w_1, \ldots, w_T$
- **improvement:** use finite precision but store the risidual

Random Search
- much better than grid search!

Best arm identification
- successive Halving algorithm
- remove half of the arms with worst empirical performance
- proof: bound the probability that #arms with better empirical value than the best $\geq \frac{1}{3} \times \frac{3}{4}$
- Hyperband: automatically tune n for $B/n$

Neural Architectrue Search
- ProxylessNAS directly learn on original task, instead of on several different predefined tasks
- each layer consider all possibilities, output relies on one sampled path
- (how to automatically tune depth and the major challenge solved?)

## Matrix Completion

Assumption:
- Matrix $A$ low rank
- known entries uniformly distributed
- Incoherence: A is not too sparse

Solution
- $P_\Omega(A)$:unknow entries filled with 0
- want to find $A = UV^T$ with low rank U,V
- minimize: $\min_{U,V\in R^{n\times r}} \left|\left|P_\Omega(UV^T) - P_\Omega(\mathrm{A})\right|\right|$
- Not convex, but we can alternatively minimize U,V , each subproblem is convex

## Non-convex optimization

Assumption:
- local min are equally good (robust to randomness)

Main theorem

- smooth, bounded, strict saddle, Hessian smooth
- SGD has variance in every direction, SGD will escape all saddle points and converge to local min (poly)
- proof not required

## Clustering

K means

Spectral Graph clustering (Graph Laplacian)

- because $\ell_p$ metric is not necessarily the best, we define similarities in a more general way
- relaxation of ratio cut problem (NP)

Graph Laplacian

- #zero eigenvalues = #Connected components

Ratio Cut Problem

- The Spectral graph clustering is an approximation of Ratio cut

t-SNE

- scale down dimension,
- $x \to y$, keep similarity distribution
- tune $\sigma_i$ by matching with perplexity
- Intuition of SNE: smooth measurement of effective number of neighbors
- t-SNE: use student t distribution (heavy tail distribution to distribute closer points farther)(crowded problem)

## Differential Privacy

$(\epsilon, \delta)$-differential private randomized algorithm M

- for all $S \subseteq Range(M), x, y \in N^{|X|}$, such that $||x-y||_1 \leq 1$:
- $\Pr[M(x) \in S] \leq e^{\epsilon} \Pr[M(y) \in S] + \delta$

Immunity to postprocessing

- $f(M(x))$ is also $(\epsilon, \delta)$-DP

Composition of DP:

- $(\sum \epsilon_i, \sum \delta_i)$-DP

Randomness is essential

- If determinisitic, for two database with different outputs(nontrivial), consider the path

DP promise: changing one choice does not change utility of anyone much

Laplace Machanism

- Given any function $f : N^{|X|} \to \mathbb{R}^k$, the Laplace mechanism is described as:

$$M_k(x, f, \epsilon) = f(x) + (Y_1, \ldots, Y_k)$$

- where $Y_i$ is drawn randomly from $Lap(\frac{\Delta f}{\epsilon})$
- $\Delta f = \max_{x,y \in N^{|X|}, ||x-y||_1 = 1} ||f(x) - f(y)||_1$
- $Lap(b) = \frac{1}{2b} e^{-\frac{|x|}{b}}$

## Learning Augmented Algorithm

- Design algorithms to learn data pattern, for better performance (also reasonably good for special cases)
- Completely replace / replace one gadget / Give oracle advice
- search in min-max range, many queries are wasted if doing binary search

## Neural architectures

- Stochastic depth
- Resnet vs Densenet