Large Scale Optimization for Machine Learning:

How to Use Hessian?

Meisam Razaviyayn

razaviya@usc.edu

Using Hessian in Optimization Algorithms

- Using Hessian is beneficial if possible (why?)
- Hessian is huge when the dimension is large

- How should we use Hessian in large size problems?
  - (Block) diagonal Newton
  - Conjugate gradient method
  - Use Hessian-Vector product approximation

Conjugate Directions

$\mathbf{Q} \in \mathbb{R}^{n \times n}$

Start from simple convex positive definite quadratic optimization: $\min_{\mathbf{x}} \quad \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} - \mathbf{b}^T\mathbf{x}$

Solution: $\mathbf{Q}\mathbf{x}^* = \mathbf{b}$

Definition: $\text{Directions } \mathbf{d}^0, \mathbf{d}^1, \ldots, \mathbf{d}^k \text{ are } \mathbf{Q} - \text{conjugate if}$

$$(\mathbf{d}^i)^T\mathbf{Q}\mathbf{d}^j = \mathbf{0}, \quad \forall i, j \text{ with } i \neq j$$

Lemma: If a set of vectors are Q-conjugate, then they are linearly independent

There are *n* Q-conjugate directions and they form a (non-orthogonal) basis

Finding Solution Using Conjugate Directions

Idea: Expand using conjugate directions:

$$\mathbf{x}^* = \sum_{i=1}^{n} \alpha^i \mathbf{d}^i \qquad \text{where } \alpha^k = \frac{(\mathbf{d}^k)^T \mathbf{b}}{(\mathbf{d}^k)^T \mathbf{Q} \mathbf{d}^k}, \ \forall k$$

$$\min_{\mathbf{x}} \quad \frac{1}{2}\mathbf{x}^T \mathbf{Q}\mathbf{x} - \mathbf{b}^T \mathbf{x}$$

Solution: $\mathbf{Q}\mathbf{x}^* = \mathbf{b}$

Alternative approach for computing coefficients:

$$\text{Start from } \mathbf{x}^0 = 0$$

$$\text{for } i = 0, 1, \ldots, n-1$$

$$\alpha^i = \arg\min_{\alpha} f(\mathbf{x}^i + \alpha \mathbf{d}^i)$$

$$\mathbf{x}^{i+1} = \mathbf{x}^i + \alpha^i \mathbf{d}^i$$

$$\text{return } \mathbf{x}^* = \mathbf{x}^n$$

This iterative procedure finds the solution in finite number of iterations!

figure

How can we find ***Q***-conjugate directions?

Generating Q-conjugate Directions

Gram-Schmidt Procedure:

Given linearly independent vectors $\boldsymbol{\xi}^0, \ldots, \boldsymbol{\xi}^{n-1}$

generate Q-conjugate $\mathbf{d}^k$'s such that $\boldsymbol{\xi}^k = \sum_{i=0}^{k-1} \beta_k^i \mathbf{d}^i + \mathbf{d}^k, \ \forall k$

In other words,

$$\mathbf{d}^0 = \boldsymbol{\xi}^0$$

$$\mathbf{d}^k = \boldsymbol{\xi}^k - \sum_{i=0}^{k-1} \frac{(\mathbf{d}^i)^T \mathbf{Q} \boldsymbol{\xi}^k}{(\mathbf{d}^i)^T \mathbf{Q} \mathbf{d}^i} \mathbf{d}^i$$

A Simple Recursion for Generating Q-conjugate Directions

$$\text{Start with } \boldsymbol{\xi}^0 = -\nabla f(\mathbf{x}^0), \ldots, \boldsymbol{\xi}^{n-1} = -\nabla f(\mathbf{x}^{n-1})$$

Gram-Schmidt Procedure:

$$\mathbf{d}^0 = -\nabla f(\mathbf{x}^0)$$

$$\mathbf{d}^k = -\nabla f(\mathbf{x}^k) + \frac{\left(\nabla f(\mathbf{x}^k)\right)^T \left(\nabla f(\mathbf{x}^k) - \nabla f(\mathbf{x}^{k-1})\right)}{\left(\nabla f(\mathbf{x}^{k-1})\right)^T \left(\nabla f(\mathbf{x}^{k-1})\right)} \mathbf{d}^{k-1}$$

$$\text{for } k = 1, \ldots, n-1$$

Conjugate Gradient Method for Quadratic Optimization

Start with $\mathbf{d}^0 = -\nabla f(\mathbf{x}^0)$

for $k = 1, 2, \ldots, n$

$$\alpha^{k-1} = \arg\min_{\alpha} f(\mathbf{x}^{k-1} + \alpha \mathbf{d}^{k-1})$$

$$\mathbf{x}^k = \mathbf{x}^{k-1} + \alpha^{k-1}\mathbf{d}^{k-1}$$

$$\mathbf{d}^k = -\nabla f(\mathbf{x}^k) + \frac{\left(\nabla f(\mathbf{x}^k)\right)^T \left(\nabla f(\mathbf{x}^k) - \nabla f(\mathbf{x}^{k-1})\right)}{\left(\nabla f(\mathbf{x}^{k-1})\right)^T \left(\nabla f(\mathbf{x}^{k-1})\right)} \mathbf{d}^{k-1}$$

return $\mathbf{x}^* = \mathbf{x}^n$

Solves quadratic problems in exactly *n* iterations

For general optimization:

- It is similar to approximating the function with second order Taylor expansion
- The Hessian is changing
- Do not stop
- Restart at some m <n (when not enough decrease)

Other ways to use Hessian information

For Newton step, we need to compute $\left(\nabla^2 f(\mathbf{x})\right)^{-1} \nabla f(\mathbf{x})$

- Approach 1: Approximate Hessian with diagonal or block diagonal matrices

- Approach 2: Use Hessian-vector product

$$\nabla^2 f(\mathbf{x})\mathbf{v} \approx \frac{\nabla f(\mathbf{x} + \alpha\mathbf{v}) - \nabla f(\mathbf{x})}{\alpha} \text{ for small } \alpha$$

Large Scale Optimization for Machine Learning:

GANs and Min-Max Problems

Meisam Razaviyayn

razaviya@usc.edu

Generative Adversarial Networks

Goal: Generate samples that look like real samples $\mathbf{x}_1, \ldots, \mathbf{x}_n \sim \mathbb{P}_x$

$\mathbf{z}$: input random variable, e.g., $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

We need $G(\mathbf{z})$ to have the same distribution as $\mathbb{P}_x$

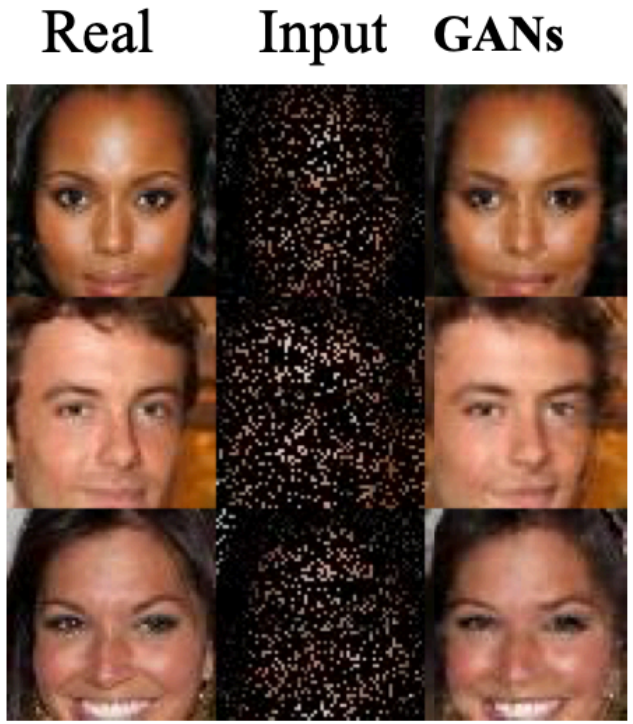
$$\min_G \ \text{dist}\left(G(\mathbf{z}), \mathbf{x}\right)$$

Generative Adversarial Networks

Neural Network

Applications of GANs:

- Simulated environments and training data
- Dealing with missing data [Dai et al. 2017]
- Super-resolution

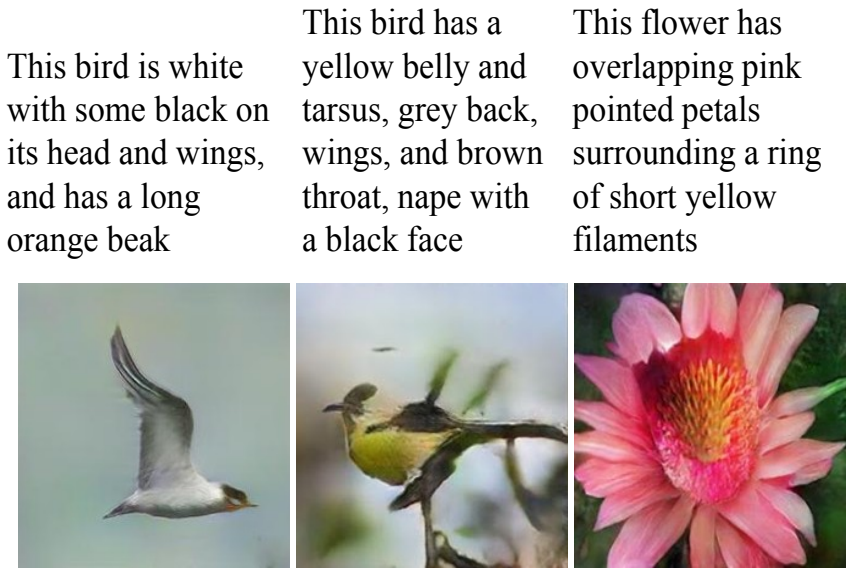


Yeh et al. 2017

Generative Adversarial Networks

Applications of GANs:

- Simulated environments and training data
- Dealing with missing data
- Super-resolution
- Text-to-image synthesis



Zhang et al. 2017

GAN Formulation

$$\min_G \quad \text{dist}\left(G(\mathbf{z}), \mathbf{x}\right)$$

How to define a distance between two distributions?

Many “distances” between distributions:

KL-divergence, Jensen-Shannon divergence, TV distance, Wasserstein divergence,…

Kullback-Leibler Divergence

Definition: $$\mathrm{KL}(P\|Q) = \int P(x) \log\left(\frac{P(x)}{Q(x)}\right) dx = \mathbb{E}_{x\sim P}\left[\log\left(\frac{P(x)}{Q(x)}\right)\right]$$

Discrete case: $\mathrm{KL}(P\|Q) = \sum_i p_i \log\left(\frac{p_i}{q_i}\right)$

This divergence is zero if and only if P = Q almost everywhere.

Drawbacks:

Not clear how to optimize it based on samples for GANs

It is a non-symmetric divergence

Jensen-Shannon Divergence

Definition: $$\mathrm{JS}(P\|Q) = \frac{1}{2}\mathrm{KL}(P\|M) + \frac{1}{2}\mathrm{KL}(Q\|M)$$

$$\text{where } M = \frac{P+Q}{2}.$$

Drawback: Not clear how to minimize it based on samples.

Lemma: $$\mathrm{JS}(P\|Q) = 1 + \frac{1}{2}\max_{D\in\mathbb{D}} \; \mathbb{E}_{x\sim P}\left[\log D(x)\right] + \mathbb{E}_{y\sim Q}\left[\log(1-D(y))\right]$$

$$\text{where } \mathbb{D} = \text{ set of all functions with range } (0,1)$$

Proof

More appropriate formulation for optimization

Wasserstein Distance

Definition: $$\mathrm{W}(P\|Q) = \min_{\pi \in \Pi} \int c(x,y)\pi(x,y)\, dy\, dx$$

$$\text{where } \Pi = \{\pi(x,y) \mid \int \pi(x,y)dy = P(x),\ \int \pi(x,y)dx = Q(y)\}$$

Discrete case: $$\mathrm{W}(P\|Q) = \min_{\pi} \sum_{x,y} c(x,y)\pi(x,y)$$

$$\text{s.t. } \pi(x,y) \geq 0,\ \sum_{y} \pi(x,y) = P(x), \forall x,\ \sum_{x} \pi(x,y) = Q(y), \forall y$$

Intuition and earth mover's distance

Wasserstein Distance

Definition: $\mathrm{W}(P\|Q) = \min_{\pi\in\Pi} \int c(x,y)\pi(x,y)\,dy\,dx$

$$\text{where } \Pi = \{\pi(x,y) \mid \int \pi(x,y)dy = P(x),\ \int \pi(x,y)dx = Q(y)\}$$

Issue: Need to work with probabilities directly and it also requires $O(n^2)$ variables

Dual formulation:

$$W(P\|Q) = \max_{\gamma,\lambda} \mathbb{E}_{x\sim P}\left[\gamma(x)\right] + \mathbb{E}_{y\sim Q}\left[\lambda(y)\right]$$

$$\text{s.t.} \quad \gamma(x) + \lambda(y) \le c(x,y), \forall x, y$$

Proof

When $c(x,y) = \|x-y\|_2$, $W(P\|Q) = \max_{\gamma} \mathbb{E}_{x\sim P}\left[\gamma(x)\right] - \mathbb{E}_{y\sim Q}\left[\gamma(y)\right]$

$$\text{s.t.} \quad \gamma(x) - \gamma(y) \le \|x-y\|_2, \forall x, y$$

GAN Formulation

$$\min_{G} \quad \text{dist}\left(G(\mathbf{z}), \mathbf{x}\right)$$

Jensen-Shannon GANs: $$\min_{G} \quad \max_{D \in \mathbb{D}} \quad \mathbb{E}_{\mathbf{x}}\left[\log D(\mathbf{x})\right] + \mathbb{E}_{\mathbf{z}} \log\left(1 - D(G(\mathbf{z}))\right)$$

$$\mathbb{D} = \text{ set of all functions with range } (0, 1)$$

Wasserstein GANs: $$\min_{G} \quad \max_{\gamma} \quad \mathbb{E}_{\mathbf{x}}\left[\gamma(\mathbf{x})\right] - \mathbb{E}_{\mathbf{z}}\left[\gamma(G(\mathbf{z}))\right]$$

$$\text{s.t.} \quad \gamma(\mathbf{x}) - \gamma(\mathbf{y}) \leq \|\mathbf{x} - \mathbf{y}\|_2, \forall \mathbf{x}, \mathbf{y}$$

- How to solve in practice over the set of functions?
  - Parameterized G, γ, D with neural network
- Why do we call it Generative Adversarial Networks? Discriminator vs. Generator

GAN Formulation

Parameterizing the functions:

$$\min_{G} \quad \text{dist}\left(G(\mathbf{z}), \mathbf{x}\right)$$

Jensen-Shannon GANs:

$$\min_{\theta} \quad \max_{\beta} \quad \mathbb{E}_{\mathbf{x}}\left[\log D_{\beta}(\mathbf{x})\right] + \mathbb{E}_{\mathbf{z}} \log\left(1 - D_{\beta}(G_{\theta}(\mathbf{z}))\right)$$

$$\text{range}(D_{\beta}(\cdot)) = (0, 1)$$

Wasserstein GANs:

$$\min_{\theta} \quad \max_{\beta} \quad \mathbb{E}_{\mathbf{x}}\left[\gamma_{\beta}(\mathbf{x})\right] - \mathbb{E}_{\mathbf{z}}\left[\gamma_{\beta}(G_{\theta}(\mathbf{z}))\right]$$

$$\text{s.t.} \quad \gamma_{\beta}(\mathbf{x}) - \gamma_{\beta}(\mathbf{y}) \leq \|\mathbf{x} - \mathbf{y}\|_2, \forall \mathbf{x}, \mathbf{y}$$

How to make it unconstrained optimization?

Resulting Optimization Problem

$$\min_{\theta} \max_{\beta} f(\theta, \beta)$$

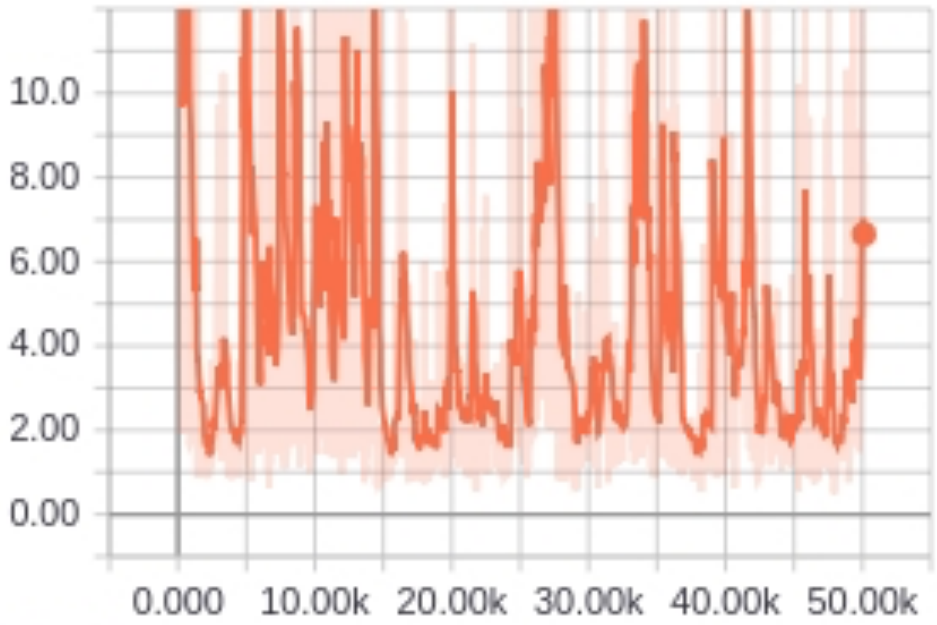
Many other GAN formulations as well

- How to solve?
  - Gradient Descent/Ascent used in practice

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta f(\theta_t, \beta_t)$$

$$\beta_{t+1} = \beta_t + \alpha \nabla_\beta f(\theta_t, \beta_t)$$

- Very unstable in practice (even its variants)

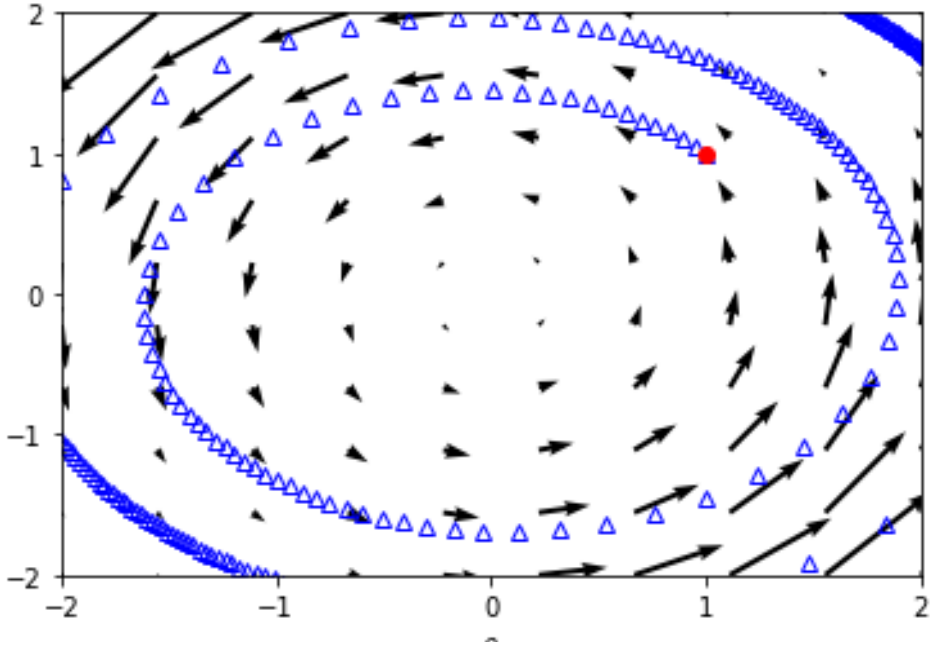


Should GDA Work?

$$\min_{\theta} \max_{\beta} f(\theta, \beta)$$

Simple example: $f(\theta, \beta) = \theta\beta$

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta f(\theta_t, \beta_t)$$
$$\beta_{t+1} = \beta_t + \alpha \nabla_\beta f(\theta_t, \beta_t)$$



- Where should the algorithm converge to?
- Does the algorithm converge?

How to solve?

$$\min_{\theta} \max_{\beta} f(\theta, \beta)$$

- What should we hope for computing?
  - Finding the global min?
  - Finding a first order stationary point with respect to θ, β
    - Open problem in the general case.

Another Point of View

$$\min_{\theta} \max_{\beta} f(\theta, \beta)$$

$$g(\theta)$$

- Can we do gradient descent on $g(\theta)$?
- Find a point that $||\nabla g(\theta)||$ is small enough?
- Danskin's theorem requires solving the max problem exactly

$$\theta_{t+1} = \theta_t - \alpha \nabla g(\theta_t) \longleftrightarrow$$

$$\beta_{t+1} = \arg\max_{\beta} f(\theta_t, \beta)$$

$$\theta_{t+1} = \theta_t - \alpha \nabla_{\theta} f(\theta_t, \beta_{t+1})$$

- In practice, we may not be able to solve the max
  - No hope for this approach when non-concave w.r.t. β. Even under concavity, we cannot solve the inner problem exactly

Another Point of View

$$\min_{\theta} \max_{\beta} f(\theta, \beta)$$

$$g(\theta)$$

- Simpler case: what happens if $f(\theta, \beta)$ is strongly concave in β?

$$\theta_{t+1} = \theta_t - \alpha \nabla g(\theta_t)$$

$$\beta_{t+1} = \arg\max_{\beta} f(\theta_t, \beta)$$

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta f(\theta_t, \beta_{t+1})$$

$$\text{for } t = 1, 2, \ldots,$$

$$\widetilde{\beta}_0 = \beta_t$$

$$\text{for } i = 0, 1, \ldots, T-1$$

$$\widetilde{\beta}_{i+1} = \widetilde{\beta}_i + \alpha \nabla_\beta f(\theta_t, \widetilde{\beta}_i)$$

$$\beta_{t+1} = \widetilde{\beta}_T$$

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta f(\theta_t, \beta_{t+1})$$

Theorem: Designing T and step-size carefully, the algorithm converges to ϵ −stationarity in polynomial time.

Can be extended to stochastic setting and using mini-batches

Experiments: Sanjabi-Ba-Razaviyayn-Lee NeurIPS2018

Thank You!