CS 6501 Natural Language Processing

Optimization for Deep Learning

Yangfeng Ji

November 7, 2019

Department of Computer Science
University of Virginia

Overview

1. Stochastic Gradient Descent

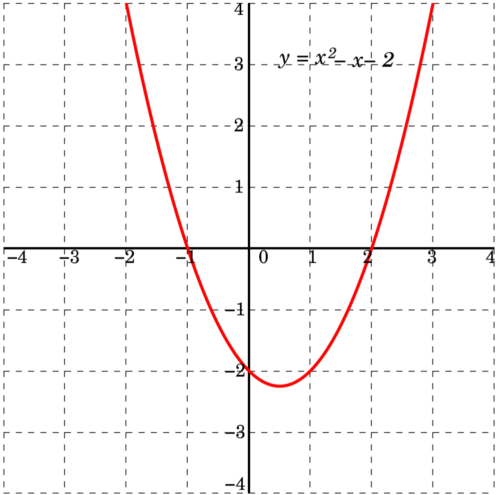
2. Adaptive Learning Rates

3. Other Tricks

4. Learning via Optimization

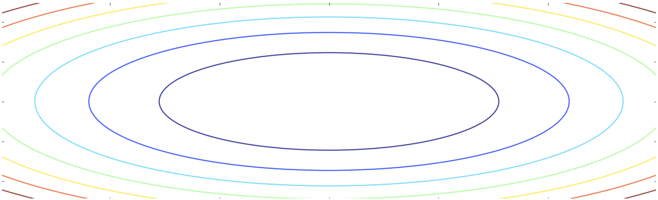
Stochastic Gradient Descent

Warmup: Gradient of a 1-D Function



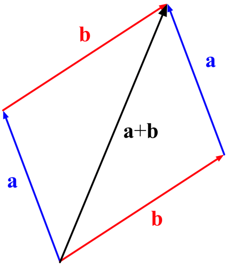
Warmup: Gradient of a 2-D Function

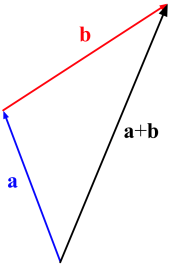
$$y = x_1^2 + 10x_2^2$$



Warmup: Vector Addition

The parallelogram law





Loss Function

Given a training set $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$, the loss function is defined as

$$\ell(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), y^{(i)}) \tag{1}$$

where $L(\cdot, \cdot)$ is the loss function for a single example and $\boldsymbol{\theta}$ denotes the parameters in f.

Some examples of $L(\cdot, \cdot)$

- Negative log-likelihood
- Cross entropy

SGD

To learn the parameter $\boldsymbol{\theta}$, we can compute the gradient with respect to one training example and then use stochastic gradient descent as

$$\boldsymbol{\theta}^{(t)} \leftarrow \boldsymbol{\theta}^{(t-1)} - \eta_t \cdot \boldsymbol{g}^{(t-1)} \tag{2}$$

where

- t: timestep
- $\boldsymbol{g}^{(t-1)} = \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t-1)})$ is the gradient of the single-example loss L
- η_t is the learning rate

Learning Rate

The usual conditions on the learning rates are

$$\sum_{t=1}^{\infty} \eta_t = \infty \tag{3}$$

$$\sum_{t=1}^{\infty} \eta_t^2 \leq \infty \tag{4}$$

A simplest function that satisfies these conditions is

$$\eta_t = \frac{1}{t} \tag{5}$$

[Bottou, 1998]

SGD with Momentum

Given the loss function $L(\boldsymbol{\theta})$ to be minimized, SGD with momentum is given by

$$\begin{aligned} \boldsymbol{v}^{(t)} &= \mu \boldsymbol{v}^{(t-1)} + \boldsymbol{g}^{(t-1)} \quad (6) \\ \boldsymbol{\theta}^{(t)} &= \boldsymbol{\theta}^{(t-1)} - \eta_t \boldsymbol{v}^{(t)} \quad (7) \end{aligned}$$

where

- η_t is still the learning rate
- $\mu \in [0, 1]$ is the momentum coefficient. Usually, $\mu = 0.99$ or 0.999.

Intuitive Explanation

The effect of momentum in SGD: reduce the fluctuation

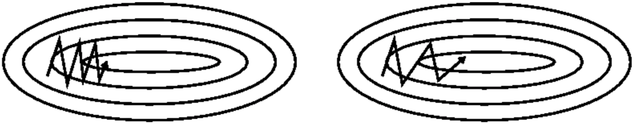


Figure: Left: SGD without momentum. Right: SGD with momentum.
(Credit: Genevieve B. Orr)

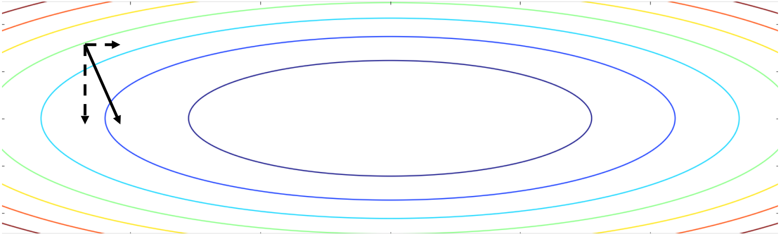
Note: the arrow show the opposite direction of the gradient

Another Example with Contour Plot

$$y = x_1^2 + 10x_2^2 \tag{8}$$

$$\frac{\partial y}{\partial x_1} = 2x_1 \tag{9}$$

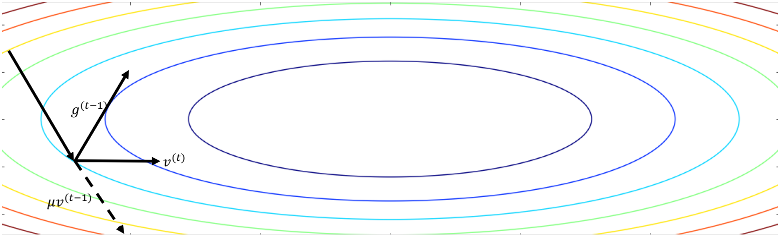
$$\frac{\partial y}{\partial x_2} = 20x_2 \tag{10}$$



Note: the arrow show the opposite direction of the gradient

Another Example with Contour Plot (Cont.)

$$v^{(t)} = \mu v^{(t-1)} + g^{(t-1)} \tag{11}$$



Note: the arrow show the opposite direction of the gradient

Adaptive Learning Rates

Basic Idea

For neural networks, the motivation of picking a different learning rate for each θ_k (the k-th component of parameter θ) is not new [LeCun et al., 2012] (the article was originally published in 1998).

- The basic idea is to make sure that all θ_k's converge roughly at the *same* speed.
- Depending on the *curvature* of the error surface, some θ_k's may require a small learning rate in order to avoid divergence, while others may require a large learning rate in order to converge fast.

AdaGrad

The basic idea of **AdaGrad** is to modify the learning rate η for $\boldsymbol{\theta}_k$ by using the history of $\partial_{\boldsymbol{\theta}_k} L$

$$\boldsymbol{\theta}_k^{(t)} = \boldsymbol{\theta}_k^{(t-1)} - \frac{\eta_0}{\sqrt{G_{k,k}^{(t-1)} + \epsilon}} \boldsymbol{g}_k^{(t-1)} \qquad (12)$$

AdaGrad

The basic idea of **AdaGrad** is to modify the learning rate η for $\boldsymbol{\theta}_k$ by using the history of $\partial_{\boldsymbol{\theta}_k} L$

$$\boldsymbol{\theta}_k^{(t)} = \boldsymbol{\theta}_k^{(t-1)} - \frac{\eta_0}{\sqrt{G_{k,k}^{(t-1)} + \epsilon}} \boldsymbol{g}_k^{(t-1)} \tag{12}$$

where

- $\boldsymbol{g}_k^{(t-1)} = [\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t-1)})]_k$ is the k-th component of $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t-1)})$

AdaGrad

The basic idea of **AdaGrad** is to modify the learning rate η for $\boldsymbol{\theta}_k$ by using the history of $\partial_{\boldsymbol{\theta}_k} L$

$$\boldsymbol{\theta}_k^{(t)} = \boldsymbol{\theta}_k^{(t-1)} - \frac{\eta_0}{\sqrt{G_{k,k}^{(t-1)} + \epsilon}} \boldsymbol{g}_k^{(t-1)} \tag{12}$$

where

- $\boldsymbol{g}_k^{(t-1)} = [\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t-1)})]_k$ is the k-th component of $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t-1)})$
- $G_{k,k}^{(t-1)} = \sum_{i=1}^{t-1} (\boldsymbol{g}_k^{(i)})^2$

AdaGrad

The basic idea of **AdaGrad** is to modify the learning rate η for $\boldsymbol{\theta}_k$ by using the history of $\partial_{\boldsymbol{\theta}_k} L$

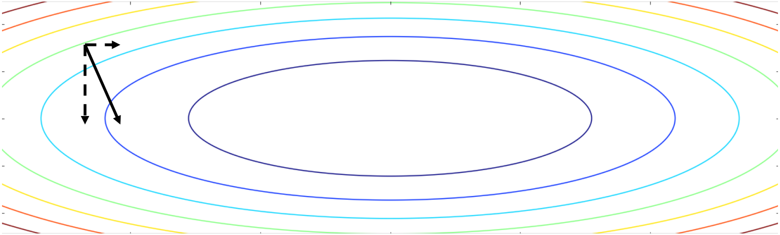
$$\boldsymbol{\theta}_k^{(t)} = \boldsymbol{\theta}_k^{(t-1)} - \frac{\eta_0}{\sqrt{G_{k,k}^{(t-1)} + \epsilon}} \boldsymbol{g}_k^{(t-1)} \tag{12}$$

where

- $\boldsymbol{g}_k^{(t-1)} = [\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t-1)})]_k$ is the k-th component of $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t-1)})$
- $G_{k,k}^{(t-1)} = \sum_{i=1}^{t-1} (\boldsymbol{g}_k^{(i)})^2$
- η_0 is the initial learning rate
- ϵ is a smoothing parameter usually with order 10^{-6}

AdaGrad: Intuitive Explanation

$$\theta_k^{(t)} = \theta_k^{(t-1)} - \frac{\eta_0}{\sqrt{G_{k,k}^{(t-1)} + \epsilon}} g_k^{(t-1)} \tag{13}$$



RMSProp

RMSProp uses a moving average over the past gradients

$$\boldsymbol{\theta}_k^{(t)} = \boldsymbol{\theta}_k^{(t-1)} - \frac{\eta_0}{\sqrt{\boldsymbol{r}_k^{(t)} + \epsilon}} \boldsymbol{g}_k^{(t-1)} \tag{14}$$

where

$$\boldsymbol{r}_k^{(t)} = \rho \boldsymbol{r}_k^{(t-1)} + (1 - \rho)[\boldsymbol{g}_k^{(t-1)}]^2 \tag{15}$$

and $\rho \in (0, 1)$

[Hinton et al., 2012]

Adam

$$
\begin{aligned}
\boldsymbol{v}_k^{(t)} &= \mu \boldsymbol{v}_k^{(t-1)} + (1-\mu)\boldsymbol{g}_k^{(t-1)} && (16)\\
\boldsymbol{r}_k^{(t)} &= \rho \boldsymbol{r}_k^{(t-1)} + (1-\rho)[\boldsymbol{g}_k^{(t-1)}]^2 && (17)\\
\hat{\boldsymbol{v}}_k^{(t)} &= \frac{\boldsymbol{v}_k^{(t)}}{1-\mu^t} && (18)\\
\hat{\boldsymbol{r}}_k^{(t)} &= \frac{\boldsymbol{r}_k^{(t)}}{1-\rho^t} && (19)\\
\boldsymbol{\theta}_k^{(t)} &= \boldsymbol{\theta}_k^{(t-1)} - \eta_0 \frac{\hat{\boldsymbol{v}}_k^{(t)}}{\sqrt{\hat{\boldsymbol{r}}_k^{(t)} + \epsilon}} && (20)
\end{aligned}
$$

The default values of μ and ρ are 0.9 and 0.999 respectively.

How to Choose a Optimization Algorithm?

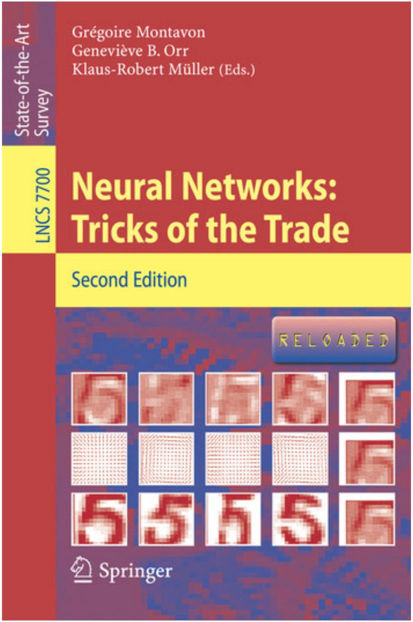
- User's familiarity with the algorithms [Goodfellow et al., 2016]
- Start from SGD first (my opinion)

Other Tricks

Other Tricks

1. Shuffle training examples in each epoch
2. Normalize inputs
3. Initialization

Further Reference



Learning via Optimization

Expected Loss

For a distribution of $\mathcal{D}$ over $(\boldsymbol{x}, y)$, where $\boldsymbol{x}$ denotes the input and y is the corresponding output/label.

The ideal prediction function is the one that minimize the expected loss $E(f) = \int_{\mathcal{D}} L(f(\boldsymbol{x}), y)$

$$f^* = \operatorname*{argmin}_f E(f) \tag{21}$$

where $L(\cdot, \cdot)$ is the loss function.

However, $\mathcal{D}$ is unknown.

Empirical Loss

Instead of minimizing the expected loss in Equation 21, which is also impossible, we can minimize the empirical loss

$$E_n(f) = \frac{1}{n} \sum_{i=1}^{n} L(f(\boldsymbol{x}^{(i)}), y^{(i)}), \tag{22}$$

as

$$f_n = \operatorname*{argmin}_f E_n(f) \tag{23}$$

where $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{n} \sim \mathcal{D}$ is the training set.

Question

How far from f_n to f^*

Question

How far from f_n to f^*

Three steps in machine learning

1. collect data
2. design a model
3. optimize an objective function

Question

How far from f_n to f^*

Three steps in machine learning

1. collect data: *Do we have enough data?*
2. design a model: *Does the model have enough data?*
3. optimize an objective function: *Is the objective fully optimized?*

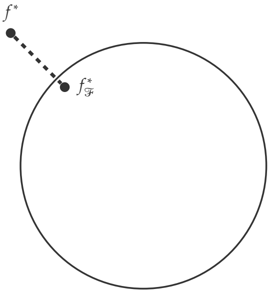
Errors (the difference between f_n and f^*) can be introduced in any of these three steps.

Approximation Error

Due to the model design: since we do not know the actual f^*, our starting point with all learnable function class $\mathcal{F}$ is predefined, such as logistic regression models or neural network models

$$f^* = \operatorname*{argmin}_{f} E(f) \tag{24}$$

$$f^*_{\mathcal{F}} = \operatorname*{argmin}_{f \in \mathcal{F}} E(f) \tag{25}$$



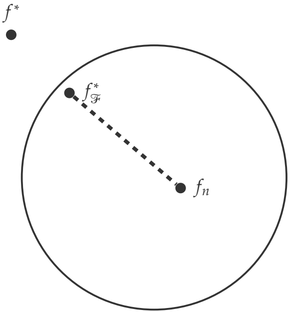
[Kearns et al., 1994, Agnostic ML]

Estimation Error

Due to the data collection: we can only have finite number of training examples and we have no idea about the real data distribution $\mathcal{D}$

$$f^*_{\mathcal{F}} = \operatorname*{argmin}_{f\in\mathcal{F}} E(f) \quad (26)$$

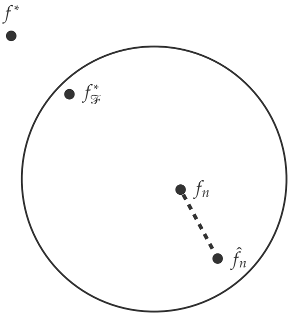
$$f_n = \operatorname*{argmin}_{f\in\mathcal{F}} E_n(f) \quad (27)$$



Optimization Error

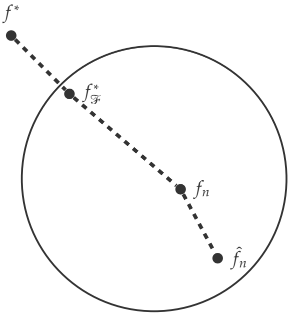
Due to the limited power of optimization methods

$$f_n = \operatorname{argmin} f \in \mathcal{F} E_n(f) \tag{28}$$



Error Decomposition

$$\underbrace{E[E(f^*_{\mathcal{F}}) - E(f^*)]}_{\text{approximation error}} + \underbrace{E[E(f_n) - E(f^*_{\mathcal{F}})]}_{\text{estimation error}} + \underbrace{E[E(\hat{f}_n) - E(f_n)]}_{\text{optimization error}}$$

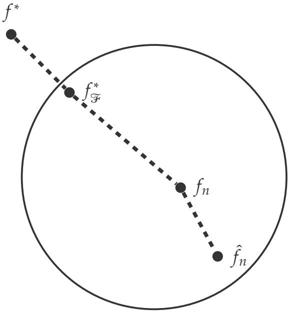


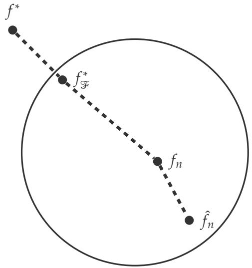
[Bottou, 2012, Sec. 18.3.1]

For a given machine learning problem,

- there is no way to know the oracle function f^* or $f^*_{\mathcal{F}}$, and
- it is difficult to get f_n, especially when $\mathcal{F}$ is a collection of deep neural networks

But it is useful to think about this decomposition.





For example, in the context of neural network learning,

- to reduce the approximation error is the motivation to design your neural network model carefully;
- to reduce the estimation error is the reason we should have enough data;
- to reduce the optimization error is why we need to know the optimization algorithms.

Summary

1. Stochastic Gradient Descent
2. Adaptive Learning Rates
3. Other Tricks
4. Learning via Optimization

Reference

Bottou, L. (1998).
Online learning and stochastic approximations.
On-line learning in neural networks, 17(9):142.

Bottou, L. (2012).
Stochastic gradient descent tricks.
In *Neural networks: Tricks of the trade*, pages 421–436. Springer.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016).
Deep Learning, volume 1.
MIT press Cambridge.

Hinton, G., Srivastava, N., and Swersky, K. (2012).
Neural networks for machine learning lecture 6a overview of mini-batch gradient descent.

Kearns, M. J., Schapire, R. E., and Sellie, L. M. (1994).
Toward efficient agnostic learning.
Machine Learning, 17(2-3):115–141.

LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. (2012).
Efficient backprop.
In *Neural networks: Tricks of the trade*, pages 9–48. Springer.

Reddi, S. J., Kale, S., and Kumar, S. (2018).
On the convergence of adam and beyond.
In *ICLR*.