Driving Neural Networks: An Exploration of Optimizers in Deep Learning & Ramblings in Optimization

Wang MA

23 Summer Study - Week6

maw2020@mail.sustech.edu.cn

July 27, 2023

Overview

Gradient Descent Methods

We consider the following non-constrained optimization problem:

$$\min_{x \in \mathbb{R}^n} f(x), \tag{1}$$

where $f(x) : \mathbb{R}^n \to \mathbb{R}$.

For a smooth function $f(x)$, at x^k, we want choose a good direction d^k as the descent direction. Notice that for $f(x^{k+1}) = f(x^k + \alpha d^k)$, we have its Taylor Expansion

$$f(x^{k+1}) = f(x^k) + \alpha \nabla f(x^k)^T d^k + O(\alpha^2 ||d^k||^2). \tag{2}$$

By Cauchy Inequality, choosing $d^k = -\nabla f(x^k)$ makes the function decrease the fastest. So the recursive equation for GD is:

$$x^{k+1} = x^k - \alpha_k \nabla f(x^k). \tag{3}$$

Simple Neural Network (NN)

- Fully Connected Network/ Dense Network
- Multilayer-Perceptron(MLP)

A Simple Neural Network mainly has following components:

- Input Layer: $x_1, x_2, ...$;
- Hidden Layer (Linear Transformation): $h_j = f(x) = W_j x + b_j$;
- Activation Function (Nonlinear Transformation): $y_k = \sigma_k(W_k h + b_k)$;
- Output Layer: y_k represents some scores or real-valued target.

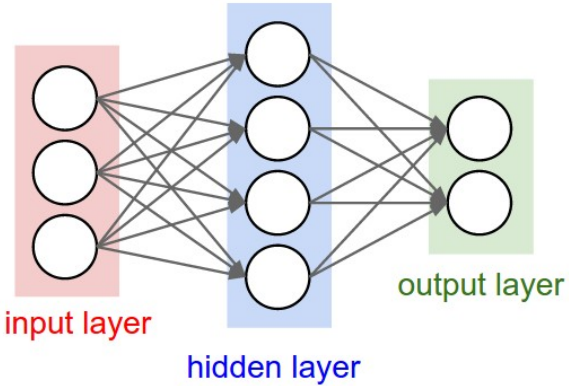

Image Source: CS231n, Stanford

Backpropagation

- Training a Neural Network can be seen as a **function-fitting** process. Given inputs $\{x_i\}_{i=1}^n$ with their corresponding labels $\{y_i\}_{i=1}^n$, we want to find a function f_ω that fits best on $f_\omega(x_i) = y_i$, where ω is a group of parameters we want to optimize.
- An intuitive **Loss** (objective function): $L = ||f_\omega(x_i) - y_i||^2$ (to minimize over ω)
- We apply **Gradient Descent Methods** on the Loss function, which needs to calculate the gradients of L over all the parameters ω, $\frac{\partial L}{\partial \omega}$
- Multi layers networks means function compositions, here we employ the **Chain Rule** to calculate gradients across layers.
- **Backpropagation** performs a backward pass to adjust the model's parameters. (from wiki)

An Example of Backpropagation on 3-layer Network

Optimization Problem in Supervised Machine Learning

- Assume $(x, y) \sim P$, where x is input, y is the corresponding label.
- Task: given x, and predict y, that is, we want to find an optimal function ϕ to minimize $\mathbb{E}[L(\phi(a), b)]$, where L is the loss function.
- We do not know the real distribution P, only have a dataset:

$$\mathcal{D} = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$$

- Then the overall problem is

$$\min_x \frac{1}{N} \sum_{i=1}^{N} L(\phi_\omega(x_i), y_i). \tag{4}$$

In later discussion, we will only consider the following stochastic optimization problem:

$$\min_{x \in \mathbb{R}^n} f(x) \hat{=} \frac{1}{N} f_i(x), \tag{5}$$

where $f_i(x)$ represents the loss function of i-th sample.

Stochastic Gradient Descent

Assume every $f_i(x)$ is convex and differentiable, then we apply GD on Eq.5, and obtain

$$x^{k+1} = x^k - \alpha_k \nabla f(x^K), \tag{6}$$

where $\nabla f(x^k) = \frac{1}{N} \sum_{i=1}^{N} \nabla f_i(x^k)$. To compute this gradient, we need to calculate all the $\nabla f_i(x^k), i = 1, 2, ..., N$ and then add them up. However, in Machine Learning tasks, we have enormous samples in dataset, so it's very **Expensive** to get the exact $\nabla f(x^k)$.

SGD recursive form

The basic recursive form for SGD is

$$x^{k+1} = x^k - \alpha_k \nabla f_{s_k}(x^k), \tag{7}$$

where s_k is equally sampled from $1, 2, ..., N$, and α_k is the stepszie.

Mini-batch SGD

Remark. $\mathbb{E}_{s_k}[\nabla f_{s_k}(x^k)] = \nabla f(x^k)$.

However, only picking one sample each time makes the training unstable, the common way is to use SGD with **mini-batch**. We pick a small set $\mathcal{I}_k \subset 1, 2, ..., N$, and operate

$$x^{k+1} = x^k - \frac{\alpha_k}{|\mathcal{I}_k|} \sum_{s \subset \mathcal{I}_k} \nabla f_s(x^k), \tag{8}$$

where $|\mathcal{I}_k|$ represents the number of the elements in $\mathcal{I}_k$.

SGD with Momentum

Above gradient based algorithms converge slowly, especially when the problem is ill-conditioned. To accelerate the training, here comes the **SGD with Momentum**.

SGD with Momentum

Momentum. $m^{k+1} = \mu_k m^k - \alpha_k \nabla f_{s_k}(x^k)$. (normally $1 \geq \mu_k \geq 0.5$)
Updating rule. $x^{k+1} = x^k + m^{k+1}$.

The idea of SGD with Momentum is to keep some of the previous descent direction, and adjust it with current gradient. This will provide the model with more stability and thus accelerate the training. Even more, this approach also have ability to avoid local optimal.

Comments on SGD with Momentum

Comments. Normally $\mu_k \in [0.5, 1]$, this means we have a large momentum during each step, much of he previous gradient information is kept. Every time we just take a litte adjustment on previous direction.

When many consecutive gradients point in the same direction, the step size becomes large, which also makes sense intuitively

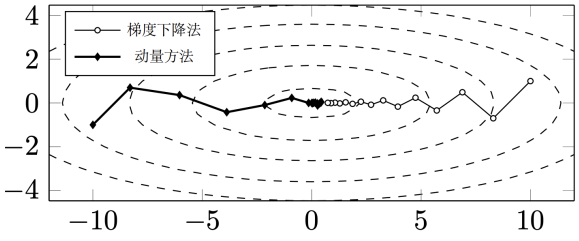

图 8.11　动量方法在海瑟矩阵病态条件下的表现

Image Source: OptBook, Zaiwen Wen, etc.

Nesterov Accelerated Gradient (NAG)

The Recursive Form of NAG

$y^{k+1} = x^k + \mu_k(x^k - x^{k-1})$,
$x^{k+1} = y^{k+1} - \alpha_k \nabla f_{s_k}(y^{k+1})$,
where $\mu_k = \frac{k-1}{k+2}$, α_k is the stepsize.

We can then rewrite the NAG alogroithm to the form of SGD with Momentum:

Rewrite NAG

$m^{k+1} = \mu_k m^k + \alpha_k \nabla f_{s_k}(x^k + \mu_k m^k)$,
$x^{k+1} = x^k + m^{k+1}$.

SGD with Momentum VS NAG

Different Momentum.

- SGD with Momentum: $m^{k+1} = \mu_k m^k + \alpha_k \nabla f_{s_k}(x^k)$
- NAG: $m^{k+1} = \mu_k m^k + \alpha_k \nabla f_{s_k}(x^k + \mu_k m^k)$

Same Updating. $x^{k+1} = x^k + m^{k+1}$

Comments.

- The main difference is the choice of gradient;
- NAG considers the gradient of future step (prediction), and then updates based on it. This strategy helps to reduce the oscillations caused by excessive momentum when reaching the minimum, thus improving the stability and speed of the optimization.

Motivation

- Parameter adjustment is a major difficulty, the quality of the parameters has a significant impact on the result, so we hope that the model can **automatically adjust the parameters** during training;
- The goal is to find the point where the gradient is 0, but the speed at which each direction of the gradient converges to 0 is different, and we need to consider it for each direction, e.g.
  - when the direction has large value, then we need small stepsize;
  - when the direction has small value, then we need large stepsize.

Let $g^k = \nabla f_{s_k}(x^k)$, here we introduce $G^k = \sum_{i=1}^{k} g^i \odot g^i$, sometimes it is called the second order momentum. This can record the accumulation of gradients in all directions during the update process

Adaptive Gradient Algorithms (AdaGrad)

The Recursive Form of AdaGrad

$x^{k+1} = x^k - \frac{\alpha}{\sqrt{G^k + \epsilon \mathbf{1}_n}} \odot g^k$,

$G^{k+1} = G^k + g^{k+1} \odot g^{k+1}$, where the subtraction in $\frac{\alpha}{\sqrt{G^k + \epsilon \mathbb{1}_n}}$ is element-wise. We introduce $\epsilon \mathbf{1}_n$ to avoid $G^k = 0$.

Remarks

- Pros: adaptive methods / different stepsizes in different direction / good theoretical property in convex problem
- Cons: monotonically decreasing stepsize / accumulating squared gradients from the beginning of training can lead to premature or excessive step size reductions

Root Mean Square Propagation (RMSProp)

Recall. Monotonically decreasing stepsize in Adagrad. Then the stepsize will become very small, which increase the cost on the computation.

Is is better to mostly consider the points that relatively near current point? Now we introduce

$$M^{k+1} = \rho M^k + (1-\rho) g^{k+1} \odot g^{k+1},$$

where ρ is decay rate and here is RMSProp:

The Recursive Form of RMSProp

$x^{k+1} = x^k - \frac{\alpha}{\sqrt{M^k + \epsilon \mathbf{1}_n}} \odot g^k,$
$M^{k+1} = \rho M^k + (1-\rho) g^{k+1} \odot g^{k+1},$
usually, we set $\rho = 0.9$ and $\alpha = 0.001$.

Adaptive Moment Estimation (Adam)

The Algorithm of Adam

Set $S^0 = 0, M^0 = 0$.

Momentum. $S^k = \rho_1 S^{k-1} + (1-\rho_1) g^k$;

2nd-order Momentum. $M^{k+1} = \rho_2 M^k + (1-\rho_2) g^{k+1} \odot g^{k+1}$;

Bias Correction.

$$\hat{S}^k = \frac{S^k}{1-\rho_1^k}, \quad \hat{M}^k = \frac{m^k}{1-\rho_2^k};$$

Update. $x^{k+1} = x^k - \frac{\alpha}{\sqrt{\hat{M}^k + \epsilon \mathbf{1}_n}} \odot \hat{S}^k$.

Usually we set $\rho_1 = 0.9, \rho_2 = 0.999, \alpha = 0.001$.

Remarks on Adam

- Pros of Adam
  - combines the advantages of two optimization algorithms, Momentum and RMSProp
  - adaptively adjust the learning rate of each parameter
  - bias correction
- Cons of Adam
  - the cons of adaptive learning rate (local minimal / self-regularization)
  - Adam requires more computing resources

Convexity & Smoothness

Newton's Method

Constrained Problem & KKT Conditions

Conclusion

- Backpropogation - the application of Chain Rule
- SGD & SGD with Mini-batch
- Momentum (SGD with Momentum, NAG)
- Adaptive Learning (AdaGrad, RMSProp)
- Adam