华南理工大学

South China University of Technology

The Experiment Report of *Deep Learning*

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Qi Chen

Supervisor:
Mingkui Tan

Student ID:
201720144924

Grade:
Graduate

December 14, 2017

Logistic Regression, Linear Classification and Stochastic Gradient Descent

***Abstract*—Stochastic gradient descent optimization algorithms, while increasingly popular, are often used as black-box optimizers. In this report, to compare the difference between various optimized methods, we discuss different variants of gradient descent in equations and experiments. Moreover, we try to provide some results intuitively with regard to the behavior of different methods in logistic regression and linear classification task for further discussion.**

I. Introduction

STOCHASTIC gradient descent is one of the most popular algorithms to perform optimization and by far the most common way to optimize neural networks. At the same time, every state-of-the-art Deep Learning library contains implementations of various algorithms to optimize gradient descent. These algorithms, however, are often used as black-box optimizers, as practical explanations of their strengths and weaknesses are hard to come by.

This report aims to explore the performance of different optimized algorithms by conducting some experiments in logistic regression and linear classification task. In the second part, we will first exhibit the loss function of logistic regression and linear classification as well as the derivative function. Moreover, we will also illustrate the equations of various optimized methods and compare the difference between each other. In the third *Experiments* part, for intuitively observing, we will provide some experimental results about the mentioned methods. In the finally part, we will draw some conclusions about the whole report.

The main purposes of this report can be concluded as the following:

- Further understand the process of logistic regression, linear classification and stochastic gradient descent.
- Conduct some experiments under small scale dataset to explore the performance of different optimized algorithm.
- Realize and analyze the process of optimization and adjusting parameters in different tasks.

II. Methods and Theory

In this section, we will give a complete introduction to these experiments including the chosen methods, the related theories, the related equations(loss function), the derivation process(taking the gradient), etc.

The loss function of logistic regression is:

$$L_{reg} = -\frac{1}{N}\sum_{i=1}^{N} y^{(i)} log(h(x^{(i)})) + (1-y^{(i)}) log(1-h(x^{(i)})), \tag{1}$$

where $h(x^{(i)}) = \frac{1}{1+e^{-w^T x}}$

The corresponding gradient with respect to weight in logistic regression:

$$\frac{\partial L_{reg}}{\partial w} = \frac{1}{N}\sum_{i=1}^{N}(h(x^{(i)}) - y^{(i)}) * x_j^{(i)} \tag{2}$$

The loss function of linear classification, e.g.Support Vector Machine (SVM):

$$L_{cls} = \frac{\|w\|^2}{2} + C\frac{1}{N}\sum_{i=1}^{N} max(0, 1 - y_i(w^T x_i + b)) \tag{3}$$

The corresponding gradient with respect to weight in Support Vector Machine (SVM):

$$X \in \mathbb{R}^{n\times d}, w \in \mathbb{R}^d, y \in \mathbb{R}^n, s \in \mathbb{R}^n \tag{4}$$

$$s_i = \begin{cases} 0, \ 1 - y_i(w^T x_i + b) \leq 0 \\ 1, \ 1 - y_i(w^T x_i + b) > 0 \end{cases} \tag{5}$$

$$\frac{\partial L_{cls}}{\partial w} = w + CX^T(y \odot s) \tag{6}$$

Stochastic Gradient Descent (SGD). Stochastic gradient descent (SGD) in contrast performs a parameter update for each training example $x^{(i)}$ and $y^{(i)}$:

$$\theta = \theta - \eta \nabla_\theta L(\theta; x^{(i)}; y^{(i)}) \tag{7}$$

Batch gradient descent performs redundant computations for large datasets, as it recomputes gradients for similar examples before each parameter update. SGD does away with this redundancy by performing one update at a time. It is therefore usually much faster and can also be used to learn online.

Mini-batch Gradient Descent. Mini-batch gradient descent finally takes the best of both worlds and performs an update for every mini-batch of n training examples:

$$\theta = \theta - \eta \nabla_\theta L(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \tag{8}$$

In this way, it can get some advantages as the following:

1) it can reduce the variance of the parameter updates, which can lead to more stable convergence
2) it can make use of highly optimized matrix optimizations common to state-of-the-art deep learning libraries that make computing the gradient w.r.t. a mini-batch very efficient.

Nesterov accelerated gradient (NAG). In the SGD optimized methodn, a ball that rolls down a hill, blindly following the slope, is highly unsatisfactory. We would like to have a smarter ball, a ball that has a notion of where it is going so that it knows to slow down before the hill slopes up again.

Nesterov accelerated gradient (NAG) is a way to give our momentum term this kind of prescience. We know that we will use our momentum term γv_{t-1} to move the parameters θ. Computing $\theta - \gamma v_{t-1}$ thus gives us an approximation of the next position of the parameters (the gradient is missing for the full update), a rough idea where our parameters are going to be. We can now effectively look ahead by calculating the gradient not w.r.t. to our current parameters θ but w.r.t. the approximate future position of our parameters:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_\theta L(\theta - \gamma v_{t-1}) \\ \theta &= \theta - v_t \end{aligned} \tag{9}$$

Again, we set the momentum term γ to a value of around 0.9. While Momentum first computes the current gradient and then takes a big jump in the direction of the updated accumulated gradient, NAG first makes a big jump in the direction of the previous accumulated gradient, measures the gradient and then makes a correction. This anticipatory update prevents us from going too fast and results in increased responsiveness, which has significantly increased the performance on a number of tasks. Now that we are able to adapt our updates to the slope of our error function and speed up SGD in turn, we would also like to adapt our updates to each individual parameter to perform larger or smaller updates depending on their importance.

Adadelta. Adadelta is an extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate. Instead of accumulating all past squared gradients, Adadelta restricts the window of accumulated past gradients to some fixed size w.

Instead of inefficiently storing w previous squared gradients, the sum of gradients is recursively defined as a decaying average of all past squared gradients. The running average g_t at time step t then depends (as a fraction γ similarly to the Momentum term) only on the previous average and the current gradient. The steps of updating in the following:

$$\begin{aligned} g_t &= \nabla L(\theta_{t-1}) \\ G_t &= \gamma G_t + (1-\gamma) g_t \odot g_t \\ \Delta\theta_t &= -\frac{\sqrt{\Delta} + \epsilon}{\sqrt{G_t + \epsilon}} \odot g_t \\ \theta_t &= \theta_{t-1} + \Delta\theta_t \\ \Delta_t &= \gamma \Delta_{t-1} + (1-\gamma) \Delta\theta_t \odot \Delta\theta_t \end{aligned} \tag{10}$$

With Adadelta, we do not even need to set a default learning rate, as it has been eliminated from the update rule.

RMSProp. RMSprop is an unpublished, adaptive learning rate method proposed by Geoff Hinton in Lecture 6e of his Coursera Class.

RMSprop and Adadelta have both been developed independently around the same time stemming from the need to resolve Adagrads radically diminishing learning rates. RMSprop in fact is identical to the first update vector of Adadelta that we derived above:

$$\begin{aligned} g_t &= \nabla L(\theta_{t-1}) \\ G_t &= \gamma G_t + (1-\gamma) g_t \odot g_t \\ \theta_t &= \theta_{t-1} - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t \end{aligned} \tag{11}$$

RMSprop as well divides the learning rate by an exponentially decaying average of squared gradients. Hinton suggests γ to be set to 0.9, while a good default value for the learning rate η is 0.001.

Adaptive Moment Estimation (Adam). Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients v_t like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients m_t, similar to momentum. We show the updating steps as the following:

$$\begin{aligned} g_t &= \nabla L(\theta_{t-1}) \\ m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ G_t &= \gamma G_t + (1-\gamma) g_t \odot g_t \\ \alpha &= \eta \frac{\sqrt{1-\gamma^t}}{1-\beta^t} \\ \theta_t &= \theta_{t-1} - \alpha \frac{m_t}{\sqrt{G_t + \epsilon}} \end{aligned} \tag{12}$$

The authors propose default values of 0.9 for β_1, 0.999 for γ, and 10^{-8} for ϵ. They show empirically that Adam works well in practice and compares favorably to other adaptive learning-method algorithms.

III. Experiments

In the this section, because we always call mini-batch gradient descent as *SGD* in deep learning research, for convenient, we use *SGD* to represent mini-batch gradient descent as what we do in deep learning papers.

A. Dataset

Experiment uses a9a of LIBSVM Data, including 32561/16281(testing) samples and each sample has 123/123 (testing) features.

B. Implementation

The steps of our experiments are as the following:

1) Load the training set and validation set of a9a.
2) Initialize logistic regression or SVM model parameters with normal distribution.
3) Define the loss function and calculate its derivation.
4) Compute the gradient with respect to the weight using different optimized method (SGD, NAG, RMSProp, AdaDelta and Adam).
5) Using gradient descent to update the weight.
6) Repeat step (4) and (5) for several times until convergence.

C. Experimental Results

In this section, we conduct several experiments to compare the performance using different optimized algorithms in both logistic regression and linear classification task.

In logistic regression task, we first discuss the impact with different value of learning rate η in the same optimized algorithm. By visualized as a figure, we try to get a proper value of learning rate and analyze the causes of different results.

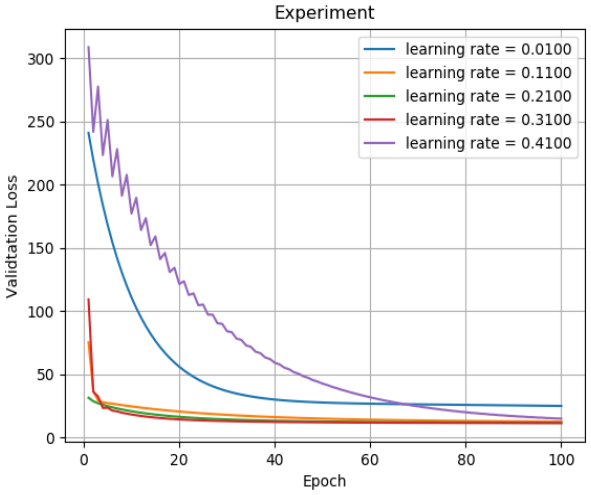

Fig. 1. We use Stochastic Gradient Descent (SGD) to optimize the logistic regression task with various learning rate ([0.01, 0.11, 0.21, 0.32, 0.41]).

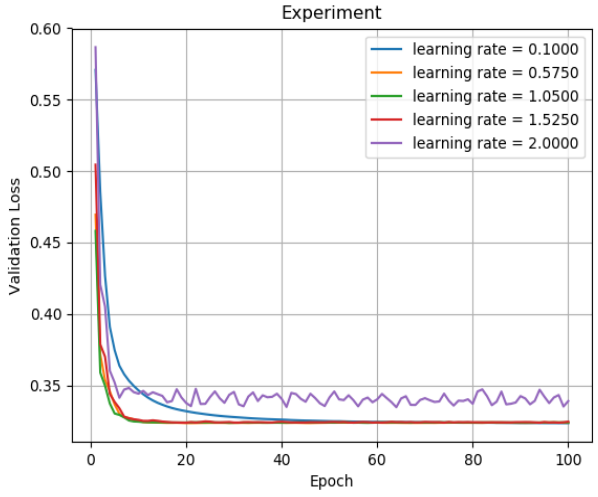

Fig. 2. We use Nesterov accelerated gradient (NAG) to optimize the logistic regression task with various learning rate ([0.1, 0.575, 1.05, 1.525, 2.0]).

In figure 1 2 3 5, we can obverse that during the learning rate increasing, the validation loss decrease progressively. However, when the learning rate is out of a boundary, the loss will be vibrating and can not achieve the best performance. Otherwise, in figure 4, we only draw one curve due to the needlessness of learning rate.

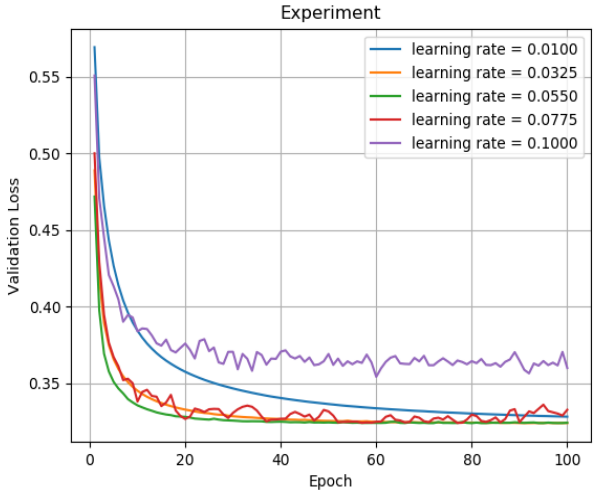

Fig. 3. We use RMSProp to optimize the logistic regression task with various learning rate ([0.01, 0.0325, 0.055, 0.0775, 0.1]).

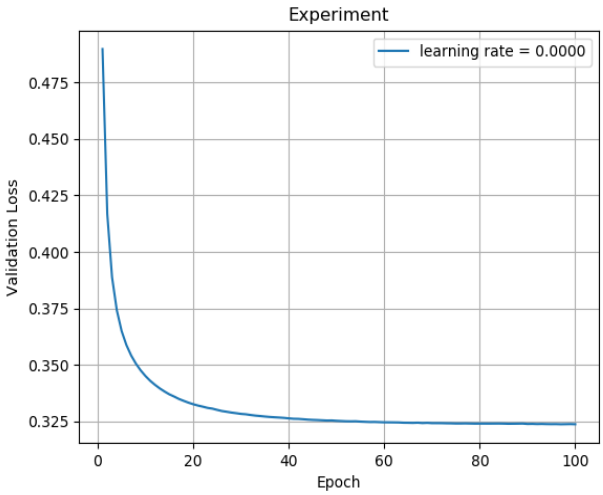

Fig. 4. We only draw one curve in this figure since it is unnecessary for the optimized method Adadelta to indicate a specific learning rate.

For further comparsion, we conduct an experiment to compare the performance of convergence intuitively. All of the methods use the same learning rate 0.01. Shown in figure 6, we can observe that all the variants of SGD are able to converge faster than the original one. That means the improved optimized methods outperform the original one in the logistic regression task by using their own tricks.

For the linear classification task, we use the uniform distribution from 0 to 1 to initialize weights and bias. And then we also compare the performance of different optimized methods in this task by visulizing the validation loss curve and accuracy curve. All of the settings in these method are the same. We choose $learningrate = 0.001$, $batchsize = 10000$ and hyper-parameter $C = 2^{-1}$. Every methods iterate 100 times in practical implement.

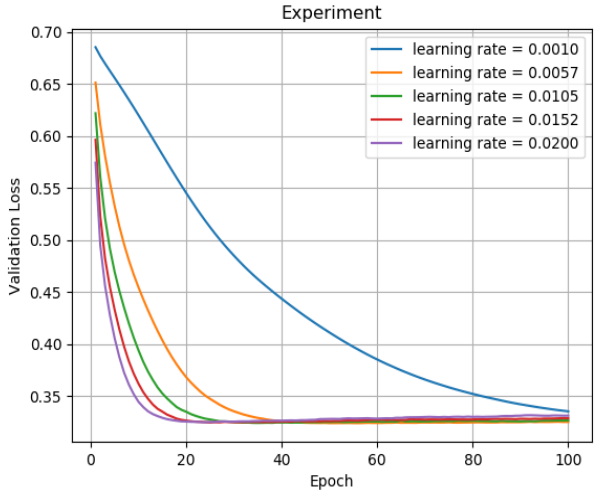

Fig. 5. We use Adam to optimize the logistic regression task with various learning rate. ([0.001, 0.0057, 0.0105, 0.0152, 0.02])

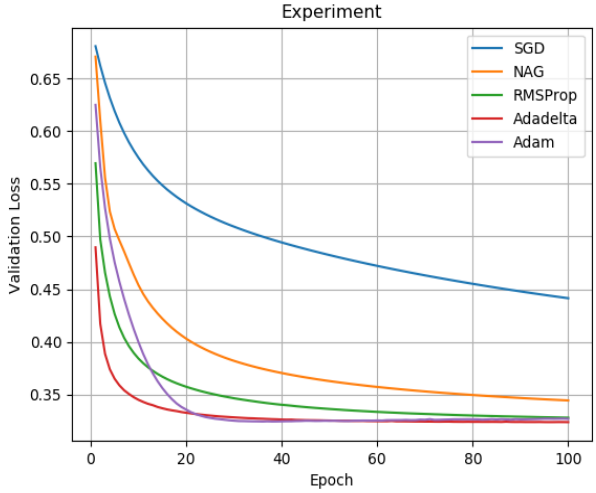

Fig. 6. To compare the performance of different optimized methods, we conduct an experiments to visualize the loss curve of these algorithm with the same settings.

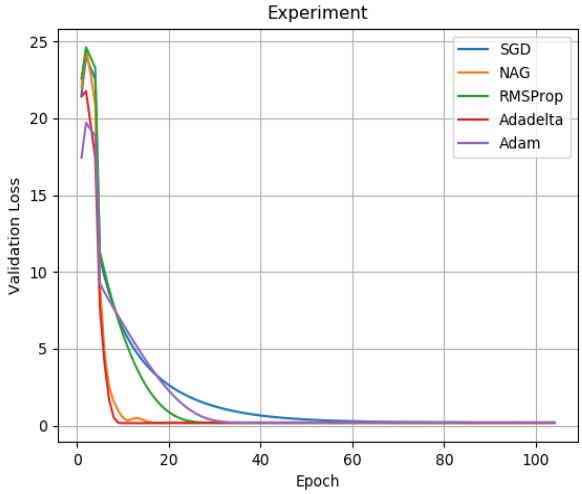

Fig. 7. To compare the performance of different optimized methods in the linear classification problem, we conduct an experiments to visualize the validation loss curve of these algorithm with the same settings.

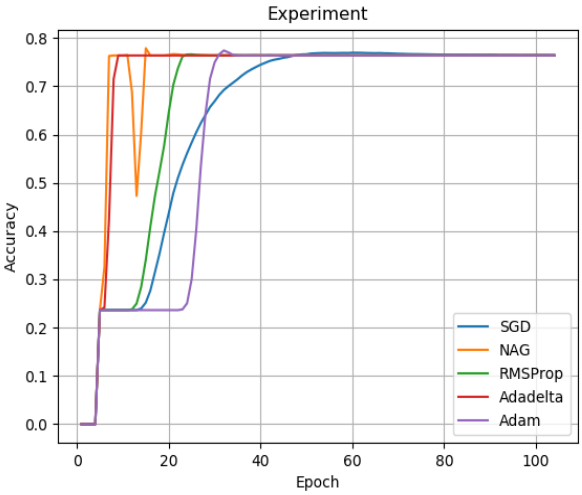

Fig. 8. To compare the performance of different optimized methods in the linear classification problem, we conduct an experiments to visualize the accuracy curve of these algorithm with the same settings.

In the figure 7, we use 0.01 as the learning rate in all the methods except Adadelta. As we can see, NAG and Adadelta can achieve the best performance of convergence by using linear SVM classifier. Although Adam and RMSProp can not obtain the fastest speed of convergence, they can also bit the SGD optimized algorithm in the same settings.

Besides, we also conduct an experiment about classification accuracy in a9a dataset. As shown in the figure 8, the accuracies of this 5 algorithm is similar. But the accuracy curve of Adadelta is most smooth and increases fastly. For NAG method, the curve is not smooth, although it also has a high speed to achieve the best accuracy. For Adam and RMSProp, they can take the same accuracy with others, but they are neither fastest ones nor slowest ones. Of course, the slowest one is SGD.

IV. Conclusion

In this reprot, we mentioned some variants of SGD algorithm, which aims to compare the difference between the original one and each other. For esaier to understand, we try to conduct some experiments and visualize the results of these optimized methods in two tasks logistic regression and linear classification. The results show that in the field of convergence, all the variants of SGD outperform the original one easily.