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:
Yihan Zheng

Supervisor:
Mingkui Tan

Student ID:
201720145105

Grade:
Graduate

December 15, 2017

Logistic Regression, Linear Classification and Stochastic Gradient Descent

***Abstract*—To train the models of Logistic Regression and Linear Classification, we commonly use mini-batch stochastic gradient descent to optimize the parameters. In this report, we discuss four different variants of SGD in equations: NAG, RMSProp, AdaDelta and Adam. By doing experiments in regression and classification tasks respectively, we compare the performances of different optimized methods.**

I. Introduction

THIS report aims to compare the difference between four various optimized methods, which are the variants of stochastic gradient descent. We explore the performance of different optimized algorithms by conducting some experiments in logistic regression and linear classification task.

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.

This report contains four parts. In the second part, we firstly introduce the principles of logistic regression and linear classification in detail. Then we illustrate the equations of various optimized methods and compare the differences. The third part exhibits the results of four different optimized methods in regression and classification tasks. Finally,we draw some conclusions of the report.

II. Methods and Theory

In this section, we will introduce the principles of logistic regression and linear classification in detail. In addition, we will illustrate the equations of four various optimized methods and compare the differences.

A. Logistic regression

The logistic regression is a non-linear model which can simulate the mapping between input X and output y. The logistic function is defined as follows:

$$h_W(x_i) = \frac{1}{\left(1 + e^{-W^T x_i}\right)}. \tag{1}$$

The loss function of logistic regression is:

$$L = \frac{1}{n}\sum_{i=1}^{n} log\left(1 + e^{-y_i W^T x_i}\right) + \frac{\lambda}{2}||W||^2, \tag{2}$$

where $y_i \in \{-1, 1\}$. The corresponding gradient with respect to weight in logistic regression:

$$G = \frac{1}{n}\sum_{i=1}^{n} \frac{-y_i x_i}{1 + e^{y_i W^T x_i}} + \lambda W. \tag{3}$$

B. Linear Classification(SVM)

SVM learns a representation of the examples as points in space, mapping the examples of the separate categories divided by a clear gap. The linear fuction is defined as follows:

$$h_{W,b}(x_i) = W^T x_i + b. \tag{4}$$

And the loss function is:

$$L = \frac{\lambda}{2}||W||^2 + \frac{1}{n}\sum_{i=1}^{n} max\left(0, 1 - y_i(W^T x_i + b)\right). \tag{5}$$

The corresponding gradient with respect to the parameters:

$$G_W = \begin{cases} \lambda W & y_i(W^T x_i + b) \geq 1 \\ \lambda W + \frac{1}{n}\sum_{i=1}^{n} -y_i x_i & y_i(W^T x_i + b) < 1, \end{cases} \tag{6}$$

$$G_b = \begin{cases} 0 & y_i(W^T x_i + b) \geq 1 \\ \frac{1}{n}\sum_{i=1}^{n} -y_i & y_i(W^T x_i + b) < 1. \end{cases} \tag{7}$$

C. Stochastic Gradient Descent

Stochastic gradient descent(SGD). Stochastic gradient descent performs a parameter update for each training example x_i and y_i:

$$W = W - \eta G_i, \tag{8}$$

where η is the learning rate, which is pre-defined.

Gradient descent performs redundant computations for large datasets, as it recomputes gradients for all examples before each parameter update. But SGD reduces 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:

$$W = W - \sum_{i \in batch} \eta G_i, \tag{9}$$

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

- Compared to SGD, it reduces the number of the parameter updates in each epoch and lead to more stable convergence.
- Compared to gradient descent, it makes use of part of training examples, which makes the computation of the gradient 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 W. Computing $W - \gamma v_{t-1}$ 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 W but w.r.t. the approximate future position of our parameters:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla_W J(W - \gamma v_{t-1}), \\ W &= W - v_t, \end{aligned} \tag{10}$$

where η is the learning rate. γ is the momentum. In addition, 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:

$$\begin{aligned} \Delta W_t &= -\frac{RMS\left[\Delta W\right]_{t-1}}{RMS\left[g\right]_t} g_t, \\ W_{t+1} &= W_t + \Delta W_t, \end{aligned} \tag{11}$$

where

$$\begin{aligned} RMS\left[g\right]_t &= \sqrt{E\left[g^2\right]_t + \epsilon}, \\ E\left[g^2\right]_t &= \gamma E\left[g^2\right]_{t-1} + (1-\gamma)\, g_t^2 \end{aligned} \tag{12}$$

and

$$\begin{aligned} RMS\left[\Delta W\right]_t &= \sqrt{E\left[\Delta W^2\right]_t + \epsilon}, \\ E\left[\Delta W^2\right]_t &= \gamma E\left[\Delta W^2\right]_{t-1} + (1-\gamma)\, \Delta W_t^2, \\ \Delta W_t &= -\frac{\eta}{RMS\left[g\right]_t} g_t. \end{aligned} \tag{13}$$

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.

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:

$$\begin{aligned} E\left[g^2\right]_t &= 0.9E\left[g^2\right]_{t-1} + 0.1g_t^2, \\ W_{t+1} &= W_t - \frac{\eta}{\sqrt{E\left[g^2\right]_t + \epsilon}} g_t, \end{aligned} \tag{14}$$

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:

$$W_{t+1} = W_t - \frac{\eta}{\sqrt{\hat{v_t}} + \epsilon} \hat{m_t}, \tag{15}$$

where

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)\, g_t, \\ \hat{m_t} &= \frac{m_t}{1-\beta_1} \end{aligned} \tag{16}$$

and

$$\begin{aligned} v_t &= \beta_2 v_{t-1} + (1-\beta_2)\, g_t^2, \\ \hat{v_t} &= \frac{v_t}{1-\beta_2}. \end{aligned} \tag{17}$$

The authors propose default values of 0:9 for β_1 and β_2, 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 this section, we firstly introduce the dataset using in this experiment. And then we illustrate the steps of the implementation. Finally, we exhibit the results of four different optimized methods in Logistic Regression and Linear Classification.

A. Dataset

The experiment uses a9a of LIBSVM Data, including 32561 training samples and 16281 validation samples. The feature dimension of each sample in both training set and validation set is 123.

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 all experiment in this report, we use mini-batch gradient descent with batch size= 128, which performs an update for every mini-batch of training examples.

In logistic regression task, we use the uniform distribution from 0 to 1 to initialize weights and bias. For each optimized method, we discuss the impact with different value of learning rate η. By visualized as a figure, we try to get a proper value of learning rate and analyze the causes of different results.

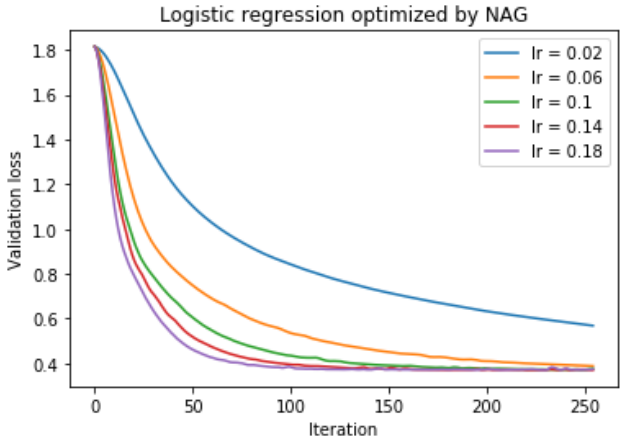

Fig. 1. Logistic regression optimized by Nesterov accelerated gradient (NAG) with various learning rate ([0.02, 0.06, 0.1, 0.14, 0.18]).

In figure 1, 2, 3 and 4, we exhibit the results of four different optimized methods respectively. We can obverse that during the learning rate increasing, the validation loss decrease progressively in all methods. However, when the learning rate is out of a boundary, the loss curve is oscillating and can not achieve the best performance.

In this way, we can get a proper value of learning rate in four methods. For NAG, RMSProp, AdaDelta and Adam, the proper values of learning rate are set to 0.18, 0.06, 0.07 and 0.08 respectively.

In figure 5, we compare the performances of four different optimized methods, which are in their own proper values of learning rate. We can obverse that four methods can converge

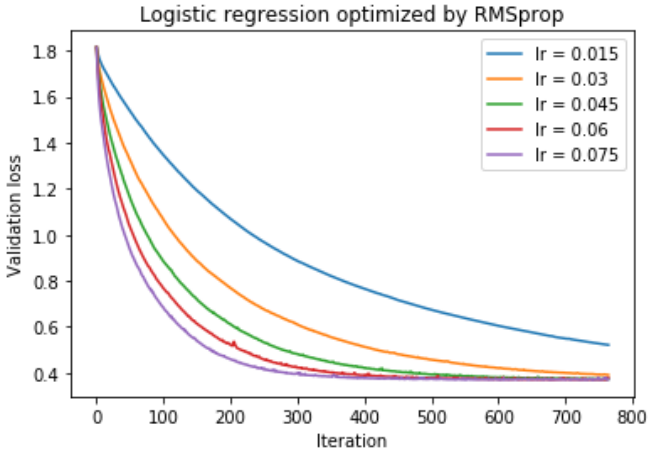

Fig. 2. Logistic regression optimized by RMSProp with various learning rate ([0.015, 0.03, 0.045, 0.06, 0.075]).

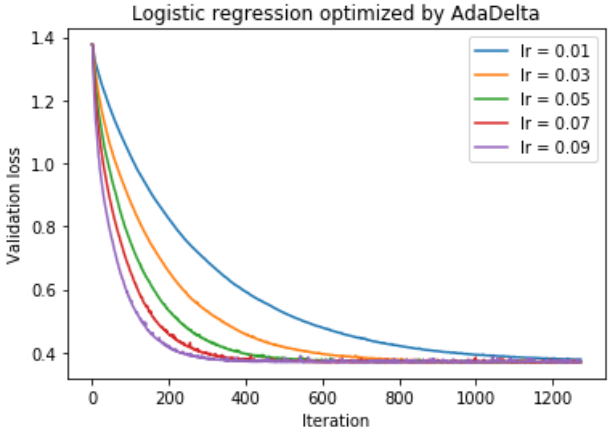

Fig. 3. Logistic regression optimized by Adadelta with various learning rate ([0.01, 0.03, 0.05, 0.07, 0.09]).

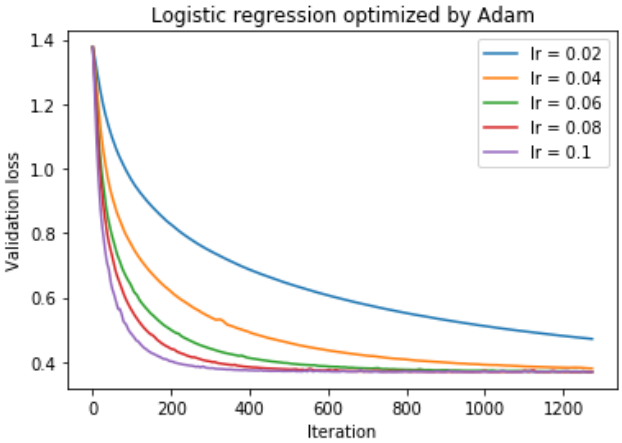

Fig. 4. Logistic regression optimized by Adaptive Moment Estimation (Adam) with various learning rate ([0.02, 0.04, 0.06, 0.08, 0.1]).

to similar results, but the convergence rate are different NAG

$>$ Adam $>$ AdaDelta $>$ RMSProp.

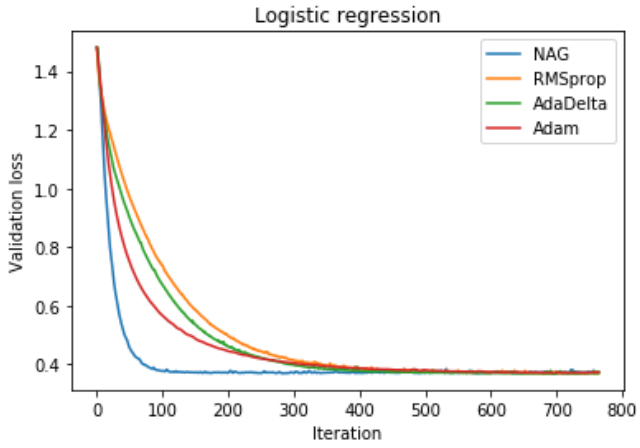

Fig. 5. Logistic regression optimized by four optimized methods.

For the linear classification task, we use the uniform distribution from 0 to 1 to initialize weights and bias. Using the same way as logistic regression task, we discuss the impact with different value of learning rate η and get a proper value of learning rate for each method.

In figure 6, 7, 8 and 9, we exhibit the results of four different optimized methods in proper values of learning rate respectively. We can obverse that the training loss curves in four different optimized methods are oscillating because of the use of mini-batch gradient descent. And the validation loss curves and accuracy curves are smooth and increases fast.

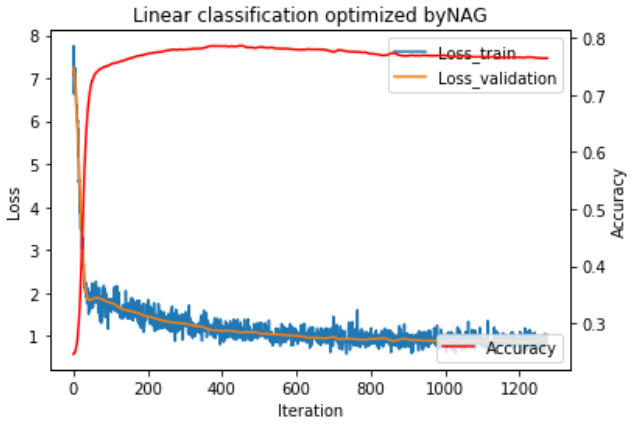

Fig. 6. Linear classification optimized by by Nesterov accelerated gradient (NAG) with η = 0.01.

And then we also compare the performances of different optimized methods in this task by the accuracy curve. In figure 6, 7, 8 and 9, we can obverse that Adam, RMSProp and Adadelta can achieve the better performance than NAG by using linear SVM classifier. The convergence rate in four optimized methods are different NAG $>$ Adam $>$ RMSProp $>$ AdaDelta.

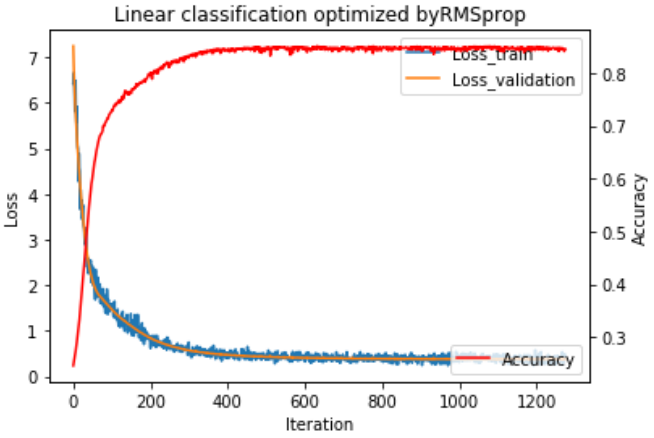

Fig. 7. Linear classification optimized by RMSProp with η = 0.06.

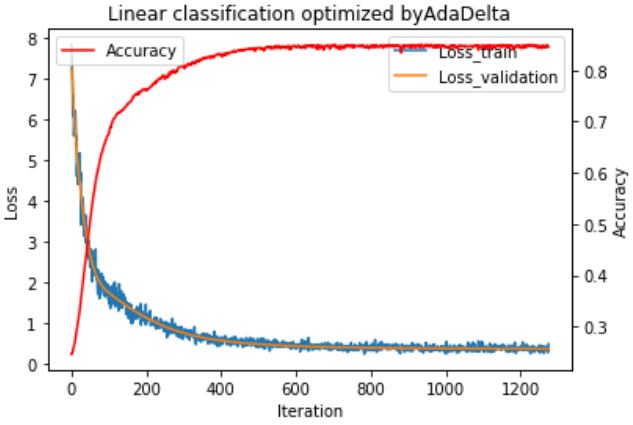

Fig. 8. Linear classification optimized by AdaDelta with η = 0.04.

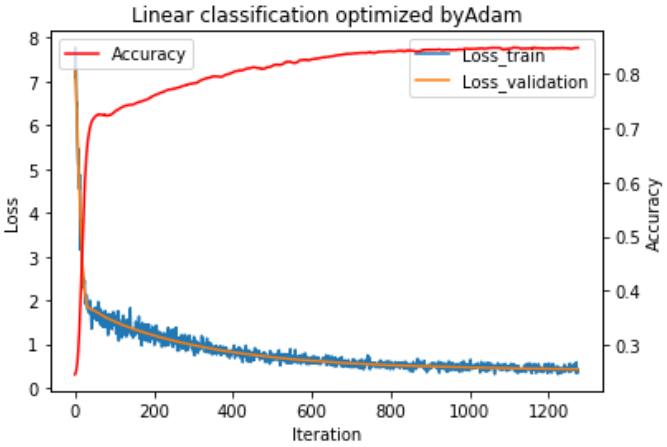

Fig. 9. Linear classification optimized by Adaptive Moment Estimation (Adam) with η = 0.07.

IV. Conclusion

In this report, we compare the difference between four various optimized methods, which are the variants of SGD. For better understand the differences, we explore the performances of algorithms in regression and classification tasks.