The Experiment Report of *Deep Learning*

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:	*Supervisor:*
TengyunWang	Mingkui Tan
Student ID:	*Grade:*
201710106574	Graduate

December 15, 2017

Linear Regression, Linear Classification and Gradient Descent

***Abstract*—Optimizer algorithm matters a lot in machine learning. Gradient decent which is one of the most famous algorithms to perform optimization has been applied to solve the optimization problem in machine learning. However, there are many disadvantages in gradient decent. The literature has proposed many gradient decent variances to relieve the insufficient of gradient decent. In this paper, we implement 4 prevailing gradient decent variances and reveal the efficiency of them by conduct experiment in public dataset.**

I. Introduction

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 (e.g. lasagne's, caffe's, and keras' documentation). These algorithms, however, are often used as black-box optimizers, as practical explanations of their strengths and weaknesses are hard to come by.

II. Methods and Theory

There are three variants of gradient descent, which differ in how much data we use to compute the gradient of the objective function. Depending on the amount of data, we make a trade-off between the accuracy of the parameter update and the time it takes to perform an update.

A. Stochastic gradient descent

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

$$\theta = \theta - \eta \nabla_\theta J(\theta; x_i, y_i) \tag{1}$$

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.

B. 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:

$$\theta = \theta - \eta \nabla_\theta J(\theta; x_{i:i+n}, y_{i:i+n}) \tag{2}$$

This way, it a) reduces the variance of the parameter updates, which can lead to more stable convergence; and b) 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.

C. Nesterov accelerated gradient

Nesterov accelerated gradient(NAG)[1] 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:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1}) \tag{3}$$

$$\theta_{t+1} = \theta_t - v_t \tag{4}$$

D. Adadelta

Adadelta[2] is an extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate.

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 $E[g^2]_t$ at time step t then depends (as a fraction γ similarly to the Momentum term) only on the previous average and the current gradient:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2 \tag{5}$$

We set γ to a similar value as the momentum term, around 0.9. For clarity, we now rewrite our vanilla SGD update in terms of the parameter update vector $\nabla \theta_t$:

$$\Delta\theta_t = g_t \tag{6}$$

$$\theta_{t+1} = \theta_t - \eta \Delta\theta_t \tag{7}$$

The parameter update vector of Adagrad that we derived previously thus takes the form:

$$\Delta\theta_t = \frac{1}{\sqrt{G_t + \epsilon}} \odot g_t \tag{8}$$

We now simply replace the diagonal matrix G_t with the decaying average over past squared gradients $E[g^2]_t$:

$$\Delta\theta_t = \frac{1}{\sqrt{E[g^2]_t + \epsilon}} \odot g_t \tag{9}$$

As the denominator is just the root mean squared (RMS) error criterion of the gradient, we can replace it with the criterion short-hand:

$$\Delta\theta_t = \frac{1}{RMS[g]_t} g_t \tag{10}$$

The authors note that the units in this update (as well as in SGD, Momentum, or Adagrad) do not match, i.e. the update should have the same hypothetical units as the parameter. To realize this, they first define another exponentially decaying average, this time not of squared gradients but of squared parameter updates:

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2 \tag{11}$$

The root mean squared error of parameter updates is thus:

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon} \tag{12}$$

Since $RMS[\Delta\theta]_t$ is unknown, we approximate it with the RMS of parameter updates until the previous time step. Replacing the learning rate η in the previous update rule with $RMS[\Delta\theta]_{t-1}$ finally yields the Adadelta update rule:

$$\Delta\theta_t = \frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t \tag{13}$$

$$\theta_{t-1} = \theta_t - \eta\Delta\theta_t \tag{14}$$

E. 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 Adagrad's radically diminishing learning rates. RMSprop in fact is identical to the first update vector of Adadelta that we derived above:

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2 \tag{15}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \tag{16}$$

F. Adam

Adaptive Moment Estimation(Adam)[3] 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:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)g_t \tag{17}$$

$$v_t = \beta_2 m_{t-1} + (1-\beta_2)g_t^2 \tag{18}$$

m_t and v_t are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively, hence the name of the method. As m_t and v_t are initialized as vectors of 0's, the authors of Adam observe that they are biased towards zero, especially during the initial time steps, and especially when the decay rates are small.

They counteract these biases by computing bias-corrected first and second moment estimates:

$$\hat{m_t} = \frac{m_t}{1-\beta_1^t} \tag{19}$$

$$\hat{v_t} = \frac{v_t}{1-\beta_2^t} \tag{20}$$

They then use these to update the parameters just as we have seen in Adadelta and RMSprop, which yields the Adam update rule:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v_t}} + \epsilon}\hat{m_t} \tag{21}$$

III. Experiments

A. Dataset

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

B. Implementation

In this experiment, we adopt logistic regression and soft margin linear support vector machine as our basic model to exam the efficiency of different algorithm.

1) Logistic Regression: The hypothesis of logistic regression is

$$h(x; w, b) = \frac{1}{1+e^{-(w^T x + b)}} \tag{22}$$

The negative value is -1, and the positive value is $+1$. We use the negative likelihood loss and the loss function is derived as follows

$$\begin{aligned} L =& \frac{1}{2N} * \frac{1}{2}\sum_{i=1}^{N} -log(h(x_i))(1+y_i) - log(1-h(x_i))(1-y_i) \\ &+ \frac{1}{2}\lambda(\|w\|^2 + \|b\|^2) \end{aligned} \tag{23}$$

The derivative of loss function is

$$L' = \frac{1}{N}\sum_{i=1}^{N} x_i(2h(x_i) - (y_i+1)) + \lambda(w+b) \tag{24}$$

2) Soft-Margin SVM: The optimization objective of SVM is to find a hyperplane which achieves a good separation and has the largest distance to the nearest training-data point of any class. Soft-margin SVM allows misclassification and the optimization objective is defined as follows

$$\begin{aligned} \underset{w,b,\varepsilon}{\text{argmin}} \quad & \frac{\|w\|^2}{2} + C\sum_{i=1}^{N}\varepsilon_i \\ \text{subject to} \quad & y_i(w^T x_i + b) \geqslant 1 - \varepsilon_i, \\ & \varepsilon_i \geqslant 0 \end{aligned} \tag{25}$$

Thus the loss function of soft-margin SVM is

$$L = \frac{1}{N}\sum_{i=1}^{N} C * hinge\_loss(x_i, y_i) + \frac{1}{2}\lambda(\|w\|^2 + \|b\|^2) \tag{26}$$

$$hinge\_loss(x, y; w, b) = max(0, 1 - y * (w^T x + b)) \quad (27)$$

Loss function‘s derivative is

$$L' = -\frac{C}{N}\sum_{i=1}^{N} x_i y_i \mathbb{I}[hinge\_loss(x_i, y_i) > 0] + \lambda(w+b) \quad (28)$$

C. Experiment Setup

We dived experiment into two parts. In the first part, we empirically initialize the hyper-parameters which will be used in mini-batch gradient decent, NAGRMSPropAdaDelta and Adam. Then we draw the loss‘s changing curve of NAGRMSPropAdaDelta and Adam accordingly. In the second part, we tuning the hyper-parameters by exhaustive gird search. Then we give the loss's changing curve too.

In both parts, we set the iterations to 50 which means that we use the whole training set 50 times to perform the gradient decent. The batch size is set to 8000, means that we use 8000 records in each epoch.

D. Experiment Part One

The un-tuned logistic regression models‘ performance are display in Fig.1

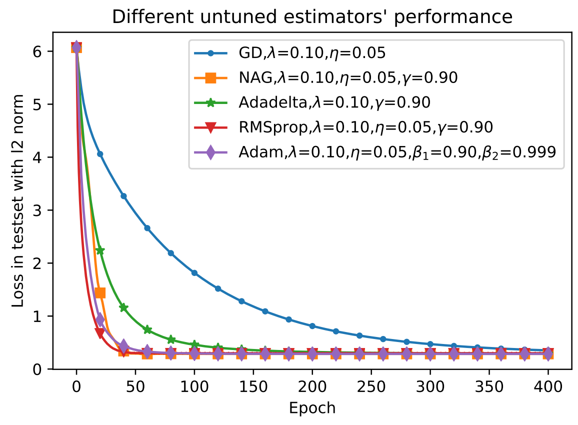


Fig. 1: LR models‘ simulation results on public dataset. The hyper-parameters are set by expert knowledge.

The un-tuned SVM models‘ performance are display in Fig.2

We can figure out from Fig.1 and Fig.2 that with mini-bath gradient decent and no tuning, RMSprop, Adam, NAG reach convergence within 50 epochs. Adadelta needs 100+ epochs to converge and GD needs 400+ epochs. Thus, RMSprop, Adam, NAG make fast and satisfied convergence.

E. Experiment Part Two

In this section, we use exhaustive grid search to tuning the hyper-parameters of models. We should define the search grid firstly. We share the hyper-parameters‘ grid in LR and SVM. The grid is list in Tab. I. For Adam, there are two

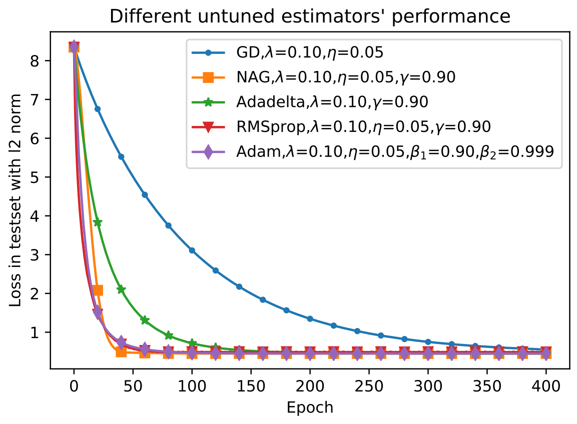


Fig. 2: SVM models‘ simulation results on public dataset. The hyper-parameters are set by expert knowledge.

additional hyper-parameters which are β_1 and β_2. And the candidates are $0.9, 0.95$ and $0.99, 0.999$ accordingly. Then, we use 3 fold cross validation to pick out the best combination of hyper-parameters in each algorithm. The judging metric is the classification accuracy in validation set. The loss changing curves and accuracy changing curves of GD, NAG, Adadelta, RMSprop, Adam in LR models and SVM models are displayed in Fig. 5 and Fig. 6 accordingly. It is interesting that the accuracy does not take a strict inverse ratio to loss for SVM model, and three of the algorithm vibrate a lot although the loss function has reach a convergence. This means that there are many uncertain samples near the decision boundary.

Finally, we leave out the accuracy changing curves and put the all loss changing curves together in Fig.3 and Fig.4 to demonstrate the performance clearly. And this two figures show the performance of tuned models. In these two figures we can see that all of five algorithm perform similarly for LR model. And for SVM model, Adam and Adadelta work a little better. Gradient decent has high-efficiency if given a appropriate learning rate.

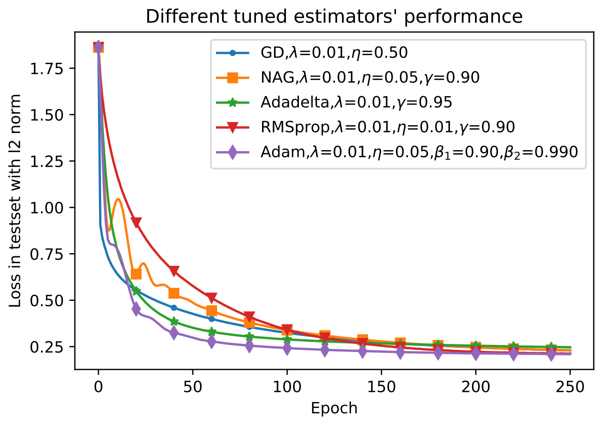


Fig. 3: LR models‘ simulation results on public dataset. The hyper-parameters are tuned by exhaustive gird search.

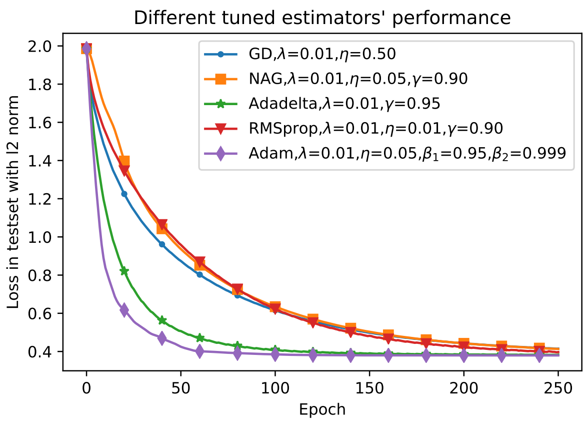


Fig. 4: SVM models‘ simulation results on public dataset. The hyper-parameters are tuned by exhaustive gird search.

TABLE I: Search Grid of LR

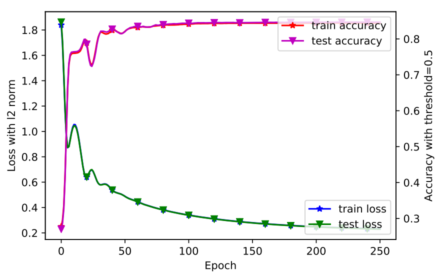
	λ	η	γ	*threshold*
NAG	0.01, 0.1	0.01, 0.05	0.8, 0.9, 0.95	0.5, 0.6
Adadelta	0.01, 0.1	None	0.8, 0.9, 0.95	0.5, 0.6
RMSprop	0.01, 0.1	0.01, 0.05	None	0.5, 0.6
Adam	0.01, 0.1	0.01, 0.05	None	0.5, 0.6
GD	0.01, 0.1, 0.5	0.1, 0.2, 0.4, 0.5	None	0.4, 0.5, 0.6

IV. Conclusion

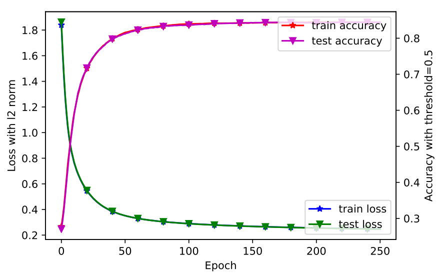
Gradient decent is indeed a resultful optimization algorithm. But it has strongly dependency on hyper-parameters especially η. The four gradient decent variants work remarkable although without the tuning. In practice if we cant tuning the hyper-parameters due to the resource limits, we can propose RM-Sprop or Adam. Both of the algorithm are not so sensitive to hyper-parameters especially learning rate.

References

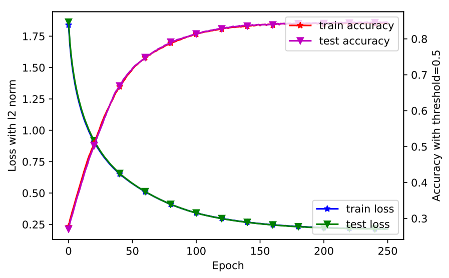
[1] Y. Nesterov, “A method of solving a convex programming problem with convergence rate o (1/k2),” in *Soviet Mathematics Doklady*, vol. 27, no. 2, 1983, pp. 372–376.
[2] M. D. Zeiler, “Adadelta: an adaptive learning rate method,” *arXiv preprint arXiv:1212.5701*, 2012.
[3] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.



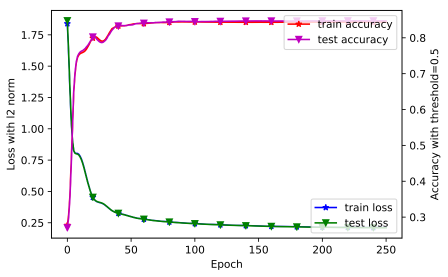
(a) NAG



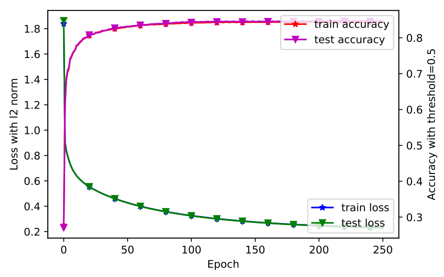
(b) Adadelta



(c) RMSprop

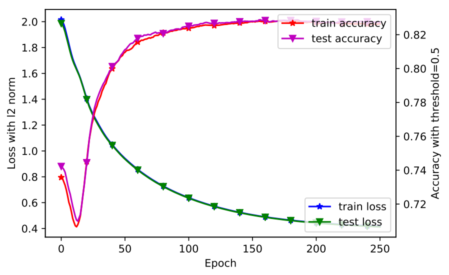


(d) Adam

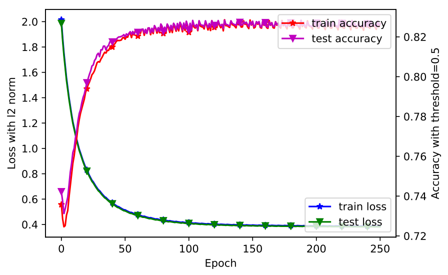


(e) GD

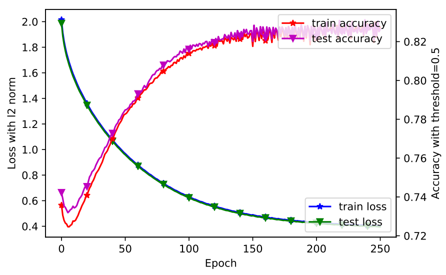
Fig. 5: The loss changing curve and accuracy changing curve in LR.



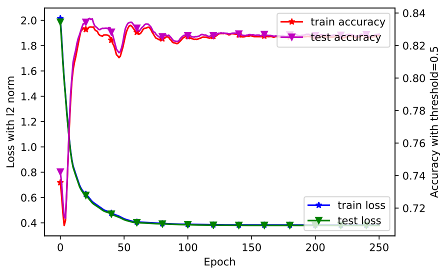
(a) NAG



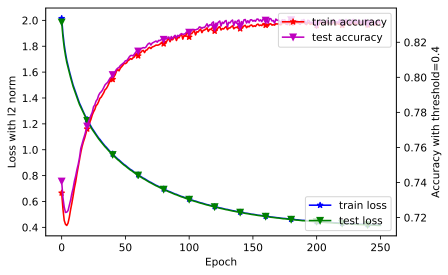
(b) Adadelta



(c) RMSprop



(d) Adam



(e) GD

Fig. 6: The loss changing curve and accuracy changing curve in SVM.