The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
黄启琛

Supervisor:
Mingkui Tan

Student ID：
201530611777

Grade:
Undergraduate

December 14, 2017

Comparison of Various Stochastic Gradient Descent Methods for Solving Classification Problems

Abstract—Gradient Descent is one of the most popular algorithms to perform optimizition. Because of the weeknesses of Batch Gradient Descent and Stochastic Gradient Descent, Mini-batch Stochastic Gradient Descent(SGD) seems to be a better choice.SGD also addressed several challenges during the optimization. Thus, many optimization methods were proposed previously. In this paper, several optimization methods are compared and summarized, including NAG, RMSprop, Adadelta, Adam etc. Considering the influence caused bydifferent models, the experiments are conducted on Logistic Regression and SVM for more believable conclusion.

I. INTRODUCTION

This part will introduce and analyse several optimization methods. For completeness of this paper, the theory of SVM for Linear Classification and Logistic Regresion are briefly introduce in the next part.

Nesterov Accelerated Gradient(NAG)

NAG method is based on momentum method. When using momentum, we push a ball down a hill. The ball accumulates momentum as it rolls downhill, becoming faster and faster on the way. The same thing happens to our parameter updates: The momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions.

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

Note: $J(\theta)$ is the object function, η is learning rate, θ is the parameter vector needs to be update, v_t is the momentum term.

NAG is a way to give our moment term presciently. 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. We can now calculate the gradient to the approximation of futher position

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

NAG maintains the strength of momentum, like faster convergence and reduced oscillation, and also prevents the parameters from updating too fast, which result in increased responsiveness.

Adadelta

Adadelta is an extension of Adagrad. Adagrad is an algorithm for gradient-based optimization that adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters.

$$g_t = \nabla_\theta J(\theta_{t-1})$$

$$G_t = \sum_i^t g_i \times g_i$$

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{G_t + \delta}} \times g_t$$

Note: the “×” here is element wise multiply

Adagrad's main weakness is its accumulation of the squared gradients. Since every added term is positive, the accumulated sum keeps growing during training. This in turn causes the learning rate to shrink and eventually become infinite small, at which point the algorithm is no longer able to acquire additional knowledge.

Adadelta aims to resolve this flaw. Instead of accumulating all past squared gradients, the sum of gradients is recursively defined as a decaying average of all past squared gradients.

$$G_t = \rho G_{t-1} + (1-\rho) g_t \times g_t$$

Futhermore, consider previous update steps the parameters θ actually takes. The smaller steps it takes to update previously, the more reasonable to believe that futher steps need to be small.

$$g_t = \nabla_\theta J(\theta_{t-1})$$
$$g'_t = \frac{\sqrt{\Delta x_{t-1} + \delta}}{\sqrt{G_t + \delta}} \times g_t$$
$$\nabla x_t = \rho \nabla x + (1-\rho)\, g'_t \times g'_t$$
$$\theta_t = \theta_{t-1} - g'$$

Note: δ is a small number, void dividing zero.

RMSprop

Like Adadelta, RMSprop also aims to Adagrad's radically diminishing learning rates.

$$g_t = \nabla_\theta J(\theta_{t-1})$$
$$G_t = \rho G_{t-1} + (1-\rho) g_t \times g_t$$
$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{G_t + \delta}} \times g_t$$

The difference between Adadelta and RMSprop is that RMSprop still needs to specify a global learning rate η .

Adaptive Moment Estimation (Adam)

Adam is a method that combines Momentum and RMSprop.

$$g_t = \nabla_\theta J(\theta_{t-1})$$
$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t \times g_t$$

m_t and v_t are biased towards zero, especially during the initial time steps, and especially when the decay rates are small (i.e. β_1 and β_2 are close to 1).

We take the bias-corrected parameters

$$\hat{m}_t = \frac{m_t}{1 - {\beta_1}^t}$$
$$\hat{v}_t = \frac{v_t}{1 - {\beta_2}^t}$$

The update rule is

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t + \delta}} \times \hat{m}_t$$

The default values of $\beta_1$and β_2 is 0.9 and 0.999 respectively, and 10^{-8} for δ. They show empirically that Adam works well in practice and compares favorably to other adaptive learning-method algorithms.

II. METHODS AND THEORY

Support Vector Machine

Object Function

SVM is a method of Linear Classification. Linear Classification is classify some data x_i with a linear function $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$, such that

$$f(x_i) = \begin{cases} \geq 0 & y_i = +1 \\ < 0 & y_i = -1 \end{cases}$$

Let f(x) = 0,we get $\mathbf{w}^T\mathbf{x} + b = 0$. It is a hyperplane in multi-dimension space. Select another two parallel hyperplanes that separate the two classes of data and let the distance between tham as large as possible,which is called "Support Vector". The region bounded by these two hyperplanes is called the "margin". A good function $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$ to classify the data is that it has maximum margin, which is most stable under perturbation of the inputs.

Choose normalization such that $\mathbf{w}^T\mathbf{x}_+ + b = +1$ and $\mathbf{w}^T\mathbf{x}_- + b = -1$ for the positive and negative support vectors respectively. Then the magin is given by

$$\frac{w}{\|w\|}(\mathbf{x}_+ - \mathbf{x}_-) = \frac{2}{\|w\|}$$

Learning the SVM can be formulate as an optimization:

$$\max \frac{2}{\|w\|}$$

$$\text{s.t. } \mathbf{w}^T\mathbf{x_i} + b \begin{cases} \geq 1 & y_i = +1 \\ \leq -1 & y_i = -1 \end{cases}$$

Or equivalently:

$$\min \frac{\|w\|^2}{2}$$

$$\text{s.t. } y_i(\mathbf{w}^T\mathbf{x_i} + b) \geq 1$$

Introduce variable $\xi_i \geq 0$, which represents how much example i is on "wrong side" of margin boundary

The opeimization problem becomes:

$$\min \frac{\|w\|^2}{2} + C\sum_{i=1}^{n}\xi_i$$

$$\text{s.t. } y_i(\mathbf{w}^T\mathbf{x_i} + b) \geq 1 - \xi_i$$

using the Hinge Loss, the optimization problem becomes:

$$\min \frac{\|\mathbf{w}\|^2}{2} + \frac{C}{n}\sum_{i=1}^{n}\max(0, 1 - y_i(\mathbf{w}^T\mathbf{x} + b))$$

Gradient Computation

The hinge loss is $\xi_i = \max(0, 1 - y_i(\mathbf{w}^T\mathbf{x} + b))$

Let $g_\mathbf{w}(\mathbf{x}_i) = \frac{\partial \xi_i}{\partial \mathbf{w}}$

if $1 - y_i(\mathbf{w}^T\mathbf{x} + b) \geq 0$:

$$g_w(\mathbf{x}_i) = \frac{\partial(1 - y_i(\mathbf{w}^T\mathbf{x} + b))}{\partial \mathbf{w}}$$

$$g_w(\mathbf{x}_i) = -y_i\mathbf{x}_i$$

if $1 - y_i(\mathbf{w}^T\mathbf{x} + b) < 0$:

$$g_w(\mathbf{x}_i) = 0$$

so we have:

$$g_\mathbf{w}(\mathbf{x}_i) = \begin{cases} -y_i\mathbf{x}_i & 1 - y_i(\mathbf{w}^T\mathbf{x} + b) \geq 0 \\ 0 & 1 - y_i(\mathbf{w}^T\mathbf{x} + b) < 0 \end{cases}$$

Let $g_b(\mathbf{x}_i) = \frac{\partial \xi_i}{\partial b}$

$$g_b(\mathbf{x}_i) = \begin{cases} -y_i & 1 - y_i(\mathbf{w}^T\mathbf{x} + b) \geq 0 \\ 0 & 1 - y_i(\mathbf{w}^T\mathbf{x} + b) < 0 \end{cases}$$

Optimization problem:

$$\min L(\mathbf{w}, b) = \min \frac{\|\mathbf{w}\|^2}{2} + \frac{C}{n}\sum_{i=1}^{n}\max(0, 1 - y_i(\mathbf{w}^T\mathbf{x} + b))$$

So we have:

$$\nabla_\mathbf{w} L(\mathbf{w}, b) = \mathbf{w} + \frac{C}{n}\sum_{i=1}^{n} g_\mathbf{w}(\mathbf{x}_i)$$

$$\nabla_b L(\mathbf{w}, b) = \mathbf{w} + \frac{C}{n}\sum_{i=1}^{n} g_b(\mathbf{x}_i)$$

Logistic Regression

Object Function

Logistic Regression is another method of Linear Classification. However, the output of Logistic Regression is the likelyhood of Classification.

Define the Losgistic function:

$$g(z) = \frac{1}{1 + e^{-z}}$$

We have prediction function:

$$h_\mathbf{w}(\mathbf{x}) = g\left(\sum_{i=1}^{m} w_i x_i\right) = g(\mathbf{w}^T\mathbf{x})$$

Probability function:

$$p(y_i|\mathbf{x}_i) = \begin{cases} h_\mathbf{w}(\mathbf{x_i}) & y_i = 1 \\ 1 - h_\mathbf{w}(\mathbf{x_i}) & y_i = 0 \end{cases}$$

which equivalent to:

$$p(y_i|x_i) = h_\mathbf{w}(\mathbf{x_i})^{y_i} \cdot (1 - h_\mathbf{w}(\mathbf{x_i}))^{(1-y_i)}$$

Maximum log-likehood loss function:

$$\max \prod_{i=1}^{n} p(y_i|x_i) \Leftrightarrow \min -\frac{1}{n}\sum_{i=1}^{n}\log(p(y_i|x_i))$$

Finally, we get our object function:

$$J(\mathbf{w}) = -\frac{1}{n}\left[\sum_{i=1}^{n} y_i \log h_\mathbf{w}(\mathbf{x_i}) + (1 - y_i)\log(1 - h_\mathbf{w}(\mathbf{x_i}))\right]$$

Gradient Computation

For a sample :

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = -\frac{1}{\partial \mathbf{w}} \cdot \partial[y_i \log h_\mathbf{w}(\mathbf{x_i}) + (1 - y_i)\log(1 - h_\mathbf{w}(\mathbf{x_i}))]$$

$$= (h_\mathbf{w}(\mathbf{x}) - y)\mathbf{x}$$

Update rule :

$$\mathbf{w} = \mathbf{w} - \eta\,(h_\mathbf{w}(\mathbf{x}) - y)\mathbf{x}$$

For all sample :

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = \frac{1}{n}\sum_{i=1}^{n}(h_\mathbf{w}(\mathbf{x_i}) - y_i)\mathbf{x_i}$$

Update rule :

$$\mathbf{w} = \mathbf{w} - \frac{\eta}{n}\sum_{i=1}^{n}(h_{\mathbf{w}}(\mathbf{x}_i) - y_i)\mathbf{x}_i$$

III. EXPERIMENT

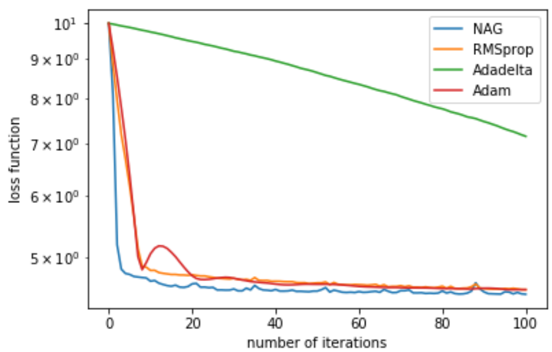
The Experiment uses a9a of LIBSVM Data, including 32561(training)/16281(testing) samples and each sample has 123 features. The environment of experiment is python 3, including python package: sklearn， numpy， jupyter， matplotlib.

Experiment Step

1. Load the training set and validation set.
2. Initalize SVM/Logistic Regression model parameters, here we consider zeros initialization.
3. Calculate gradient toward loss function ,as metioned above, from partial samples.
4. Update model parameters using NAG，RMSProp，AdaDelta and Adam optimization methods.
5. Predict the result under the validation set and calculate the loss function of difference optimization method.
6. Repeat step 3 to 5 for several times, and draw graph of these loss function with the number of iterations

Experiment Result

The graph of loss function in SVM looks like:



Final prediction accuracy is:

NAG accuracy： 0.800503654567

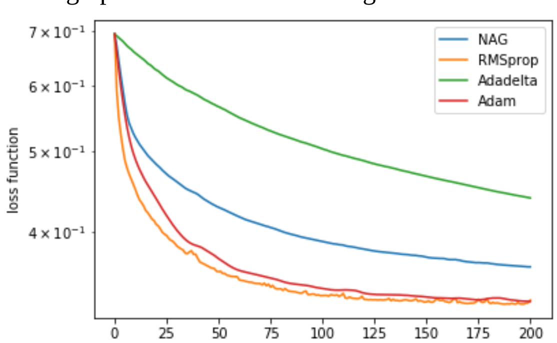
RMSprop accuracy： 0.778821939684

Adadelta accuracy： 0.763773723973

Adam accuracy： 0.794545789571

Conclusion is that the performance of NAG is the best,next is Adam, Adadelta performs worst. From the graph, we can also know that the loss function of Adadelta goes down slower than others.

The graph of loss function in Logistic function looks like:



Final prediction accuracy is :

NAG accuracy: 0.831705669185

RMSprop accuracy: 0.84527977397

Adadelta accuracy: 0.781340212518

Adam accuracy: 0.848412259689

The difference between Logistic Regression and SVM is that NAG no longer performs the best. Both RMSprop and Adam have higher accuracy than NAG. What is the same is that Adadelta still performs worst.

Because the slow speed of convergence in Adadelta, I make a little change of the update rule:

$$\theta_t = \theta_{t-1} - 2 * g'$$

but it seems have no help in the performance of Adadelta.

IV. CONCLUSION

The Experiment result show us that in different model, an optimization method has difference performance. Considering the two Experiment , Adam seems perform better in difference model, while Adadelta performs worst. We can also that NAG and RMSprop perform well in SVM and Logistic Regression respectively. As we know that Adam is a combination of RMSprop and momentum, a futher work is to have a try on combining RMSprop and NAG, which is to calculate the gradient of the estimation of futher position. In theory, it should have better performance than Adam.