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:
Junteng Pang

Supervisor:
Mingkui Tan

Student ID：
201720145174

Grade:
Graduate

December 15, 2017

# Logistic Regression, Linear Classification and Stochastic Gradient Descent

**Abstract— this experiment shows the difference between gradient descent and stochastic gradient descent as well as the differences and relationships between Logistic regression and linear classification. What's more, it's a good way to have a further understanding in the principles of SVM and practice it on larger data.**

## I. INTRODUCTION

The purpose of the experiment is to compare and understand the difference between gradient descent and stochastic gradient descent. Also, it compares the differences and relationships between Logistic regression and linear classification.

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

## II. METHODS AND THEORY

The Logistic Regression is used in binary classification problems. The loss function is defined as below:

$$J(w) = \frac{1}{n}\sum_{n=1}^{n} log\left(1 + e^{-y_i * w^T x_i}\right)$$

There are many ways to optimize the model. Firstly, SGD picks an initial value $w^0$ randomly, then compute

$$w^{i+1} \leftarrow w^i - \eta \frac{dL}{dw}|_{w=w^i}$$

Update the value of w and b in each iterations. In this project, in each calculation of gradient, use d samples:

$$\frac{\partial L}{\partial w} = -\frac{1}{n}\sum_{i*d}^{(i+1)d}\left(\frac{y_i x_i}{e^{y_i * w^T x_i} + 1}\right)$$

the argument like initial value of w, and $\eta$ can be change in each iterations so that the loss can be minimized.

Also, there are different optimized methods like NAG，RMSProp, AdaDelta and Adam.

In NAG(Nesterov accelerated gradient), it use Momentum to forecast the next gradient instead of the current gradient:

$$g_t = \nabla J(\theta_{t-1} - \gamma v_{t-1})$$

$$v_t = \gamma v_{t-1} + \eta g_t$$

$$\theta_t = \theta_{t-1} - v_t$$

RMSProp is the updated version of AdaGrad, it can solve the proplem in AdaGrad that the v tend to zero.

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

$$G_t = \gamma G_t + (1-\gamma) g_t \odot g_t$$

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

AdaDelta also can solve the problem of AdaGrad, it doesn't need a original learning rate but sometimes is slow.

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

$$\mathrm{G}_t = \gamma G_t + (1-\gamma) g_t \odot g_t$$

$$\Delta\theta_t = -\frac{\sqrt{\Delta_{t-1} + \varepsilon}}{\sqrt{\mathrm{G}_t + \varepsilon}} \odot g_t$$

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

$$\theta_t = \gamma \Delta\theta_{t-1} + (1-\gamma)\Delta\theta_t \odot \Delta\theta_t$$

Adam perform better in the correction of the initialization of bias.

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

$$\mathrm{m}_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$\mathrm{G}_t = \gamma G_t + (1-\gamma) g_t \odot g_t$$

$$\alpha = \eta \frac{\sqrt{1-\gamma^t}}{1-\beta_t}$$

$$\theta_t = \theta_{t-1} - \alpha \frac{\mathrm{m}_t}{\sqrt{\mathrm{G}_t + \varepsilon}}$$

The second part of the experiment is to use SVM model in classification problem. Linear Classification is used to forecast the class of input data. Giving training data $(x_i, y_i)$ for i = 1..n with $x_i \in R^m$ and $y_i \in \{-1,1\}$, learn a classfier f(x)
such that

$$\mathrm{f(x)} \begin{cases} \geq 0 & y_i = +1 \\ \leq 0 & y_i = -1 \end{cases}$$

The Hinge Loss is defined as below:

$$\text{Hinge loss} = \max(0, 1 - y_i(w^T x_i + \mathrm{b}))$$

The optimization problem becomes:

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

An optimization problem can be considered in two ways, primal problem and dual problem. And primal problem is selected in this experiment.

$$g_w(x_i) = \begin{cases} -y_i x_i & 1 - y_i(w^T x_i + b) \geq 0 \\ 0 & 1 - y_i(w^T x_i + b) < 0 \end{cases}$$

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

$$\nabla_w L(w,b) = w + \frac{C}{n} \sum_{i=1}^{n} g_w(x_i)$$

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

$$\mathrm{w} = \mathrm{w} - \eta \nabla_w L(w,b)$$

$$\mathrm{b} = \mathrm{b} - \eta \nabla_b L(w,b)$$

## III. EXPERIMENTS

**Logistic Regression and Stochastic Gradient Descent**

a. Load the training set and validation set.

```
def get_data(filename):
    data = load_svmlight_file(filename,123)
```

```
    return data[0], data[1]

#Import data
X_train, y_train = get_data("a9a")
X_test, y_test = get_data("a9a.t")
```

b. Initialize logistic regression model parameters to zero.
c. Select the loss function and calculate its derivation.

$$LossFunction = \frac{1}{n}\sum_{i=1}^{n} log(1 + e^{-y_i \cdot w^T X_i})$$

d. Calculate gradient towards loss function from partial samples.
e. Update model parameters using different optimized methods (NAG, RMSProp, AdaDelta and Adam).
SGD:

```
def
logisticRegressionSGD(delta,n,X_train,y_train,X
_test,y_test,batchSize):
    #Initialize linear model parameters. Set all
parameter into zero
    W = np.zeros(X_train.shape[1])
    trainCost = []
    testCost = []
    for i in range(n):

trainCost.append(np.sum(np.log(np.exp(X_train.
dot(W)*y_train*(-1))+1))/X_train.shape[0])

x_batch=(X_train[i*batchSize:(i+1)*batchSize,:])

y_batch=(y_train[i*batchSize:(i+1)*batchSize])

W=W+delta*x_batch.T.dot(y_batch/(np.exp(x_b
atch.dot(W)*y_batch)+1))/batchSize



testCost.append(np.sum(np.log(np.exp(X_test.dot
(W)*y_test*(-1))+1))/X_test.shape[0])

    return trainCost,testCost
```

NAG:

```
def
logisticRegressionNAG(delta,n,X_train,y_train,X
_test,y_test,batchSize):
    W=np.zeros(X_train.shape[1])
    trainCost=[]
    testCost=[]
    gama=0.9
    V=np.zeros(W.shape)

    for i in range(n):

trainCost.append(np.sum(np.log(np.exp(X_train.
dot(W)*y_train*(-1))+1))/X_train.shape[0])

X_batch=(X_train[i*batchSize:(i+1)*batchSize,:]
)

y_batch=(y_train[i*batchSize:(i+1)*batchSize])
        wforecast=W-gama*V
        g=(-
1)*X_batch.T.dot(y_batch/(np.exp(X_batch.dot(
wforecast)*y_batch)+1))/batchSize
        V=gama*V+delta*g
        W=W-V



testCost.append(np.sum(np.log(np.exp(X_test.dot
(W)*y_test*(-1))+1))/X_test.shape[0])
    return trainCost,testCost
```

RMSProp:

```
def
logisticRegressionRMSProp(delta,num_iters,x_tr
ain,y_train,x_test,y_test,batch_size):
    w=np.ones(x_train.shape[1])
    w=w/20
    trainCost=[]
    testCost= []
    epsilon=1/(10**8)
    gama=0.9
    G=np.zeros(w.shape)

    for i in range(num_iters):

trainCost.append(np.sum(np.log(np.exp(x_train.d
ot(w)*y_train*(-1))+1))/x_train.shape[0])

x_batch=(x_train[i*batch_size:(i+1)*batch_size,:
])

y_batch=(y_train[i*batch_size:(i+1)*batch_size])
```

```
        g=(-
1)*x_batch.T.dot(y_batch/(np.exp(x_batch.dot(w
)*y_batch)+1))/batch_size
        G=G*gama+(1-gama)*np.square(w)
        w=w-delta/(np.sqrt(G+epsilon))*w


testCost.append(np.sum(np.log(np.exp(x_test.dot
(w)*y_test*(-1))+1))/x_test.shape[0])
    return trainCost,testCost
```

AdaDelta:

```
def
logisticRegressionAdaDelta(delta,num_iters,x_tra
in,y_train,x_test,y_test,batch_size):
    w=np.ones(x_train.shape[1])
    w=w/20
    trainCost=[]
    testCost=[]
    epsilon=1/(10**8)
    gama=0.95
    G=np.zeros(w.shape)
    dt=np.zeros(w.shape)

    for i in range(num_iters):

trainCost.append(np.sum(np.log(np.exp(x_train.d
ot(w)*y_train*(-1))+1))/x_train.shape[0])

testCost.append(np.sum(np.log(np.exp(x_test.dot
(w)*y_test*(-1))+1))/x_test.shape[0])

x_batch=(x_train[i*batch_size:(i+1)*batch_size,:
])

y_batch=(y_train[i*batch_size:(i+1)*batch_size])


        g=(-
1)*x_batch.T.dot(y_batch/(np.exp(x_batch.dot(w
)*y_batch)+1))/batch_size
        G=G*gama+(1-gama)*np.square(g)
        dw=(-
1)*np.sqrt(dt+epsilon)/np.sqrt(G+epsilon)*g
        w=w+dw
        dt=gama*dt+(1-gama)*dw*dw
    return trainCost,testCost
```

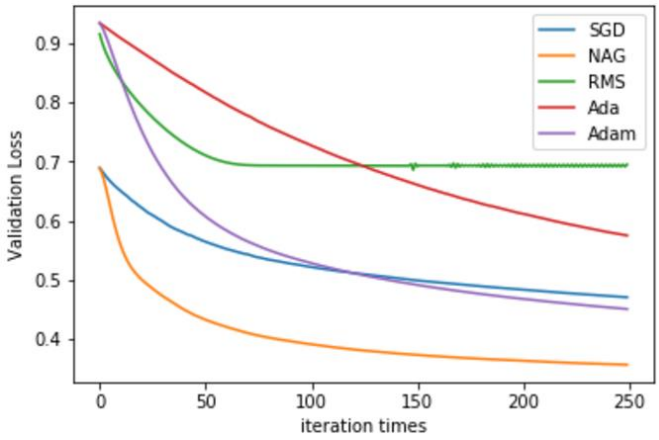
Adam:

```
def
logisticRegressionAdam(delta,num_iters,x_train,
y_train,x_test,y_test,batch_size):
    w=np.ones(x_train.shape[1])
    w=w/20
    trainCost=[]
    testCost=[]
    epsilon=1/(10**8)
    gama=0.999
    beta=0.9
    delta=0.001
    m=np.zeros(w.shape)
    G=np.zeros(w.shape)

    for i in range(num_iters):

trainCost.append(np.sum(np.log(np.exp(x_train.d
ot(w)*y_train*(-1))+1))/x_train.shape[0])

testCost.append(np.sum(np.log(np.exp(x_test.dot
(w)*y_test*(-1))+1))/x_test.shape[0])

x_batch=(x_train[i*batch_size:(i+1)*batch_size,:
])

y_batch=(y_train[i*batch_size:(i+1)*batch_size])


        g=(-
1)*x_batch.T.dot(y_batch/(np.exp(x_batch.dot(w
)*y_batch)+1))/batch_size
        m=beta*m+(1-beta)*g
        G=G*gama+(1-gama)*np.square(g)
        #alpha=delta*np.sqrt(1-gama**i)/(1-
beta**i)
        alpha=delta*np.sqrt(1-gama)/(1-beta)
        w=w-alpha*m/(np.sqrt(G+epsilon))
    return trainCost,testCost
```

f. Select the appropriate threshold, mark the sample whose predict scores greater than the threshold as positive, on the contrary as negative. Predict under validation set and get the different optimized method loss $L_{NAG}$ , $L_{RMSProp}$, $L_{AdaDelta}$ and$L_{Adam}$ .

g. Repeat step 4 to 6 for several times, and drawing graph of $L_{NAG}$ , $L_{RMSProp}$ ,

$L_{AdaDelta}$ and $L_{Adam}$ with the number of iterations.




**Linear Classification and Stochastic Gradient Descent**

a. Load the training set and validation set.

```
def get_data(filename):
    data = load_svmlight_file(filename,123)
    return data[0], data[1]

#Import data
X_train, y_train = get_data("a9a")
X_test, y_test = get_data("a9a.t")
```

b. Initialize SVM model parameters to zero.

c. Select the loss function and calculate its derivation.

$$LossFunction = \frac{\|w\|^2}{2} + \frac{C}{m}\sum_{i=1}^{m} max(0, 1 - y_i(W^T x_i + b))$$

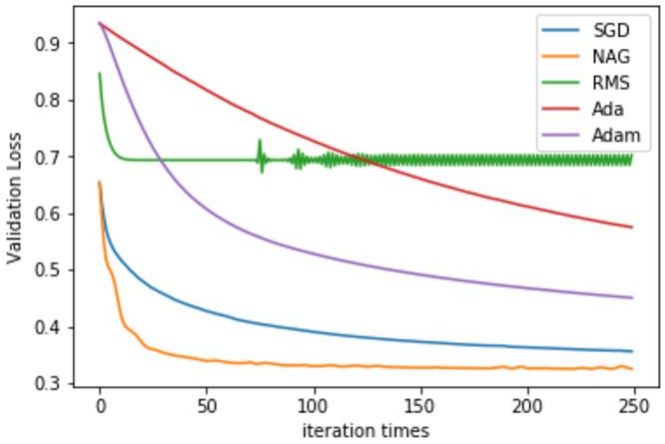
d. Calculate gradient towards loss function from partial samples.

f. Update model parameters using different optimized methods (NAG, RMSProp, AdaDelta and Adam).

g. Select the appropriate threshold, mark the sample whose predict scores greater than the threshold as positive, on the contrary as negative. Predict under validation set and get the different optimized method loss $L_{NAG}$ , $L_{RMSProp}$, $L_{AdaDelta}$ and$L_{Adam}$.

h. Repeat step 4 to 6 for several times, and drawing graph of $L_{NAG}$ , $L_{RMSProp}$ ,

$L_{AdaDelta}$ and $L_{Adam}$ with the number of iterations.




**The initialization method of model parameters:**

Parameters

| | argument | value |
|---|---|---|
| | Num_iters | 250 |
| | Batch_size | 130 |
| SGD | $\eta$ | 0.01 |
| | c | 0.5 |
| NAG | $\eta$ | 0.01 |
| | $c$ | 0.5 |
| | gama | 0.9 |
| RMSProp | $\eta$ | 0.001 |
| | c | 0.5 |
| | gama | 0.9 |
| | epsilon | 1E-8 |
| AdaDelta | $\eta$ | 0.01 |
| | c | 0.5 |
| | gama | 0.95 |
| | epsilon | 1E-8 |
| Adam | $\eta$ | 0.01 |
| | c | 0.5 |
| | gama | 0.999 |
| | beta | 0.9 |

## Ⅳ. CONCLUTION

Logistic regression is usually used for the binary classification model while linear regression is used to fit data, and the objective function is plain and error. In classification case , it's allowed to set a classification threshold manually so that the

classification would be totally different.

Both the logistic regression and SVM are used in classification problems. The difference between them is that logistic regression uses logistical loss to train the model, while the SVM uses the hinge loss.

These five optimized methods have their own advantages and disadvantages. In this experiment, the NAG has the best performance, with little loss and training the model fast. RMS can only uses a small learning rate, or the parameters are hard to converge.