The Experiment Report of
Machine Learning

College Software College

Subject Software Engineering

Members JINGJING HU（胡菁菁）

Student ID 201530611609

E-mail 767816313@qq.com

Tutor Prof.Mingkui Tan

Date submitted 2017.12 .15

1. Topic: Logistic Regression, Linear Classification and Stochastic Gradient Descent

2. Time: 2017.12.15

3. Reporter: JINGJING HU

4. Purposes:

1) Compare and understand the difference between gradient descent and stochastic gradient descent.
2) Compare and understand the differences and relationships between Logistic regression and linear classification.
3) Further understand the principles of SVM and practice on larger data.

5. Data sets and data analysis:

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

6. Experimental steps:

Logistic Regression and Stochastic Gradient Descent

1) Load the training set and validation set.
2) Initialize logistic regression model parameters, you can consider initializing zeros, random numbers or normal distribution.
3) Select the loss function and calculate its derivation, find more

detail in PPT.

4) Calculate gradient G toward loss function from partial samples.
5) Update model parameters using different optimized methods(NAG，RMSProp，AdaDelta and Adam).
6) 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}.

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

1) Load the training set and validation set.
2) Initialize SVM model parameters, you can consider initializing zeros, random numbers or normal distribution.
3) Select the loss function and calculate its derivation, find more detail in PPT.
4) Calculate gradient G toward loss function from partial samples.
5) Update model parameters using different optimized

methods(NAG， RMSProp， AdaDelta and Adam).

6) 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} .

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.

7. Code:

<u>*Logistic Regression and Stochastic Gradient Descent*</u>

1) NAG

```
%matplotlib inline
from sklearn import datasets as ds
from sklearn.model_selection import train_test_split
import math
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt


#load the file
x_train,y_train =
ds.load_svmlight_file("/Users/humeng/Desktop/a9a.txt")
```

```
x_validation,y_validation =
ds.load_svmlight_file("/Users/humeng/Desktop/a9a.t")

# Add bias
x_train = np.hstack((x_train.toarray(), np.ones((y_train.shape[0], 1))))
x_validation = np.hstack((x_validation.toarray(), np.zeros(
        (x_validation.shape[0], max(x_train.shape[1] - 1,
x_validation.shape[1]) - x_validation.shape[1]))))
x_validation =
np.hstack((x_validation ,np.ones((x_validation.shape[0],1))))

#change data format
y_validation = y_validation.reshape(-1,1)
y_train = y_train.reshape(-1,1)

#parameter initialization
batch_size = 100
learning_rate = 0.05
epochs =2500
gamma = 0.9
threshold = 0.0
best_acc = 0.0
```

```
w = np.zeros((x_train.shape[1],1))
v = np.zeros((x_train.shape[1],1))
l_NAG = []

#sigmoid function
def sigmoid(n):
    return 1.0 / (1 + np.exp(-n))

def gradient(w,x,y):
    h = sigmoid(y * np.dot(x, w))
    grad = -np.sum(np.multiply(1 - h, y * x), axis = 0).T / x.shape[0]
    return grad.reshape(-1, 1)

def loss(w,x,y):
    h2 = sigmoid(np.multiply(y, np.dot(x, w)))
    l = -np.sum(np.log(h2)) / x.shape[0]
    return l

for i in range(epochs):
    # Choose batch samples randomly
    batch_sample = np.random.choice(x_train.shape[0], batch_size)
    batch_x = x_train[batch_sample]
```

```
            batch_y = y_train[batch_sample]
            #update parameters
            v = gamma * v + learning_rate * gradient(w - gamma *
v ,batch_x ,batch_y)
            w = w - v
            # mark the sample with predict scores
            y_pred = np.ones((y_validation.shape[0],1))
            for j in range (x_validation.shape[0]):
                score = np.dot(x_validation[j,:],w)
                if score < threshold :
                    y_pred[j] = -1
            #calculate the accuracy
            acc = np.mean(y_pred == y_validation)
            if acc > best_acc:
                best_acc = acc
            l_NAG.append(loss(w,x_validation,y_validation))

    print('beat acc is %f' %best_acc)



    #print the figure
    plt.rcParams['figure.figsize'] = (10.0, 8.0)
```

```
plt.plot(np.arange(epochs),l_NAG,label='NAG loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc='best')
plt.grid()
plt.show()
```

2) RMSProp

…略…

```
#parameter initialization
best_acc = 0.0
batch_size = 100
learning_rate = 0.001
epochs = 2500
gamma = 0.9
epsilon = 1e-8
threshold = 0.0
w = np.zeros((x_train.shape[1],1))
G = np.zeros((x_train.shape[1],1))
l_RMSProp = []
```

…略…

```
#update parameters
g = gradient(w ,batch_x ,batch_y)
```

```
G = gamma * G + (1 - gamma) * (g**2)
w = w - learning_rate * g / np.sqrt(G + epsilon)
```

…略…

3) AdaDelta

…略…

```
#parameter initialization
best_acc = 0.0
batch_size = 100
epochs = 2500
gamma = 0.95
epsilon = 1e-6
threshold = 0.0
dw = np.zeros((x_train.shape[1],1))
w = np.zeros((x_train.shape[1],1))
G = np.zeros((x_train.shape[1],1))
l_AdaDelta = []
```

…略…

```
#update parameters
g = gradient(w ,batch_x ,batch_y)
G = gamma * G + (1 - gamma) * (g**2)
a = np.sqrt(dw + epsilon) * g / np.sqrt(G + epsilon)
w = w - a
```

dw = gamma * dw + (1 - gamma) * (dw**2)

…略…

4) Adam

…略…

```
#parameter initialization

best_acc = 0.0

batch_size = 100

t = 1

learning_rate = 0.002

epochs = 2500

gamma = 0.999

beta = 0.9

epsilon = 1e-8

threshold = 0.0

m = np.zeros((x_train.shape[1],1))

w = np.zeros((x_train.shape[1],1))

G = np.zeros((x_train.shape[1],1))

l_Adam = []
```

…略…

```
#update parameters

g = gradient(w ,batch_x ,batch_y)

m = beta * m + (1- beta) * g
```

```
G = gamma * G + (1 - gamma) * (g**2)
a = learning_rate * np.sqrt(1 - gamma**t) / np.sqrt(1 - beta**t)
w = w - a * m / np.sqrt(G + epsilon)
t += 1
…略…
```

Linear Classification and Stochastic Gradient Descent

1) NAG

```
%matplotlib inline
from sklearn import datasets as ds
from sklearn.model_selection import train_test_split
import math
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

#load the file
x_train,y_train =
ds.load_svmlight_file("/Users/humeng/Desktop/a9a.txt")
x_validation,y_validation =
ds.load_svmlight_file("/Users/humeng/Desktop/a9a.t")
```

```
# Add bias
x_train = np.hstack((x_train.toarray(), np.ones((y_train.shape[0], 1))))
x_validation = np.hstack((x_validation.toarray(), np.zeros(
        (x_validation.shape[0], max(x_train.shape[1] - 1,
x_validation.shape[1]) - x_validation.shape[1]))))
x_validation =
np.hstack((x_validation ,np.ones((x_validation.shape[0],1))))

#change data format
y_validation = y_validation.reshape(-1,1)
y_train = y_train.reshape(-1,1)

#parameter initialization
batch_size = 10
C = 10
learning_rate = 1e-5
epochs =1000
gamma = 0.9
threshold = 0.0
best_acc = 0.0
w = np.zeros((x_train.shape[1],1))
v = np.zeros((x_train.shape[1],1))
```

```
l_NAG = []
def gradient(w,C,x,y):
    condition = 1 - np.multiply(y, np.dot(x, w))
    y[condition < 0] = 0
    grad = w - C * np.dot(x.T, y)
    grad[-1] -= w[-1]
    return grad

def loss(w,C,x,y):
    hinge = np.maximum(0, 1 - np.multiply(y,np.dot(x,w)))
    l = float(0.5 * np.dot(w.T,w) + C * np.sum(hinge) / x.shape[0])
    return l

for i in range(epochs):
    # Choose batch samples randomly
    batch_sample = np.random.choice(x_train.shape[0], batch_size)
    batch_x = x_train[batch_sample]
    batch_y = y_train[batch_sample]
    #update parameters
    v = gamma * v + learning_rate * gradient(w - gamma *
v ,C ,batch_x ,batch_y)
    w = w - v
```

```
        # mark the sample with predict scores
        y_pred = np.ones((y_validation.shape[0],1))
        for j in range (x_validation.shape[0]):
            score = np.dot(x_validation[j,:],w)
            if score < threshold :
                y_pred[j] = -1
        acc = np.mean(y_pred == y_validation)
        if acc > best_acc:
            best_acc = acc
        l_NAG.append(loss(w,C,x_validation,y_validation))
print('beat acc is %f' %best_acc)


#print the figure
plt.rcParams['figure.figsize'] = (10.0, 8.0)
plt.plot(np.arange(epochs),l_NAG,label='NAG loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(loc='best')
plt.grid()
plt.show()
```

2) RMSProp

…略…

```
#parameter initialization
batch_size = 10
C = 10
learning_rate = 0.001
epochs = 1000
gamma = 0.9
epsilon = 1e-8
threshold = 0.0
best_acc = 0.0
w = np.zeros((x_train.shape[1],1))
G = np.zeros((x_train.shape[1],1))
l_RMSProp = []
…略…
#update parameters
    g = gradient(w ,C ,batch_x ,batch_y)
    G = gamma * G + (1 - gamma) * (g**2)
    w = w - learning_rate * g / np.sqrt(G + epsilon)
…略…
```

3) AdaDelta

```
…略…
#parameter initialization
best_acc = 0.0
```

```
batch_size = 10

C = 10

epochs = 1000

gamma = 0.95

epsilon = 1e-6

threshold = 0.0

dw = np.zeros((x_train.shape[1],1))

w = np.zeros((x_train.shape[1],1))

G = np.zeros((x_train.shape[1],1))

l_AdaDelta = []
```

…略…

```
#update parameters

g = gradient(w ,C ,batch_x ,batch_y)

G = gamma * G + (1 - gamma) * (g**2)

a = np.sqrt(dw + epsilon) * g / np.sqrt(G + epsilon)

w = w - a

dw = gamma * dw + (1 - gamma) * (dw**2)
```

…略…

4) Adam

…略…

```
#parameter initialization

best_acc = 0.0
```

```
batch_size = 100
C = 10
t = 1
learning_rate = 0.0005
epochs = 1000
gamma = 0.999
beta = 0.9
epsilon = 1e-8
threshold = 0.0
m = np.zeros((x_train.shape[1],1))
w = np.zeros((x_train.shape[1],1))
G = np.zeros((x_train.shape[1],1))
l_Adam = []
```

…略…

```
#update parameters
g = gradient(w ,C ,batch_x ,batch_y)
m = beta * m + (1- beta) * g
G = gamma * G + (1 - gamma) * (g**2)
a = learning_rate * np.sqrt(1 - gamma**t) / np.sqrt(1 - beta**t)
w = w - a * m / np.sqrt(G + epsilon)
t += 1
```

…略…

(Fill in the contents of 8-11 respectively for logistic regression and linear classification)

8. The initialization method of model parameters:

Logistic Regression and Stochastic Gradient Descent

Initializing zero

Linear Classification and Stochastic Gradient Descent

Initializing zero

9. The selected loss function and its derivatives:

Logistic Regression and Stochastic Gradient Descent

Loss: $\frac{1}{\mathrm{N}}\sum_{\mathrm{n}=1}^{\mathrm{N}} \ln(1+e^{-\mathrm{y_n}\cdot\mathrm{w^T x_n}})$

Gradient :$\frac{1}{\mathrm{N}}\sum_{\mathrm{n}=1}^{\mathrm{N}} \frac{\mathrm{y_n}\cdot\mathrm{x_n}}{1+e^{\mathrm{y_n}\cdot\mathrm{w^T x_n}}}$

Linear Classification and Stochastic Gradient Descent

Loss: $\min_{w,b} \frac{1}{2} \| \boldsymbol{w} \|_2^2 + C\sum_{i=1}^{n} \max(0, 1-y_i(\mathrm{w}^{\mathrm{T}}x_i+b))$

Gradient : $\boldsymbol{w} - C\boldsymbol{X}^T\boldsymbol{y}$

10. Experimental results and curve:(Fill in this content for various methods of gradient descent respectively)

Logistic Regression and Stochastic Gradient Descent

Hyper-parameter selection:

1) NAG

epochs =2500, gamma = 0.9, threshold = 0.0

2) RMSProp

learning_rate = 0.001, epochs = 2500, gamma = 0.9, epsilon = 1e-8, threshold = 0.0

3) AdaDelta

epochs = 2500, gamma = 0.95, epsilon = 1e-6, threshold = 0.0

4) Adam

learning_rate = 0.002, epochs = 2500, gamma = 0.999, beta = 0.9, epsilon = 1e-8, threshold = 0.0

Predicted Results (Best Results):

1) NAG

Best acc is 0.852405

2) RMSProp

Best acc is 0.852159

3) AdaDelta

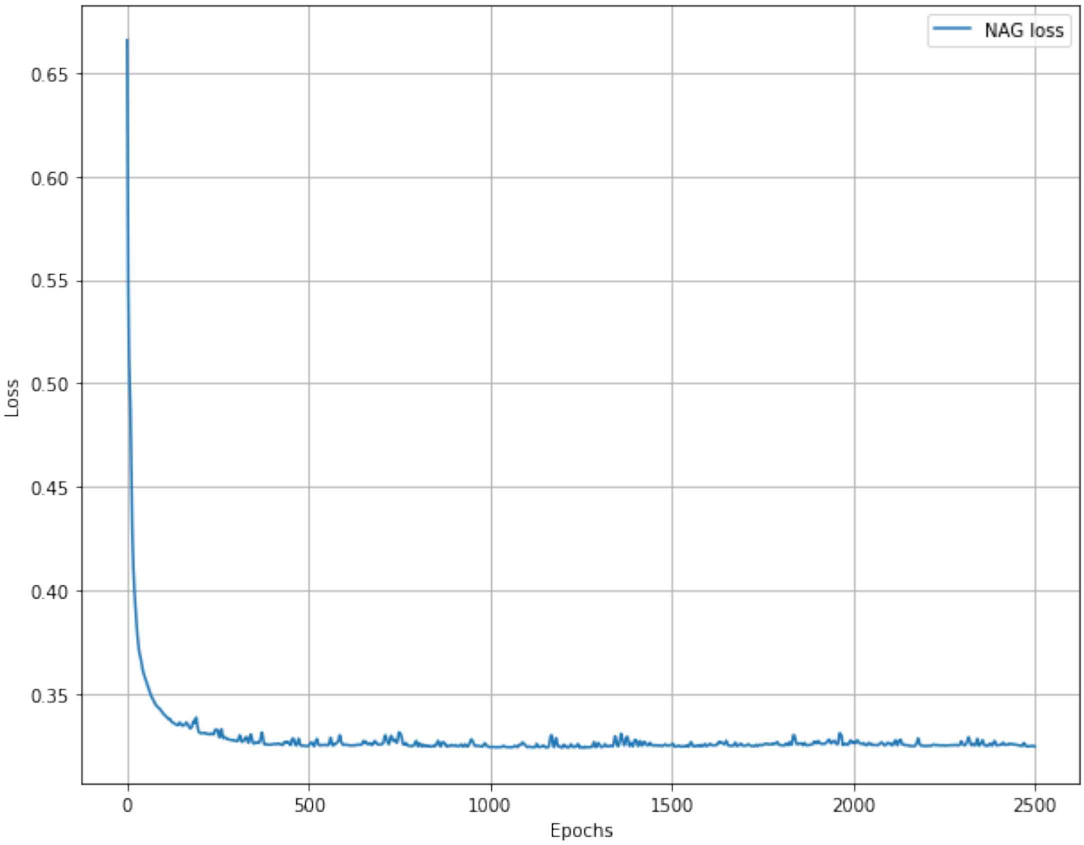
Beat acc is 0.852466

4) Adam

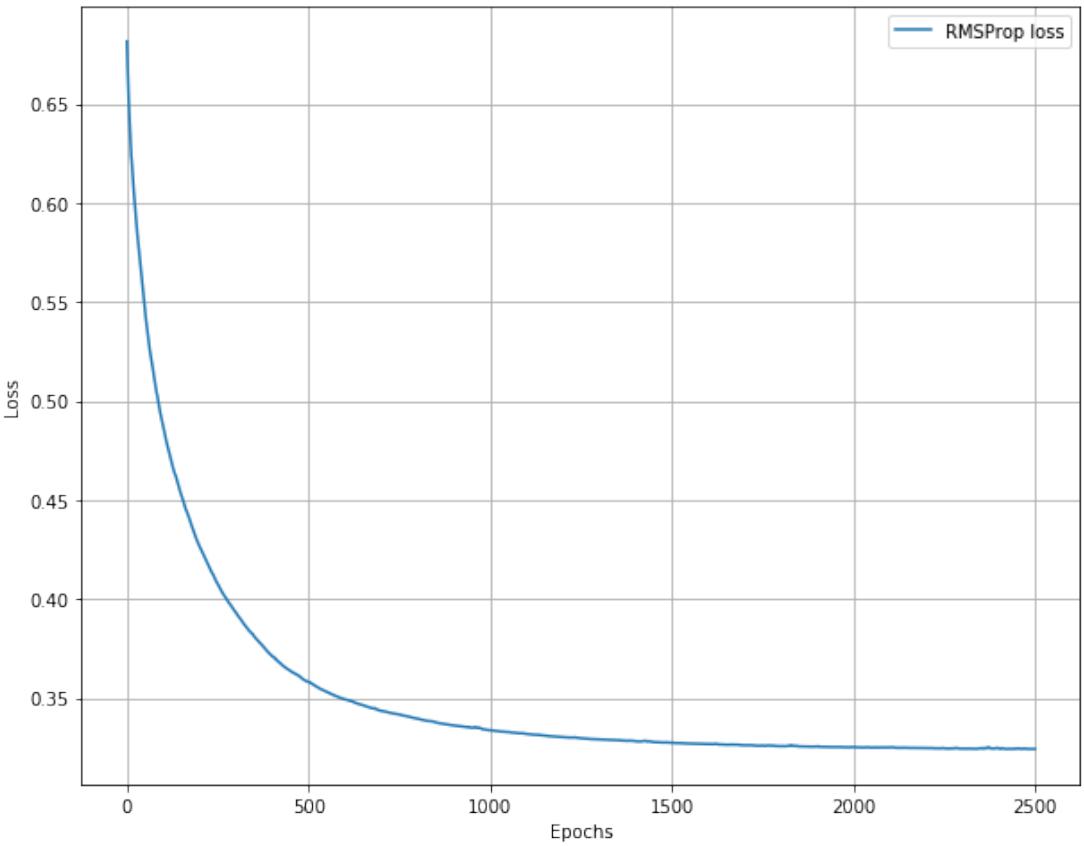
Best acc is 0.851852

Loss curve:

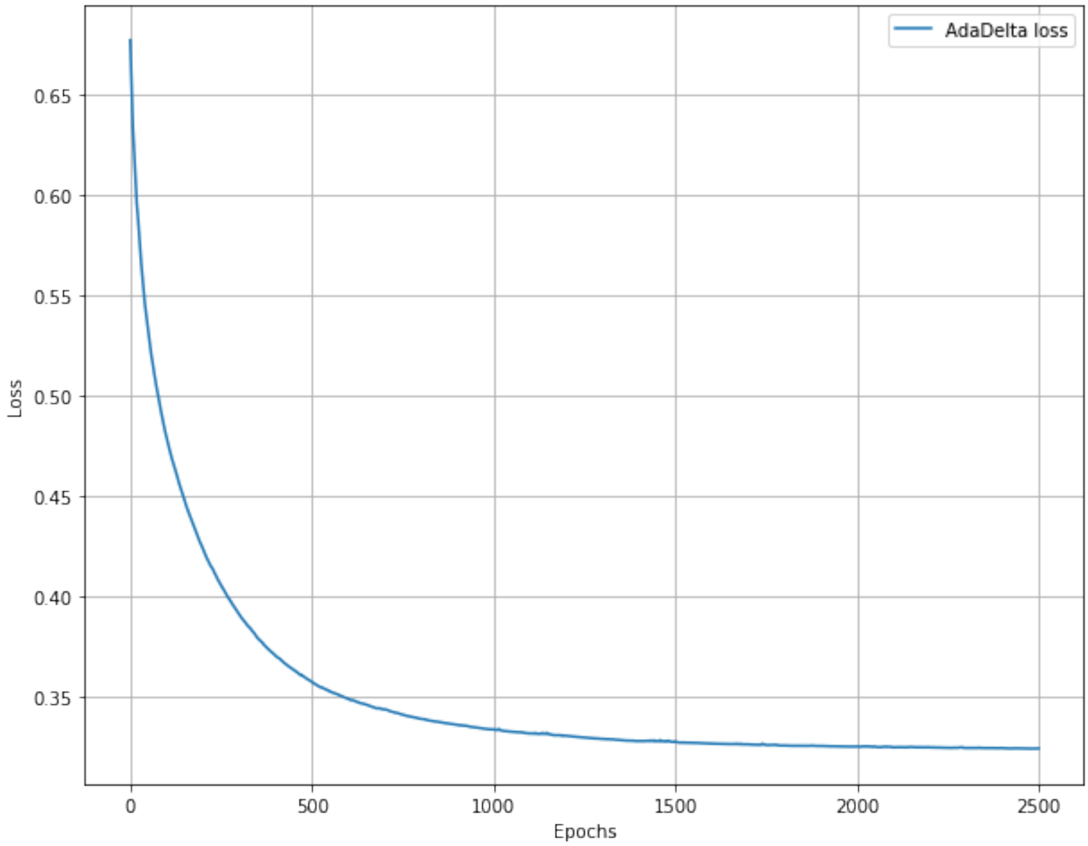
1) NAG


NAG loss
0.65
0.60
0.55
0.50
0.45
0.40
0.35
Loss
0
500
1000
1500
2000
2500
Epochs

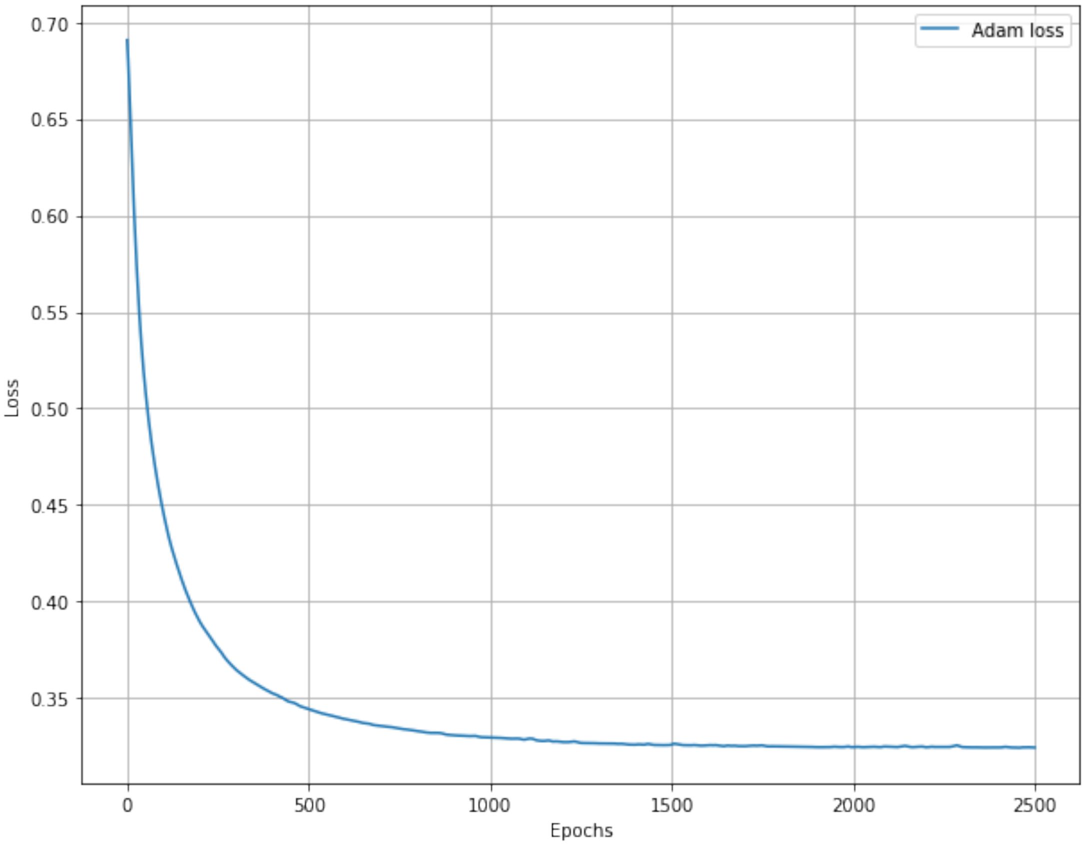
2) RMSProp


RMSProp loss
0.65
0.60
0.55
0.50
0.45
0.40
0.35
Loss
0
500
1000
1500
2000
2500
Epochs

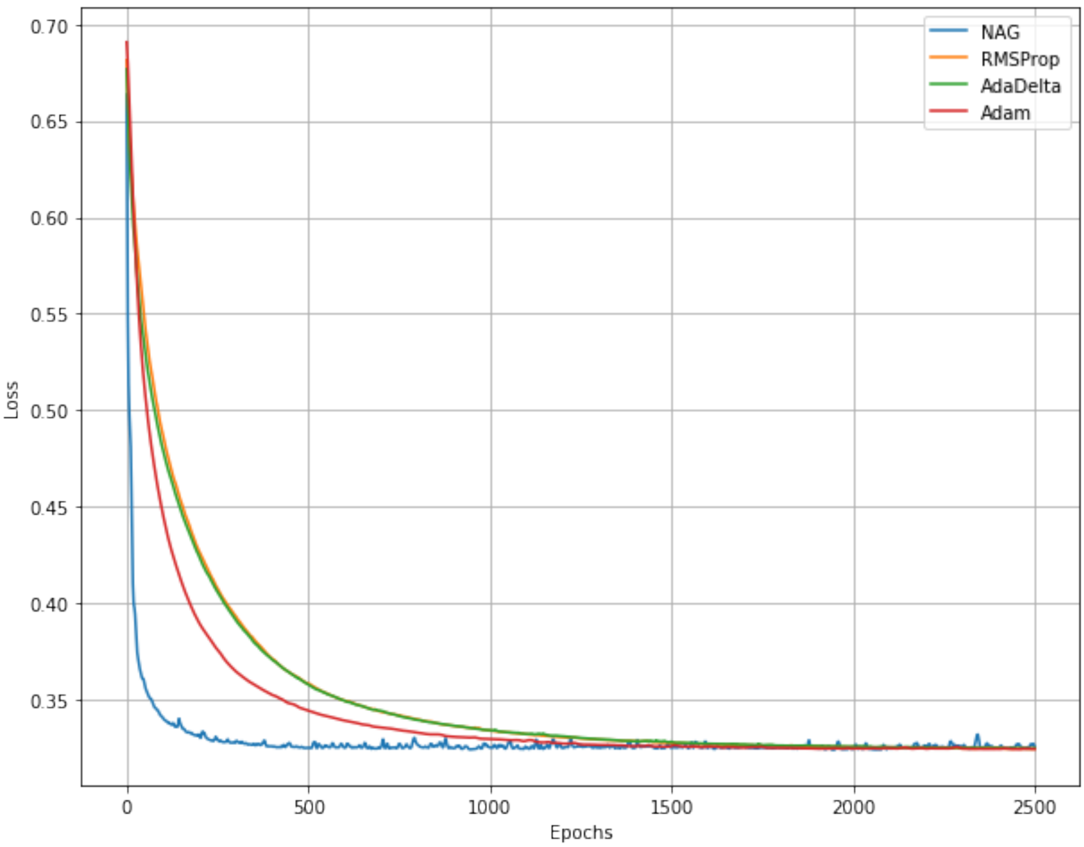
3) AdaDelta



4) Adam



5) All



<u>Linear Classification and Stochastic Gradient Descent</u>

Hyper-parameter selection:

1) NAG

learning_rate = 1e-5, epochs =1000, gamma = 0.9, threshold = 0.0, C = 10

2) RMSProp

learning_rate = 0.001, epochs = 1000, gamma = 0.9, epsilon = 1e-8, threshold = 0.0, C = 10

3) AdaDelta

epochs = 1000, gamma = 0.95, epsilon = 1e-6, threshold = 0.0

4) Adam

learning_rate = 0.0005, epochs = 1000, gamma = 0.999, beta = 0.9, epsilon = 1e-8, threshold = 0.0

Predicted Results (Best Results):

1) NAG

Best acc is 0.827468

2) RMSProp

Best acc is 0.807260

3) AdaDelta

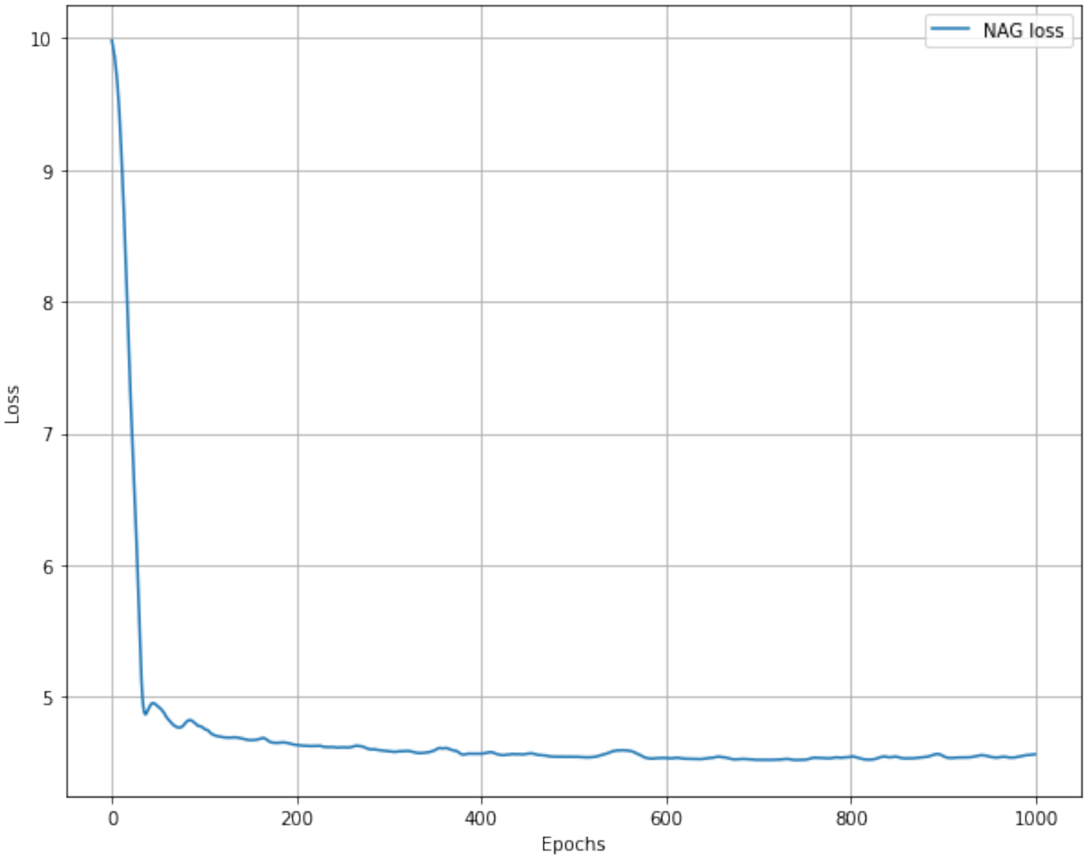
Beat acc is 0.795283

4) Adam

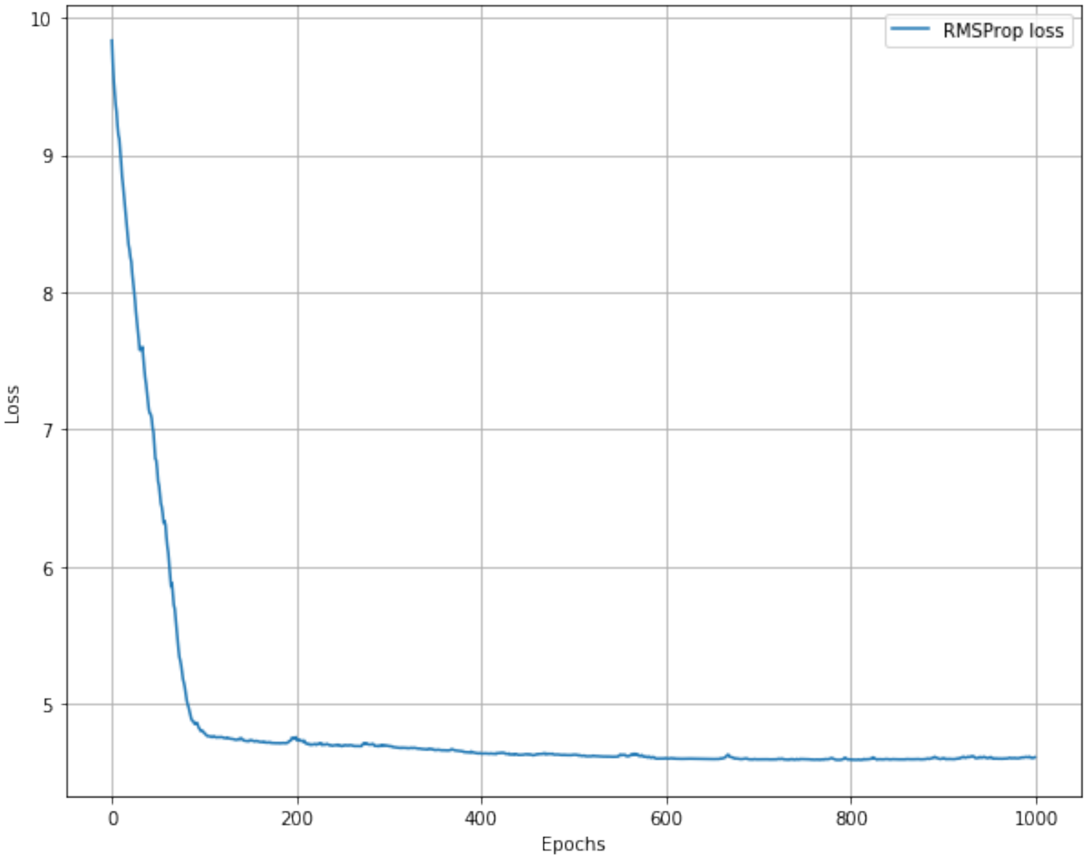
Best acc is 0.802285

Loss curve:

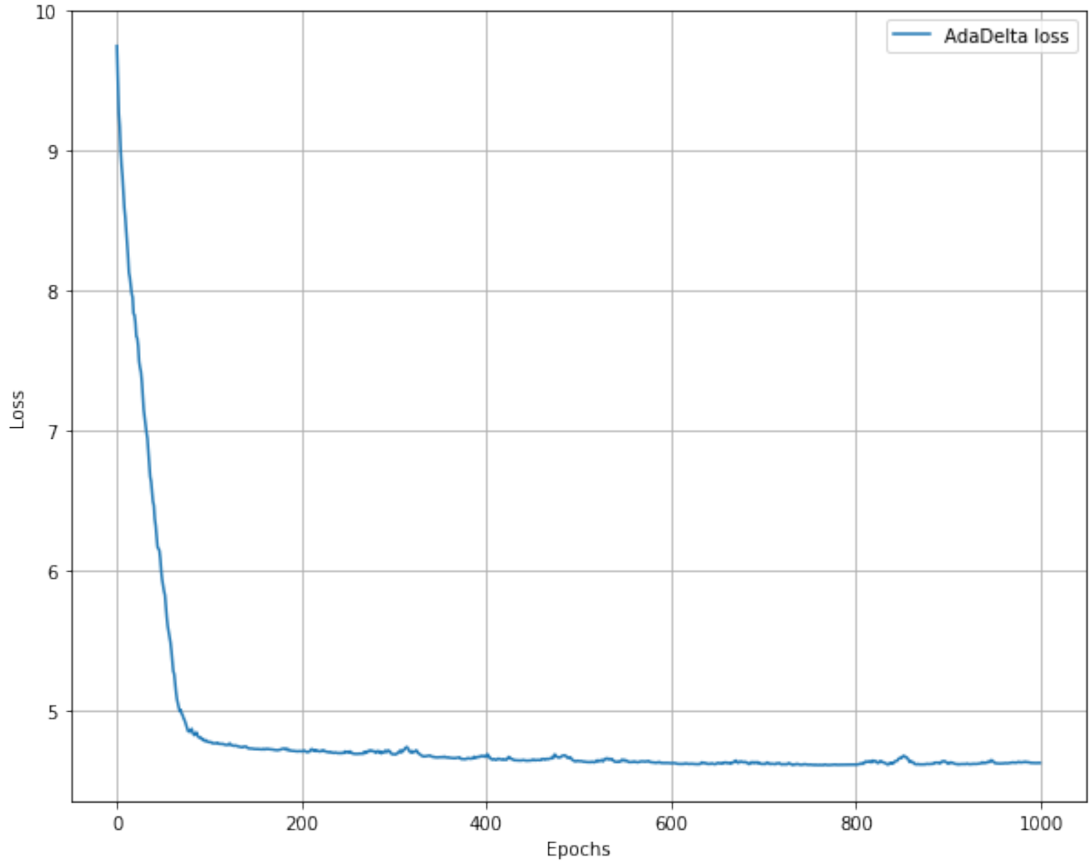
1) NAG



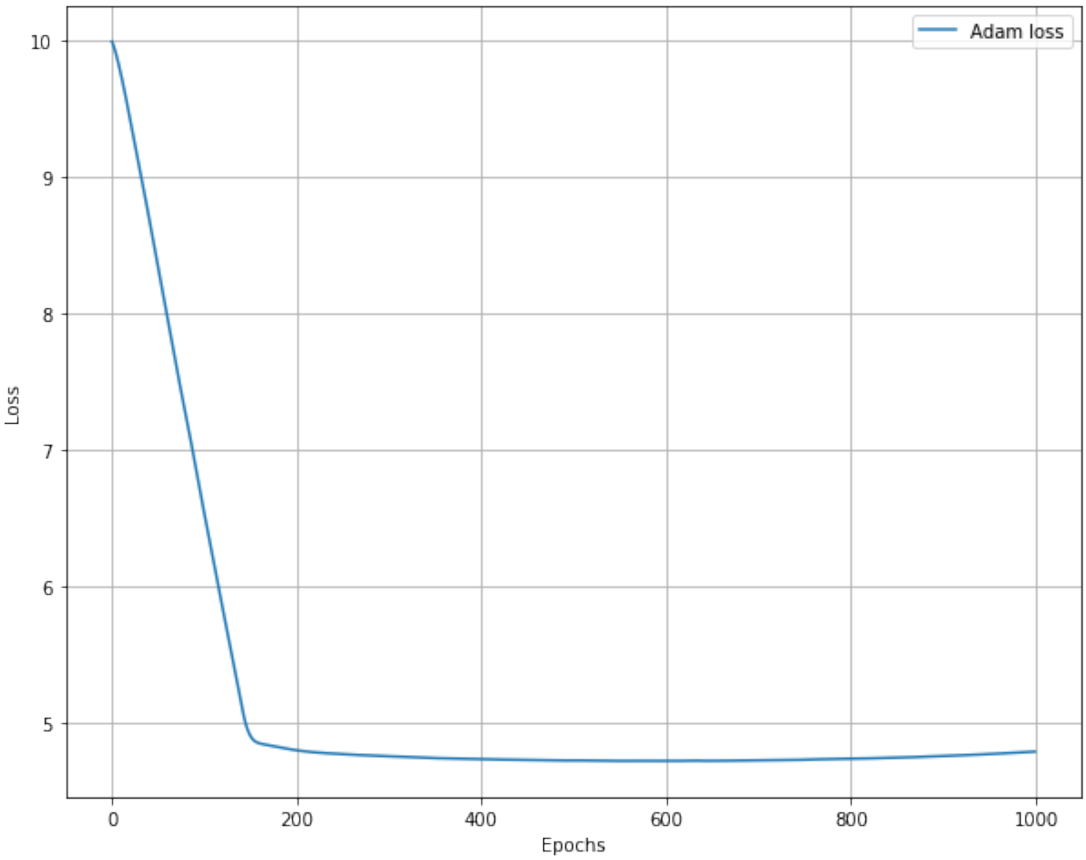
2) RMSProp



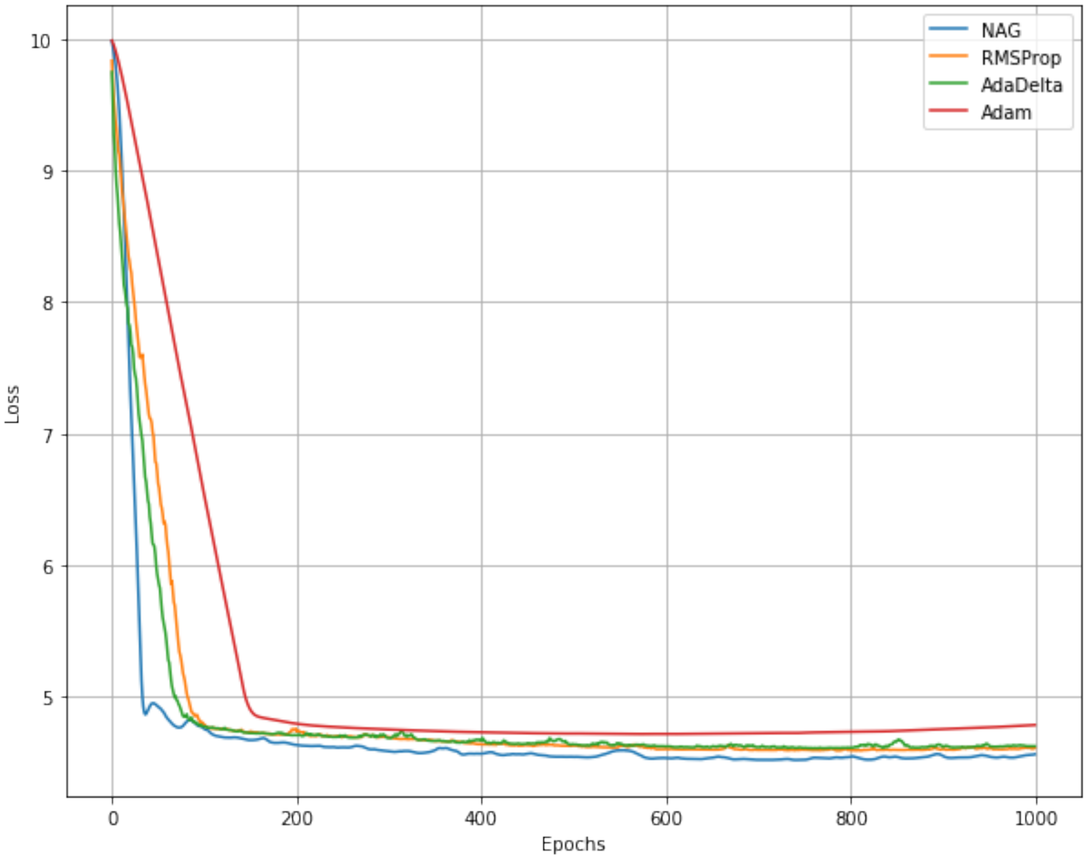
3) AdaDelta



4) Adam



5) All



11. Results analysis:

Logistic Regression and Stochastic Gradient Descent

Adam shows the best performance, AdaDelta works as well as RMSProp, NAG converges very fast, but it's loss vibrates slightly, which might occur for the other parameters.

Linear Classification and Stochastic Gradient Descent

NAG shows the best performance and converge fast too. This may be due to the simple linear model.

12. Similarities and differences between logistic regression and linear classification:

1) logistic regression uses sigmoid function while linear classification uses linear function.
2) They are both the classifier essentially.

13. Summary:

It is a little hard for me to understand what the difference between this four optimized methods is but easy to programme with the formula. By doing so, I can realize the SDG better.