The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Shihhong Chen

Supervisor:
Qingyao Wu

Student ID：201530611234

Grade:
Undergraduate

December 9, 2017

Logistic Regression, Linear Classification and Stochastic Gradient Descent

Abstract—This experiment is designed for students to have a comprehension understanding of logistic regression, linear classification and stochastic gradient descent.

I. Introduction

The motivation of experiment:
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.

II. Methods and Theory

Logistic Regression and Stochastic Gradient Descent

1. Load the training set and validation set.
2. Initalize logistic regression model parameters, you can consider initalizing 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}.
7. Repeate 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. Initalize SVM model parameters, you can consider initalizing 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}.
7. Repeate 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.

III. Experiment

A. Dataset

Experiment uses a9a of LIBSVM Data, including 32561/16281(testing) samples and each sample has 123/123 (testing) features. Please download the training set and validation set.

B. Implementation

RegressionExperiment:

```
import numpy
import random
import jupyter
import math
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import train_test_split
from matplotlib import pyplot

# x为 1*d 矩阵  y为数字
def compute_grad(x, y, w, C):
    gradient = (-y / (1+math.exp(y*x.dot(w.T))))*x + C*w
    return gradient

# x为m*d   y为长度m的list
def compute_loss(x, y, w, C, random_i):
    loss = 0
    n = len(random_i)
    for m in range(n):
        loss += math.log(1+math.exp(-y[m]*(x[m,:].dot(w.T))))
    loss = loss/n + C/2 * w.dot(w.T)
    return loss[0,0]

def NAG_train(x, y, x_test, y_test, w, C, lr, gamma, threshold, iteration):
    vt = numpy.zeros(w.shape)
    loss_history = []
    test_loss_history = []
    random_index = []
    random_test_index = []
    for i in range(iteration):
        random_num = random.randint(0, x.shape[0]-1)
        random_test_num = random.randint(0, x_test.shape[0]-1)
        random_index.append(random_num)
        random_test_index.append(random_test_num)
    for i in range(iteration):
        gradient = compute_grad(x[random_index[i],:], y[random_index[i]], w-gamma*vt, C)
        vt = gamma*vt - lr*gradient
        w += vt
        loss = compute_loss(x, y, w, C, random_index)
        loss_history.append(loss)
        test_loss_history.append(compute_loss(x_test, y_test, w, C, random_test_index))
        if loss < threshold :
            break
    return w, loss_history, test_loss_history
```

```python
def RMSProp_train(x, y, x_test, y_test, w, C, lr, gamma, threshold, iteration):
    Gt = 0
    loss_history = []
    test_loss_history = []
    random_index = []
    random_test_index = []
    for i in range(iteration):
        random_num = random.randint(0, x.shape[0]-1)
        random_test_num = random.randint(0, x_test.shape[0]-1)
        random_index.append(random_num)
        random_test_index.append(random_test_num)
    for i in range(iteration):
        gradient = compute_grad(x[random_index[i],:], y[random_index[i]], w, C)
        Gt = gamma*Gt + (1-gamma)*gradient.dot(gradient.T)
        w -= lr * gradient / math.sqrt(Gt+1e-8)
        loss = compute_loss(x, y, w, C, random_index)
        loss_history.append(loss)
        test_loss_history.append(compute_loss(x_test, y_test, w, C, random_test_index))
        if loss < threshold :
            break
    return w, loss_history, test_loss_history

def AdaDelta_train(x, y, x_test, y_test, w, C, lr, gamma, threshold, iteration):
    Gt = 0
    variable_t = 0
    loss_history = []
    test_loss_history = []
    random_index = []
    random_test_index = []
    for i in range(iteration):
        random_num = random.randint(0, x.shape[0]-1)
        random_test_num = random.randint(0, x_test.shape[0]-1)
        random_index.append(random_num)
        random_test_index.append(random_test_num)
    for i in range(iteration):
        gradient = compute_grad(x[random_index[i],:], y[random_index[i]], w, C)
        Gt = gamma*Gt + (1-gamma)*gradient.dot(gradient.T)
        variable_w = - math.sqrt(variable_t + 1e-8) * gradient / math.sqrt(Gt + 1e-8)
        w += variable_w
        variable_t = gamma*variable_t + (1-gamma)*variable_w.dot(variable_w.T)
        loss = compute_loss(x, y, w, C, random_index)
        loss_history.append(loss)
        test_loss_history.append(compute_loss(x_test, y_test, w, C, random_test_index))
        if loss < threshold :
            break
    return w, loss_history, test_loss_history
```

```python
def Adam_train(x, y, x_test, y_test, w, C, lr, gamma, threshold, iteration):
    Gt = 0
    moment = numpy.zeros((1, x.shape[1]))
    B = 0.9
    loss_history = []
    test_loss_history = []
    random_index = []
    random_test_index = []
    for i in range(iteration):
        random_num = random.randint(0, x.shape[0]-1)
        random_test_num = random.randint(0, x_test.shape[0]-1)
        random_index.append(random_num)
        random_test_index.append(random_test_num)
    for i in range(iteration):
        gradient = compute_grad(x[random_index[i],:], y[random_index[i]], w, C)
        moment = B*moment + (1-B)*gradient
        Gt = gamma*Gt + (1-gamma)*gradient.dot(gradient.T)
        a = lr * math.sqrt(1 - pow(gamma, iteration)) / (1-pow(B, iteration))
        w -= a * moment / math.sqrt(Gt + 1e-8)
        loss = compute_loss(x, y, w, C, random_index)
        loss_history.append(loss)
        test_loss_history.append(compute_loss(x_test, y_test, w, C, random_test_index))
        if loss < threshold :
            break
    return w, loss_history, test_loss_history
```

```python
x, y_train = load_svmlight_file("a9a.txt")
x_train = x.toarray()
x, y_test = load_svmlight_file("a9a_test.txt")
x_test = x.toarray()

X_train = numpy.hstack([x_train, numpy.ones((x_train.shape[0], 1))])
X_test = numpy.hstack([x_test, numpy.zeros((x_test.shape[0], 1))])
X_test = numpy.hstack([X_test, numpy.ones((x_test.shape[0], 1))])

iteration = 3000
# NAG
NAG_w = numpy.zeros((1, X_train.shape[1]))
NAG_w, NAG_loss_history, NAG_test_loss_history = NAG_train(X_train, y_train, X_test, y_test, NAG_w, 0.3, 0.001, 0.9, 0.001, iteration)

# RMSProp
RMS_w = numpy.zeros((1, X_train.shape[1]))
RMS_w, RMS_loss_history, RMS_test_loss_history = RMSProp_train(X_train, y_train, X_test, y_test, RMS_w, 0.3, 0.001, 0.9, 0.001, iteration)

# AdaDelta
AdaDelta_w = numpy.zeros((1, X_train.shape[1]))
AdaDelta_w, AdaDelta_loss_history, AdaDelta_test_loss_history = AdaDelta_train(X_train, y_train, X_test, y_test, AdaDelta_w, 0.3, 0.001, 0.9, 0.001, iteration)

#Adam
Adam_w = numpy.zeros((1, X_train.shape[1]))
Adam_w, Adam_loss_history, Adam_test_loss_history = Adam_train(X_train, y_train, X_test, y_test, Adam_w, 0.3, 0.001, 0.9, 0.001, iteration)


# pyplot.plot(NAG_loss_history, label = 'loss')
pyplot.plot(NAG_test_loss_history, label = 'NAG_validation_loss')
# pyplot.plot(RMS_loss_history, label = 'loss')
pyplot.plot(RMS_test_loss_history, label = 'RMSProp_validation_loss')
# pyplot.plot(AdaDelta_loss_history, label = 'loss')
pyplot.plot(AdaDelta_test_loss_history, label = 'AdaDelta_validation_loss')
# pyplot.plot(Adam_loss_history, label = 'loss')
pyplot.plot(Adam_test_loss_history, label = 'Adam_validation_loss')
pyplot.legend(loc='upper right')
pyplot.ylabel('loss')
pyplot.xlabel('iteration')
pyplot.title('graph')
pyplot.show()
```

ClassificationExperiment:

```python
import numpy
import random
import jupyter
import math
from sklearn.datasets import load_svmlight_file
from sklearn.model_selection import train_test_split
from matplotlib import pyplot

# x为 1*d 矩阵  y为数字
def compute_grad(x, y, w):
    gradient = x * (y - x.dot(w.T))
    return gradient

# x为m*d   y为长度m的list
def compute_loss(x, y, w, random_i):
    loss = 0
    a = len(random_i)
    for m in range(a):
        loss += 0.5 * ((y[random_i[m]] - x[random_i[m],:].dot(w.T)) ** 2)
    return loss/a

def NAG_train(x, y, x_test, y_test, w, C, lr, gamma, threshold, iteration):
    vt = numpy.zeros(w.shape)
    loss_history = []
    test_loss_history = []
    random_index = []
    random_test_index = []
    for i in range(iteration):
        random_num = random.randint(0, x.shape[0]-1)
        random_test_num = random.randint(0, x_test.shape[0]-1)
        random_index.append(random_num)
        random_test_index.append(random_test_num)
    for i in range(iteration):
        gradient = compute_grad(x[random_index[i],:], y[random_index[i]], w-gamma*vt)
        vt = gamma*vt - lr*gradient
        w -= vt
        loss = compute_loss(x, y, w, random_index)
        loss_history.append(loss)
        test_loss_history.append(compute_loss(x_test, y_test, w, random_test_index))
        if loss < threshold :
            break
    return w, loss_history, test_loss_history
```

```python
def RMSProp_train(x, y, x_test, y_test, w, C, lr, gamma, threshold, iteration):
    Gt = 0
    loss_history = []
    test_loss_history = []
    random_index = []
    random_test_index = []
    for i in range(iteration):
        random_num = random.randint(0, x.shape[0]-1)
        random_test_num = random.randint(0, x_test.shape[0]-1)
        random_index.append(random_num)
        random_test_index.append(random_test_num)
    for i in range(iteration):
        gradient = compute_grad(x[random_index[i],:], y[random_index[i]], w)
        Gt = gamma*Gt + (1-gamma)*gradient.dot(gradient.T)
        w += lr * gradient / math.sqrt(Gt+1e-8)
        loss = compute_loss(x, y, w, random_index)
        loss_history.append(loss)
        test_loss_history.append(compute_loss(x_test, y_test, w, random_test_index))
        if loss < threshold :
            break
    return w, loss_history, test_loss_history

def AdaDelta_train(x, y, x_test, y_test, w, C, lr, gamma, threshold, iteration):
    Gt = 0
    variable_t = 0
    loss_history = []
    test_loss_history = []
    random_index = []
    random_test_index = []
    for i in range(iteration):
        random_num = random.randint(0, x.shape[0]-1)
        random_test_num = random.randint(0, x_test.shape[0]-1)
        random_index.append(random_num)
        random_test_index.append(random_test_num)
    for i in range(iteration):
        gradient = compute_grad(x[random_index[i],:], y[random_index[i]], w)
        Gt = gamma*Gt + (1-gamma)*gradient.dot(gradient.T)
        variable_w = - math.sqrt(variable_t + 1e-8) * gradient / math.sqrt(Gt + 1e-8)
        w -= variable_w
        variable_t = gamma*variable_t + (1-gamma)*variable_w.dot(variable_w.T)
        loss = compute_loss(x, y, w, random_index)
        loss_history.append(loss)
        test_loss_history.append(compute_loss(x_test, y_test, w, random_test_index))
        if loss < threshold :
            break
    return w, loss_history, test_loss_history
```

```python
def Adam_train(x, y, x_test, y_test, w, C, lr, gamma, threshold, iteration):
    Gt = 0
    moment = numpy.zeros((1, x.shape[1]))
    B = 0.9
    loss_history = []
    test_loss_history = []
    random_index = []
    random_test_index = []
    for i in range(iteration):
        random_num = random.randint(0, x.shape[0]-1)
        random_test_num = random.randint(0, x_test.shape[0]-1)
        random_index.append(random_num)
        random_test_index.append(random_test_num)
    for i in range(iteration):
        gradient = compute_grad(x[random_index[i],:], y[random_index[i]], w)
        moment = B*moment + (1-B)*gradient
        Gt = gamma*Gt + (1-gamma)*gradient.dot(gradient.T)
        a = lr * math.sqrt(1 - pow(gamma, iteration)) / (1-pow(B, iteration))
        w += a * moment / math.sqrt(Gt + 1e-8)
        loss = compute_loss(x, y, w, random_index)
        loss_history.append(loss)
        test_loss_history.append(compute_loss(x_test, y_test, w, random_test_index))
        if loss < threshold :
            break
    return w, loss_history, test_loss_history
```

```python
x, y_train = load_svmlight_file("a9a.txt")
x_train = x.toarray()
x, y_test = load_svmlight_file("a9a_test.txt")
x_test = x.toarray()

X_train = numpy.hstack([x_train, numpy.ones((x_train.shape[0], 1))])
X_test = numpy.hstack([x_test, numpy.zeros((x_test.shape[0], 1))])
X_test = numpy.hstack([X_test, numpy.ones((x_test.shape[0], 1))])

iteration = 1000
# NAG
NAG_w = numpy.zeros((1, X_train.shape[1]))
NAG_w, NAG_loss_history, NAG_test_loss_history = NAG_train(X_train, y_train, X_test, y_test, NAG_w, 0.3, 0.001, 0.9, 0.001, 3000)

# RMSProp
RMS_w = numpy.zeros((1, X_train.shape[1]))
RMS_w, RMS_loss_history, RMS_test_loss_history = RMSProp_train(X_train, y_train, X_test, y_test, RMS_w, 0.3, 0.001, 0.9, 0.001, iteration)

# AdaDelta
AdaDelta_w = numpy.zeros((1, X_train.shape[1]))
AdaDelta_w, AdaDelta_loss_history, AdaDelta_test_loss_history = AdaDelta_train(X_train, y_train, X_test, y_test, AdaDelta_w, 0.3, 0.001, 0.9, 0.001, iteration)

#Adam
Adam_w = numpy.zeros((1, X_train.shape[1]))
Adam_w, Adam_loss_history, Adam_test_loss_history = Adam_train(X_train, y_train, X_test, y_test, Adam_w, 0.3, 0.001, 0.9, 0.001, iteration)


# pyplot.plot(NAG_loss_history, label = 'loss')
pyplot.plot(NAG_test_loss_history, label = 'NAG_validation_loss')
# pyplot.plot(RMS_loss_history, label = 'loss')
pyplot.plot(RMS_test_loss_history, label = 'RMSProp_validation_loss')
# pyplot.plot(AdaDelta_loss_history, label = 'loss')
pyplot.plot(AdaDelta_test_loss_history, label = 'AdaDelta_validation_loss')
# pyplot.plot(Adam_loss_history, label = 'loss')
pyplot.plot(Adam_test_loss_history, label = 'Adam_validation_loss')
pyplot.legend(loc='upper right')
pyplot.ylabel('loss')
pyplot.xlabel('iteration')
pyplot.title('graph')
pyplot.show()
```

Regression Experiment:

Loss function:

$$J(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\log(1+e^{-y_i \cdot \mathbf{w}^\top \mathbf{x}_i}) + \frac{\lambda}{2}||\mathbf{w}||_2^2$$

Derivative:

$$\frac{y_i \mathbf{x}_i}{1+e^{y_i \cdot \mathbf{w}^\top \mathbf{x}_i}} + \lambda \mathrm{w}$$

Classification Experiment:

Loss function:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(y^i - h_\theta(x^i))^2$$

$$h(\theta) = \sum_{j=0}^{n}\theta_j x_j$$

Derivative:

$$\frac{1}{m}\sum_{i=1}^{m}(y^i - h_\theta(x^i))x_j^i$$

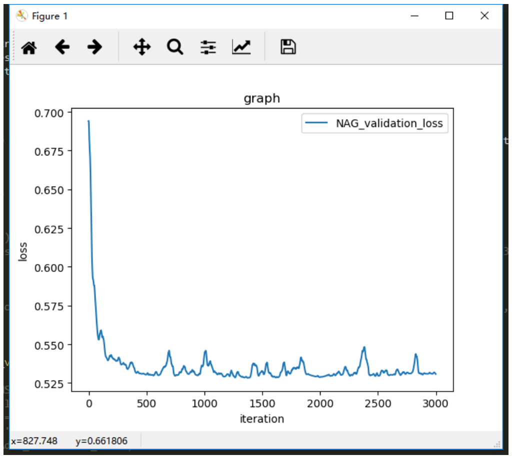
Regression Experiment		
NAG	λ	0.3
	learning rate	0.001
	gamma	0.9
	threshold	0.001
	iteration	3000
RMSProp	λ	0.3
	learning rate	0.001
	gamma	0.9
	threshold	0.001
	iteration	3000
AdaDelta	λ	0.3
	learning rate	0.001
	gamma	0.9
	threshold	0.001
	iteration	3000
Adam	λ	0.3
	learning rate	0.001
	gamma	0.9
	threshold	0.001
	iteration	3000

Classification Experiment		
NAG	λ	0.3
	learning rate	0.001
	gamma	0.9
	threshold	0.001
	iteration	3000
RMSProp	λ	0.3
	learning rate	0.001
	gamma	0.9
	threshold	0.001
	iteration	3000
AdaDelta	λ	0.3
	learning rate	0.001
	gamma	0.9
	threshold	0.001
	iteration	3000
Adam	λ	0.3
	learning rate	0.001
	gamma	0.9
	threshold	0.001
	iteration	3000

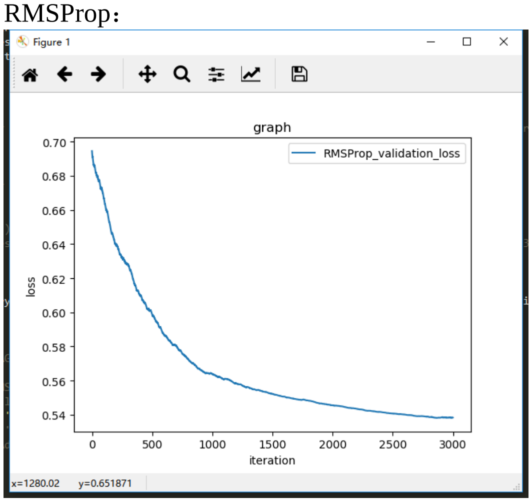
Predicted Result:

Regression Experiment:

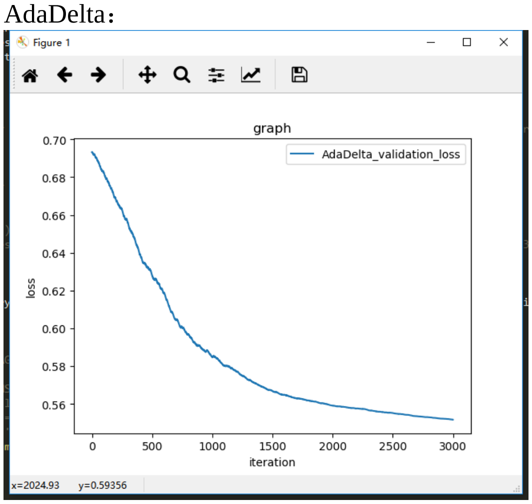
NAG:



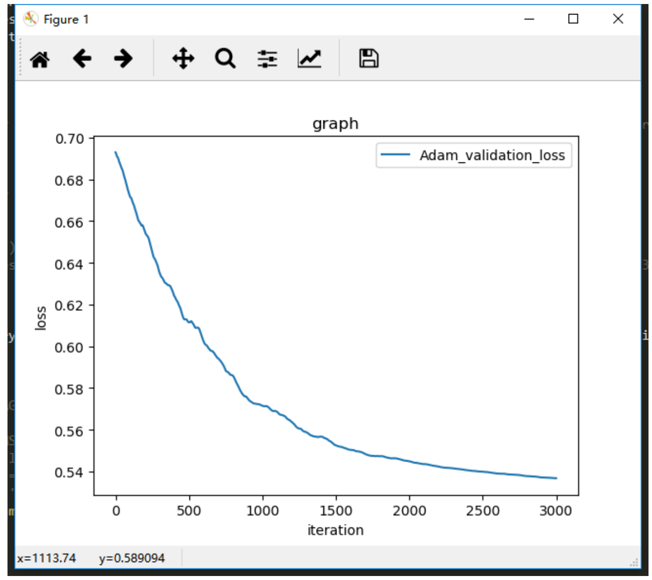
RMSProp:



AdaDelta:

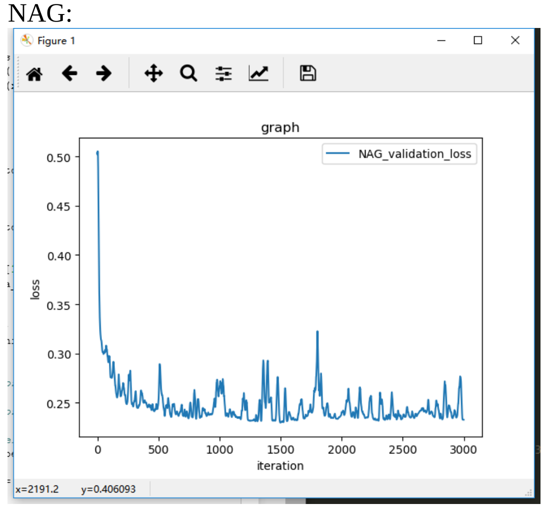


Adam:

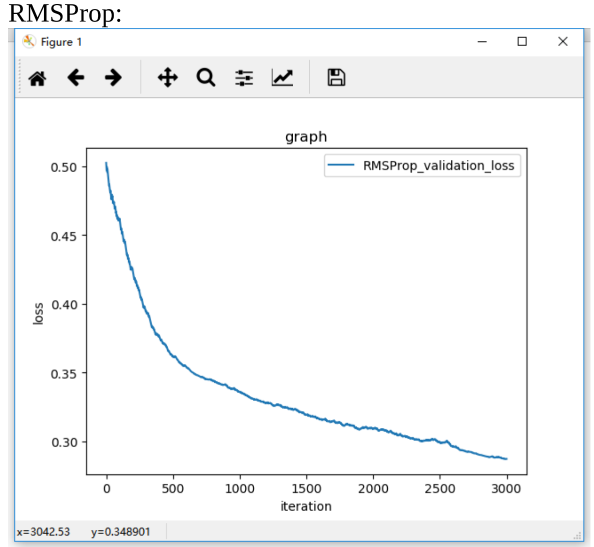


Classification Experiment:

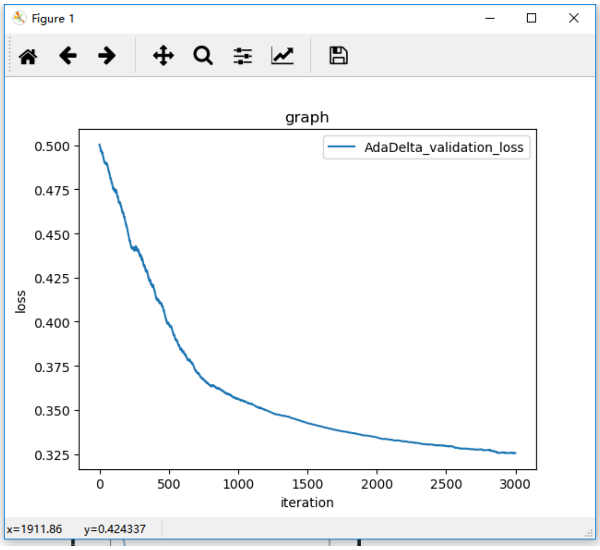
NAG:



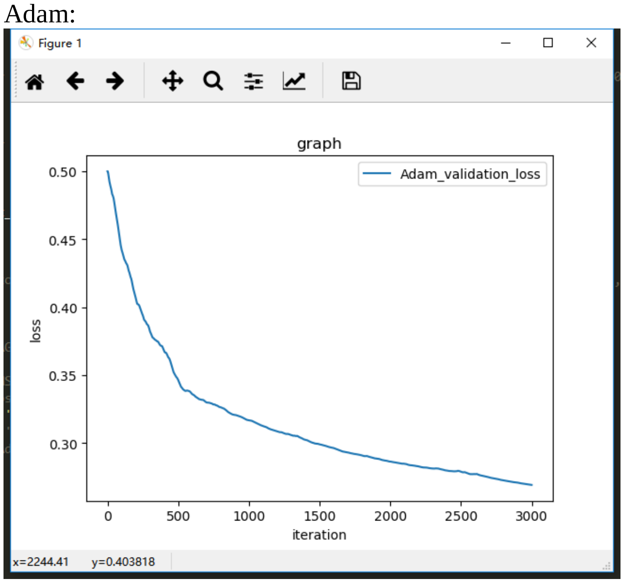
RMSProp:

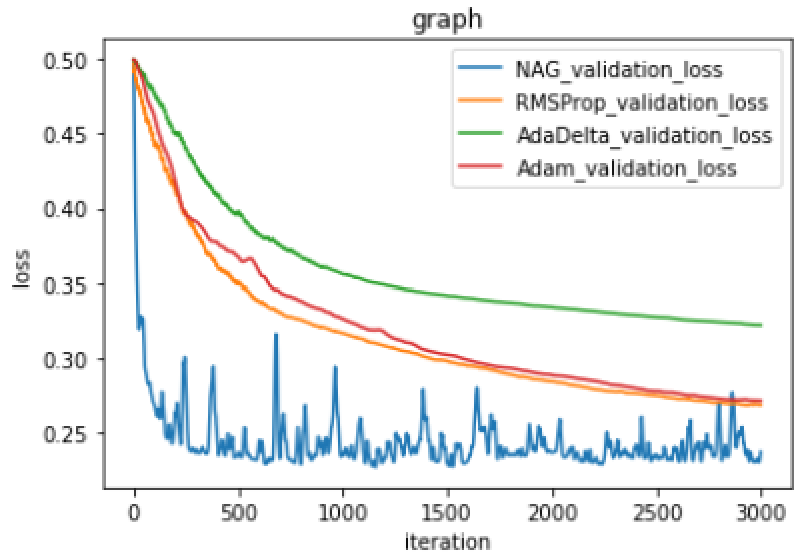


AdaDelta:



Adam:



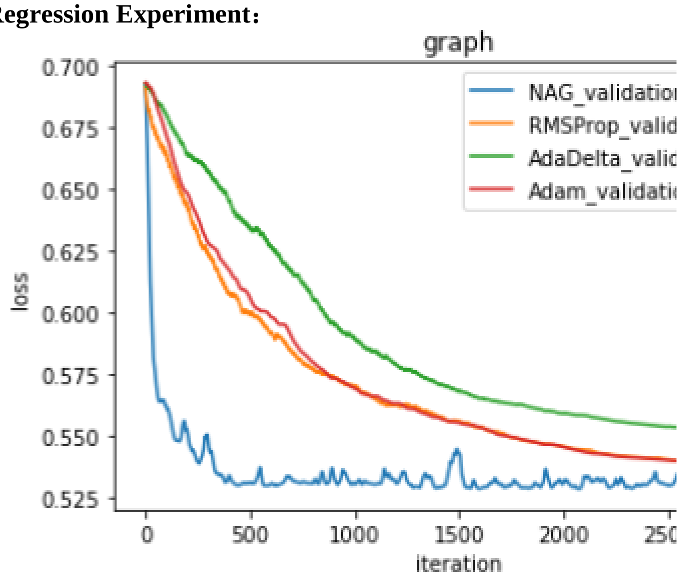


IV. CONCLUSION

The convergence speed of NAG is the fastest, while the concussion is obvious. The convergence speed of RMSProp and Adam is almost the same and the concussion is not obvious. AdaDelta converges slowly but steadily.

Loss Graph:

Regression Experiment:



Classification Experiment: