华南理工大学

South China University of Technology

《机器学习》课程实验报告

学　　院　软件学院

专　　业　软件工程

组　　员　吴国斌

学　　号　201530613030

邮　　箱　854370642@qq.com

指导教师　吴庆耀

提交日期　2017 年 12 月 2 日

1. 实验题目: 逻辑回归、线性分类与随机梯度下降

2. 实验时间： 2017 年 12 月 2 日

3. 报告人:吴国斌

4. 实验目的:

1. 对比理解梯度下降和随机梯度下降的区别与联系。
2. 对比理解逻辑回归和线性分类的区别与联系。
3. 进一步理解 SVM 的原理并在较大数据上实践。

5. 数据集以及数据分析：

实验使用的是 LIBSVM Data 的中的 a9a 数据，包含 32561 / 16281(testing)个样本，每个样本有 123/123 (testing)个属性。请自行下载训练集和验证集。其中 label 表示该记录的类别，+1 为正类，-1 为负类。其中正类有 11687 个样本，负类有 37155 个样本。

6. 实验步骤:

逻辑回归与随机梯度下降

1. 读取实验训练集和验证集。
2. 逻辑回归模型参数初始化，可以考虑全零初始化，随机初始化或者正态分布初始化。
3. 选择Loss函数及对其求导，过程详见课件ppt。
4. 求得部分样本对Loss函数的梯度G。
5. 使用不同的优化方法更新模型参数（NAG，RMSProp，AdaDelta和Adam）。
6. 选择合适的阈值，将验证集中计算结果大于阈值的标记为正类，反之为负类。在验证集上测试并得到不同优化方法的Loss函数值L_{NAG}，$L_{RMSProp}$，$L_{AdaDelta}$和L_{Adam}。
7. 重复步骤4-6若干次，画出L_{NAG}，$L_{RMSProp}$，$L_{AdaDelta}$和L_{Adam}随迭代次数的变化图。

线性分类与随机梯度下降

1. 读取实验训练集和验证集。
2. 支持向量机模型参数初始化，可以考虑全零初始化，随机初始化或者正态分布初始化。
3. 选择Loss函数及对其求导，过程详见课件ppt。
4. 求得部分样本对Loss函数的梯度G。
5. 使用不同的优化方法更新模型参数（NAG，RMSProp，AdaDelta和Adam）。
6. 选择合适的阈值，将验证集中计算结果大于阈值的标记为正类，反之为负类。在验证集上测试并得到不同优化方法的Loss函数值L_{NAG}，$L_{RMSProp}$，$L_{AdaDelta}$和L_{Adam}。
7. 重复步骤4-6若干次，画出L_{NAG}，$L_{RMSProp}$，$L_{AdaDelta}$和L_{Adam}随迭代次数的变化图。

7. 代码内容:

逻辑回归：

```python
from sklearn import datasets, model_selection, linear_model
import numpy as np
import jupyter
import matplotlib.pyplot as plt
import math
import random

X_train, y_train = datasets.load_svmlight_file("a9atrain.txt")

# turn the csr_matrix into array for futher processing
x_ = np.array(X_train.toarray(), np.float32).reshape((-1, 123))
y_ = np.array(y_train, np.float32).reshape((-1, 1))

for i in range(y_.shape[0]):
    if y_[i,0] == -1.0 : y_[i,0] = 0.

X_ = np.hstack([x_, np.ones((x_.shape[0], 1))])


X_test, y_test = datasets.load_svmlight_file("a9atest.txt")

xt_ = np.array(X_test.toarray(), np.float32).reshape((-1, 122))
yt_ = np.array(y_test, np.float32).reshape((-1, 1))
for i in range(yt_.shape[0]):
    if yt_[i,0] == -1.0 : yt_[i,0] = 0.
# X_ = (x_;1) to fit the constent item

Xt_ = np.hstack([xt_, np.zeros((xt_.shape[0], 1)),np.ones((xt_.shape[0], 1))])


# hθ(X) = e^(Theta * X) / (1 + e^(Theta * X)) = 1 / (1 + e^(- Theta * X))
def h_theta(Xi, Theta):
    e_t = math.exp(Xi.dot(Theta.T))
    return e_t / (1 + e_t)

# L(θ) = - (1/m) ∑ (yi * log(hθ(Xi)) + (1 - yi) * log(1 - hθ(Xi)))
def compute_loss(X, y, Theta):
    m = y.shape[0]
    loss = 0.
    for i in range(m):
        loss +=  (y[i] * math.log(h_theta(X[i, :], Theta))) + ((1 - y[i]) * math.log(1 - h_theta(X[i,:], Theta)))
    loss /= - m
    return loss

# ∂L/∂θ = (1/m) ∑ (hθ(Xi) - yi) * Xi
def compute_gradient(X, y, Theta):
    m = y.shape[0]
    gradient = np.zeros(Theta.shape)
    for i in range(m):
        gradient += (h_theta(X[i, :], Theta) - y[i]) * (X[i, :])
    gradient /= m
    return gradient


def train_model_nag(X, y, Theta, learning_rate, gamma, iteration = 10000):
    test_loss_history = np.zeros((iteration, 1))
    v = np.zeros(Theta.shape)
    Theta_gradient = np.zeros(Theta.shape)
    for iter in range(iteration):
        index = random.randint(0, y.shape[0]-10)
        Theta = Theta - gamma * v
        v = gamma * v - learning_rate * compute_gradient(X[index:index+10,:], y[index:index+10], Theta)
        Theta = Theta + v
        test_loss_history[iter] = compute_loss(Xt_, yt_, Theta)[-1:]
    return test_loss_history, Theta


def train_model_rmsprop(X, y, Theta, learning_rate, gamma, epsilon, iteration = 10000):
    test_loss_history = np.zeros((iteration, 1))
    G_t = 0.
    Theta_gradient = np.zeros(Theta.shape)
    for iter in range(iteration):
        index = random.randint(0, y.shape[0]-10)
        Theta_gradient = compute_gradient(X[index:index+10,:], y[index:index+10], Theta)
        G_t = gamma * G_t + (1 - gamma) * Theta_gradient.dot(Theta_gradient.T)
        Theta = Theta - (learning_rate / np.sqrt(G_t + epsilon)) * Theta_gradient
        test_loss_history[iter] = compute_loss(Xt_, yt_, Theta)[-1:]
    return test_loss_history, Theta
```

```python

def train_model_adadelta(X, y, Theta, gamma, epsilon, iteration):
    test_loss_history = np.zeros((iteration, 1))
    Theta_gradient = np.zeros(Theta.shape)
    G_t = 0.
    delta_theta = np.zeros(Theta.shape)
    delta_t = 0.03
    for iter in range(iteration):
        index = random.randint(0, y.shape[0]-10)
        Theta_gradient = compute_gradient(X[index:index+10,:], y[index:index+10], Theta)
        G_t = gamma * G_t + (1 - gamma) * Theta_gradient.dot(Theta_gradient.T)
        delta_theta = - (np.sqrt(delta_t + epsilon) / np.sqrt(G_t + epsilon)) * Theta_gradient
        Theta = Theta + delta_theta
        delta_t = gamma * delta_t + (1 - gamma) * (delta_theta.dot(delta_theta.T))
        test_loss_history[iter] = compute_loss(Xt_, yt_, Theta)[-1:]
    return test_loss_history, Theta

def train_model_adam(X, y, Theta, learning_rate, beta1, beta2, epsilon, iteration):
    test_loss_history = np.zeros((iteration, 1))
    Theta_gradient = np.zeros(Theta.shape)
    v_t = 0.
    m_t = np.zeros(Theta.shape)
    for iter in range(iteration):
        index = random.randint(0, y.shape[0]-10)
        Theta_gradient = compute_gradient(X[index:index+10,:], y[index:index+10], Theta)
        m_t = beta1 * m_t + (1 - beta1) * Theta_gradient
        v_t = beta2 * v_t + (1 - beta2) * Theta_gradient.dot(Theta_gradient.T)
        mt_estimate = m_t / (1 - pow(beta1, iter + 1))
        vt_estimate = v_t / (1 - pow(beta2, iter + 1))
        Theta = Theta - learning_rate * mt_estimate / (np.sqrt(vt_estimate) + epsilon)
        test_loss_history[iter] = compute_loss(Xt_, yt_, Theta)[-1:]
    return test_loss_history, Theta

iteration = 1000

be1 = 0.9
be2 = 0.999
ep = 1e-8
```

```python
t_nag = np.zeros((1, 124))
t_rmsprop = np.zeros((1, 124))
t_adadelta = np.zeros((1, 124))
t_adam = np.zeros((1, 124))
# for i in range(t.shape[0]):
    # t[i] = [-0.03]

nag_loss_history, t_nag = train_model_nag(X_, y_, t_nag, 0.005, be1, iteration)
rmsprop_loss_history, t_rmsprop = train_model_rmsprop(X_, y_, t_rmsprop, 0.005, be1, ep, iteration)
adadelta_loss_history, t_adadelta = train_model_adadelta(X_, y_, t_adadelta, be1, ep, iteration)
adam_loss_history, t_adam = train_model_adam(X_, y_, t_adam, 0.005, be1, be2, ep, iteration)


plt.plot(nag_loss_history, 'g', label='NAG')
plt.plot(rmsprop_loss_history, 'b', label='RMSProp')
plt.plot(adadelta_loss_history, 'r', label='AdaDelta')
plt.plot(adam_loss_history, 'y', label='Adam')


plt.legend(loc='upper right')

plt.ylabel('lost');

plt.xlabel('iteration count')

plt.title('loss graph')

plt.show()
```

线性分类:

```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, y_train = load_svmlight_file("a9atrain.txt")
x_train = x.toarray()
x, y_test = load_svmlight_file("a9atest.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))])

def compute_grad(x, y, w):
    gradient = x * (y - x.dot(w.T))
    return gradient

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
```

```python
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
```

```python
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
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_t

# 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,

# AdaDelta
AdaDelta_w = numpy.zeros((1, X_train.shape[1]))
AdaDelta_w, AdaDelta_loss_history, AdaDelta_test_loss_history = AdaDelta_train(X_train,

#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,


pyplot.plot(NAG_test_loss_history, label = 'NAG_validation_loss')
pyplot.plot(RMS_test_loss_history, label = 'RMSProp_validation_loss')
pyplot.plot(AdaDelta_test_loss_history, label = 'AdaDelta_validation_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()
```

逻辑回归：

8. 模型参数的初始化方法:

模型参数的初始化方法采用的是全零初始化。

9.选择的 loss 函数及其导数:

loss function:

$$L(\theta) = -\frac{1}{m}\sum_{i=0}^{n}\left(y_i \log\left(h_\theta(X_i)\right) + (1-y_i)\log\left(1-\log\left(h_\theta(X_i)\right)\right)\right)$$

其中，$h_\theta(X) = \frac{1}{1+e^{-\theta^T \cdot X}}$

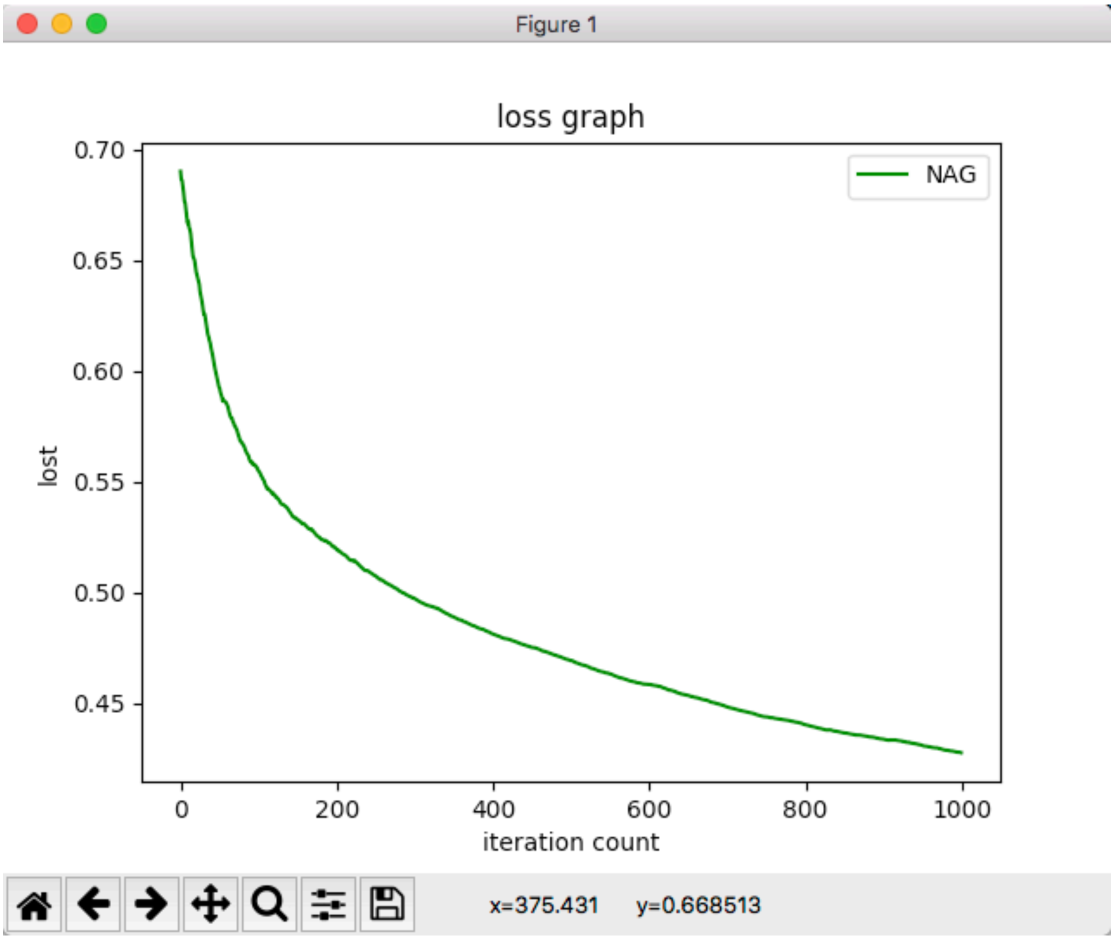
gradient: $\frac{\partial L}{\partial \theta} = \frac{1}{n}\sum_{i=0}^{n} X_i\left(h_\theta(X) - y_i\right)$

10.实验结果和曲线图:

NAG:

超参数选择： η=0.005 γ =0.9 epoch = 1000

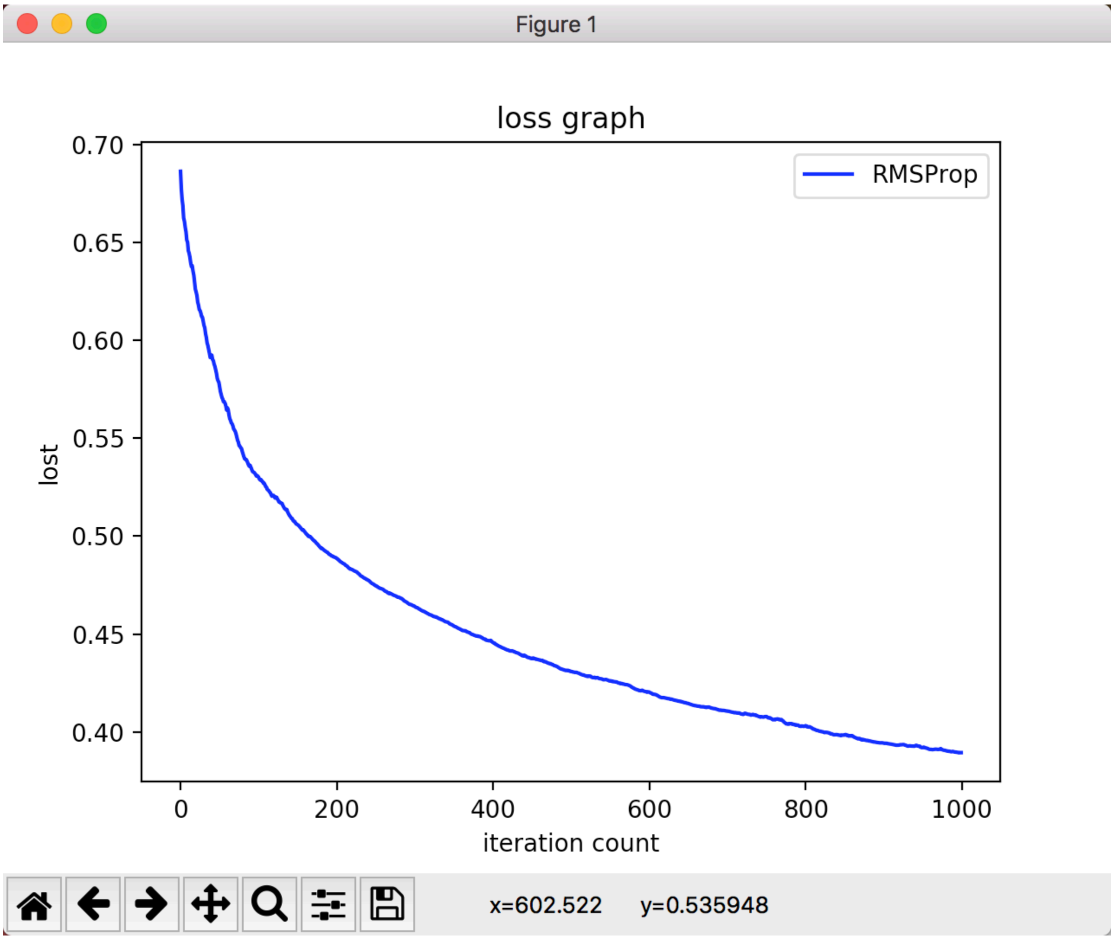
预测结果（最佳结果）：



RMSProp:

超参数选择：η=0.005 γ = 0.9 ε = 1e-8 epoch = 1000

预测结果（最佳结果）：



AdaDelta:

超参数选择：γ = 0.9 ε = 1e-8 epoch=1000

预测结果（最佳结果）：

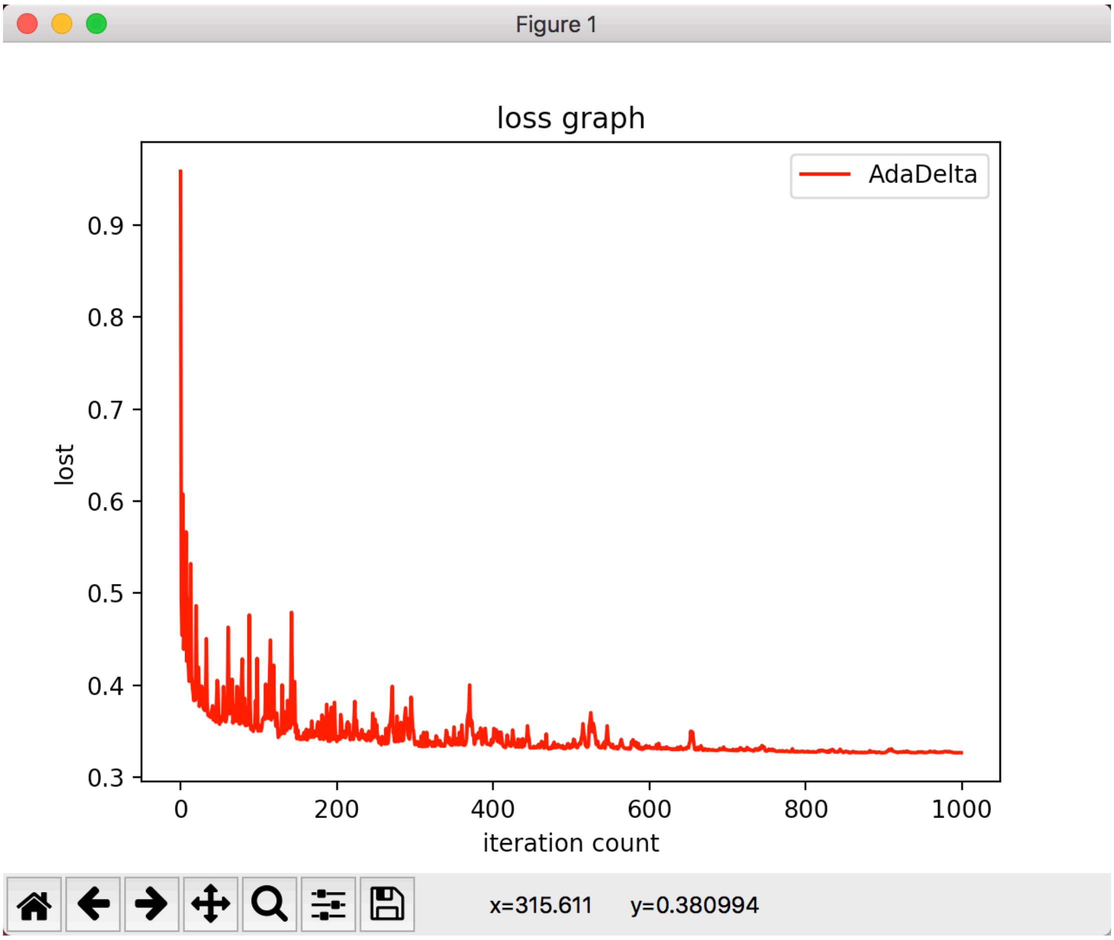
loss 曲线图：

Adam:

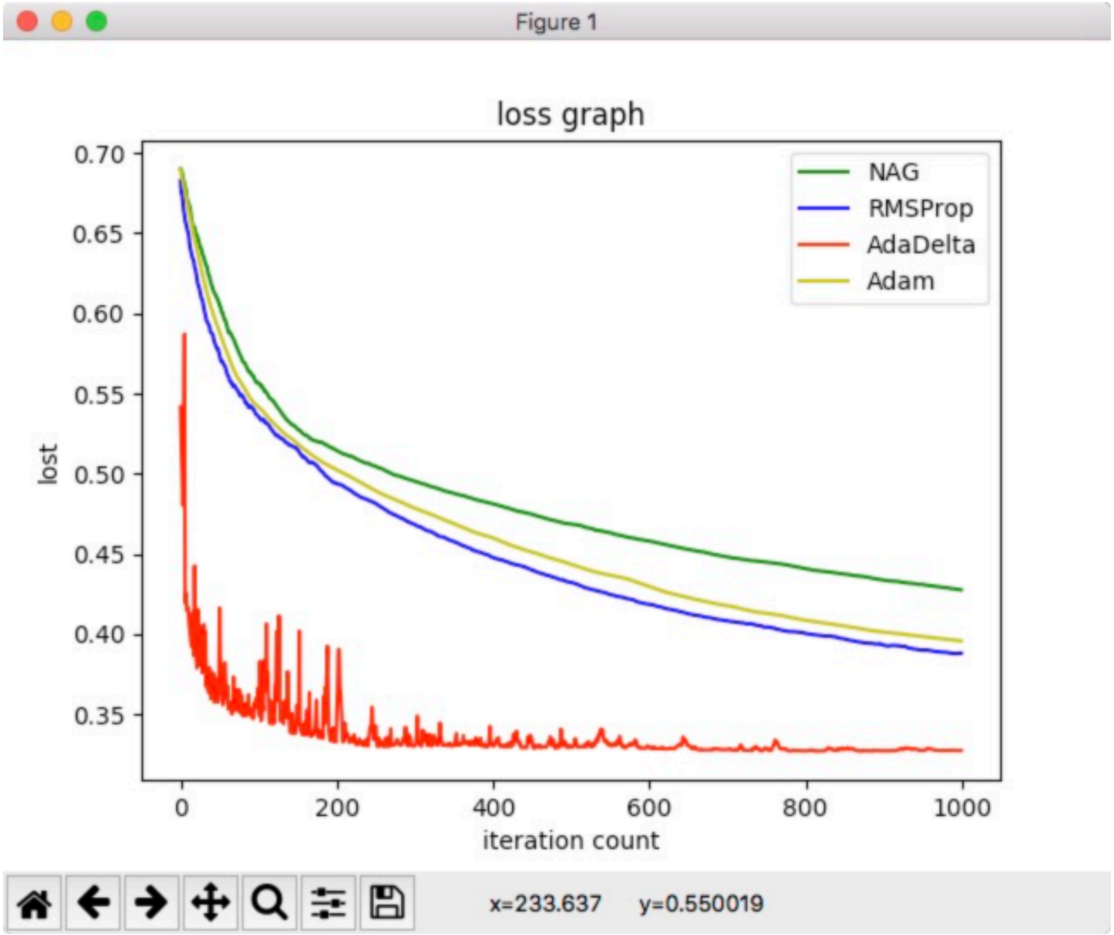
超参数选择： η=0. 005 $\boldsymbol{\beta_1}$ = 0. 9 $\boldsymbol{\beta_2}$ = 0. 999

$\boldsymbol{\varepsilon}$ = 1e−8 epoch=1000

预测结果（最佳结果）：



loss 曲线图：



11.实验结果分析:

AdaDelta 较快的达到了局部最优解，但是震荡明显；NAG 在收敛速度和收敛幅度上都较小，但是曲线较为平滑；RMSProp 和 Adam 基本重合，收敛速度比 NAG 快，但是比 AdaDelta 慢。

总体而言，四种方法都可以较为迅速的收敛到局部最优解，比基本的梯度下降法更加优秀。

线性分类：

8. 模型参数的初始化方法:

模型参数的初始化方法采用的是全零初始化。

9.选择的 loss 函数及其导数:

loss function:

$$L(\theta) = \frac{1}{2n}\sum_{i=0}^{n}(y_i - h_\theta(X_i))^2$$

其中，$h_\theta(X) = \sum_{i=0}^{n} \theta_i X_i$

gradient:

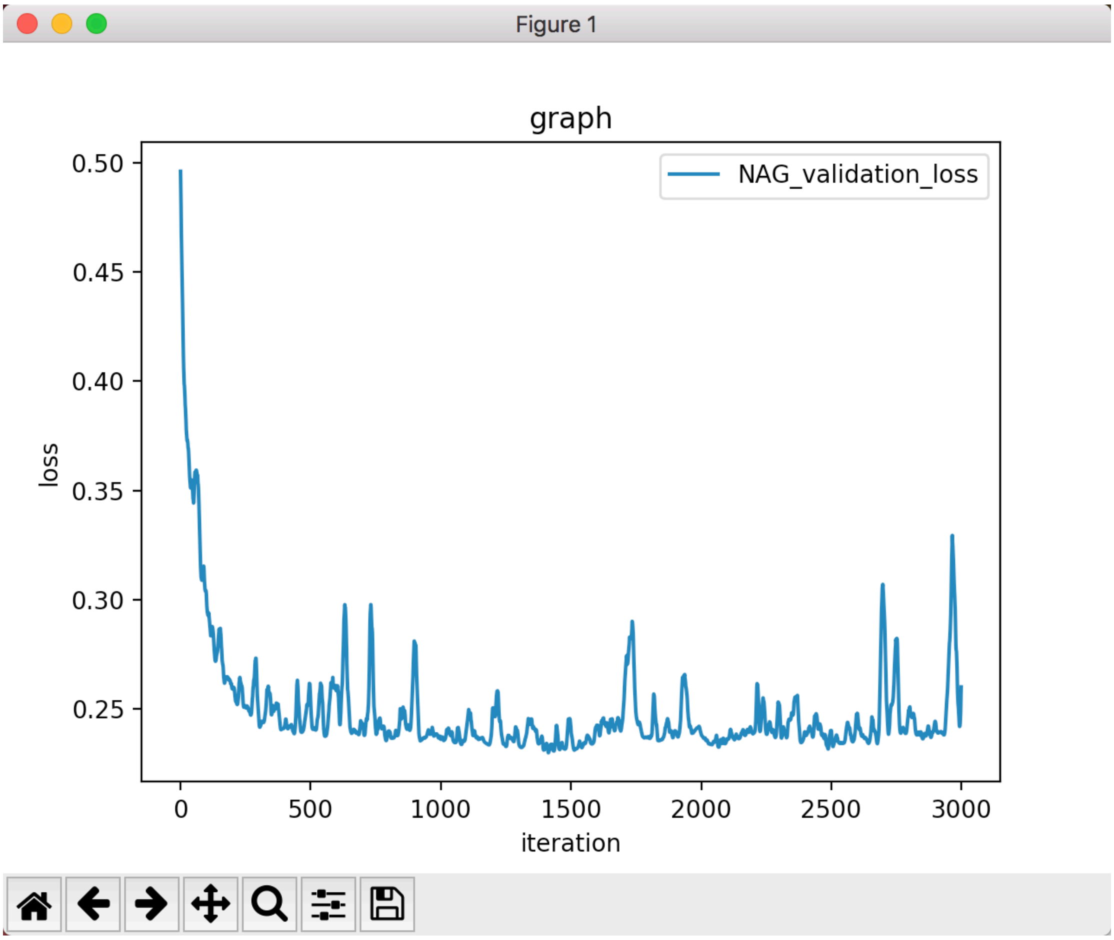
$$\frac{\partial L}{\partial \theta} = \frac{1}{n}\sum_{i=0}^{n}(y_i - h_\theta(X_i)) \cdot X_i$$

10.实验结果和曲线图:

NAG:

超参数选择： η=0.001 γ =0.9 epoch = 3000

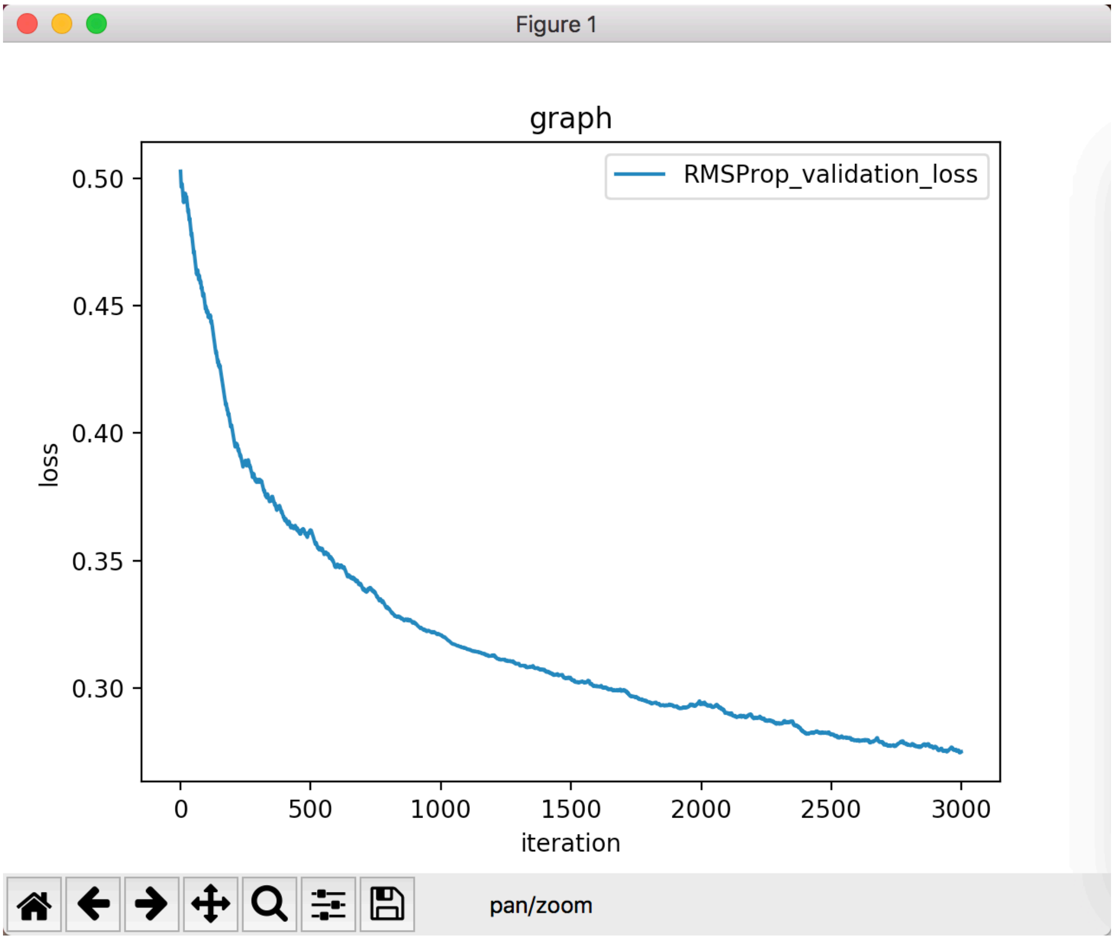
预测结果（最佳结果）:



RMSProp:

超参数选择：η=0.001 γ = 0.9 ε = 1e-8 epoch = 3000

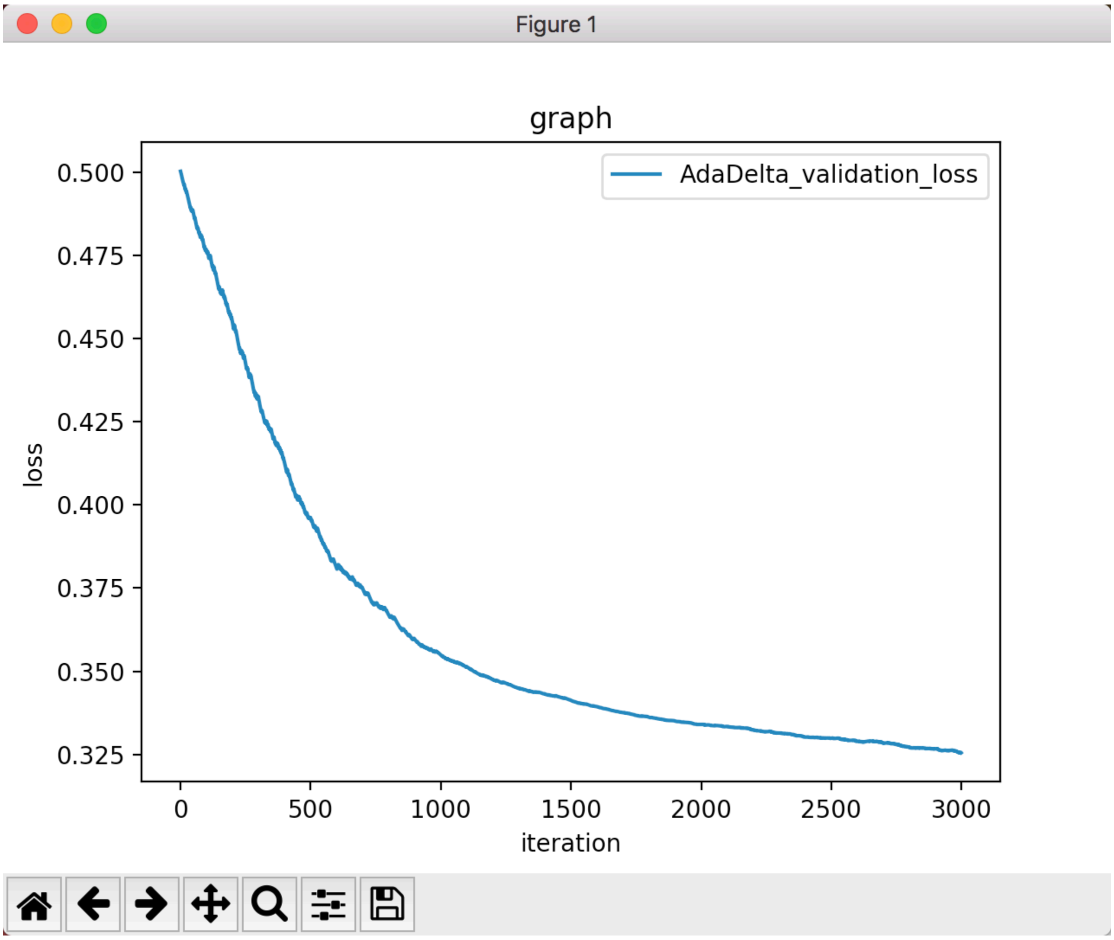
预测结果（最佳结果）：



AdaDelta:

超参数选择：$\gamma = 0.9$ $\varepsilon = 1e{-}8$ epoch=3000

预测结果（最佳结果）：

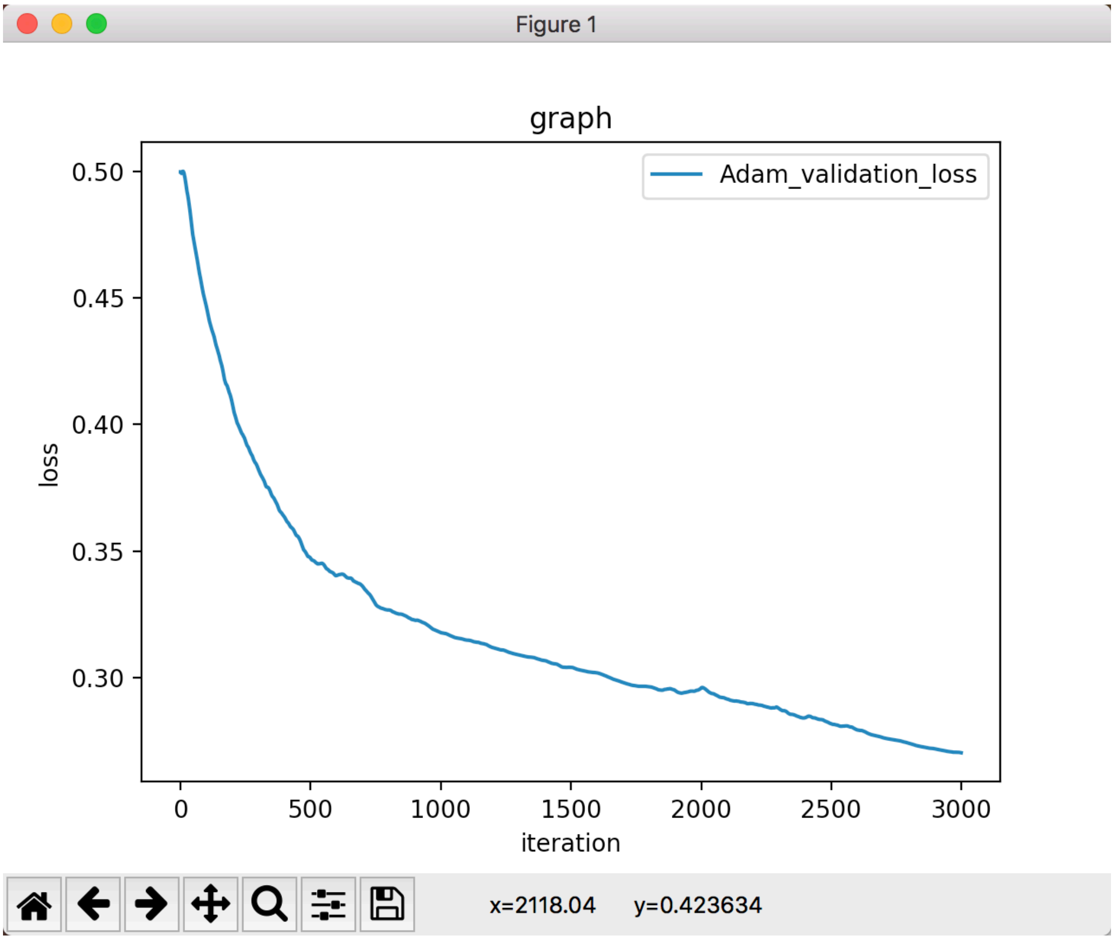


Adam:

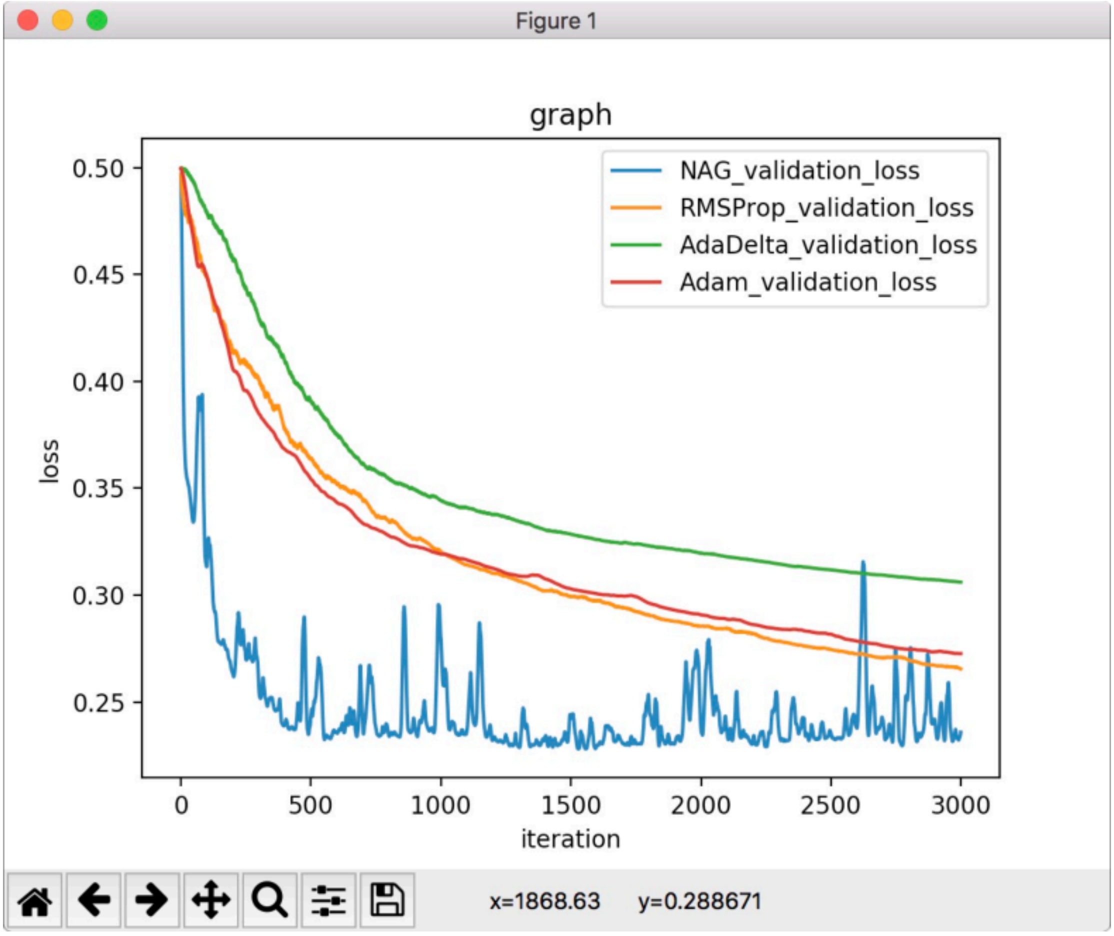
超参数选择： η=0.005 $\boldsymbol{\beta_1}$ = 0.9 $\boldsymbol{\beta_2}$ = 0.999

$\boldsymbol{\varepsilon}$ = 1e−8 epoch=3000

预测结果（最佳结果）：



loss 曲线：



11.实验结果分析:

NAG 收敛速度最快，但是震荡明显;；RMSProp 与 Adam 的收敛速度几乎持平，震荡不明显；AdaDelta 收敛较慢但是曲线比较平滑。相比基本的随机梯度下降算法，这四种算法在不同程度上做了改进，更加优秀。

12.对比逻辑回归和线性分类的异同点：

同：

都属于分类问题，都用于预测。

异：

找最优超平面的方法不同，，形象点说，logistic 模型找的

那个超平面，是尽量让所有点都远离它，而 SVM 线性分类寻找的那个超平面，是只让最靠近中间分割线的那些点尽量远离，即只用到那些“支持向量”的样本。
逻辑回归只可以处理线性可分情况；SVM 则二者皆可。

13.实验总结：

本次实验当中，我学习到了很多 SGD 在实践运用的经验，将课程中学习到的知识运用在实际问题上。但是由于自身对于知识把握懂得程度不高，在实现的过程中遇到了诸如无法正确实现 SGD 优化算法，调参不够灵活的问题，在总结反思之后解决了问题并顺利完成了实验。、

在对模型的训练过程中，我体会到了灵活调参的重要性。在一开始，因为超参数 C, learning rate 等设置得不合理，导致 loss 图像与预期相差甚远，模型参数无法收敛或者收敛过慢，跑数据集时间过长等问题，导致无法拟合数据；或者是因为迭代次数过少，参数还未收敛便停止了训练。在进行数次的不同的调参后，模型往预期方向改变，我也从中学得一些经验。