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　李可欣

学　　号　201530611999

邮　　箱　609186106@qq.com

指导教师　吴庆耀

提交日期　2017 年 12 月 15 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 下午 2:00-5:00

3. 报告人：李可欣

4. 实验目的：

- 对比理解梯度下降和随机梯度下降的区别与联系。
- 对比理解逻辑回归和线性分类的区别与联系。
- 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。

6. 实验步骤：

1）逻辑回归和随机梯度下降：

- 读取实验训练集和验证集。
- 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。
- 选择 **Loss** 函数及对其求导，过程详见课件 **ppt**。
- 求得部分样本对 **Loss** 函数的梯度 **G**。
- 使用不同的优化方法更新模型参数(**NAG**,**RMSProp**,**AdaDelta** 和 **Adam**）。
- 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

LNAG，**LRMSProp**，**LAdaDelta** 和 **LAdam**。

- 重复步骤 **4-6** 若干次，画出 **LNAG**，**LRMSProp**，**LAdaDelta** 和 **LAdam** 随迭代次数的变化图。

2）线性分类和梯度下降：

- 读取实验训练集和验证集。
- 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。
- 选择 **Loss** 函数及对其求导，过程详见课件 **ppt**。
- 求得部分样本对 **Loss** 函数的梯度。
- 使用不同的优化方法更新模型参数（**NAG**，**RMSProp**，**AdaDelta** 和 **Adam**）。
- 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 **LNAG**，**LRMSProp**，**LAdaDelta** 和 **LAdam**。
- 重复步骤 **4-6** 若干次，画出 **LNAG**，**LRMSProp**，**LAdaDelta** 和 **LAdam** 随迭代次数的变化图。

7. 代码内容:

1）逻辑回归和随机梯度下降：

```python
# !python3
# -*- coding:utf-8 -*-
from numpy import *
import math
import pandas as pd
import sklearn.datasets
import matplotlib.pyplot as plt
from numpy import random
# %matplotlib inline

NUM_ITERATIONS = 100


def get_data(filename):
    data = sklearn.datasets.load_svmlight_file(filename)
    return data[0], data[1]


def compute_loss(X, y, theta):
    loss = 0
    for dx, dy in zip(X, y):
        loss += log1p(exp(-(dy*dot(dx, theta))))
    n = shape(X)[0]
    loss = loss / n + (linalg.norm(theta) ** 2) / 2
    return mean(loss)
```

```python
def compute_gradient(X, y, theta, rand):
    grad = zeros(shape=(1, len(theta)))
    for i in rand:
        grad += (y[i]*X[i]/(1+exp(y[i]*dot(X[i],theta))))
    grad /= len(rand)
    grad = -grad + theta.T
    return grad.T


def SGD(X_train, y_train,X_test, y_test, ini_theta, num_iter):
    learning_rate = 0.1
    theta = ini_theta
    Ltest = []
    random.seed(0)
    for i in range(num_iter):
        rand = random.randint(shape(X_train)[0], size=10)
        theta = theta - learning_rate*compute_gradient(X_train,y_train,theta,rand)
        Ltest.append(compute_loss(X_test,y_test,theta))
        print('SGD: Loss of the ', i+1, ' iteration for test:', Ltest[i])
    return Ltest
```

```python
def NAG(X_train, y_train,X_test, y_test, ini_theta, num_iter):
    theta = ini_theta
    learning_rate = 0.01
    momentum = zeros(shape=(shape(theta)[0], 1))
    miu = 0.9

    Ltest = []
    for i in xrange(num_iter):
        rand = random.randint(shape(X_train)[0], size=10)
        grad = compute_gradient(X_train, y_train, theta-miu*momentum, rand)
        momentum = momentum*miu + learning_rate*grad
        theta = theta-momentum
        Ltest.append(compute_loss(X_test, y_test, theta))
        print('NAG:Loss of the ', i+1, ' iteration for test:', Ltest[i])
    return Ltest


def RMSprop(X_train, y_train, X_test, y_test, ini_theta, num_iter):
    theta = ini_theta
    learning_rate = 0.1
    g = zeros(shape=(len(theta), 1))
    eps = 1e-5
    dr = 0.9  # decay_rate
    Ltest = []

    for i in xrange(num_iter):
        rand = random.randint(shape(X_train)[0], size=10)
        grad = compute_gradient(X_train, y_train, theta, rand)
        g = dr*g +(1-dr)*grad*grad
        theta = theta - learning_rate*grad/sqrt(g+eps)
        Ltest.append(compute_loss(X_test, y_test, theta))
        print('RMSprop: Loss of the ', i+1, ' iteration for test:', Ltest[i])
    return Ltest
```

```python
def Adam(X_train, y_train,X_test, y_test, ini_theta, num_iter):
    theta = ini_theta
    m = zeros(shape=(len(theta), 1))
    g = zeros(shape=(len(theta), 1))
    learning_rate = 0.1
    eps = 1e-5
    beta1 = 0.9
    beta2 = 0.999
    Ltest = []

    for i in range(1,num_iter+1):
        rand = random.randint(shape(X_train)[0], size=10)
        grad = compute_gradient(X_train, y_train, theta, rand)
        m = beta1*m+(1-beta1)*grad
        g = beta2*g+(1-beta2)*grad*grad
        alpha = learning_rate*sqrt(1-beta2**i)/(1-beta1**i)
        theta = theta - alpha*m/sqrt(g+eps)
        Ltest.append(compute_loss(X_test, y_test, theta))
        print('Adam: Loss of the ', i, ' iteration for test:', Ltest[i - 1])
    return Ltest
```

```python
def AdaDelta(X_train, y_train,X_test, y_test, ini_theta, num_iter):
    theta = ini_theta
    g = zeros(shape=(len(theta), 1))
    s = zeros(shape=(len(theta), 1))
    eps = 1e-5
    r = 0.95
    # learning_rate = 50

    Ltest = []
    for i in xrange(num_iter):
        rand = random.randint(shape(X_train)[0], size=10)
        grad = compute_gradient(X_train, y_train, theta, rand)
        g = r*g+(1-r)*grad*grad
        diff_theta = -sqrt(s+eps)*grad/sqrt(g+eps)
        theta=theta+diff_theta
        s = r*s+(1-r)*diff_theta*diff_theta
        Ltest.append(compute_loss(X_test, y_test, theta))
        print('AdaDelta: Loss of the ', i+1, ' iteration for test:', Ltest[i])
    return Ltest


def main():
    # Logistic regression
    X_train, y_train = get_data('./data/a9a')
    X_train = X_train.todense()
    temp = ones(shape=[shape(X_train)[0], 1], dtype=float32)
    X_train = concatenate([X_train, temp], axis=1)
    y_train = y_train.reshape(len(y_train), 1)
```

```python
    X_test, y_test = get_data('./data/a9at')
    X_test = X_test.todense()
    X_test = append(zeros(shape=(shape(X_test)[0], 1)), X_test, 1)
    temp = ones(shape=[shape(X_test)[0], 1], dtype=float32)
    X_test = concatenate([X_test, temp], axis=1)
    y_test = y_test.reshape(len(y_test), 1)
    for i in range(shape(y_test)[0]):
        if y_test[i,0] == -1:
            y_test[i,0] = 0

    # initialize parameters
    num_iter = NUM_ITERATIONS
    initial_theta = random.rand(shape(X_test)[1])
    initial_theta = initial_theta.reshape(len(initial_theta),1)

    LSGD = SGD(X_train,y_train,X_test,y_test,initial_theta,num_iter)
    LNAG = NAG(X_train, y_train, X_test, y_test, initial_theta, num_iter)
    LRMSprop = RMSprop(X_train, y_train,X_test, y_test, initial_theta, num_iter)
    LAdam = Adam(X_train, y_train, X_test, y_test, initial_theta, num_iter)
    LAdaDelta = AdaDelta(X_train, y_train, X_test, y_test, initial_theta, num_iter)

    # visualization
    num_iter=xrange(num_iter)
    plt.plot(num_iter, LSGD, label='SGD')
    plt.plot(num_iter, LNAG, label='NAG')
    plt.plot(num_iter,LRMSprop, label='RMSprop')
    plt.plot(num_iter, LAdam,label='Adam')
    plt.plot(num_iter, LAdaDelta,label='AdaDelta')
    plt.title('Loss for test')
    plt.xlabel('iteration')
    plt.ylabel('loss')
    plt.legend()
    plt.show()
```

```python
if __name__=='__main__':
    main()
```

2）线性分类和梯度下降：

```python
# !python3
# -*- coding:utf-8 -*-
from numpy import *
import math
import pandas as pd
import sklearn.datasets
import sklearn.model_selection
import matplotlib.pyplot as plt
from numpy import random
# %matplotlib inline

NUM_ITERATIONS = 200


def get_data(filename):
    data = sklearn.datasets.load_svmlight_file(filename)
    return data[0], data[1]


def compute_loss(X, y, theta):
    loss = 0.0
    for i in range(shape(X)[0]):
        pred = dot(X[i], theta)
        loss += max(0, y[i,0]*pred)
    loss += 0.5 * linalg.norm(theta) ** 2
    return mean(loss)/shape(X)[0]
```

```python
def compute_gradient(X, y, theta,i):
    grad = zeros(shape=(1, len(theta)))
    gamma = 1
    pred = dot(X[i],theta)
    grad = max(0,1-y[i,0]*pred)*(-y[i,0]*X[i])+gamma*theta.T
    return grad.T


def SGD(X_train, y_train,X_test, y_test, ini_theta, num_iter):
    learning_rate = 0.05
    theta = ini_theta
    Ltest = []
    for i in range(num_iter):
        j = random.randint(0,shape(X_train)[0])
        theta = theta - learning_rate*compute_gradient(X_train,y_train,theta,j)
        Ltest.append(compute_loss(X_test,y_test,theta))
        print('SGD: Loss of the ', i+1, ' iteration for test:', Ltest[i])
    return Ltest
```

```python
def NAG(X_train, y_train,X_test, y_test, ini_theta, num_iter):
    theta = ini_theta
    learning_rate = 0.01
    momentum = zeros(shape=(shape(theta)[0], 1))
    miu = 0.9

    Ltest = []
    for i in xrange(num_iter):
        j = random.randint(0, shape(X_train)[0])
        grad = compute_gradient(X_train, y_train, theta-miu*momentum,j)
        momentum = momentum*miu + learning_rate*grad
        theta = theta-momentum
        Ltest.append(compute_loss(X_test, y_test, theta))
        print('NAG:Loss of the ', i+1, ' iteration for test:', Ltest[i])
    return Ltest


def RMSprop(X_train, y_train, X_test, y_test, ini_theta, num_iter):
    theta = ini_theta
    learning_rate = 0.01
    g = zeros(shape=(len(theta), 1))
    eps = 1e-7
    dr = 0.95  # decay_rate
    Ltest = []

    for i in xrange(num_iter):
        j = random.randint(0, shape(X_train)[0])
        grad = compute_gradient(X_train, y_train, theta,j)
        g = dr*g +(1-dr)*square(grad)
        theta = theta - learning_rate*grad/sqrt(g+eps)
        Ltest.append(compute_loss(X_test, y_test, theta))
        print('RMSprop: Loss of the ', i+1, ' iteration for test:', Ltest[i])
    return Ltest
```

```python
def Adam(X_train, y_train, X_test, y_test, ini_theta, num_iter):
    theta = ini_theta
    m = zeros(shape=(len(theta), 1))
    g = zeros(shape=(len(theta), 1))
    learning_rate = 0.01
    eps = 1e-8
    beta1 = 0.9
    beta2 = 0.999
    Ltest = []

    for i in xrange(num_iter):
        j = random.randint(0, shape(X_train)[0])
        grad = compute_gradient(X_train, y_train, theta,j)
        m = beta1*m+(1-beta1)*grad
        g = beta2*g+(1-beta2)*square(grad)
        arpha = learning_rate*sqrt(1-beta2)/(1-beta1)
        theta = theta - arpha*m/sqrt(g+eps)
        Ltest.append(compute_loss(X_test, y_test, theta))
        print('Adam: Loss of the ', i+1, ' iteration for test:', Ltest[i])
    return Ltest
```

```python
def AdaDelta(X_train, y_train, X_test, y_test, ini_theta, num_iter):
    theta = ini_theta
    g = zeros(shape=(len(theta), 1))
    s = zeros(shape=(len(theta), 1))
    eps = 1e-8
    r = 0.95
    learning_rate = 50
    Ltest = []
    for i in xrange(num_iter):
        j = random.randint(0, shape(X_train)[0])
        grad = compute_gradient(X_train, y_train, theta,j)
        g = r*g+(1-r)*square(grad)
        diff_theta = -multiply(sqrt(s+eps),grad)/sqrt(g+eps)
        theta=theta+learning_rate*diff_theta
        s = r*s+(1-r)*square(diff_theta)
        Ltest.append(compute_loss(X_test, y_test, theta))
        print('AdaDelta: Loss of the ', i+1, ' iteration for test:', Ltest[i])
    return Ltest
```

```python
def main():
    # Logistic regression
    X_train, y_train = get_data('./data/a9a')
    X_train = X_train.todense()
    temp = ones(shape=[shape(X_train)[0], 1], dtype=float32)
    X_train = concatenate([X_train, temp], axis=1)
    y_train = y_train.reshape(len(y_train), 1)
    for i in range(shape(y_train)[0]):
        if y_train[i,0] == -1:
            y_train[i,0] = 0

    X_test, y_test = get_data('./data/a9at')
    X_test = X_test.todense()
    X_test = append(zeros(shape=(shape(X_test)[0], 1)), X_test, 1)
    temp = ones(shape=[shape(X_test)[0], 1], dtype=float32)
    X_test = concatenate([X_test, temp], axis=1)
    y_test = y_test.reshape(len(y_test), 1)
    for i in range(shape(y_test)[0]):
        if y_test[i,0] == -1:
            y_test[i,0] = 0

    # initialize parameters
    num_iter = NUM_ITERATIONS
    initial_theta = random.rand(shape(X_test)[1])
    initial_theta = initial_theta.reshape(len(initial_theta),1)
    LSGD = SGD(X_train,y_train,X_test,y_test,initial_theta,num_iter)
    LNAG = NAG(X_train, y_train, X_test, y_test, initial_theta, num_iter)
    LRMSprop = RMSprop(X_train, y_train,X_test, y_test, initial_theta, num_iter)
    LAdam = Adam(X_train, y_train, X_test, y_test, initial_theta, num_iter)
    LAdaDelta = AdaDelta(X_train, y_train, X_test, y_test, initial_theta, num_iter)
```

```python
    #  visualization
    num_iter=xrange(num_iter)
    plt.plot(num_iter, LSGD, label='SGD')
    plt.plot(num_iter, LNAG, label='NAG')
    plt.plot(num_iter,LRMSprop, label='RMSprop')
    plt.plot(num_iter, LAdam,label='Adam')
    plt.plot(num_iter, LAdaDelta,label='AdaDelta')
    plt.title('Loss for test')
    plt.xlabel('iteration')
    plt.ylabel('loss')
    plt.legend()
    plt.show()


if __name__=='__main__':
    main()
```

8. 模型参数的初始化方法:

- 逻辑回归和随机梯度下降：正态分布初始化
- 线性分类和随机梯度下降：随机初始化

9.选择的 loss 函数及其导数:

- 逻辑回归和随机梯度下降：

Log-likehood loss function:

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

导数计算:

$$\mathbf{w}' \to \mathbf{w} - \eta\frac{\partial J(\mathbf{w})}{\partial\mathbf{w}} = (1-\eta\lambda)\mathbf{w} + \eta\frac{1}{n}\sum_{i=1}^{n}\frac{y_i\mathbf{x}_i}{1+e^{y_i\cdot\mathbf{w}^\top\mathbf{x}_i}}$$

- 线性分类和随机梯度下降：

Hinge loss:

$$Hinge\ loss = \xi_i = \max(0, 1 - y_i(\mathbf{w}^\top\mathbf{x}_i + b))$$

导数计算:

$$\frac{\partial f(\mathbf{w},b)}{\mathbf{w}} = \begin{cases}\mathbf{w}^\top - C\mathbf{y}^\top\mathbf{X} & 1-y_i(\mathbf{w}^\top\mathbf{x}_i+b) >= 0\\ \mathbf{w}^\top & 1-y_i(\mathbf{w}^\top\mathbf{x}_i+b) < 0\end{cases}$$

$$\frac{\partial f(\mathbf{w},b)}{b} = \begin{cases}-C\sum_{i=1}^{N} y_i & 1-y_i(\mathbf{w}^\top\mathbf{x}_i+b) >= 0\\ 0 & 1-y_i(\mathbf{w}^\top\mathbf{x}_i+b) < 0\end{cases}$$

10.实验结果和曲线图:

- **逻辑回归和随机梯度下降:**

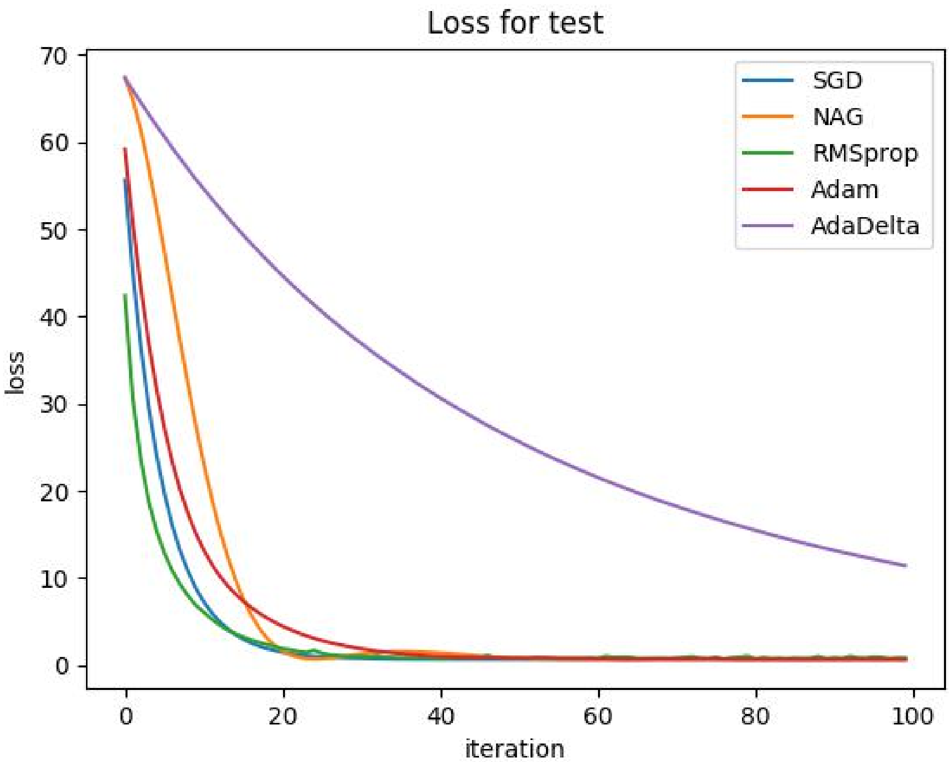
  - **SGD:**
    learning_rate = 0.1

  - **NAG:**
    learning_rate = 0.01
    momentum = zeros(shape=(shape(theta)[0], 1))
    miu = 0.9

  - **RMSprop:**
    learning_rate = 0.1
    g = zeros(shape=(len(theta), 1))
    eps = 1e-5
    dr = 0.9   # decay_rate

  - **Adam:**
    m = zeros(shape=(len(theta), 1))
    g = zeros(shape=(len(theta), 1))
    learning_rate = 0.1
    eps = 1e-5
    beta1 = 0.9
    beta2 = 0.999

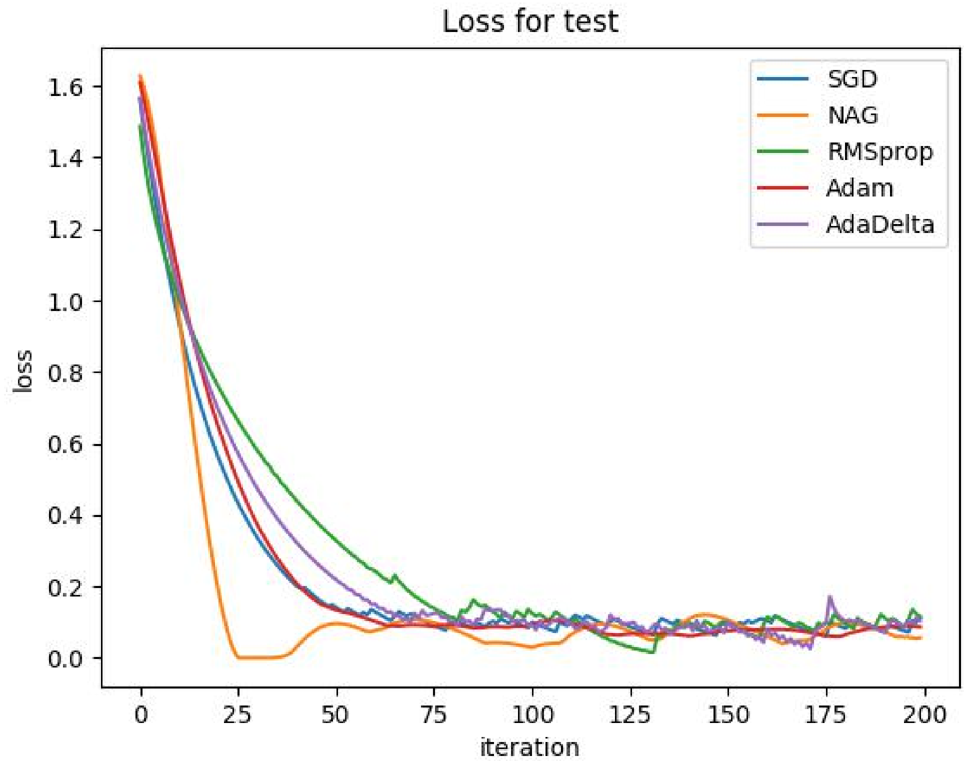
  - **AdaDelta:**
    theta = ini_theta
    g = zeros(shape=(len(theta), 1))
    s = zeros(shape=(len(theta), 1))
    eps = 1e-5
    r = 0.95



- **线性分类和随机梯度下降：**
  - **SGD:**
    learning_rate = 0.05
  - **NAG:**
    learning_rate = 0.01
    momentum = zeros(shape=(shape(theta)[0], 1))
    miu = 0.9
  - **RMSprop:**
    learning_rate = 0.01
    g = zeros(shape=(len(theta), 1))
    eps = 1e-7
    dr = 0.95   # decay_rate
  - **Adam:**
    m = zeros(shape=(len(theta), 1))
    g = zeros(shape=(len(theta), 1))
    learning_rate = 0.01
    eps = 1e-8
    beta1 = 0.9
    beta2 = 0.999

◆ **AdaDelta:**

```
theta = ini_theta
g = zeros(shape=(len(theta), 1))
s = zeros(shape=(len(theta), 1))
eps = 1e-8
r = 0.95
learning_rate = 50
```



11.实验结果分析:

- **逻辑回归和随机梯度下降：**

  实验的结果从图像看，除 AdaDelta 方法是自适应调整学习率收敛速度比较慢外，其他优化方法都在一百次迭代内较快地完成了收敛。这个实验我采取了随机小批量梯度下降的方法为基础，可以看出相比任务二的震荡情况会好很多，收敛的效果更好些。

- **线性分类和随机梯度下降：**

  实验的结果从最终图像上看效果一般，loss 有轻微震荡的情况，六

个随机梯度下降的方法在第七十五次迭代时收敛。其中 SGD，NAG，Adam 的收敛速度相较较快。这个实验我采取的是随机梯度下降，相较任务一的实验，震荡情况会明显些。对于 AdaDelta 收敛速度慢的情况，我在更新 theta 时乘了个比较合适的 learning_rate，实验结果的效果上看收敛速度基本能跟其他方法持平。

12.对比逻辑回归和线性分类的异同点：

- 相同点：

两种方法都是常见的分类算法,从目标函数来看,区别在于逻辑回归采用的是 logistical loss，SVM 采用的是 hinge loss。这两个损失函数的目的都是增加对分类影响较大的数据点的权重,减少与分类关系较小的数据点的权重。SVM 的处理方法是只考虑 support vectors，也就是和分类最相关的少数点,去学习分类器。而逻辑回归通过非线性映射，大大减小了离分类平面较远的点的权重，相对提升了与分类最相关的数据点的权重。

- 不同点：

分类和回归的区别在于输出变量的类型。定量输出称为回归，或者说是连续变量预测；定性输出称为分类，或者说是离散变量预测。

13.实验总结：

首先这次实验体验了看论文实现代码的过程，对各算法的原理过程都深刻了解了许多，比起单单上课听讲要收益许多。整个实验过程，我对调参的过程也体验了很多，深刻感受到各参数的毫厘之差对最终的实验结果造成的

影响确是巨大的。找到一组合适的参数是非常不容易的事情。