**Introduction to Machine Learning (Spring 2019)**

**Homework #1 (Due date: April 8)**

**Student ID 2014313303**

**Name Taeha, Hong**

**Instruction:** We provide all codes and datasets in Python. Please write your code to complete two models: linear regression and logistic regression. Besides, please measure the performance for each model.

1. **[30 pts]** Implementation

**(a)[Linear reression]** Implement training and evaluation function in ‘models/LinearRegression.py’ (‘train’ and ‘eval’ respectively).

def train(self, x, y, epochs, batch\_size, lr, optim):

final\_loss = None # loss of final epoch

W = self.W

for i in range(epochs):

minibatch\_loss = 0

for j in range(len(x)//batch\_size+1):

if j\*batch\_size >= len(x):

break

updated\_x = x[j \* batch\_size:(j + 1) \* batch\_size]

updated\_y = y[j \* batch\_size:(j + 1) \* batch\_size].reshape(-1,1)

grad = np.dot(updated\_x, W) - updated\_y

grad = np.multiply(updated\_x, grad)

grad = grad.mean(axis=0).reshape(-1,1)

W = optim.update(W, grad, lr)

loss = np.square(np.dot(updated\_x, W) - updated\_y)

minibatch\_loss += loss.mean(axis=0)

final\_loss = minibatch\_loss/(len(x)//batch\_size+1)

self.W = W

return final\_loss

def eval(self, x):

pred = None

# Evaluation Function

# Given the input 'x', the function should return prediction for 'x'

# ========================= EDIT HERE ========================

pred = np.dot(x, self.W)

# ============================================================

return pred

**(b)[Logistic reression]** Implement training and evaluation function in ‘models/LogisticRegression.py’ (‘train’ and ‘eval’ respectively).

def train(self, x, y, epochs, batch\_size, lr, optim):

final\_loss = None # loss of final epoch

W = self.W

epsilon = 0.00001

for i in range(epochs):

minibatch\_loss = 0.

for j in range(len(x)//batch\_size+1):

updated\_x = x[j\*batch\_size:(j+1)\*batch\_size]

updated\_y = y[j\*batch\_size:(j+1)\*batch\_size]

h = self.\_sigmoid(np.dot(updated\_x, W).T)- updated\_y

grad = np.multiply(updated\_x, h.T)

grad = grad.sum(axis=0)

for l in range(len(W)):

W[l] = optim.update(W[l], grad[l], lr)

loss\_h = self.\_sigmoid(np.dot(updated\_x, W).T)

loss = np.multiply(np.log(loss\_h+epsilon), updated\_y) + np.multiply((1-updated\_y), np.log(1-loss\_h+epsilon))

minibatch\_loss += -loss.sum(axis=1)

final\_loss = minibatch\_loss/(len(x)//batch\_size+1)

self.W = W

return final\_loss

def eval(self, x):

threshold = 0.5

pred = None

res = np.dot(x, self.W)

pred = res

for i in range(len(x)):

if res[i] >= threshold:

pred[i] = 1

else:

pred[i] = 0

return pred

**(c)[Optimization]** Implement SGD, Momentum, RMS Prop optimizers in ‘optim/Optmizer.py’. Training should be based on the minibatch, not the whole data.

class SGD:

def \_\_init\_\_(self, gamma, epsilon):

self.gamma = gamma

self.epsilon = epsilon

def update(self, w, grad, lr):

updated\_weight = None

updated\_weight = w - lr\*grad

return updated\_weight

class Momentum:

def \_\_init\_\_(self, gamma, epsilon):

self.gamma = gamma

self.epsilon = epsilon

self.v = 0

def update(self, w, grad, lr):

updated\_weight = None

self.v = self.gamma\*self.v + lr\*grad

updated\_weight = w - self.v

return updated\_weight

class RMSProp:

def \_\_init\_\_(self, gamma, epsilon):

self.gamma = gamma

self.epsilon = epsilon

self.G = 0

def update(self, w, grad, lr):

updated\_weight = None

self.G = self.gamma\*self.G + (1-self.gamma)\*(grad\*\*2)

updated\_weight = w - lr\*grad/((self.G+self.epsilon)\*\*0.5)

return updated\_weight

NOTE: You should write your codes in ‘EDIT HERE’ signs. It is not recommended to edit other parts. Once you complete your implementation, run the main codes to check if it is done correctly (‘linear\_main.py’ for Linear Regression and ‘logistic\_main.py’ for Logistic Regression).

1. **[30 pts]** Experimental results
2. **[Linear Regression]** For ‘Graduate’ and ‘Concrete’ dataset, adjust the number of training epochs and learning rate to minimize RMSE. Report your best results for each optimizer.   
   (Batch size = 10, epsilon = 0.01, gamma = 0.9)

**Answer: Fill the blank in the table.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dataset** | **Optimizer** | **# of epochs** | **Learning rate** | **MSE** |
| **Graduate** | SGD | 50 | 0.1 | 0.08 |
| Momentum | 500 | 0.1 | 0.08 |
| RMSProp | 100 | 0.01 | 0.08 |
| **Concrete** | SGD | 3000 | 0.0005 | 11.88 |
| Momentum | 1000 | 0.0005 | 11.68 |
| RMSProp | 1000 | 0.005 | 11.64 |

1. **[Logistic Regression]** For ‘Titanic’ and ‘Digit’ dataset, adjust the number of training epochs and learning rate to maximize accuracy. Report your best results for each optimizer.  
   (Batch size = 10, epsilon = 0.01, gamma = 0.9)

**Answer: Fill the blank in the table.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dataset** | **Optimizer** | **# of epochs** | **Learning rate** | **Acc.** |
| **Titanic** | SGD | 2000 | 0.001 | 0.852 |
| Momentum | 2000 | 0.005 | 0.833 |
| RMSprop | 1000 | 0.005 | 0.806 |
| **Digit** | SGD | 10 | 0.001 | 0.992 |
| Momentum | 10 | 0.001 | 0.992 |
| RMSprop | 10 | 0.001 | 0.992 |

(c) **[Optimization]** For ‘Titanic’ dataset, execute the logistic regression with three optimization methods. Given the following parameter settings, draw two plots : a plot whose x-axis and y-axis are epochs and accuracy, and a plot whose x-axis and y-axis are epochs and cross-entropy loss. Explain which optimization method shows the best accuracy.

|  |  |
| --- | --- |
| **Parameter Settings** | |
| Batch size | 10 |
| Learning rate | 0.0005 |
| Epsilon | 0.01 |
| Gamma | 0.9 |
| # of Epochs | 30, 60, 90, …, 300 |

SGD optimizer가 가장 좋은 성능을 보이고 있고 RMSProp optimizer가 그보다 조금 낮지만 비슷한 흐름이다. Momentum optimizer는 epoch 초반에 들쑥날쑥한 경향을 보이지만 epoch 90이후로 꾸준히 성능이 좋아진다. Accuracy와 cross-entropy loss는 반비례한 경향을 보인다. loss값이 떨어지면 accuracy값은 올라간다.

**Answer: draw the plot and explain the result, especially about the correlation with loss and accuracy according to different optimization methods.**