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

**Homework #2 (40 Pts, April 29)**

**Student ID 2014313303**

**Name 홍태하**

**Instruction:** We provide all codes and datasets in Python. Please write your code to complete the softmax classifier. Compress ‘models/SoftmaxClassifier.py’ and submit with the filename ‘HW2\_STUDENT\_ID.zip’.

1. **[20 pts]** Implement five functions in ‘models/SoftmaxClassifier.py’. (‘train’, ‘eval’, ‘softmax\_loss’, ‘compute\_grad’ and ‘\_softmax’ respectively). Copy ‘optim/Optimizer.py’ from the previous assignment if you have implemented.

**Answer: Fill your code here. You also have to submit your code to i-campus.**

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

"""

N : # of training data

D : # of features

C : # of classes

[INPUT]

x : (N, D), input data (first column is bias for all data)

y : (N, )

epochs: (int) # of training epoch to execute

batch\_size : (int) # of minibatch size

lr : (float), learning rate

optimizer : (Python class) Optimizer

[OUTPUT]

final\_loss : (float) loss of last training epoch

[Functionality]

Given training data, hyper-parameters and optimizer, execute training procedure.

Training should be done in minibatch (not the whole data at a time)

Procedure for one epoch is as follow:

- For each minibatch

- Compute probability of each class for data

- Compute softmax loss

- Compute gradient of weight

- Update weight using optimizer

\* loss of one epoch = Mean of minibatch losses

(minibatch losses = [0.5, 1.0, 1.0, 0.5] --> epoch loss = 0.75)

"""

print('========== TRAINING START ==========')

final\_loss = None # loss of final epoch

num\_data, num\_feat = x.shape

losses = []

for epoch in range(1, epochs + 1):

batch\_losses = [] # list for storing minibatch losses

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

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)

prob = np.zeros((len(updated\_x), self.num\_label))

prob = np.dot(updated\_x, self.W)

prob = self.\_softmax(prob)

batch\_losses.append(self.softmax\_loss(prob, updated\_y))

grad\_weight = self.compute\_grad(updated\_x, self.W, prob, updated\_y)

self.W = optimizer.update(self.W, grad\_weight, lr)

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

epoch\_loss = sum(batch\_losses) / len(batch\_losses) # epoch loss

# print loss every 10 epoch

if epoch % 10 == 0:

print('Epoch %d : Loss = %.4f' % (epoch, epoch\_loss))

# store losses

losses.append(epoch\_loss)

final\_loss = losses[-1]

return final\_loss

def eval(self, x):

"""

[INPUT]

x : (N, D), input data

[OUTPUT]

pred : (N, ), predicted label for N test data

[Functionality]

Given N test data, compute probability and make predictions for each data.

"""

pred = None

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

pred = np.zeros((len(x), 1))

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

for i in range(len(x)):

pred[i][0] = np.where(prob[i,:] == np.max(prob[i,:]))[0][0]

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

return pred

def softmax\_loss(self, prob, label):

"""

N : # of minibatch data

C : # of classes

[INPUT]

prob : (N, C), probability distribution over classes for N data

label : (N, ), label for each data

[OUTPUT]

softmax\_loss : scalar, softmax loss for N input

[Functionality]

Given probability and correct label, compute softmax loss for N minibatch data

"""

softmax\_loss = 0.0

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

epsilon = 0.00000001

for n in range(len(label)):

softmax\_loss += -(np.log(prob[n][label[n]] + epsilon))

softmax\_loss /= len(label)

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

return softmax\_loss

def compute\_grad(self, x, weight, prob, label):

"""

N : # of minibatch data

D : # of features

C : # of classes

[INPUT]

x : (N, D), input data

weight : (D, C), Weight matrix of classifier

prob : (N, C), probability distribution over classes for N data

label : (N, ), label for each data. (0 <= c < C for c in label)

[OUTPUT]

gradient of weight: (D, C), Gradient of weight to be applied (dL/dW)

[Functionality]

Given input (x), weight, probability and label, compute gradient of weight.

"""

grad\_weight = np.zeros\_like(weight, dtype=np.float32) # (D, C)

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

for n in range(len(label)):

prob[n][label[n]] -= 1

grad\_weight = np.dot(x.T, prob)/len(label)

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

return grad\_weight

def \_softmax(self, x):

"""

[INPUT]

x : (N, C), score before softmax

[OUTPUT]

softmax : (same shape with x), softmax distribution over axis-1

[Functionality]

Given an input x, apply softmax function over axis-1 (classes).

"""

softmax = None

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

softmax = np.zeros((x.shape))

epsilon = 0.00000001

for n in range(len(x)):

sum\_prob = 0

for c in range(len(x[0])):

softmax[n][c] = np.exp(x[n][c] - np.max(x[n,:]))

sum\_prob += softmax[n][c]

softmax[n,:] /= (sum\_prob + epsilon)

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

return softmax

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 (‘main.py’) to check if it is done correctly.

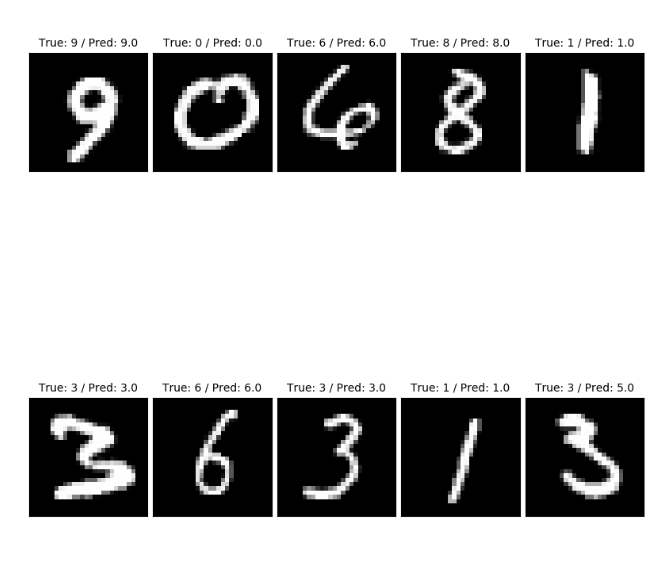
1. **[20 pts]** Writre your experimental results.
2. For ‘Iris’ and ‘Digit’ dataset, adjust the number of training epochs and learning rate to maximize accuracy. Report your best results for each optimizer.  
   (Batch size = 10 for Iris & 256 for Digit, epsilon = 0.01, gamma = 0.9)

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dataset** | **Optimizer** | **# of epochs** | **Learning rate** | **Acc.** |
| **Iris** | SGD | 100 | 0.01 | 0.97 |
| Momentum | 200 | 0.005 | 0.93 |
| RMSprop | 300 | 0.005 | 1 |
| **Digit** | SGD | 20 | 0.000001 | 0.91 |
| Momentum | 30 | 0.000001 | 0.89 |
| RMSprop | 50 | 0.00001 | 0.92 |

(b)For ‘Digit’ dataset, execute the softmax classifier with a given parameter setting. Using the code provided in ‘main.py’, show 10 sample images for true labels and corresponding predicted labels. (Set the variable ‘show\_plot’ as ‘True’ to show sample images.).

|  |  |
| --- | --- |
| **Parameter Settings** | |
| Batch size | 256 |
| Learning rate | 0.00001 |
| Optimizer | RMSProp |
| Epsilon | 0.01 |
| Gamma | 0.9 |
| # of Epochs | 50 |



**Answer: Show the result image.**