Introduction to Machine Learning (Spring 2019)

Homework #2 (40 Pts, April 29)

Student ID ______________________

Name ______________________

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.

- models/SoftmaxClassifier.py

1) train

```
# ========================= EDIT HERE ========================
    index = 0

    while index < num_data:
        # Selects the minibatch size
        data = x[index: min(index + batch_size, num_data)]
        label = y[index: min(index + batch_size, num_data)]
        index += batch_size

        prob = self._softmax(np.matmul(data, self.W))           # Calculate softmax
        loss = self.softmax_loss(prob, label)                   # Calculate loss using softmax value
        grad_weight = self.compute_grad(data, self.W, prob, label)
                                                   # Calculate gradient of weight using softmax value

        self.W = optimizer.update(self.W, grad_weight, lr)      # Update weight using given optimizer
        batch_losses.append(loss)                               # Save each batch losses

# ============================================================
```

2) eval

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

# Calculate the softmax value of total data
pred = np.array([])
softval = self._softmax(np.matmul(x, self.W))

# For each data, select one which has max probability
for i in range(len(softval)):
    pred = np.append(pred, np.argmax(softval[i]))

# ============================================================
```

3) softmax_loss

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

len_data, _ = prob.shape

# For each data, calculate negative log likelihood (NLL)
for i in range(len_data):
    softmax_loss -= np.log(prob[i][label[i]])

# ============================================================
```

4) compute_grad

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

grad_weight = np.transpose(grad_weight)                                    # (C, D)
len_data, _ = x.shape

for data_i in range(len_data):                                            # For each data
    for label_j in range(self.num_label):                                 # For each label
        if label[data_i] == label_j:                                      # Gradient for right label
            gradient = np.multiply(prob[data_i][label_j] - 1, x[data_i])
        else:                                                             # Gradient for wrong label
            gradient = np.multiply(prob[data_i][label_j], x[data_i])

        grad_weight[label_j] += gradient                                   # Add the gradient value

grad_weight = np.divide(grad_weight, len_data)                             # Divide by the size of data
grad_weight = np.transpose(grad_weight)                                    # (D, C)

# ============================================================
```

5) _softmax

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

num_data, _ = x.shape
softmax = []

# For each data, calculate the softmax value and append to softmax list
for i in range(num_data):
    vector = np.exp(x[i])
    vector = np.divide(vector, np.sum(vector))                            # Normalize
    softmax.append(vector)

softmax = np.asarray(softmax)

# ============================================================
```

- optim/ Optimizer.py

```
import numpy as np


class SGD:
    def __init__(self, gamma, epsilon):
        # ========================= EDIT HERE ========================
        self.gamma = gamma
        self.epsilon = epsilon

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

    def update(self, w, grad, lr):
        updated_weight = None
        # ========================= EDIT HERE ========================

        updated_weight = w - lr * grad

        # ============================================================
        return updated_weight

class Momentum:
    def __init__(self, gamma, epsilon):
        # ========================= EDIT HERE ========================
        self.gamma = gamma
        self.epsilon = epsilon
        self.velocity = []

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

    def update(self, w, grad, lr):
        updated_weight = None
        # ========================= EDIT HERE ========================

        if len(self.velocity) == 0:
            self.velocity = lr * grad

        else:
            self.velocity = self.gamma * self.velocity + lr * grad

        updated_weight = w - self.velocity

        # ============================================================
        return updated_weight


class RMSProp:
    # ========================= EDIT HERE ========================
    def __init__(self, gamma, epsilon):
        # ========================= EDIT HERE ========================
        self.gamma = gamma
        self.epsilon = epsilon
        self.G = []

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

    def update(self, w, grad, lr):
        updated_weight = None
```

```python
# ======================== EDIT HERE ========================

if len(self.G) == 0:
    self.G = np.power(grad, 2)

else:
    self.G = self.gamma * self.G + (1 - self.gamma) * np.power(grad, 2)

eps = np.asarray([self.epsilon] *len(self.G))
eps = eps.reshape(len(grad), 1)

updated_weight = np.sqrt(self.G  + eps)
updated_weight = np.divide(updated_weight, lr)
updated_weight = np.reciprocal(updated_weight)
updated_weight = np.multiply(updated_weight, grad)
updated_weight = w - updated_weight

# ============================================================
return updated_weight
```

(2) [20 pts] Writre your experimental results.

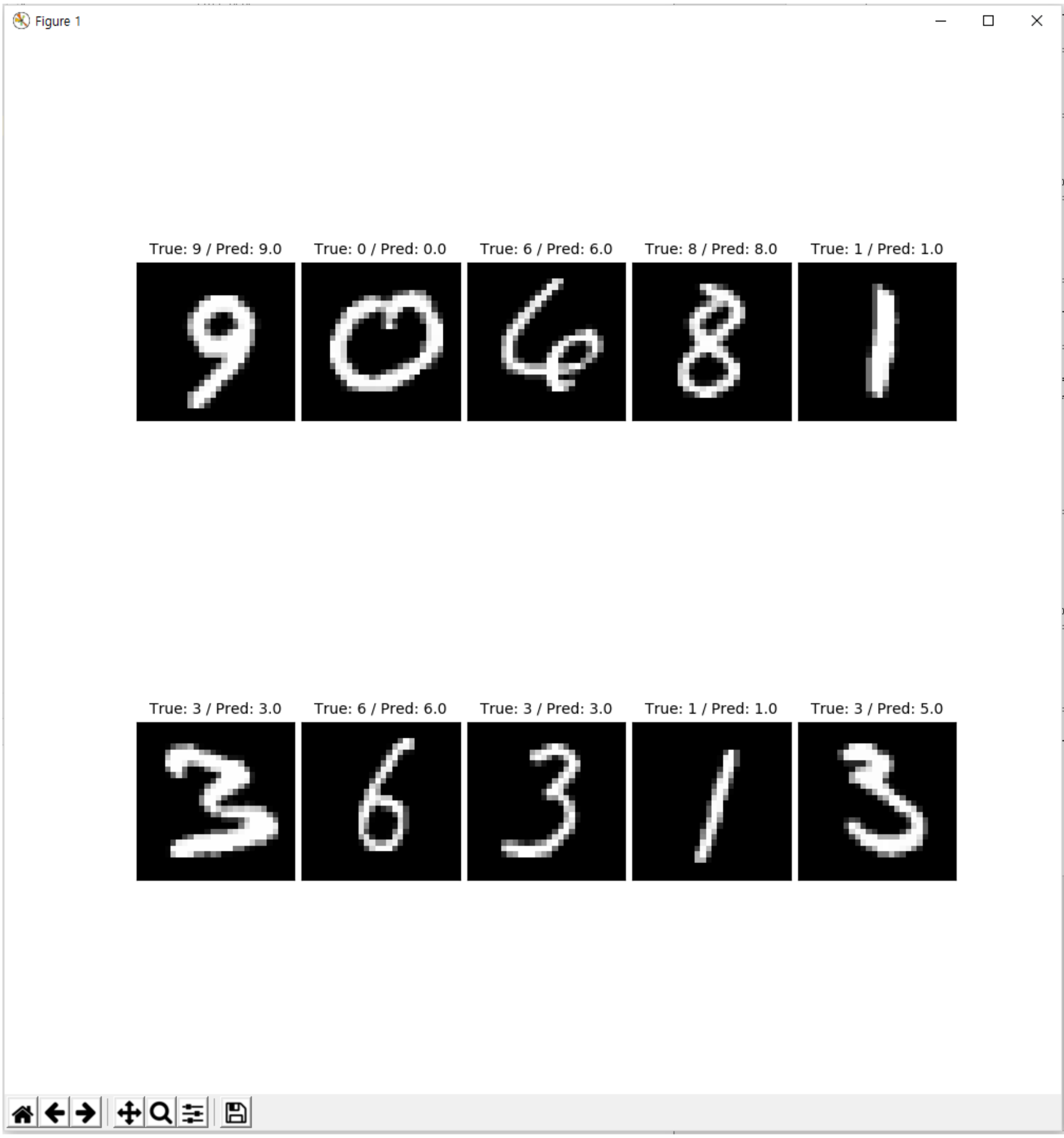
(a) For 'Iris' and 'Digit' dataset, adjust the number of training epochs and learning rate to maximize accuracy. Report your best results for each optimizer.
(Batch size = 10 for Iris & 256 for Digit, epsilon = 0.01, gamma = 0.9)

Answer: Fill the blank in the table.

Dataset	Optimizer	# of epochs	Learning rate	Acc.
Iris	SGD	100	0.1	1.00
	Momentum	100	0.05	1.00
	RMSprop	100	0.06	1.00
Digit	SGD	40	0.000008	0.93
	Momentum	60	0.000001	0.92
	RMSprop	70	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



Run: main

```
C:₩Users₩ljhjo₩anaconda3₩python.exe "C:/Users/ljhjo/OneDrive/coding/Python/2019 Spring ML_Basic/HW2_R/main.py"
# of Training data : 11501

========== TRAINING START ==========
Epoch 10 : Loss = 66.1091
Epoch 20 : Loss = 56.3555
Epoch 30 : Loss = 51.2047
Epoch 40 : Loss = 47.6277
Epoch 50 : Loss = 44.8681
Training Loss at last epoch: 44.8681
RMSProp  Accuracy on Test Data : 0.92

Process finished with exit code 0
```