CSC411: Assignment #2

Due on Monday, February 18, 2018

Chawla Dhruv, Lokman Sabrina

February 18, 2018

Part 1

Dataset Description

The dataset consists of a set of images with each image representing a digit from 0 to 9.

Each image is 28x28 and has a black background with the digit handwritten in white.

Some of the images are straightforward to analyze and decipher while some are more tricky to ascertain what the handwriting is trying to represent (consider the 7th image from the left of the digit 5).

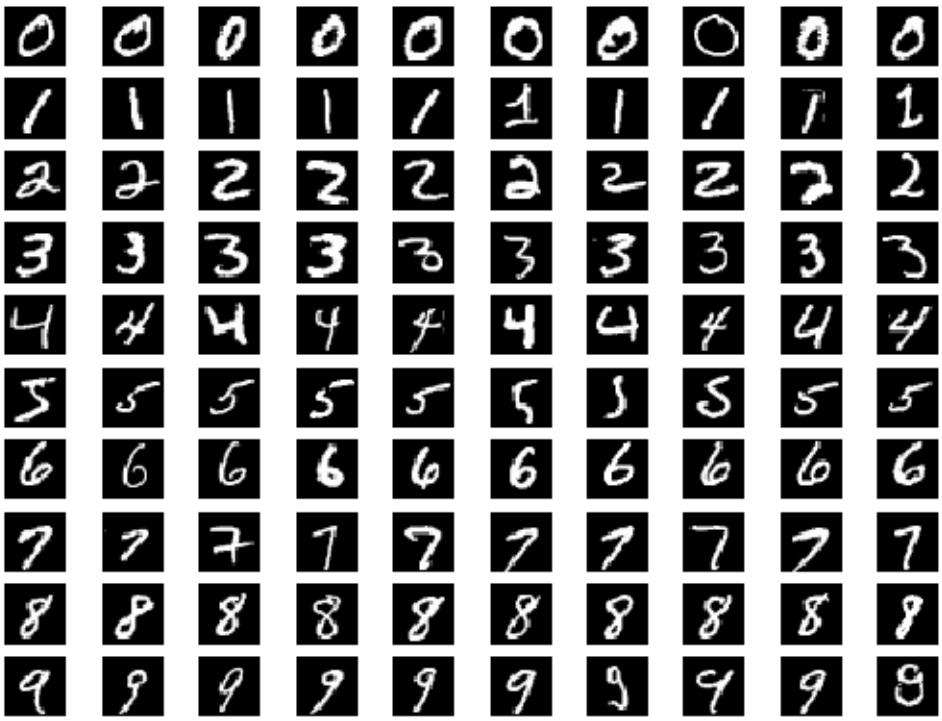

Figure 1: Images from the MNIST dataset

Part 2

Compute Simple Neural Network

The function for computing simple neural network (no hidden layers) is in `mnist_handout.py` and is reproduced here.

```
def compute_simple_network(x, W, b):
    '''Compute a simple network (with no hidden layers)
    '''
    o = np.dot(W, x) + b
    return softmax(o)
```

An example of the function in action is seen below (and in `part2()` of `digits.py`).

```
>>> x = np.random.rand(784, 1)
>>> W = np.random.rand(10, 784)
>>> b = np.random.rand(10, 1)
>>> compute_simple_network(x, W, b).shape
(10, 1)
```

`compute_simple_network(x, W, b)` returns the output layer of the neural network.

Part 3

Cost Function: Sum of negative log-probabilities of all training cases

Part 3(a)

Compute $\frac{\partial C}{\partial w_{ij}}$ (gradient of cost function with respect to a single weight)

For one training case, the cost function is (from slide 6 of One-Hot Encoding Lecture):

$$C = -\sum_j y_j log p_j \tag{1}$$

For M training examples, the cost functions gets modified to:

$$C = -\sum_{m=1}^{M} \sum_j y_j log p_j \tag{2}$$

We also know that (from slide 7 of One-Hot Encoding Lecture),

$$p_i = \frac{e^{o_i}}{\sum_j e^{o_j}} \tag{3}$$

The partial derivative will be the following:

$$\frac{\partial p_i}{\partial o_j} = \begin{cases} p_i(1-p_i) & i = j \\ -p_i p_j & i \neq j \end{cases} \tag{4}$$

Computing the cost function with respect to the output:

$$\frac{\partial C}{\partial o_i} = \sum_j \frac{\partial C}{\partial p_j} \frac{\partial p_j}{\partial o_i} \tag{5}$$

$$= \frac{\partial C}{\partial p_i} \frac{\partial p_i}{\partial o_i} - \sum_{j \neq i} \frac{\partial C}{\partial p_j} \frac{\partial p_j}{\partial o_i} \tag{6}$$

$$= -y_i(1-p_i) + \sum_{j \neq i} y_j p_j \tag{7}$$

$$= -y_i + p_i \sum_{j \neq i} y_j \tag{8}$$

$$= p_i - y_i \tag{9}$$

Computing the O with respect to weight

$$o_i = \sum_j w_{ji} x_j + b \tag{10}$$

$$\frac{\partial o_i}{\partial w_{ij}} = \sum_j x_j \tag{11}$$

Computing the cost function with respect to the weight

$$\frac{\partial C}{\partial w_{ij}} = \sum_j \frac{\partial C}{\partial o_i} \frac{\partial o_i}{\partial w_{ij}} \tag{12}$$

we will get

$$\frac{\partial C}{\partial w_{ij}} = x_j(p_i - y_i) \tag{13}$$

Part 3(b)

Compute Gradient of Cost Function with respect to Weight

The function for computing the gradient with respect to weight is in `mnist_handout.py` and is reproduced here.

```
def gradient_simple_network_w(x, W, b, y):
    p = compute_simple_network(x, W, b)

    gradient_mat = np.zeros((28*28, 10))

    for j in range(28*28):
        gradient_mat[j, :] = np.dot((p - y), x[j, :].T)

    return gradient_mat
```

Compute Gradient of Cost Function with respect to Bias

The function for computing the gradient with respect to bias is in `mnist_handout.py` and is reproduced here.

```
def gradient_simple_network_b(x, W, b, y):
    p = compute_simple_network(x, W, b)

    return np.sum((p - y), axis=1).reshape((10, 1))
```

Part 4

Train the neural network using Gradient Descent

```
def train_nn(f, df_W, df_b, x_train, y_train, x_test, y_test, init_W, init_b, alpha, max_iter =
                                                    2000):
    x = x_train
    y = y_train

    epoch, train_perf, test_perf = [], [], []

    EPS = 1e-10
    prev_W = init_W - 10 * EPS
    prev_b = init_b - 10 * EPS
    W = init_W.copy()
    b = init_b.copy()
    itr = 0

    while norm(W - prev_W) > EPS and norm(b - prev_b) > EPS and itr < max_iter:
        prev_W = W.copy()
        prev_b = b.copy()

        W -= alpha * df_W(x, W, b, y)
        b -= alpha * df_b(x, W, b, y)

        if itr % 500 == 0 or itr == max_iter - 1:
            epoch_i = itr
            train_perf_i = performance(x_train, W, b, y_train)
            test_perf_i = performance(x_test, W, b, y_test)

            epoch.append(epoch_i)
            train_perf.append(train_perf_i)
            test_perf.append(test_perf_i)

            print("Epoch: " + str(epoch_i))
            print("Training Performance:   " + str(train_perf_i) + "%")
            print("Testing Performance:    " + str(test_perf_i) + "%\n")

        itr += 1

    return W, b, epoch, train_perf, test_perf
```

Performance of the learning curves can be seen in figure 2.

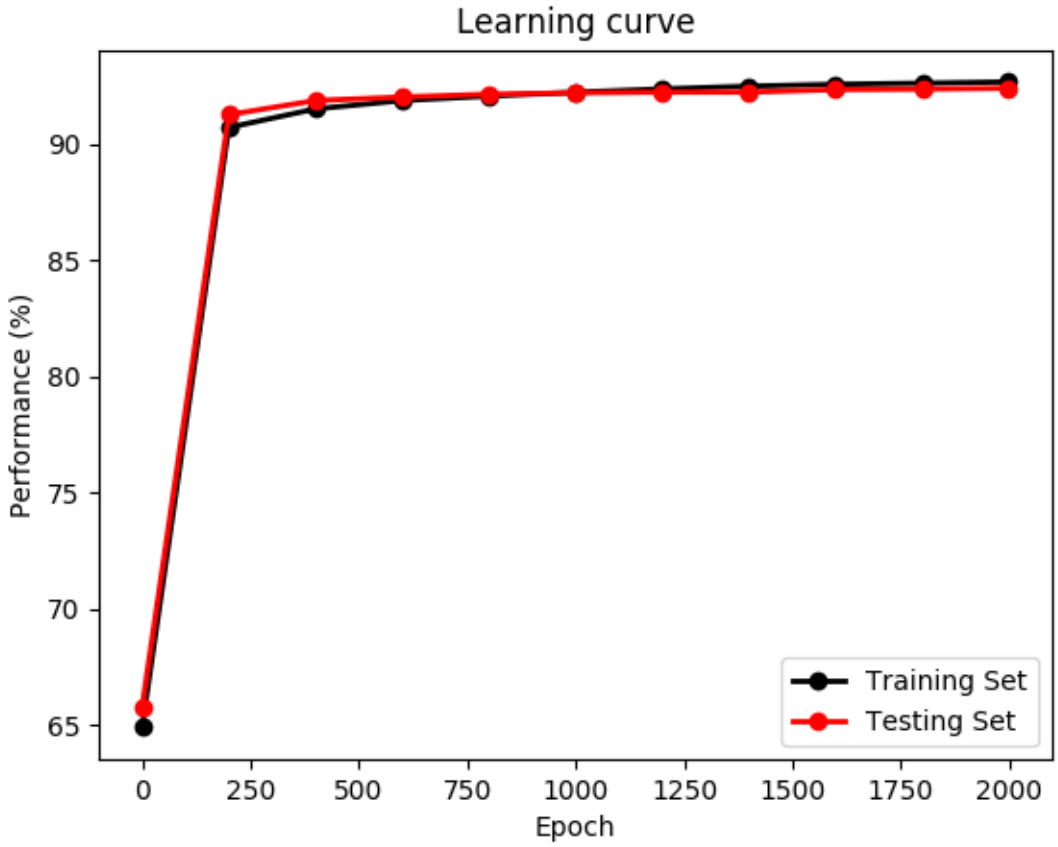

Figure 2: Learning curve for neural network using Gradient Descent

Part 5

Train the neural network using Gradient Descent with Momentum

```
def train_nn_M(f, df_W, df_b, x_train, y_train, x_test, y_test, init_W, init_b, alpha, gamma = 0.
                                                    9, max_iter = 2000):
    x = x_train
    y = y_train

    epoch, train_perf, test_perf = [], [], []

    EPS = 1e-10
    prev_W = init_W - 10 * EPS
    prev_b = init_b - 10 * EPS
    W = init_W.copy()
    b = init_b.copy()
    itr = 0

    while norm(W - prev_W) > EPS and norm(b - prev_b) > EPS and itr < max_iter:
        prev_W = W.copy()
        prev_b = b.copy()

        W = (gamma * W) - alpha * df_W(x, W, b, y)
        b -= alpha * df_b(x, W, b, y)

        if itr % 200 == 0 or itr == max_iter - 1:
            epoch_i = itr
            train_perf_i = performance(x_train, W, b, y_train)
            test_perf_i = performance(x_test, W, b, y_test)

            epoch.append(epoch_i)
            train_perf.append(train_perf_i)
            test_perf.append(test_perf_i)

            print("Epoch: " + str(epoch_i))
            print("Training Performance:   " + str(train_perf_i) + "%")
            print("Testing Performance:    " + str(test_perf_i) + "%\n")

        itr += 1

    return W, b, epoch, train_perf, test_perf
```

Performance of the learning curves can be seen in figure 3.

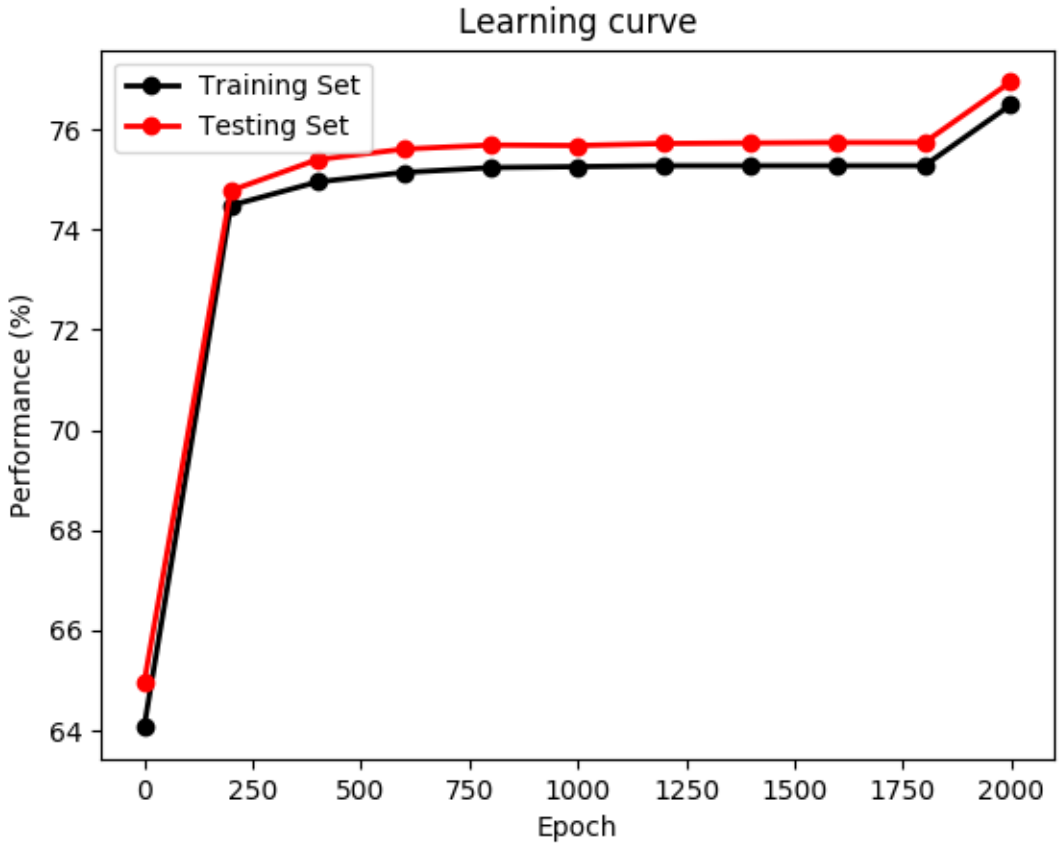

Figure 3: Learning curve for neural network using Gradient Descent with Momentum

Part 6

Part 7

Part 8

Part 9

Part 10