Artificial Neural Network

Machine Learning Lab Assignment

Mohammed Junaid Anwar Qader
21CSB0B36

1. Importing dataset.

```python
import numpy as np
import emnist

# Load the EMNIST dataset for digits (0-9) and uppercase letters (A-Z)
train_images_digits, train_labels_digits = emnist.extract_training_samples('digits')
test_images_digits, test_labels_digits = emnist.extract_test_samples('digits')
train_images_letters, train_labels_letters = emnist.extract_training_samples('letters')
test_images_letters, test_labels_letters = emnist.extract_test_samples('letters')

# Combine the datasets
# take 50,000 from each
train_images = np.concatenate((train_images_digits[:80000], train_images_letters[:80000]), axis=0)
train_labels = np.concatenate((train_labels_digits[:80000], train_labels_letters[:80000] + 10), axis=0)
test_images = np.concatenate((test_images_digits[:10000], test_images_letters[:10000]), axis=0)
test_labels = np.concatenate((test_labels_digits[:10000], test_labels_letters[:10000] + 10), axis=0)

# Shuffle the data
perm = np.random.permutation(train_images.shape[0])
train_images = train_images[perm]
train_labels = train_labels[perm]

# Reshape and normalize the images
train_images_float32 = train_images.reshape(train_images.shape[0], -1).astype(np.float32) / 255.0
test_images_float32 = test_images.reshape(test_images.shape[0], -1).astype(np.float32) / 255.0
test_images_float32 = test_images_float32.reshape(test_images_float32.shape[0], test_images_float32.shape[1], 1)
train_images_float32 = train_images_float32.reshape(train_images_float32.shape[0], train_images_float32.shape[1], 1)

# Print the shape of the new arrays
print("Train images shape (reshaped):", train_images_float32.shape)
print("Test images shape (reshaped):", test_images_float32.shape)
```

```
✓ 4.6s

Train images shape (reshaped): (160000, 784, 1)
Test images shape (reshaped): (20000, 784, 1)
```

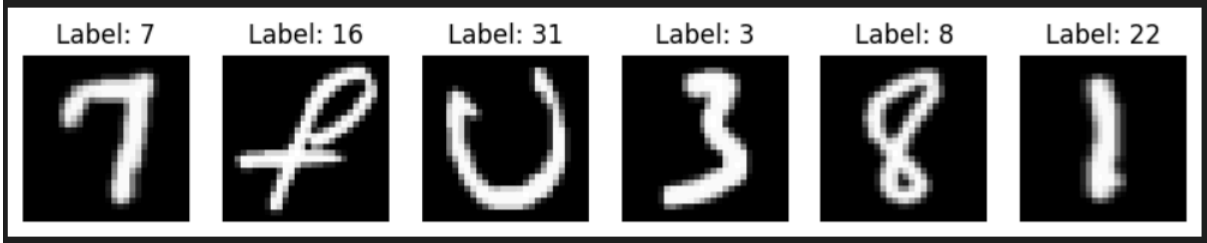
NOTE: mnist only contains digits, i have taken emnist, which has both digits and letters

2. Printing random image to test out dataset.

```python
import matplotlib.pyplot as plt
# Define a function to display sample images
def show_images(images, labels, num_images=6):
    plt.figure(figsize=(10, 5))
    for i in range(num_images):
        plt.subplot(1, num_images, i + 1)
        plt.imshow(images[i], cmap='gray')
        plt.title(f"Label: {labels[i]}")
        plt.axis('off')
    plt.show()

show_images(train_images[-6:], train_labels[-6:])
```

✓ 0.4s



3. One-hot encoding the labels.

```python
# Define input and output sizes
input_size = train_images_float32.shape[1]
output_size = len(np.unique(train_labels)) +1

# Convert labels to one-hot encoding
def one_hot_encode(labels, num_classes):
    num_samples = len(labels)
    one_hot_labels = np.zeros((num_samples, num_classes))
    for i in range(num_samples):
        one_hot_labels[i, labels[i]] = 1
    return one_hot_labels

train_labels_onehot = one_hot_encode(train_labels, output_size)
test_labels_onehot = one_hot_encode(test_labels, output_size)
# reshaping the one hot encodings
train_labels_onehot = train_labels_onehot.reshape(train_labels_onehot.shape[0], train_labels_onehot.shape[1], 1)
test_labels_onehot = test_labels_onehot.reshape(test_labels_onehot.shape[0], test_labels_onehot.shape[1], 1)
```

Example :

X : 5

one_hot_X : [0 0 0 0 0 1] data at index 5 is 1

4. NeuralNetwork class.

```python
# Create and train the neural network
class NeuralNetwork:
    # initialising the sizes of layers
    def __init__(self, input_size, hidden_size, output_size, learning_rate=0.01):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.learning_rate = learning_rate
        # Initialize weights and biases
        self.W1 = np.random.uniform(low=-0.5, high=0.5, size=(self.hidden_size, self.input_size)).astype(np.float32)
        self.b1 = np.random.uniform(low=-0.5, high=0.5, size=(self.hidden_size, 1)).astype(np.float32)
        self.W2 = np.random.uniform(low=-0.5, high=0.5, size=(self.output_size, self.hidden_size)).astype(np.float32)
        self.b2 = np.random.uniform(low=-0.5, high=0.5, size=(self.output_size, 1)).astype(np.float32)
```

Initializing the weights and the biases randomly

5. The sigmoid and the derivative of sigmoid activation.

```python
# we are using the sigmoid activation function
def sigmoid(self, x):
    return 1/(1 + np.exp(-x))

# the derivative of the sigmoid activation function
def sigmoid_derivative(self, x):
    return x * (1 - x)
```

The derivative is used in backpropagation.

6. Forward pass.

```python
# forward pass
def forward(self, X):
    # straightforward, just the matrix m
    # weights and inputs and addition wi
    self.z1 = np.dot(self.W1, X) + self.b1
    self.a1 = self.sigmoid(self.z1)
    self.z2 = np.dot(self.W2, self.a1) + self.b2
    self.a2 = self.sigmoid(self.z2)
    return self.a2
```

```
(variable) b1: NDArray[float32]
```

Forward pass is just basic matrix multiplication between weights and the inputs.

7. BackPropagation.

```
# backward pass
def backward(self, X, y, output):
    # calculating the two deltas
    delta_output = (y - output)*self.sigmoid_derivative(output)
    delta_hidden = np.dot(self.W2.T, delta_output) * self.sigmoid_derivative(self.a1)

    # the change of weights, to be added to weights later
    dW2 = self.learning_rate * np.dot(delta_output, self.a1.T)
    db2 = self.learning_rate * np.sum(delta_output, axis=1, keepdims=True)
    dW1 = self.learning_rate * np.dot(delta_hidden, X.T)
    db1 = self.learning_rate * np.sum(delta_hidden, axis=1, keepdims=True)

    # updating weights
    self.W2 += dW2
    self.b2 += db2
    self.W1 += dW1
    self.b1 += db1
```

Backpropagation based on the rule listed in the book by tom mitchell.

8. Training function

```
# training function
def train(self, X, y, epochs=50, learning_rate=0.01):
    for epoch in tqdm(range(epochs)):
        for i in range(len(X)):
            temp_output = self.forward(X[i])
            self.backward(X[i], y[i], temp_output)
        error = np.mean(np.square(y-temp_output))
        # checking accuracy on the test data
        print("accuracy (test) : ",self.evaluate(test_images_float32, test_labels_onehot),"%")
        # printing epoch and error
        print(f'Epoch: {epoch}, Error: {error}')
```

Straightforward, just forward propagating the data, and then based on the output, backpropagating the errors, to improve the weights based on the stochastic gradient descent.

9. evaluation.

```
# evaluation function, to evaluate the model on the test data
def evaluate(self, X, y):
    count = 0
    for i in range(len(X)):
        temp_output = self.forward(X[i])
        if np.argmax(temp_output) == np.argmax(y[i]):
            count += 1
    return (count/len(X))*100
```

Basic forward propagating the input and checking with label.

10. Training, now.

```
    # Define neural network parameters
    hidden_size = 32 # Adjust as needed

    # Train the neural network
    nn = NeuralNetwork(input_size, hidden_size, output_size)
    nn.train(train_images_float32, train_labels_onehot)
]   10m 24.1s
```

Creating an object of neural network to initialize the layers, i have taken the units in the hidden layers as 32.

Propagate the errors backward through the network:

2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

3. For each hidden unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where

$$\Delta w_{ji} = \eta\, \delta_j\, x_{ji}$$

11. output->

Training on 1,60,000 training examples and testing on 20,000 testing dataset.

Scenario 1: (both letters and digits) 85% accuracy

```
 10%|          | 1/10 [00:13<01:59, 13.24s/it]
accuracy (test) :  78.41818181818182 %
Epoch: 0, Error: 0.03137290512054773
 20%|          | 2/10 [00:26<01:46, 13.29s/it]
accuracy (test) :  82.18181818181817 %
Epoch: 1, Error: 0.03946107331728847
 30%|          | 3/10 [00:39<01:31, 13.08s/it]
accuracy (test) :  83.2909090909091 %
Epoch: 2, Error: 0.043762835663772974
 40%|          | 4/10 [00:52<01:17, 12.97s/it]
accuracy (test) :  83.9090909090909 %
Epoch: 3, Error: 0.045574721442975266
 50%|          | 5/10 [01:05<01:05, 13.02s/it]
accuracy (test) :  84.2 %
Epoch: 4, Error: 0.046492445153485905
 60%|          | 6/10 [01:18<00:52, 13.17s/it]
accuracy (test) :  84.30909090909091 %
Epoch: 5, Error: 0.04702490948376443
 70%|          | 7/10 [01:31<00:39, 13.18s/it]
accuracy (test) :  84.58181818181818 %
Epoch: 6, Error: 0.04735866570198203
 80%|          | 8/10 [01:45<00:26, 13.16s/it]
accuracy (test) :  84.85454545454544 %
Epoch: 7, Error: 0.04757739479428799
 90%|          | 9/10 [01:58<00:13, 13.15s/it]
accuracy (test) :  85.01818181818182 %
Epoch: 8, Error: 0.04772338446160571
100%|          | 10/10 [02:11<00:00, 13.14s/it]
accuracy (test) :  85.21818181818182 %
Epoch: 9, Error: 0.04781981142121393
```

Scenario 2: (only digits) 94.62 % accuracy

```
 10%|          | 1/10 [00:11<01:41, 11.28s/it]
accuracy (test) :  88.62 %
Epoch: 0, Error: 0.043484778285394314
 20%|          | 2/10 [00:22<01:30, 11.33s/it]
accuracy (test) :  91.44 %
Epoch: 1, Error: 0.045373270102018 57
 30%|          | 3/10 [00:33<01:19, 11.31s/it]
accuracy (test) :  92.38 %
Epoch: 2, Error: 0.046168158161546624
 40%|          | 4/10 [00:45<01:07, 11.30s/it]
accuracy (test) :  92.75999999999999 %
Epoch: 3, Error: 0.046588844055445 13
 50%|          | 5/10 [00:56<00:57, 11.46s/it]
accuracy (test) :  93.28 %
Epoch: 4, Error: 0.046857753585598 98
 60%|          | 6/10 [01:08<00:45, 11.37s/it]
accuracy (test) :  93.7 %
Epoch: 5, Error: 0.047050813965645 39
 70%|          | 7/10 [01:19<00:33, 11.22s/it]
accuracy (test) :  94.12 %
Epoch: 6, Error: 0.047195428001512725
 80%|          | 8/10 [01:30<00:22, 11.24s/it]
accuracy (test) :  94.3 %
Epoch: 7, Error: 0.04730569957151
 90%|          | 9/10 [01:41<00:11, 11.32s/it]
accuracy (test) :  94.54 %
Epoch: 8, Error: 0.047390944242411 22
100%|          | 10/10 [01:52<00:00, 11.27s/it]
accuracy (test) :  94.62 %
Epoch: 9, Error: 0.047457726215002806
```