Homework #3
EE 541: Fall 2023

Name: Nissanth Neelakandan Abirami
USC ID: 2249203582
Instructor: Dr. Franzke

1. The MNIST dataset of handwritten digits is one of the earliest and most used datasets to benchmark machine learning classifiers. Each datapoint contains 784 input features – the pixel values from a 28 × 28 image – and belongs to one of 10 output classes – represented by the numbers 0-9. In this problem you will use numpy to program the feed-forward phase for a deep multilayer perceptron (MLP) neural network. We provide you with a pre-trained MLP using the MNIST dataset. The MLP has an input layer with 784-neurons, 2 hidden layers of 200 and 100 neurons, and a 10-neuron output layer. The model assumes ReLU activation for the hidden layers and softmax function in the output layer.

(a) Extract the weights and biases of the pre-trained network from mnist network params.hdf5. The file has 6 keys corresponding to: W 1, b 1, W 2, b 2, W 3, b 3. Verify the dimension of each numpy array with the shape property.

(b) The file mnist testdata.hdf5 contains 10,000 images. xdata holds pixel intensities and ydata contains the corresponding class labels. Extract these. Note: each image vector is 784-dimensions and the label is 10-dimensional with one-hot encoded, i.e., label [0,0,0,1,0,0,0,0,0,0] means the image is class “3”.

(c) Write functions to calculate ReLU and softmax:
• ReLU (x)
• Softmax (x)
The softmax function takes a vector of size n and returns another vector of size n that you can interpret as a probability distribution. For example: Softmax([0; 1; 2]) = [0:09; 0:24; 0:67] so you can conclude that the 3rd element is the most likely outcome.

(d) Use numpy to create an MLP to classify 784-dimensional images into the target 10-dimensional
output. Use ReLU activation for the two hidden layers and softmax the output layer. Format and write your results to file result.json according to:

data = [
{"index": XXX, "activations": [YYY,..., YYY], "classification": ZZZ},
...
]

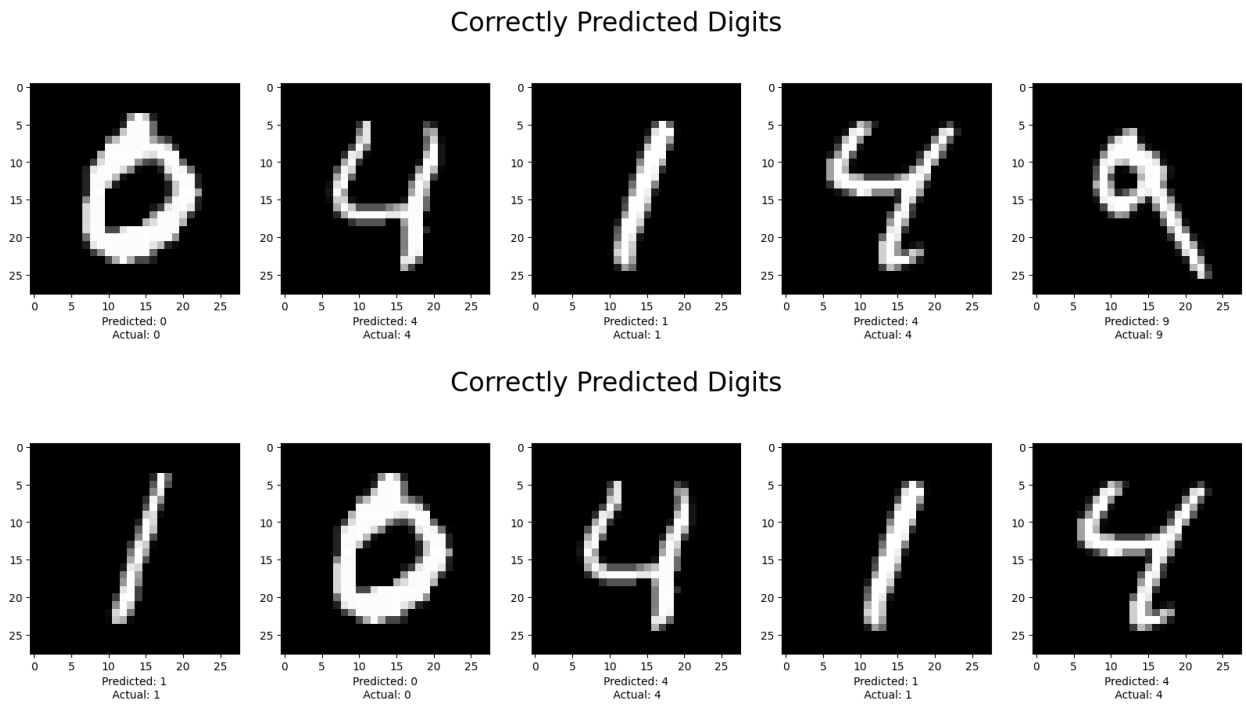
```
import json
with open("result.json", "w") as f:
f.write(json.dumps(data))
```

Output index and activation as integers (not string) and include only the 10 final-layer output activations as numeric floats.

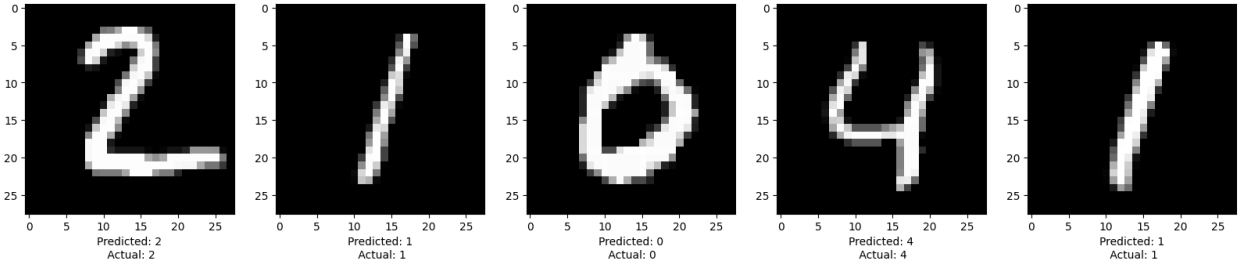
(e) Compare your prediction with the (true) ydata label. Count the classification as correct if the position of the maximum element in your prediction matches with the position of the 1 in ydata. Tally the number of correctly classified images from the whole set of 10,000. [hint: 9790 correct].

(f) Identify and investigate several datapoints that your MLP classified correctly and several it classified incorrectly. Inspect them visually. Is the correct class obvious to you in the incorrect cases? Use matplotlib to visualize:

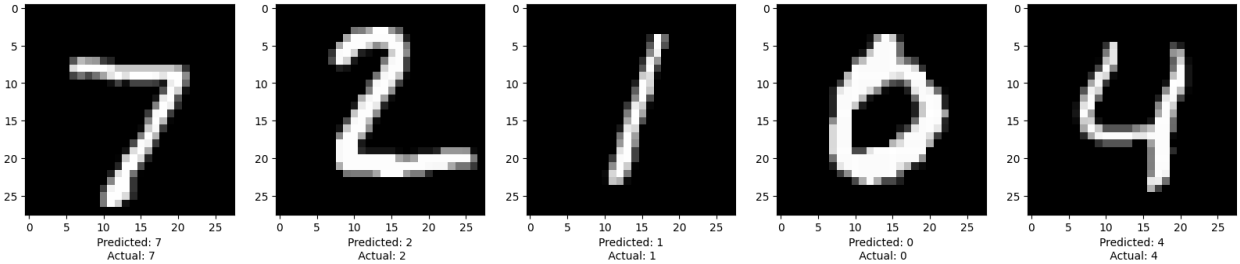
```
import matplotlib.pyplot as plt
plt.imshow(xdata[i].reshape(28,28))
plt.show()
# the index i selects which image to visualize
# xdata[i] is a 784x1 numpy array
```



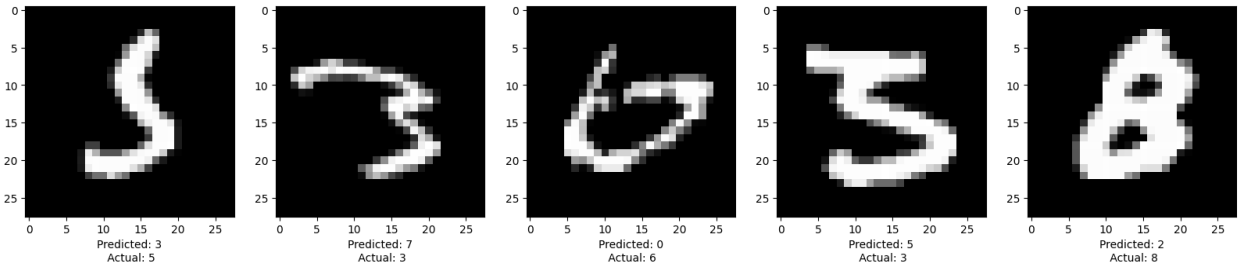
Correctly Predicted Digits



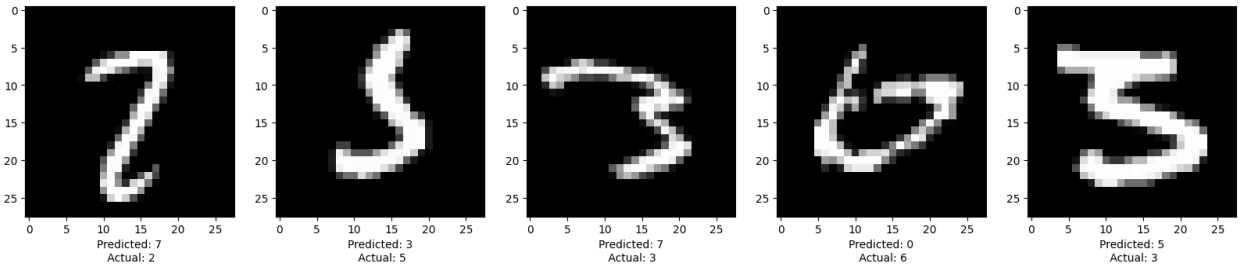
Correctly Predicted Digits

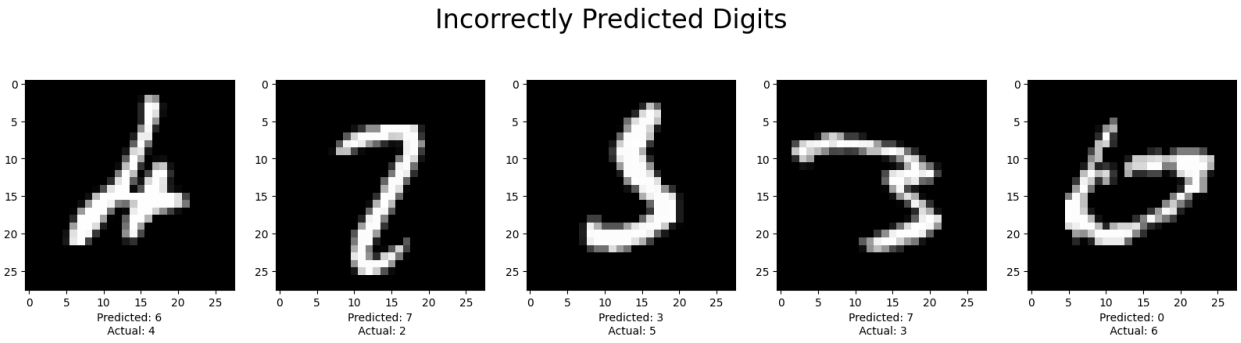


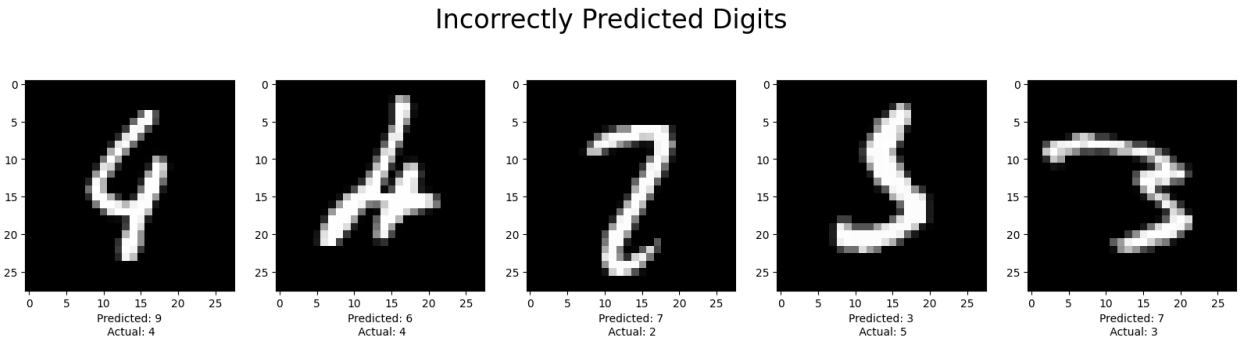
Incorrectly Predicted Digits

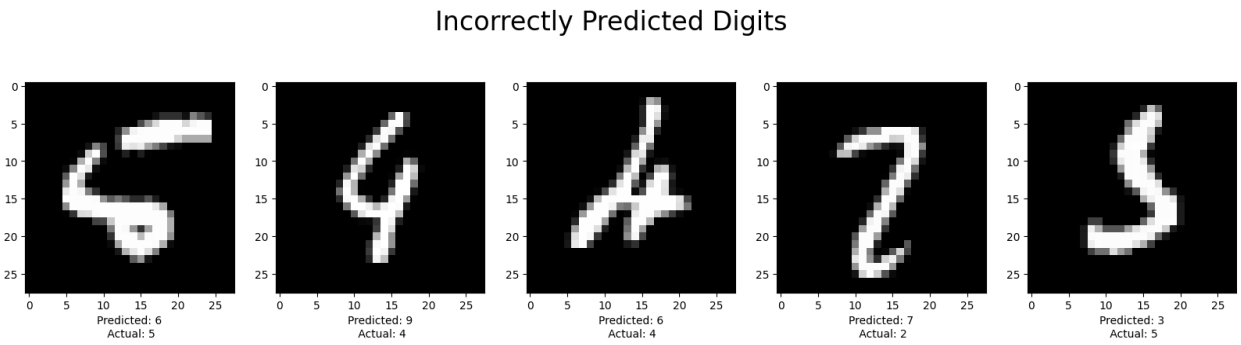


Incorrectly Predicted Digits

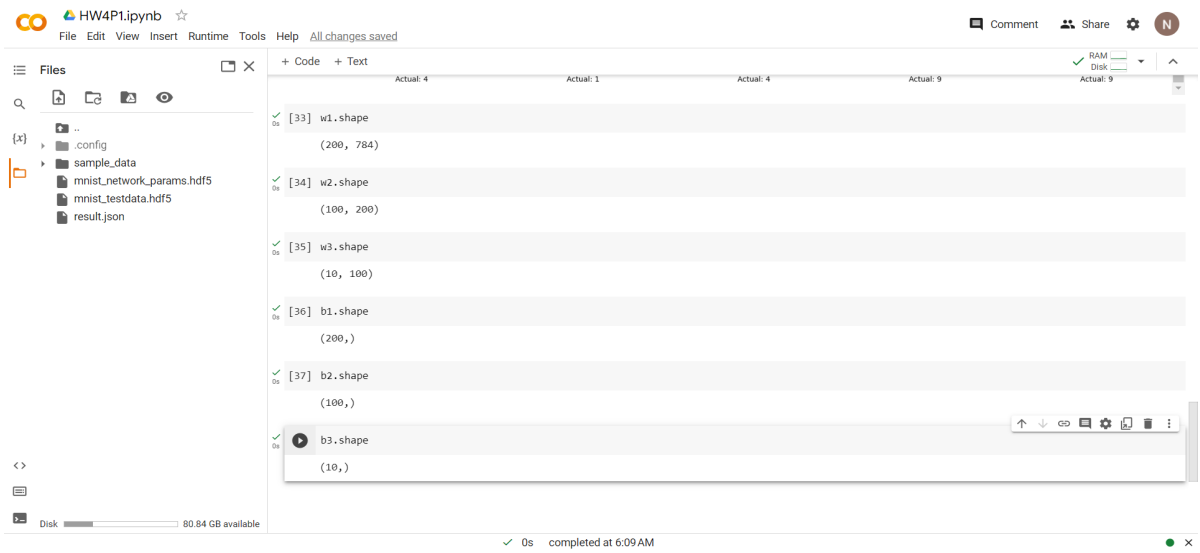



Incorrectly Predicted Digits
Predicted: 6
Actual: 4
Predicted: 7
Actual: 2
Predicted: 3
Actual: 5
Predicted: 7
Actual: 3
Predicted: 0
Actual: 6


Incorrectly Predicted Digits
Predicted: 9
Actual: 4
Predicted: 6
Actual: 4
Predicted: 7
Actual: 2
Predicted: 3
Actual: 5
Predicted: 7
Actual: 3


Incorrectly Predicted Digits
Predicted: 6
Actual: 5
Predicted: 9
Actual: 4
Predicted: 6
Actual: 4
Predicted: 7
Actual: 2
Predicted: 3
Actual: 5

a)


HW4P1.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Files
+ Code
+ Text
.config
sample_data
mnist_network_params.hdf5
mnist_testdata.hdf5
result.json
[33] w1.shape
(200, 784)
[34] w2.shape
(100, 200)
[35] w3.shape
(10, 100)
[36] b1.shape
(200,)
[37] b2.shape
(100,)
b3.shape
(10,)
Disk
80.84 GB available
0s completed at 6:09 AM

The Dimension of 6 keys has been extracted from the mnist_network_params.hdf5, The values of the keys are as follows:

W1 = (200, 784)
W2 = (100, 200)
W3 = (10, 100)
B1 = (200,)
B2 = (100,)
B3 = (10,)

Weights are 2D Array and Biases are 1D Array.

b)

```
with h5py.File('mnist_testdata.hdf5', 'r') as data2:
    x, y = map(np.asarray, (data2['xdata'], data2['ydata']))
```

10000 Images have been extracted from mnist_testdata.hdf5 and stored in 2 variables as x,y.

c)

Relu:

```
def relu(x):
    return np.maximum(x, 0)
```

The Rectified Linear Unit (ReLU) is a popular activation function in neural networks. It is a straightforward and effective function that is defined as follows:

ReLU(x) = max(0, x)

In other words, the ReLU function returns x if the input value x is positive. It yields 0 if x is negative. This function gives the model non-linearity, letting it to learn complicated patterns and representations in the data.

Softmax:

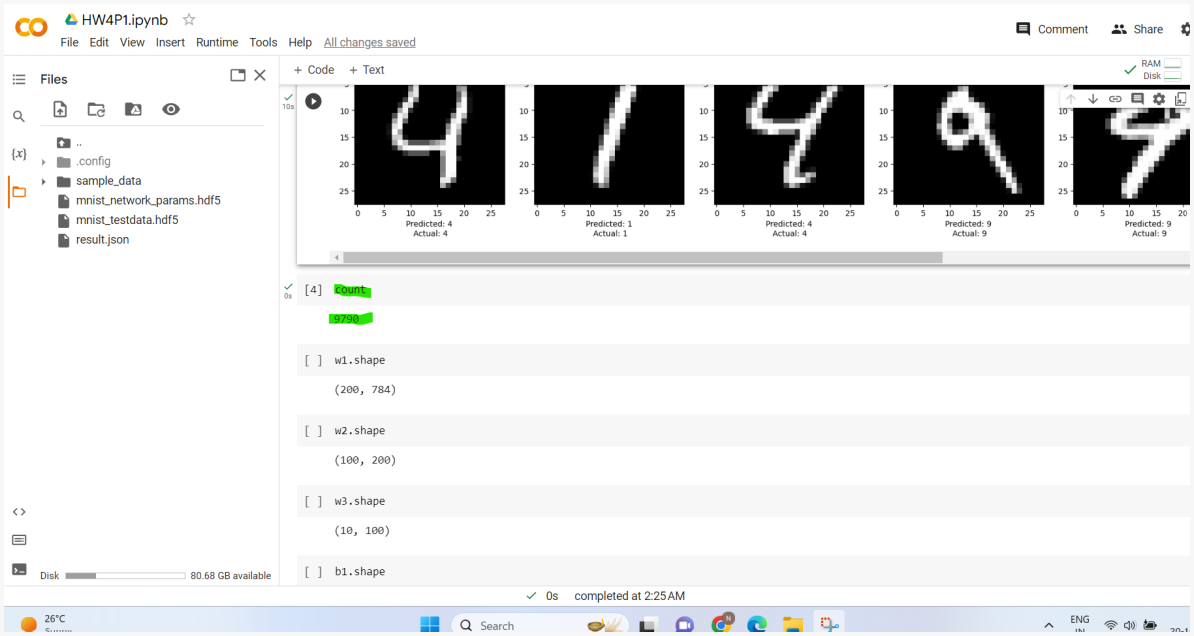
```
ex = np.exp(L3)
ex_sum = np.sum(ex, axis=1).reshape(-1, 1)
sf_out = ex / ex_sum
```

In machine learning and neural networks, the softmax function is widely used to turn a vector of real values into a probability distribution. It takes an input vector and converts it into an output vector, each member of which reflects the probability of the relevant class. The following is the definition of the softmax function:

For an input vector x of length n: Softmax(x)_i = e^(x_i) / Σ(e^(x_j)) for i = 1 to n

The above methods have been used to derive the Relu and Softmax function.

```
d)
result_list = []
for i in range(len(sf_out)):
    result_dict = {
        "activations": sf_out[i].tolist(),
        "index": i,
        "classification": int(sf_out[i].argmax())
    }
    result_list.append(result_dict)

import json

with open("result.json", "w") as json_file:
    json_file.write(json.dumps(result_list))

e)

true_lb = np.argmax(y, axis=1)
predicted_lb = np.argmax(sf_out, axis=1)

count = np.sum(true_lb == predicted_lb)
```



While printing count it shows a count of 9790.

f)

Visual Prediction of the digits make it clear that some digits can be detected correctly just by visual representation rather than the algorithm.

The code above already implements a function visualize, which accepts picture indices and titles to visualize. Matplotlib is then used to display these pictures. By calling visualize(incrt_ind, 'Incorrectly Predicted Digits'), you'll visualize some of the incorrectly classified images, and by calling visualize(crt_ind, 'Correctly Predicted Digits'), you'll visualize some of the correctly classified images.

You may use this code to visually check the images and see if the proper class is evident in the incorrect circumstances. Matplotlib will display images as well as their expected and real labels, allowing you to study the results.

APPENDIX:

```
import h5py
import numpy as np
import matplotlib.pyplot as plt

# Extracting the weights and biases from .hdf5 file
```

```
with h5py.File('mnist_network_params.hdf5', 'r') as data1:
    w1, w2, w3 = map(np.asarray, (data1['W1'], data1['W2'], data1['W3']))
    b1, b2, b3 = map(np.asarray, (data1['b1'], data1['b2'], data1['b3']))

# Extracting the xdata and ydata from the .hdf5 file
with h5py.File('mnist_testdata.hdf5', 'r') as data2:
    x, y = map(np.asarray, (data2['xdata'], data2['ydata']))

# Define the ReLU activation function
def relu(x):
    return np.maximum(x, 0)

# Calculate the inputs for hidden layer 1 using ReLU
L1 = np.dot(x, w1.T) + b1
relu1 = np.maximum(L1, 0)

# Calculate the inputs for hidden layer 2 using ReLU
L2 = np.dot(relu1, w2.T) + b2
relu2 = np.maximum(L2, 0)

# Calculate the Output
L3 = np.dot(relu2, w3.T) + b3

# Define the Softmax activation function
ex = np.exp(L3)
ex_sum = np.sum(ex, axis=1).reshape(-1, 1)
sf_out = ex / ex_sum

# Writing the outputs of the final layer to a JSON file
result_list = []
for i in range(len(sf_out)):
    result_dict = {
        "activations": sf_out[i].tolist(),
        "index": i,
        "classification": int(sf_out[i].argmax())
    }
    result_list.append(result_dict)

import json
```

```
with open("result.json", "w") as json_file:
    json_file.write(json.dumps(result_list))

# Comparing the number of correct predictions made by the model with ydata
true_lb = np.argmax(y, axis=1)
predicted_lb = np.argmax(sf_out, axis=1)

count = np.sum(true_lb == predicted_lb)


# Define the number of samples to visualize for both correct and incorrect
predictions
num = 5

# Get indices where the predictions were made incorrectly and correctly
incrt_ind = [i for i in range(len(true_lb)) if true_lb[i] !=
predicted_lb[i]]
crt_ind = [i for i in range(len(true_lb)) if true_lb[i] ==
predicted_lb[i]]

# Define a function for visualization
def visualize(ind, title):
    for i in range(0, num, 1):
        figure = plt.figure(figsize=[20, 20])
        for j, index in enumerate(ind[i:i + 5]):
            plt.subplot(1, 5, j + 1)
            plt.imshow(x[index].reshape(28, 28), cmap= 'gray')
            plt.xlabel(f'Predicted: {predicted_lb[index]}\n Actual:
{true_lb[index]}')
        plt.suptitle(title, y=0.6, va='center', fontsize=24)
        plt.show()

# Visualization of incorrectly predicted digits
visualize(incrt_ind, 'Incorrectly Predicted Digits')

# Visualization of correctly predicted digits
visualize(crt_ind, 'Correctly Predicted Digits')
```

2. Backprop by Hand

Consider an MLP with three input nodes, two hidden layers, and three outputs. The hidden layers use the ReLU activation function and the output layer users softmax. The weights and biases for this MLP are:

$$W(1) = \begin{bmatrix} 1 & -2 & 1 \\ 3 & 4 & -2 \end{bmatrix}$$

$$b(1) = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

$$W(2) = \begin{bmatrix} 1 & -2 \\ 3 & 4 \end{bmatrix}$$

$$b(2) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$W(3) = \begin{bmatrix} 2 & 2 \\ 3 & -3 \\ 2 & 1 \end{bmatrix}$$

$$b(3) = \begin{bmatrix} 0 \\ -4 \\ -2 \end{bmatrix}$$

(a) Feedforward Computation: Perform the feedforward calculation for the input vector $x = [+1 \; -1 \; +1]^T$. Fill in the following table. Follow the notation used in the slides, i.e., $s(l)$ is the linear activation, $a(l) = h(s(l))$, and $a'(l) = h'(s(l))$.

(b) Backpropagation Computation: Apply standard SGD backpropagation for the input assuming a multi-category cross-entropy loss function and one-hot labeled target: $y = [0 \; 0 \; 1]^T$. Follow the notation used in the slides, i.e., $\delta(l) = \nabla s(l) C$. Enter the delta values in the table below and provide the updated weights and biases assuming a learning rate $\eta = 0.5$.

Back Prop by hand

(1) Feed Forward Propagation

$$W^1 = \begin{bmatrix} 1 & -2 & 1 \\ 3 & 4 & -2 \end{bmatrix} \qquad b^1 = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

$$W^2 = \begin{bmatrix} 1 & -2 \\ 3 & 4 \end{bmatrix} \qquad b^2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$W^3 = \begin{bmatrix} 2 & 2 \\ 3 & -3 \\ 2 & 1 \end{bmatrix} \qquad b^3 = \begin{bmatrix} 0 \\ -4 \\ -2 \end{bmatrix}$$

$$x = \begin{bmatrix} 1 & -1 & 1 \end{bmatrix}^T$$

$$S^1 = W^1 x + b$$

$$= \begin{bmatrix} 1 & -2 & 1 \\ 3 & 4 & -2 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

$$= \begin{bmatrix} 1+2+1 \\ 3-4-2 \end{bmatrix} + \begin{bmatrix} 1 \\ -2 \end{bmatrix} = \begin{bmatrix} 4 \\ -3 \end{bmatrix} + \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

$$= \begin{bmatrix} 5 \\ -5 \end{bmatrix}$$

$$a^1 = h(s^{(1)}) = \text{Relu}(s^1)$$

$$a^1 = \begin{bmatrix} 5 \\ -5 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$$

$$a'^1 = \text{gradient Descendent} = h'(s^{(1)}) \qquad \begin{matrix} 5 > 0 \\ -5 < 0 \end{matrix}$$

$$= \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$s^2 = W^2 a^1 + b^2$$

$$= \begin{bmatrix} 1 & -2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 5+0 \\ 15+0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 6 \\ 15 \end{bmatrix}$$

$$a^2 = h(s^2) = \text{relu}(s^2)$$

$$= \begin{bmatrix} 6 \\ 15 \end{bmatrix}$$

$6 > 0$
$15 > 0$

$$a'^2 = h'(s^2) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

$$s^3 = W^3 a^2 + b^3$$

$$= \begin{bmatrix} 2 & 2 \\ 3 & -3 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 6 \\ 15 \end{bmatrix} + \begin{bmatrix} 0 \\ .4 \\ -2 \end{bmatrix}$$

$$= \begin{bmatrix} 12+30 \\ 18-45 \\ 12+15 \end{bmatrix} + \begin{bmatrix} 0 \\ -4 \\ -2 \end{bmatrix}$$

$$= \begin{bmatrix} 42 \\ -31 \\ 25 \end{bmatrix}$$

$$a^3 = \text{Softmax}(s^3)$$

$$\frac{e^{42}}{e^{42} + e^{-31} +}$$

$e^{42} = 1.739 \times 10^{18}$

$e^{-31} = 3.442 \times 10^{-14}$

$e^{25} = 7.200 \times 10^{10}$

$e^{42} \quad e^{25}$

$= 1.739 \times 10^{18}$

$$a^3 = \begin{bmatrix} \frac{e^{42}}{e^{42}+e^{-31}+e^{25}} \\ \frac{e^{-31}}{e^{42}+e^{-31}+e^{25}} \\ \frac{e^{25}}{e^{42}+e^{-31}+e^{25}} \end{bmatrix} = \begin{bmatrix} 1 \\ 1.979 \times 10^{-32} \\ 4.140 \times 10^{-8} \end{bmatrix}$$

(ii) Backpropagation Computation

$y = [0 \; 0 \; 1]^T$

$S^3 = a^3 - y$

$= \begin{bmatrix} 1 \\ 1.979 \times 10^{-32} \\ 4.140 \times 10^{-8} \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

$$= \begin{bmatrix} 1 \\ 1.979 \times 10^{-32} \\ -1 \end{bmatrix}$$

$$W^3_{new} = W^3 - \eta \delta^3 [a^2]^T$$

$$= \begin{bmatrix} 2 & 2 \\ 3 & -3 \\ 2 & 1 \end{bmatrix} - 0.5 \underset{3\times 1}{\begin{bmatrix} 1 \\ 1.979 \times 10^{-32} \\ -1 \end{bmatrix}} \underset{1\times 2}{\begin{bmatrix} 6 & 15 \end{bmatrix}}$$

$$= \begin{bmatrix} 2 & 2 \\ 3 & -3 \\ 2 & 1 \end{bmatrix} - 0.5 \begin{bmatrix} 6 & 15 \\ 1.18 \times 10^{-31} & 2.968 \times 10^{-31} \\ -6 & -15 \end{bmatrix}$$

$$= \begin{bmatrix} -1 & -5.5 \\ 3 & -3 \\ 5 & 8.5 \end{bmatrix}$$

$$b^3 = b^3 - \eta * \delta^3$$

$$= \begin{bmatrix} 0 \\ -4 \\ -2 \end{bmatrix} - 0.5 \begin{bmatrix} 1 \\ 1.9791 \times 10^{-32} \\ -1 \end{bmatrix}$$

$$= \begin{bmatrix} -0.5 \\ -4 \\ -1.5 \end{bmatrix}$$

$$\delta^2 = a^{(2)} \odot [(W^3)^T \delta^3]$$

$$= \begin{bmatrix} 1 \\ 1 \end{bmatrix} \odot \left[\underset{2\times 3}{\begin{bmatrix} 2 & 3 & 2 \\ 2 & -3 & 1 \end{bmatrix}} \underset{3\times 1}{\begin{bmatrix} 1 \\ 1.9791 \times 10^{-32} \\ -1 \end{bmatrix}} \right]$$

$$= \begin{bmatrix} 1 \\ 1 \end{bmatrix} \odot \begin{bmatrix} 24 & 5.93 \times 10^{-32} & -2 \\ 2 & -5.93 \times 10^{-32} & -1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 \\ 1 \end{bmatrix} \odot \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$W^2_{new} = W^2 - \eta \delta^2 [a^1]^T$$

$$= \begin{bmatrix} 1 & -2 \\ 3 & 4 \end{bmatrix} - 0.5 \begin{bmatrix} 0 \\ 1 \end{bmatrix} \begin{bmatrix} 5 & 0 \end{bmatrix}$$

$2 \times 1 \quad 1 \times 2$

$$= \begin{bmatrix} 1 & -2 \\ 3 & 4 \end{bmatrix} - 0.5 \begin{bmatrix} 0 & 0 \\ 5 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & -2 \\ 3 & 4 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ 2.5 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & -2 \\ 0.5 & 4 \end{bmatrix}$$

$$b^2_{new} = b^2 - \eta s^2$$

$$= \begin{bmatrix} 1 \\ 0 \end{bmatrix} - 0.5 \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 0.5 \end{bmatrix}$$

$$= \begin{bmatrix} 1 \\ -0.5 \end{bmatrix}$$

$$s^1 = a^{(1)} \odot \left[\left[w^2 \right]^T s^2 \right]$$

$$= \begin{bmatrix} 1 \\ 0 \end{bmatrix} \odot \left[\underset{2\times 2}{\begin{bmatrix} 1 & 3 \\ -2 & 4 \end{bmatrix}} \underset{2\times 1}{\begin{bmatrix} 0 \\ 1 \end{bmatrix}} \right]$$

$$= \begin{bmatrix} 1 \\ 0 \end{bmatrix} \odot \begin{bmatrix} 0+3 \\ 0+4 \end{bmatrix}$$

$$= \begin{bmatrix} 1 \\ 0 \end{bmatrix} \odot \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

$$= \begin{bmatrix} 3 \\ 0 \end{bmatrix}$$

$$b'_{new} = b' - \eta s'$$

$$= \begin{bmatrix} 1 \\ -2 \end{bmatrix} - 0.5 \begin{bmatrix} 3 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 \\ -2 \end{bmatrix} - \begin{bmatrix} 1.5 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} -0.5 \\ -2 \end{bmatrix}$$

$$W'_{new} = W_1 - \eta \times s' \times x^T$$

$$= \begin{bmatrix} 1 & -2 & 1 \end{bmatrix} - 0.5 \left(\begin{bmatrix} 3 \\ \end{bmatrix} \begin{bmatrix} 1 & -1 & 1 \end{bmatrix} \right)$$

$$\begin{bmatrix} 3 & 4 & -2 \end{bmatrix} - \begin{bmatrix} 0 \\ 2 \end{bmatrix}_{1\times 3}$$

$$= \begin{bmatrix} 1 & -2 & 1 \\ 3 & 4 & -2 \end{bmatrix} - 0.5 \begin{bmatrix} 3 & -3 & 3 \\ 0 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & -2 & 1 \\ 3 & 4 & -2 \end{bmatrix} - \begin{bmatrix} 1.5 & -1.5 & 1.5 \\ 0 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} -0.5 & -0.5 & 0.5 \\ 3 & 4 & -2 \end{bmatrix}$$