Department of electrical and computer engineering, Fall 2022

AIR UNIVERSITY ISLAMABAD

COMPLEX ENGINEERING PROBLEM

Artificial Intelligence

Group Members:

M HAMZA AMIN 191862

HAMZA SHAFIQUE 191205

M YASIR 191236

FAHAD MUSHTAQ 191235

HAMMAD KHAN 191216

Submitted to: SIR M FAROOQ KHAN

OBJECTIVE:

The objective of this complex engineering activity is to carry out research, analysis, design, investigation, and implementation of a real-world complex programming project that has the following attributes:

1. The activity requires abstract thinking, originality in analysis to formulate suitable programming models of the activity;

2. The activity involves the creative use of programming principles and research- based knowledge in novel ways;

3. The activity can extend beyond previous experiences by applying principles-based approaches.

PROBLEM STATEMENT

□ MNIST DIGIT CLASSIFICATION USING NEURAL NETWORKS

i. Develop a classifier that can accurately predict whether an image in the MNIST dataset contains a digit X or Y.

ii It is a group project, with one group having a maximum of five members.

iii. Take the sum of the last two digits of the roll numbers of all group members. Find the average; suppose the average comes out to be 57. So X=5 and Y=7. This group will need to build a classifier that can accurately predict whether an image in the MNIST dataset contains a digit 5 or 7. The neural network architecture that this group needs to use is given below:

□ Input Layer: MNIST Images (Flatten)

□ Hidden Layer 1: X Neurons; for the above example, five neurons.

□ Hidden Layer 2: Y Neurons; for the above example, seven neurons.

□ Output Layer: 1 neuron.

iv. You can use any activation function you want.

v. Initialize all the required parameters randomly or of your choice.

vi. Test your classifier on test data; see lab 1, for how you split the data into test and train images.

vii. Find the accuracy on test data, if you think the accuracy didn't come out to be good, you can change the parameters, the number of hidden layers and neurons to increase the accuracy, but you will need to show your implementations with the architecture that uses X and Y neurons as well as with the one you have used to improve accuracy.

INTRODUCTION:

The MNIST dataset is a widely used dataset for the task of handwritten digit classification. It consists of a collection of images of handwritten digits (0-9), along with their corresponding labels. The goal of this problem is to develop a classifier that can accurately predict the label of a given image, i.e., whether it contains a digit X or Y.

Our approach to solving this problem is to use a neural network, which is a type of machine learning model that is particularly well-suited for image classification tasks. In this project, we will design and train a neural network to classify images from the MNIST dataset, and evaluate its performance on a test set.

IMPLEMENTATION

NEURAL NETWORK:

A neural network is a machine learning model that is inspired by the structure and function of the human brain. It is composed of a large number of interconnected processing units, called neurons, which are organized into layers.

The input layer of a neural network receives input data, which is then processed by one or more hidden layers. The hidden layers use weights and biases to transform the input data into a new representation, which is then passed through to the output layer. The output layer produces the final output of the neural network, which can be a classification, a prediction, or some other type of output.

Neural networks are trained using an optimization algorithm, such as stochastic gradient descent, which adjusts the weights and biases of the network in order to minimize a loss function. The loss function measures the difference between the network's predicted output and the true output, and the optimization algorithm adjusts the weights and biases to reduce this difference.

Neural networks are particularly well-suited for tasks that involve pattern recognition and classification, such as image classification, speech recognition, and natural language processing. They are also used in a wide variety of other applications, including finance, healthcare, and robotics.

THEORETICALLY:

Department of electrical and computer engineering, Fall 2022

Group members have the following roll numbers

191862
191235
191205
191236
191216

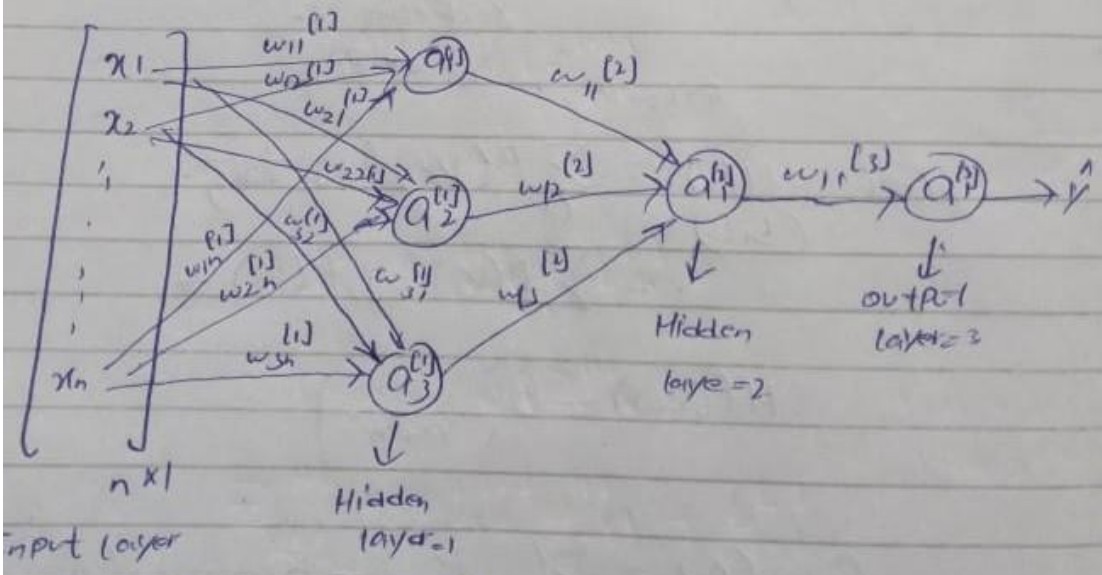
Taking last 2 digits of each roll now

$(62 + 35 + 05 + 36 + 16)/5 = \frac{154}{5}$

Add -1 to 154

$= \frac{155}{5} = 31$

Hidden layer 1 contain 3 neurons
Hidden layer 2 contain 1 neuron

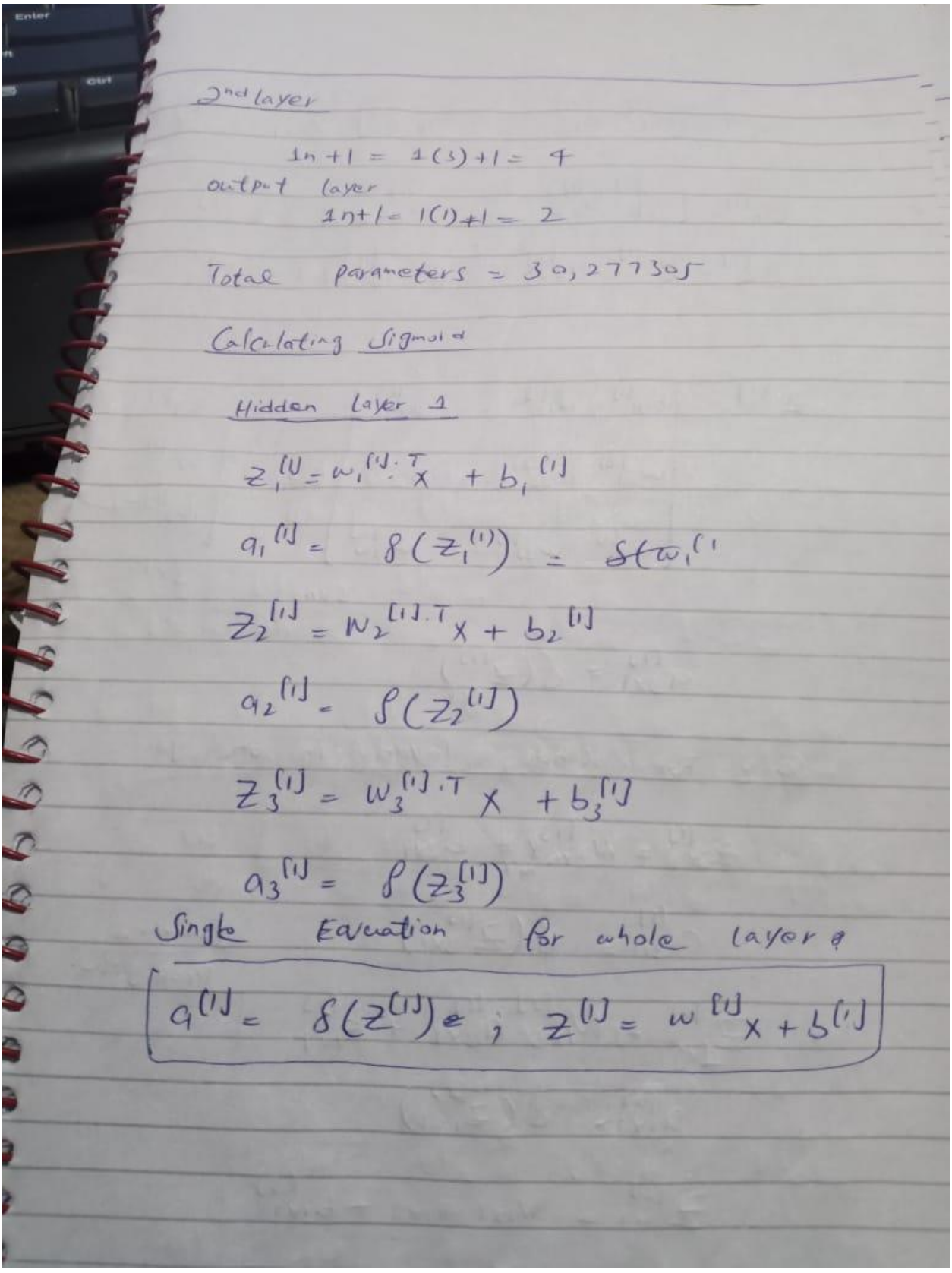


$h = (10,092,41)$

$X \rightarrow (10,092,41 \times 1)$ ⇒ Feature vector

1st layer

$3n + 3 = 3(10,092,432) + 3 = 32\,77,297$



2nd layer

$In+1 = 1(3)+1 = 4$

output layer

$In+1 = 1(1)+1 = 2$

Total parameters = 30,277305

Calculating Sigmoid

Hidden layer 1

$$z_1^{[1]} = w_1^{[1].T} x + b_1^{[1]}$$

$$a_1^{[1]} = g(z_1^{(1)}) = g(w_1^{(1}$$

$$z_2^{[1]} = w_2^{[1].T} x + b_2^{[1]}$$

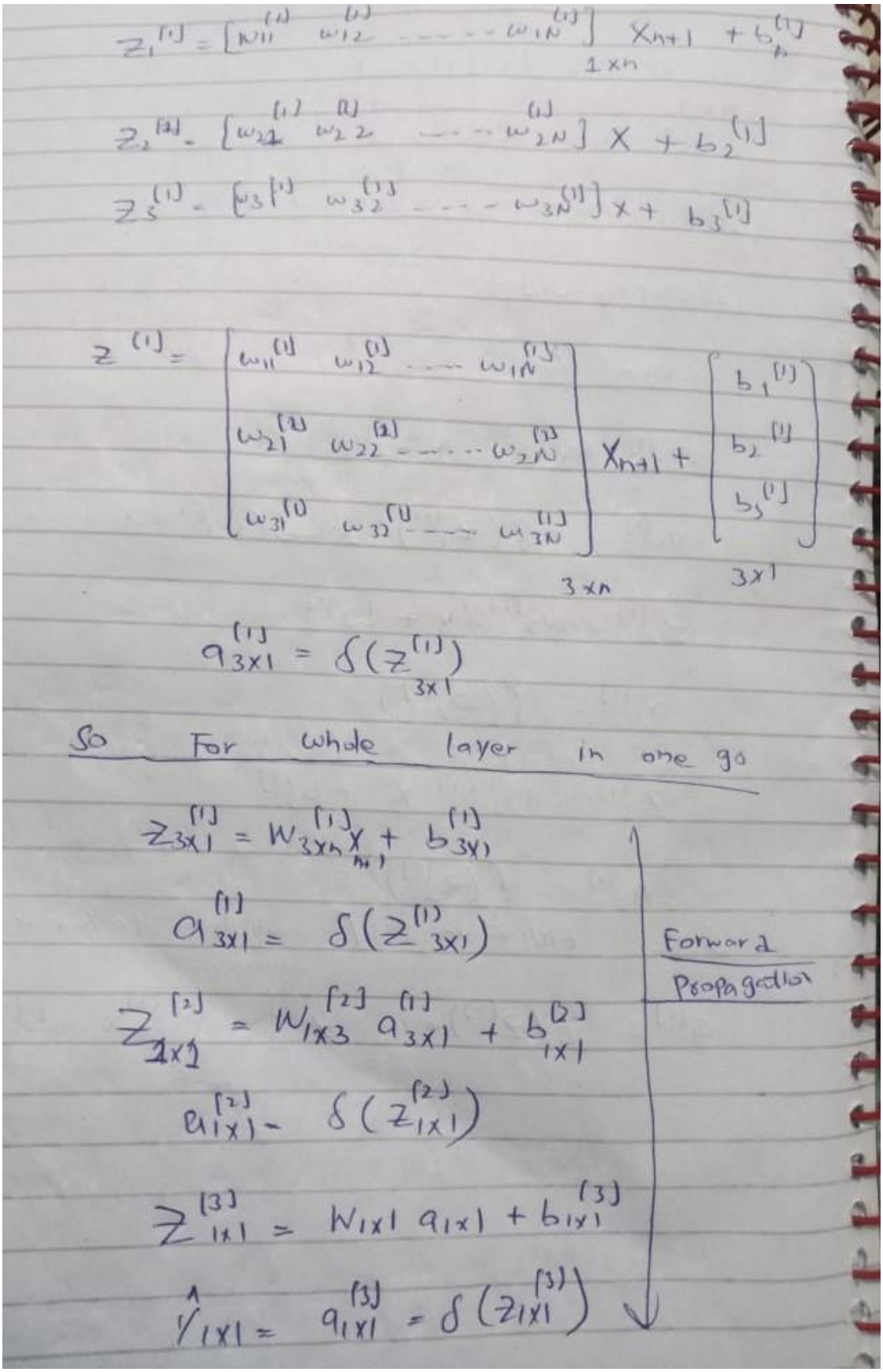
$$a_2^{[1]} = g(z_2^{[1]})$$

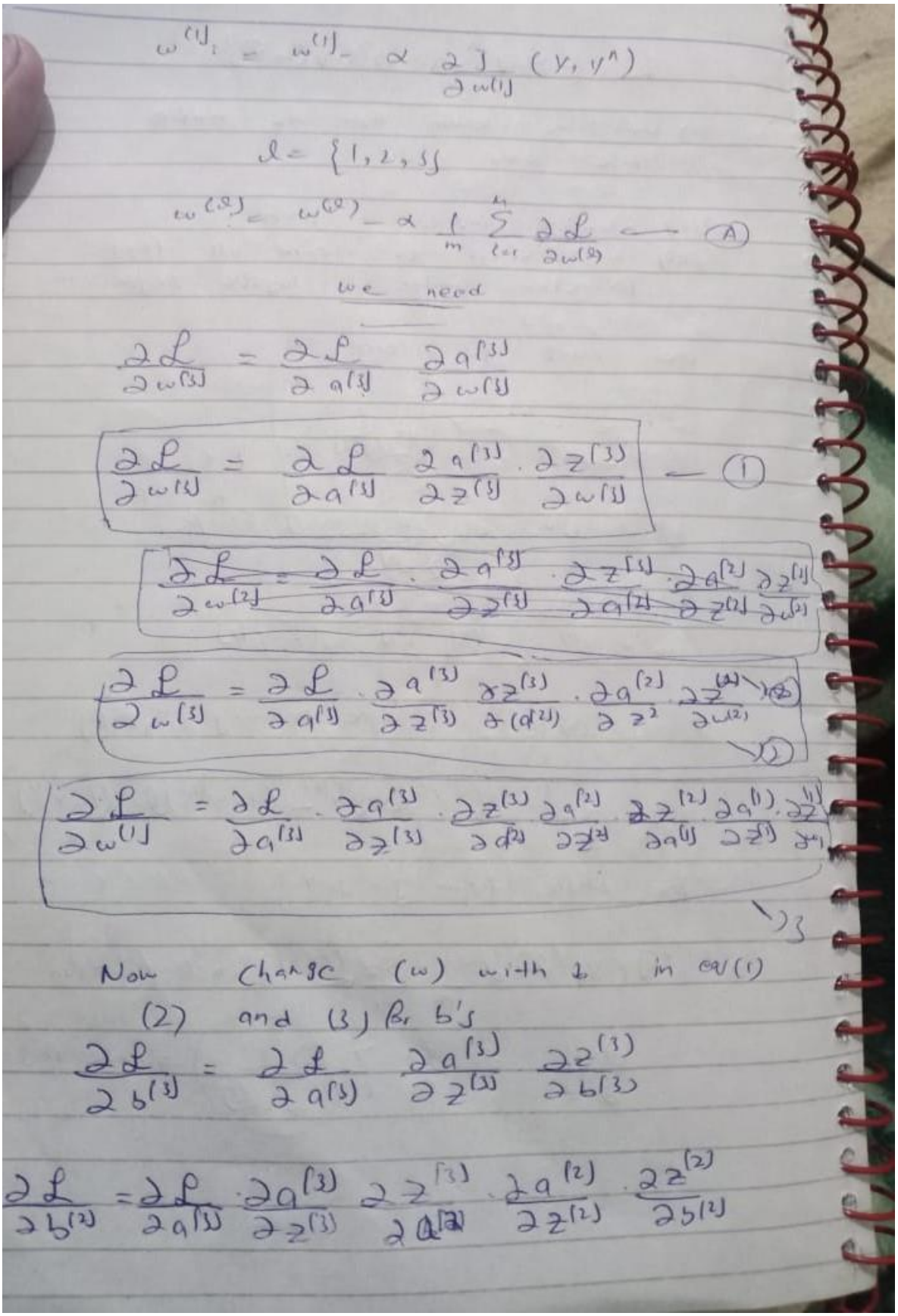
$$z_3^{[1]} = w_3^{[1].T} x + b_3^{[1]}$$

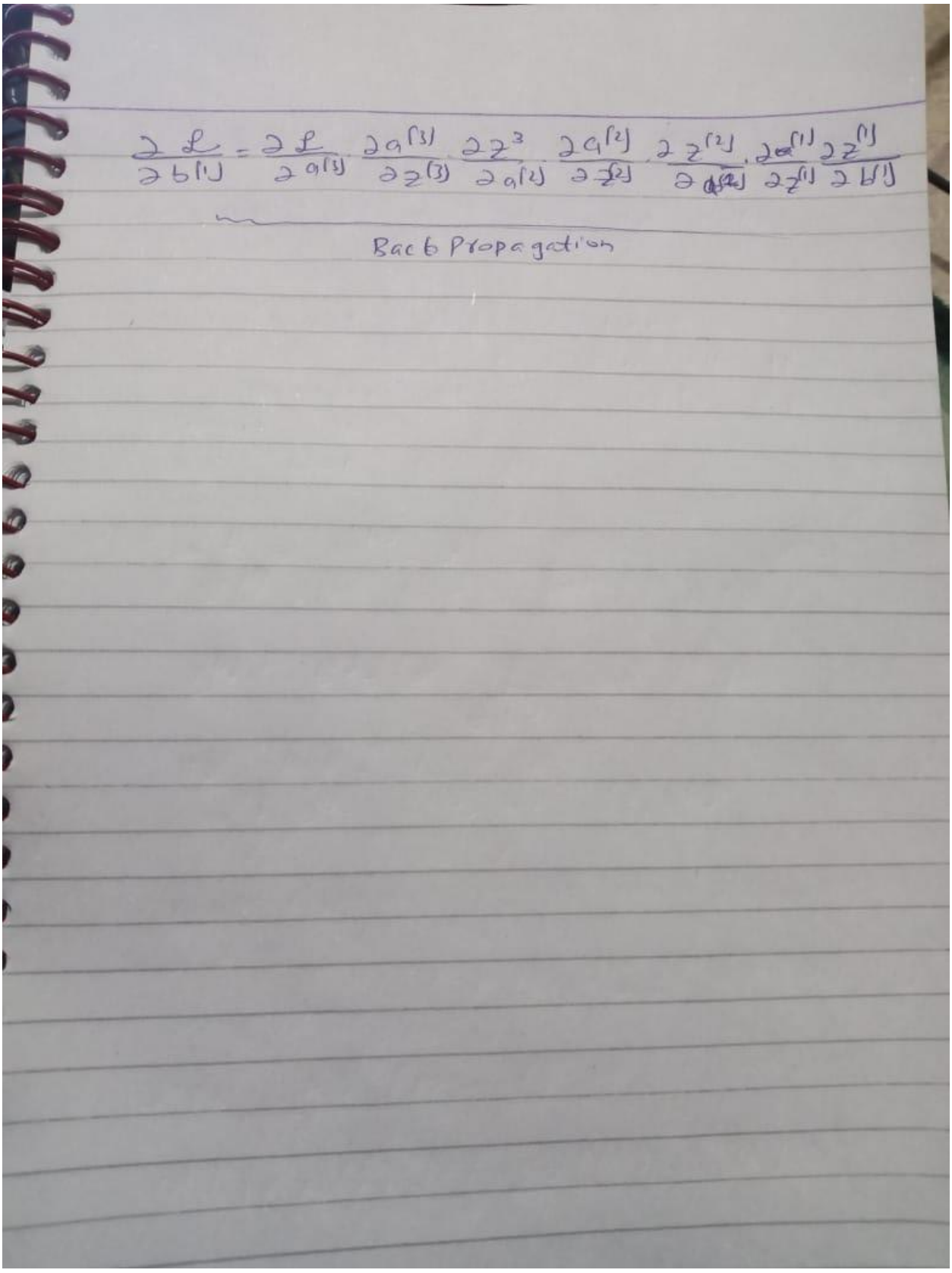
$$a_3^{[1]} = g(z_3^{[1]})$$

Single Equation for whole layer:

$$a^{[1]} = g(z^{[1]}) ; \quad z^{[1]} = w^{[1]} x + b^{[1]}$$


$z_1^{[1]} = [w_{11}^{[1]} \; w_{12}^{[1]} \; \cdots \; w_{1N}^{[1]}]_{1\times n} X_{n+1} + b_1^{[1]}$
$z_2^{[1]} = [w_{21}^{[1]} \; w_{22}^{[1]} \; \cdots \; w_{2N}^{[1]}] X + b_2^{[1]}$
$z_3^{[1]} = [w_3^{[1]} \; w_{32}^{[1]} \; \cdots \; w_{3N}^{[1]}] X + b_3^{[1]}$
$z^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} & \cdots & w_{1N}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} & \cdots & w_{2N}^{[1]} \\ w_{31}^{[1]} & w_{32}^{[1]} & \cdots & w_{3N}^{[1]} \end{bmatrix}_{3\times n} X_{n+1} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{bmatrix}_{3\times 1}$
$a_{3\times 1}^{[1]} = g(z_{3\times 1}^{[1]})$
So For whole layer in one go
$z_{3\times 1}^{[1]} = W_{3\times n}^{[1]} X_{n+1} + b_{3\times 1}^{[1]}$
$a_{3\times 1}^{[1]} = g(z_{3\times 1}^{[1]})$
$z_{1\times 1}^{[2]} = W_{1\times 3}^{[2]} a_{3\times 1}^{[1]} + b_{1\times 1}^{[2]}$
$a_{1\times 1}^{[2]} = g(z_{1\times 1}^{[2]})$
$z_{1\times 1}^{[3]} = W_{1\times 1} a_{1\times 1} + b_{1\times 1}^{[3]}$
$\hat{y}_{1\times 1} = a_{1\times 1}^{[3]} = g(z_{1\times 1}^{[3]})$
Forward Propagation


$w^{[1]}_i = w^{[1]} - \alpha \frac{\partial J}{\partial w^{[1]}} (Y, \hat{Y})$
$L = \{1, 2, 5\}$
$w^{[2]} = w^{[2]} - \alpha \frac{1}{m} \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial w^{[2]}}$ — (A)
we need
$\frac{\partial \mathcal{L}}{\partial w^{[3]}} = \frac{\partial \mathcal{L}}{\partial a^{[3]}} \frac{\partial a^{[3]}}{\partial w^{[3]}}$
$\frac{\partial \mathcal{L}}{\partial w^{[3]}} = \frac{\partial \mathcal{L}}{\partial a^{[3]}} \frac{\partial a^{[3]}}{\partial z^{[3]}} \cdot \frac{\partial z^{[3]}}{\partial w^{[3]}}$ — (1)
$\frac{\partial \mathcal{L}}{\partial w^{[2]}} = \frac{\partial \mathcal{L}}{\partial a^{[3]}} \cdot \frac{\partial a^{[3]}}{\partial z^{[3]}} \cdot \frac{\partial z^{[3]}}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial w^{[2]}}$
$\frac{\partial \mathcal{L}}{\partial w^{[3]}} = \frac{\partial \mathcal{L}}{\partial a^{[3]}} \cdot \frac{\partial a^{[3]}}{\partial z^{[3]}} \frac{\partial z^{[3]}}{\partial (a^{[2]})} \cdot \frac{\partial a^{[2]}}{\partial z^{2}} \frac{\partial z^{[2]}}{\partial w^{[2]}}$ (2)
(1)
$\frac{\partial \mathcal{L}}{\partial w^{[1]}} = \frac{\partial \mathcal{L}}{\partial a^{[3]}} \cdot \frac{\partial a^{[3]}}{\partial z^{[3]}} \cdot \frac{\partial z^{[3]}}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial z^{[1]}} \frac{\partial z^{[1]}}{\partial w^{[1]}}$
Now change (w) with b in eq (1)
(2) and (3) for b's
$\frac{\partial \mathcal{L}}{\partial b^{[3]}} = \frac{\partial \mathcal{L}}{\partial a^{[3]}} \frac{\partial a^{[3]}}{\partial z^{[3]}} \frac{\partial z^{[3]}}{\partial b^{[3]}}$
$\frac{\partial \mathcal{L}}{\partial b^{[2]}} = \frac{\partial \mathcal{L}}{\partial a^{[3]}} \cdot \frac{\partial a^{[3]}}{\partial z^{[3]}} \frac{\partial z^{[3]}}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial b^{[2]}}$


Back Propagation

CODE SCREENSHOT:

In [410]:
```
import tensorflow as tf
import numpy as np
import math
```

In [411]:
```
# Load the MNIST dataset
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Create empty lists to store the images and labels
one_images = []
three_images = []
one_labels = []
three_labels = []

# Iterate over the training data
for x, y in zip(x_train, y_train):
  # If the label is 2, append the image and label to the list of 1's
  if y == 1:
    one_images.append(x)
    one_labels.append(y)
  # If the label is 3, append the image and label to the list of 3's
  elif y == 3:
    three_images.append(x)
    three_labels.append(y)

# Print the length of the lists to verify that the images were correctly added
print(f'Number of 1 images: {len(one_images)}')
print(f'Number of 3 images: {len(three_images)}')
print("Shape of one_images",np.shape(one_images))
print("Shape of three_images",np.shape(three_images))
```

```
Number of 1 images: 6742
Number of 3 images: 6131
Shape of one_images (6742, 28, 28)
Shape of three_images (6131, 28, 28)
```

In [412]:
```
# Assume that the images are stored in a list called 'images'
# Each image is a 2D numpy array of size 28x28
flattened_1images = []
flattened_3images = []

# Iterate over the images
for image in one_images:
  # Flatten the image by reshaping it to a 1D array of size 784
  flattened_1image = image.reshape(784)
  flattened_1images.append(flattened_1image)
print(np.shape(flattened_1images[0]))
for image in three_images:
  # Flatten the image by reshaping it to a 1D array of size 784
  flattened_3image = image.reshape(784)
  flattened_3images.append(flattened_3image)
print(np.shape(flattened_3images[0]))

# The flattened_images list now contains the flattened images
```

```
(784,)
(784,)
```

```
(784,)
```

In [424]:
```
reshaped_1images = []
reshaped_3images = []

# Iterate over the flattened images
for image in flattened_1images:
  # Reshape the image from a 1D array of size 784 to a 2D array of size (784, 1)
  reshaped_1image = image.reshape(784, 1)
  reshaped_1images.append(reshaped_1image)
print(np.shape(reshaped_1images[0]))

# The reshaped_images list now contains the reshaped images
# Iterate over the flattened images
for image in flattened_3images:
  # Reshape the image from a 1D array of size 784 to a 2D array of size (784, 1)
  reshaped_3image = image.reshape(784, 1)
  reshaped_3images.append(reshaped_3image)
print(np.shape(reshaped_3images[0]))

# The reshaped_images list now contains the reshaped images
```

```
(784, 1)
(784, 1)
```

NEURAL NETWORK WITH IMAGE 1 IN MNSIT DATASET

In [444]:
```
m = 6742
num_iterations = 1000
x = []
cost = []
alpha = 0.021
def sigmoid(x):
  # Implement the sigmoid function
  return 1 / (1 + np.exp(-x))
def firstlayers(inputimage):
  num_iterations = 1000
  learning_rate = 0.01
  # Initialize the weight and bias parameters to zero
  w1_1 = np.random.rand(784)
  w2_1 = np.random.rand(784)
  w3_1 = np.random.rand(784)
  b1_1 = np.random.rand()
  b2_1 = np.random.rand()
  b3_1 = np.random.rand()
  w1_2 = np.random.rand(1)
  w2_2 = np.random.rand(1)
  w3_2 = np.random.rand(1)
  b1_2 = np.random.rand()
  b1_3 = np.random.rand()
  w1_3 = np.random.rand()
  output_1 = []
  output_2 = []
  output_3 = []
  wstack_1 = np.vstack([[w1_1] ,[w2_1], [w3_1]])
  wstack_2 = np.hstack([[w1_2] ,[w2_2], [w3_2]])
  bstack = np.vstack([[b1_1] ,[b2_1], [b3_1]])
  print(np.shape(wstack_1))
  print(np.shape(bstack))
  print(np.shape(wstack_2))
  print(np.shape(inputimage))
  # Iterate over the number of iterations
  for i in range(m):
  # Calculate the output for the all three hidden layer
      output_1 = sigmoid(np.dot(wstack_1,inputimage[i]) + bstack)
      output_2 = sigmoid(np.dot(wstack_2,output_1) + b1_2)
      output_3 = sigmoid(np.dot(w1_3,output_2) + b1_3)
      likelihood = -(np.dot(one_labels[i],np.log(output_3))+(np.dot((1-one_labels[i]),(np.log(1-output_3)))))
      cost = (1/m)*(np.sum(likelihood))
      w1_bupdated = np.dot((output_3-one_labels[i]),output_2.T)
      wstack_bupdated = w1_bupdated.T*(np.dot(output_2,(1-output_2)))*output_1.T
      wprod = (output_3-one_labels[i])*(np.dot(output_2,(1-output_2)))*(np.dot(output_1.T,(1-output_1)))
```

```
# Iterate over the number of iterations
for i in range(m):
# Calculate the output for the all three hidden layer
    output_1 = sigmoid(np.dot(wstack_1,inputimage[i]) + bstack)
    output_2 = sigmoid(np.dot(wstack_2,output_1) + b1_2)
    output_3 = sigmoid(np.dot(w1_3,output_2) + b1_3)
    likelihood = -(np.dot(one_labels[i],np.log(output_3))+(np.dot((1-one_labels[i]),(np.log(1-output_3)))))
    cost = (1/m)*(np.sum(likelihood))
    w1_3updated = np.dot((output_3-one_labels[i]),output_2.T)
    wstack_2updated = w1_3updated.T*(np.dot(output_2,(1-output_2)))*output_1.T
    wprod = (output_3-one_labels[i])*(np.dot(output_2,(1-output_2)))*(np.dot(output_1.T,(1-output_1)))
    wprod = np.resize(wprod, (3,))
# Now the array can be reshaped to a shape of (3,1)
    wprod = wprod.reshape((3,1))
    mulimg = wprod*inputimage[i].T
    wstack_1 =(np.dot(wstack_2updated.T,w1_3updated.T))*mulimg
    w1_3 = (w1_3 - alpha*(1/m)*(np.sum(w1_3updated)))
    wstack_2 = (wstack_2 - alpha*(1/m)*(np.sum(wstack_2updated)))
    wstack_1 = (wstack_1 - alpha*(1/m)*(np.sum(wstack_1)))
#   print(np.shape(likelihood))
    print("Likelihood: {}, Cost: {}".format(likelihood, cost))
return
```

```
In [415]: firstlayera(reshaped_1images)
```

```
(3, 784)
(3, 1)
(1, 3)
(6742, 784, 1)
Likelihood: [[0.37903894]], Cost: 5.622054917759653e-05
Likelihood: [[0.3807882]], Cost: 5.6480005918918806e-05
Likelihood: [[0.38875275]], Cost: 5.7661339086318054e-05
Likelihood: [[0.38078772]], Cost: 5.647993514280299e-05
Likelihood: [[0.38875236]], Cost: 5.766128199172074e-05
Likelihood: [[0.38078725]], Cost: 5.647986436682228e-05
Likelihood: [[0.38875198]], Cost: 5.766122489722221e-05
```

NEURAL NETWORK FOR IMAGE WITH 3 IN MNIST DATASET

```
In [416]: m2 = 6131
num_iterations = 1000
X = []
cost = []
alpha = 0.821
def sigmoid(x):
  # Implement the sigmoid function
  return 1 / (1 + np.exp(-x))
def firstlayerb(inputimage):
  num_iterations = 1000
  learning_rate = 0.01
  # Initialize the weight and bias parameters to zero
  w1_1 = np.random.rand(784)
  w2_1 = np.random.rand(784)
  w3_1 = np.random.rand(784)
  b1_1 = np.random.rand()
  b2_1 = np.random.rand()
  b3_1 = np.random.rand()
  w1_2 = np.random.rand(1)
  w2_2 = np.random.rand(1)
  w3_2 = np.random.rand(1)
  b1_2 = np.random.rand()
  b1_3 = np.random.rand()
  w1_3 = np.random.rand()
  output_1 = []
  output_2 = []
  output_3 = []
  wstack_1 = np.vstack([[w1_1] ,[w2_1], [w3_1]])
  wstack_2 = np.hstack([[w1_2] ,[w2_2], [w3_2]])
  bstack = np.vstack([[b1] ,[b2], [b3]])
  print(np.shape(b1_2))
  print(np.shape(wstack_1))
  print(np.shape(bstack))
  print(np.shape(wstack_2))
  print(np.shape(inputimage))
  # Iterate over the number of iterations
  for i in range(m):
  # Calculate the output for the all three hidden layer
      output_1 = sigmoid(np.dot(wstack_1,inputimage[i]) + bstack)
      output_2 = sigmoid(np.dot(wstack_2,output_1) + b1_2)
      output_3 = sigmoid(np.dot(w1_3,output_2) + b1_3)
      likelihood = -(np.dot(three_labels[i],np.log(output_3))+(np.dot((1-three_labels[i]),(np.log(1-output_3)))))
      cost = (1/m)*(np.sum(likelihood))
      w1_3updated = np.dot((output_3-three_labels[i]),output_2.T)
```

```
    print(np.shape(wstack_2))
    print(np.shape(inputimage))
    # Iterate over the number of iterations
    for i in range(m):
    # Calculate the output for the all three hidden layer
        output_1 = sigmoid(np.dot(wstack_1,inputimage[i]) + bstack)
        output_2 = sigmoid(np.dot(wstack_2,output_1) + b1_2)
        output_3 = sigmoid(np.dot(w1_3,output_2) + b1_3)
        likelihood = -(np.dot(three_labels[i],np.log(output_3))+(np.dot((1-three_labels[i]),(np.log(1-output_3)))))
        cost = (1/m)*(np.sum(likelihood))
        w1_3updated = np.dot((output_3-three_labels[i]),output_2.T)
        wstack_2updated = w1_3updated.T*(np.dot(output_2,(1-output_2)))*output_1.T
        wprod = (output_3-three_labels[i])*(np.dot(output_2,(1-output_2)))*(np.dot(output_1.T,(1-output_1)))
        wprod = np.resize(wprod, (3,))
# Now the array can be reshaped to a shape of (3,1)
        wprod = wprod.reshape((3,1))
        muling = wprod*inputimage[i].T
        wstack_1 =(np.dot(wstack_2updated.T,w1_3updated.T))*muling
        w1_3 = (w1_3 - alpha*(1/m)*(np.sum(w1_3updated)))
        wstack_2 = (wstack_2 - alpha*(1/m)*(np.sum(wstack_2updated)))
        wstack_1 = (wstack_1 - alpha*(1/m)*(np.sum(wstack_1)))
#   print(np.shape(likelihood))
        print("\likelihood:",likelihood)
        print("Cost",cost)
    return
```

In [427]:
```
firstlayerb(reshaped_3images)
```

```
()
(3, 784)
(3, 1)
(1, 3)
(6131, 784, 1)
LikeliHood: [[-2.02626198]]
Cost -0.0003005431593627246
LikeliHood: [[-1.97188295]]
Cost -0.0002924774480665060
LikeliHood: [[-1.79006589]]
Cost -0.000265509625093121
LikeliHood: [[-1.97190244]]
Cost -0.0002924803388360589
LikeliHood: [[-1.79007955]]
Cost -0.0002655116504128312
LikeliHood: [[-1.97192193]]
Cost -0.0002924832296047475
LikeliHood: [[-1.7900932]]
Cost -0.00026551367573072907
LikeliHood: [[-1.97194142]]
```


```
()
(3, 784)
(3, 1)
(1, 3)
(6131, 784, 1)
LikeliHood: [[-2.02626198]]
Cost -0.0003005431593627246
LikeliHood: [[-1.97188295]]
Cost -0.0002924774480665069
LikeliHood: [[-1.79006589]]
Cost -0.000265509625093121
LikeliHood: [[-1.97190244]]
Cost -0.0002924803388360589
LikeliHood: [[-1.79007955]]
Cost -0.0002655116504128312
LikeliHood: [[-1.97192193]]
Cost -0.0002924832296047475
LikeliHood: [[-1.7900932]]
Cost -0.00026551367573072907
LikeliHood: [[-1.97194142]]
```


In [ ]:

CONCLUSION

Neural Network is the combination of neurons at differnt levels to predict different things in the image which we cannot find through the simple logistic regression.

In our project we perform Neural Network on the aminst data set. In our case we Implement it on the Images of 1 and 3. First we load the aminst dataset and then we separate out the images of 1 and 3.In our Neural network we have 4 layers, 1st one is input layer having the input images of 1 and 3, 2nd layer is Hidden layer 1 haviny 3 neurons, 3rd layer is Hidden layer 2 having 1 neurons and the last layer is output layer.

1st we flatten the input images and then reshsape it to get the optimal shapes for the multipilication of the matrics.We initialize the value of W and b with rand function for all three layers. Then we implement the sigmoid function and get the values at each neuron. And these output is feed as input for the next layer. And then we find the liklihood (loss) for each image. And in combine form this loss is Cost for the images. And to update our perameters we use back propogation method to get the maximum accuracy.