Assignment 4 Report

Name: Bhavesh Sood
Roll No: 2019355
Branch: CSAI

Q1)

1. Generated data from multivariate gaussian with the given mean and covariance.

```
import numpy as np
np.random.seed(43)
import matplotlib.pyplot as plt
mean1=[0,0,0]
cov1 = [[1, 0.8, 0.8], [0.8, 1, 0.8],[0.8, 0.8, 1]]
X= np.random.multivariate_normal(mean1, cov1, 1000)
print(X)
```

```
[[-0.37783608  0.11676566 -0.45780845]
 [ 0.3471694   0.30203767  0.8447309 ]
 [-0.0026565  -1.33003507 -0.05867343]
 ...
 [ 0.59601504  0.73963566  0.50036625]
 [-0.00954922  0.69076099 -0.27270638]
 [-1.55886924 -1.23020385 -1.11034104]]
```

Then Normalized the data so that value in each dimension lies in [0,1]

```
X_min=np.min(X)
X_max=np.max(X)
X = (X - X_min)/(X_max - X_min)
X
```

```
array([[0.46034139, 0.53913369, 0.44760142],
       [0.57583806, 0.56864837, 0.65510187],
       [0.5201092 , 0.30865175, 0.51118545],
       ...,
       [0.6154803 , 0.63835972, 0.60024302],
       [0.51901116, 0.63057376, 0.47708902],
       [0.27219743, 0.32455532, 0.34365001]])
```

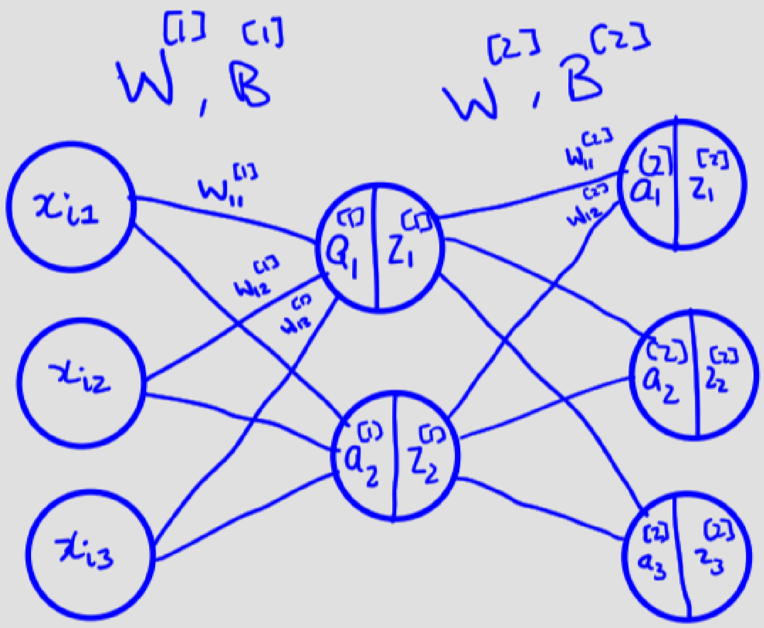
Finally divided the data into training and test-set.

```
X_train=X[:,:800]
X_test=X[:,800:]
print(X_train.shape,X_test.shape)
```

```
(3, 800) (3, 200)
```

2. We have to encode a autoencoder with one input layer which would be having 3 nodes as it's a 3 dimensional data , then one hidden layer with 2 nodes, and finally output layer with 3 nodes.

Setup



$$W^{[1]}.shape = (2,3) = \begin{bmatrix} W_{11}^{[1]} & W_{12}^{[1]} & W_{13}^{[1]} \\ W_{21}^{[1]} & W_{22}^{[1]} & W_{23}^{[1]} \end{bmatrix}, \quad B^{[1]}.shape = (2,1) = \begin{bmatrix} B_1^{[1]} \\ B_2^{[1]} \end{bmatrix}$$

$$W^{[2]}.shape = (3,2) = \begin{bmatrix} W_{11}^{[2]} & W_{12}^{[2]} \\ W_{21}^{[2]} & W_{22}^{[2]} \\ W_{31}^{[2]} & W_{32}^{[2]} \end{bmatrix}, \quad B^{[2]}.shape = (3,1) = \begin{bmatrix} B_1^{[2]} \\ B_2^{[2]} \\ B_3^{[2]} \end{bmatrix}$$

Superscript $^{[2]}$ → represents layer
Subscript w_{ij} → represents from node j to i

Forward Propagation:

The activation function used for output and hidden layer is sigmoid.

Forward Propagation:

Firstly we define Matrices A1, Z1, A2, Z2 as follows:

Let $X_i = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$ be the input vector.

Then

1) $A1 = W^{[1]} X_i + B^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \end{bmatrix}$

2) $Z1 = \sigma(A1) = \begin{bmatrix} \sigma(a_1) \\ \sigma(a_2) \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[2]} \end{bmatrix}$, $\sigma(x) \rightarrow$ sigmoid of x

3) $A2 = W^{[2]} Z1 + B^{[2]} = \begin{bmatrix} a_1^{[2]} \\ a_2^{[2]} \\ a_3^{[2]} \end{bmatrix}$

4) $Z2 = \sigma(A2) = \begin{bmatrix} z_1^{[2]} \\ z_2^{[2]} \\ z_3^{[2]} \end{bmatrix}$

Z2 gives us the final layer output.

Backpropagation:

Backpropagation:

$$\text{Error } E = \sum_{j=1}^{3} (x_{ij} - \hat{x}_{ij})^2 = \sum_{j=1}^{3} (x_{ij} - z_j^{[2]})^2$$

$$\text{"}dz_j^{[2]}\text{"} = \frac{dE}{dz_j^{[2]}} = -2(x_{ij} - z_j^{[2]}) \quad , \text{ we do this for } 1 \le j \le 3$$

$$\text{We define "}dz2\text{"} = \begin{bmatrix} dz_1^{[2]} \\ dz_2^{[2]} \\ dz_3^{[2]} \end{bmatrix}$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Now

$$\text{"}da_j^{[2]}\text{"} = \frac{dE}{da_j^{[2]}} = \frac{dE}{dz_j^{[2]}} \cdot \frac{dz_j^{[2]}}{da_j^{[2]}} = -2(x_{ij} - z_j^{[2]}) \cdot (z_j^{[2]}(1 - z_j^{[2]}))$$

$$\text{We define this as "}dA2\text{"} = \begin{bmatrix} da_1^{[2]} \\ da_2^{[2]} \\ da_3^{[2]} \end{bmatrix}$$

For $W^{[2]}$, $\frac{dE}{dW_{ji}^{[2]}} = \frac{dE}{da_j^{[2]}} \cdot \frac{da_j^{[2]}}{dW_{ji}^{[2]}} = \text{"}da_j^{[2]}\text{"} \times z_i^{[1]}$; as $a_j^{[2]} = W_{j1} z_1^{[1]} + W_{j2} z_2^{[1]} + B_j^{[2]}$

We can represent it in matrix form $\text{"}dW2\text{"} = dA2 . Z1^T$

For $B^{[2]}$, $\frac{dE}{dB_j^{[2]}} = \frac{dE}{da_j^{[2]}} \cdot \frac{da_j^{[2]}}{dB_j^{[2]}} = \frac{dE}{da_j^{[2]}} \cdot 1$

$\therefore$ "dB2" = "dA2"

For Z1

$$"dz_i^{[1]}" = \frac{dE}{dz_i^{[1]}} = \sum_{j=1}^{3} \frac{dE}{da_j^{[2]}} \cdot \frac{da_j^{[2]}}{dz_i^{[1]}} = \sum_{j=1}^{3} \frac{dE}{da_j^{[2]}} \cdot W_{ji}^{[2]} = \sum_{j=1}^{3} "da_j^{[2]}" \cdot W_{ji}^{[2]}$$

In Matrix form

$$"dZ1" = W^{[2]T} \cdot "dA2" \quad \text{(dot product)}$$

$$\frac{dE}{da_i^{[1]}} = \frac{dE}{dz_i^{[1]}} \cdot \frac{dz_i^{[1]}}{da_i^{[1]}} = "dz_i^{[1]}" \cdot \sigma(a_i^{[1]})(1-\sigma(a_i^{[1]})), \text{ as } z_i^{[1]} = \sigma(a_i^{[1]})$$

$$\therefore "dA1" = "dZ1" \cdot (Z1(1-Z1))$$

Same as W2, we get "dW1"

$$\frac{dE}{dW_{ji}^{[1]}} = \frac{dE}{da_j^{[1]}} \cdot \frac{da_j^{[1]}}{dW_{ji}^{[1]}} = "da_j^{[1]}" \cdot x_i$$

In Matrix form

$$"dW1" = "dA1" \cdot X_i^T$$

for B1

$$\frac{dE}{dB_i^{[1]}} = \frac{dE}{da_i^{[1]}} \cdot \frac{da_i^{[1]}}{dB_i^{[1]}} = \frac{dE}{da_i^{[1]}} \cdot 1$$

$$\therefore "dB1" = "dA1"$$

In Concise format forward propagation and backpropagation works like this.

Forward Propagation

```
A1 = np.dot(W1, Xi) + B1
Z1= sigmoid(A1)
A2 = np.dot(W2, Z1) + B2
Z2=sigmoid(A2)
```

Back Propagation:

```
E=(Xi-Z2)**2;
dZ2= -2*(Xi-Z2)
dA2=dZ2*(Z2*(1-Z2))
dW2=np.dot(dA2,Z1.T)
dB2=dA2
dZ1=np.dot(W2.T,dA2)
dA1=dZ1*(Z1*(1-Z1))
dW1=np.dot(dA1,Xi.T)
dB1=dA1
```

Weight Updates are like this:
(Based on Matrices defined above)

$$W2 = W2 - \alpha \text{"}dW2\text{"}$$; "dW2" is the matrix as defined above

$$B2 = B2 - \alpha \text{"}dB2\text{"}$$

$$W1 = W1 - \alpha \text{"}dW1\text{"}$$

$$B1 = B1 - \alpha \text{"}dB1\text{"}$$

This way all the weights are updated in one go, and get stored in matrix form only.

We then run this implementation in python from scratch, and make a loop to run multiple epochs.

```
def runepoch(X_given,parameters):
  assert (X_given.shape==(3,800))
  for i in range(800):
    cur_cache=forwardprop(parameters,X_given[:,i:i+1])
    cur_grad=backwardprop(cur_cache,parameters,X_given[:,i:i+1])
    parameters=updateweights(parameters,cur_grad,.03)
  return parameters
```

Note we keep the learning rate as 0.03

Using this function we run 100 epochs and plot MSE vs Epoch.

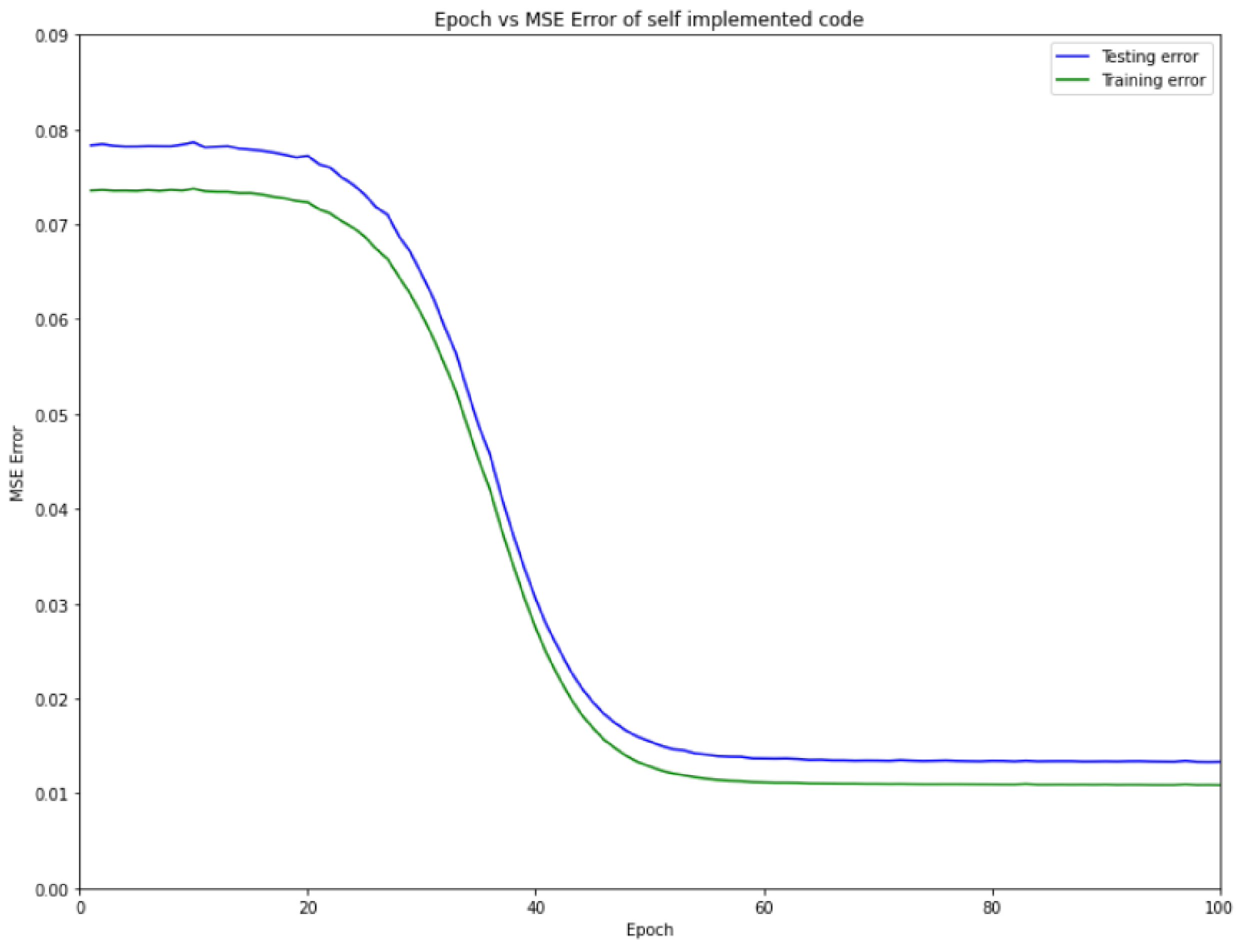
```
parameters=initialize_parameters(3, 2, 3)
testepocherror=[];
trainepocherror=[]
index=np.arange(1,101,1)
for i in range(100):
  p = np.random.permutation(800)
  parameters=runepoch(X_train[:,p],parameters)
  totalerror=0
  totaler2=0
  for i in range(200):
    curans=forwardprop(parameters,X_test[:,i:i+1])
    yhat=curans['Z2']
    Ei=(X_test[:,i:i+1]-yhat)**2;
    totalerror+=Ei
  MSE=totalerror/200
  testepocherror.append(np.sum(MSE))
  for i in range(800):
    curans=forwardprop(parameters,X_train[:,i:i+1])
    yhat=curans['Z2']
    Ei=(X_train[:,i:i+1]-yhat)**2;
    totaler2+=Ei
  MSE=totaler2/800
  trainepocherror.append(np.sum(MSE))
```

I get the following final MSE errors:

```
trainepocherror[-1],testepocherror[-1]
```

```
(0.010873232615716955, 0.013320378386287095)
```

3. Final Graph



4. We implement the same neural network but this time using the autograd functionality of pytorch to calculate the gradients.

Forward and back Prop

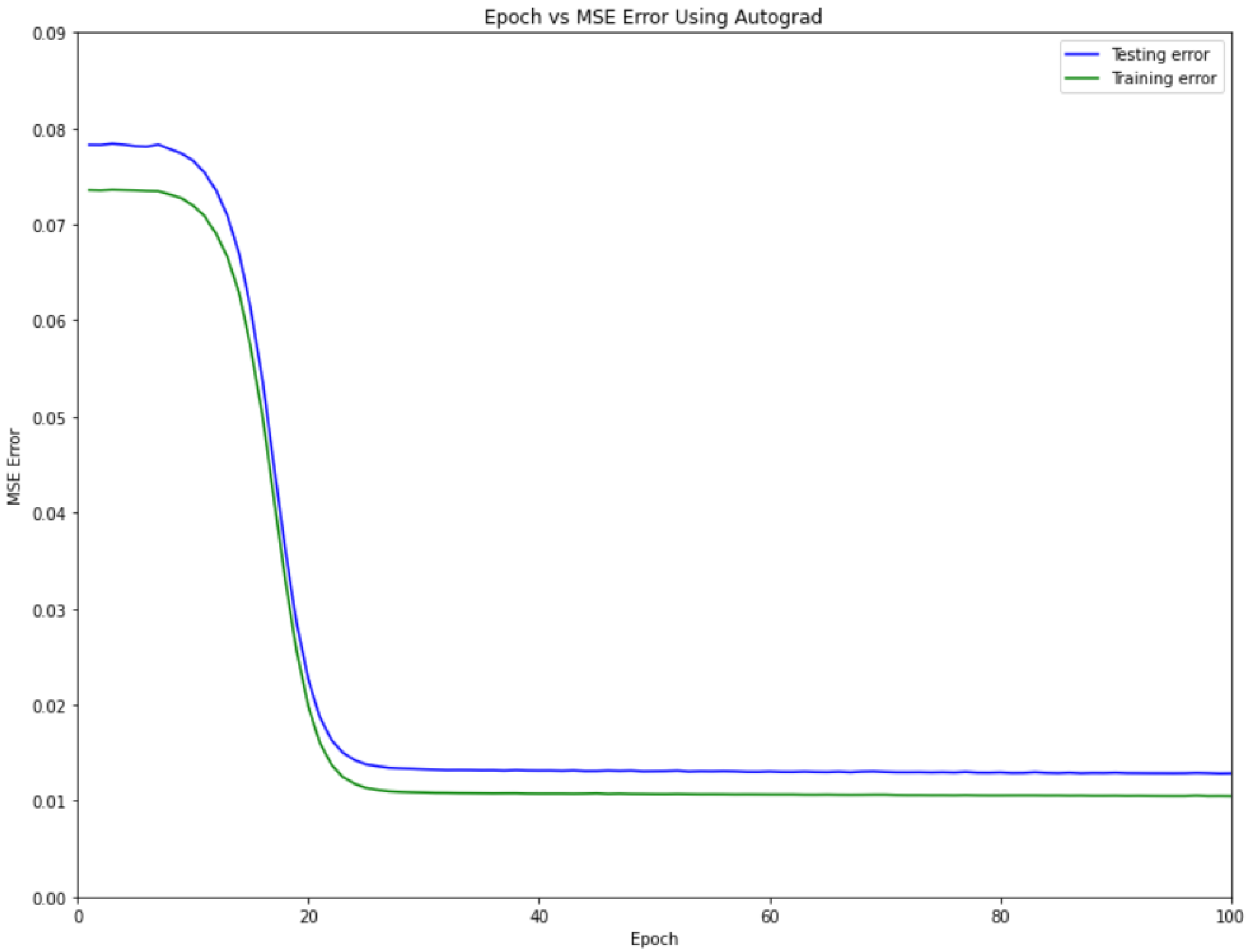
```
B2=parameters["B2"]
# print(W1.shape,W1.dtype)
A1 = torch.matmul (W1,Xi) + B1
Z1= torch.sigmoid(A1)
A2 = torch.matmul (W2,Z1)+B2
Z2=torch.sigmoid(A2)
E=(Xi-Z2)**2;
# print(E)
E.retain_grad()
Z2.retain_grad()
A2.retain_grad()
Z1.retain_grad()
A1.retain_grad()
W2.retain_grad()
W1.retain_grad()
B2.retain_grad()
B1.retain_grad()

E.sum().backward(retain_graph=True)
```

Weight Update

```
with torch.no_grad():
      # print("GRADIENT ",W2,W2.grad)
      W2-=learningrate*W2.grad
      W1-=learningrate*W1.grad
      B2-=learningrate*B2.grad
      B1-=learningrate*B1.grad
W2.grad.zero_()
W1.grad.zero_()
B1.grad.zero_()
B2.grad.zero_()
```

Graph with autograd




```
trainepocherror[-1],testepocherror[-1]
```

```
(tensor(0.0105, dtype=torch.float64, grad_fn=<SumBackward0>),
 tensor(0.0129, dtype=torch.float64, grad_fn=<SumBackward0>))
```

Inference :

We notice that we get the exact shape curve of the graph and even the final values of the MSE error are pretty much the same. (Slight difference being in that torch uses float64 dtype so handles higher precision much better)