Deep Learning
(Fall 2024)
CS898BD

Vamshi Nallaguntla
Y852D635

Assignment-1

1. Introduction:

Assignment code Link:
https://colab.research.google.com/drive/11PulmLwzm9ynnYzvfPfH0mOIG-i4Jkev?usp=sharing

How does backpropagation work:

In machine learning, backpropagation—short for "backward propagation of errors"—is a method that is employed, mostly for artificial neural network training. Using the chain rule, it quickly determines the gradient of the loss function with respect to each weight. This procedure enables the updating of weights and minimization of loss through optimization methods such as gradient descent.

Steps in Backpropagation:

- **Forward Pass:**
  The inputs are passed layer by layer in the network. The output of one layer will be the input of the next layer until you reach the output layer, which gives the final output. The weighted sum of inputs is calculated at each layer. This is used to calculate the loss

- **Backward Pass**
  Backpropagation happens at the backward pass. In the Network using the chain rule, we calculate the partial derivatives with respect to the weights. The Backward pass is used to compute gradients. Here, the error is propagated backward to the input layer from the output layer.

Deep Neural Networks (DNN):

Fully Connected Neural Network (FCNN): also called Dense Neural Network. In this type of neural network, each neuron in one layer is connected to each neuron in the next layer.

Layers in FCNN:

- **Input Layer:** The raw data is received by the input layer. In MNIST dataset, a 28x28(=784 pixels) image. So, 784 neurons will be present in the input layer of this DNN

- **Hidden Layer:** Each hidden layer consists of neurons that are connected to both the previous layer and the next layer in the case of FCNN. The number of hidden layers determines the complexity of the data. The neurons in the hidden layer receive weighted inputs and apply the activation function to produce the output.

- **Output Layer:** The output layer gives the final output. Here, probabilities are provided for each class, in this case the digits in the range 0-9. So, the output consists of 10 neurons. The highest probability class among these classes will be given as the final prediction of the digit.

Activation Function:

In a neural network, an activation function evaluates an input and decides whether to activate a neuron. It gives the network non-linearity. In the absence of an activation function, a neural network would function as a basic linear model, which would restrict its capacity to address complex problems such as image classification, and natural language processing. For Binary classification we use sigmoid, for multiclass Classification, we use SoftMax as the activation function.

SoftMax: When a task requires selecting one class from a variety of options, such as classifying images in the MNIST dataset with ten potential digits, SoftMax is utilized.

For a raw scores vector $z = [z_1, z_2, \ldots\ldots, z_n]$

the SoftMax function for the i-th score is

$$SoftMax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

Where, n is the number of classes,

z_i - is the raw score for the i-th class

Loss Function:

A loss function, commonly referred to as a cost function, is an essential part of machine learning, particularly when neural networks are being trained. It calculates the difference

between the model's expected output and the actual output. Minimizing this loss is the aim during training; it shows that the model is getting closer to the right answers with its predictions.

Categorical Cross-Entropy Loss: for multiclass classification,

Categorical Cross-Entropy = $-\sum_{i=1}^{n}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c})$

Where C is the number of classes,

$y_{i,c}$ is a binary indicator

$\hat{y}_{i,c}$ is the predicted probability

MNIST Dataset:

- The MNIST (Modified National Institute of Standards and Technology) dataset consists of 70,000 grayscale images of handwritten digits. Out of which 60,000 examples are for training set, and the remaining 10,000 examples are for test set.
- Each image resolution is 28x28 pixels. So, each image contains 784 pixels. The range of these pixels is from 0(black) to 255(White).
- The labels are from '0' to '9' corresponding to each image.

Goal:

To Design and evaluate the Deep Neural Network (DNN) for the two models with the given configuration and compare them.

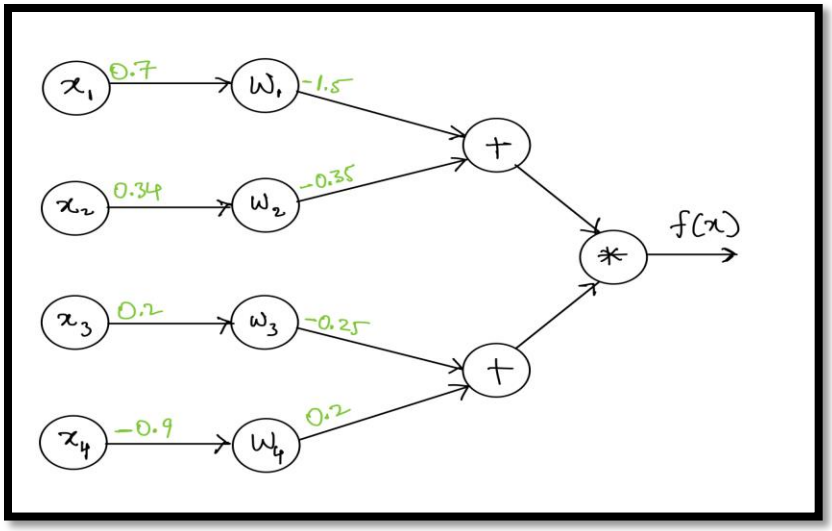
2. Methodology:

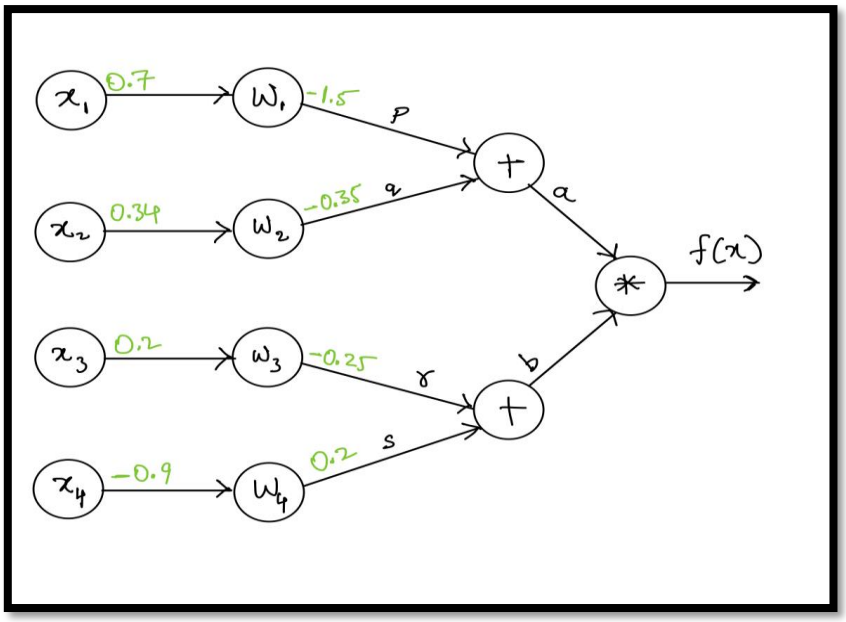
Question-1:

Computational graph: $f(x) = (x_1w_1 + x_2w_2) * (x_3w_3 + x_4w_4)$

Given x_1= 0.7, w_1 = −1.5, x_2 = 0.34, w_2 = −0.35, x_3 = 0.2, w_3 = −0.25, x_4 = −0.9, w_4 = 0.2

- Forward pass:





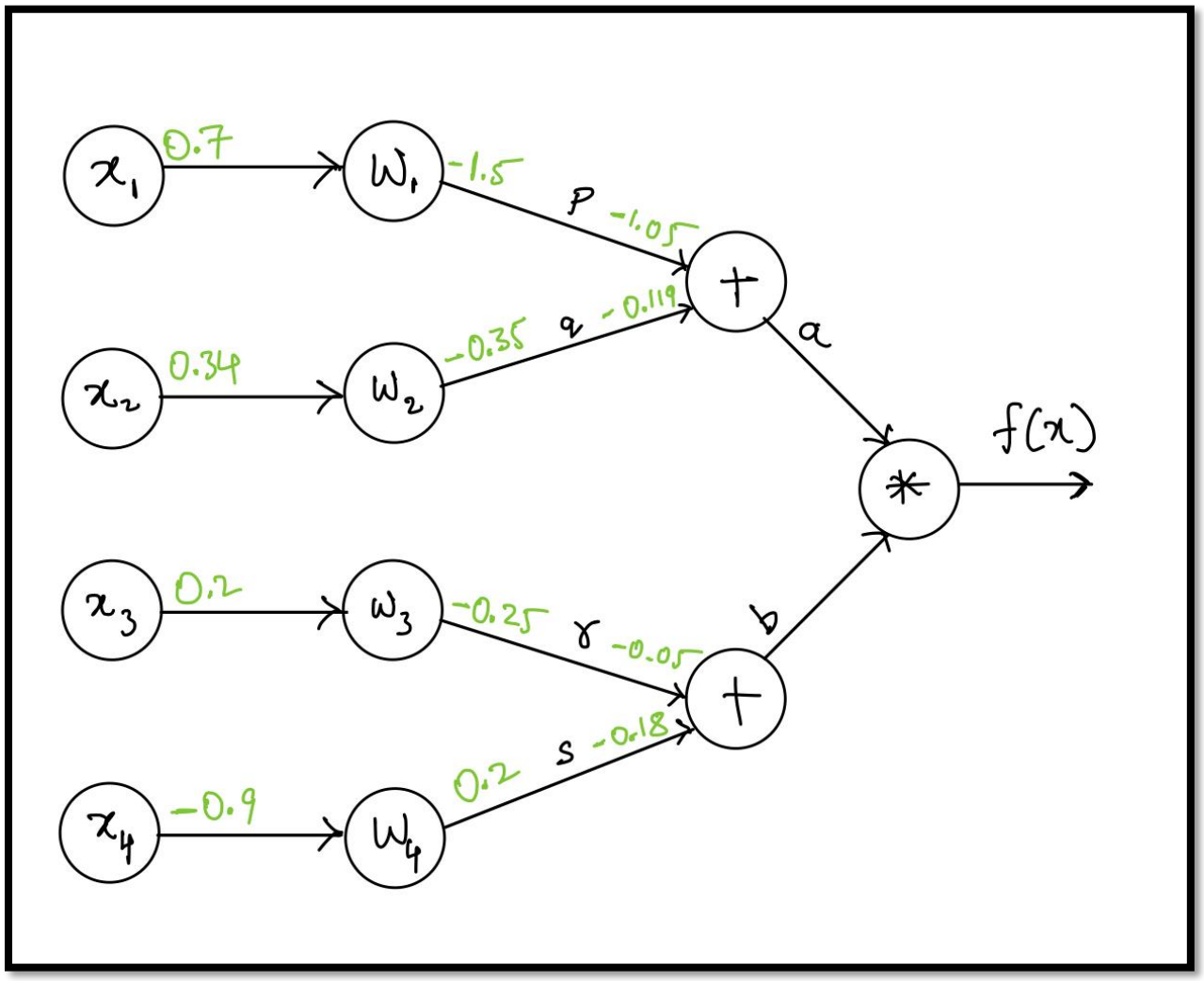
Let ‘p’, ‘q’, ‘r’ and ‘s’ be the products, then,

p= $x_1 * w_1$ = 0.7∗ (-1.5) = -1.05

q= $x_2 * w_2$ = 0.34 ∗ (-0.35) = -0.119

r= $x_3 * w_3$ = 0.2 ∗ (-0.25) = -0.05

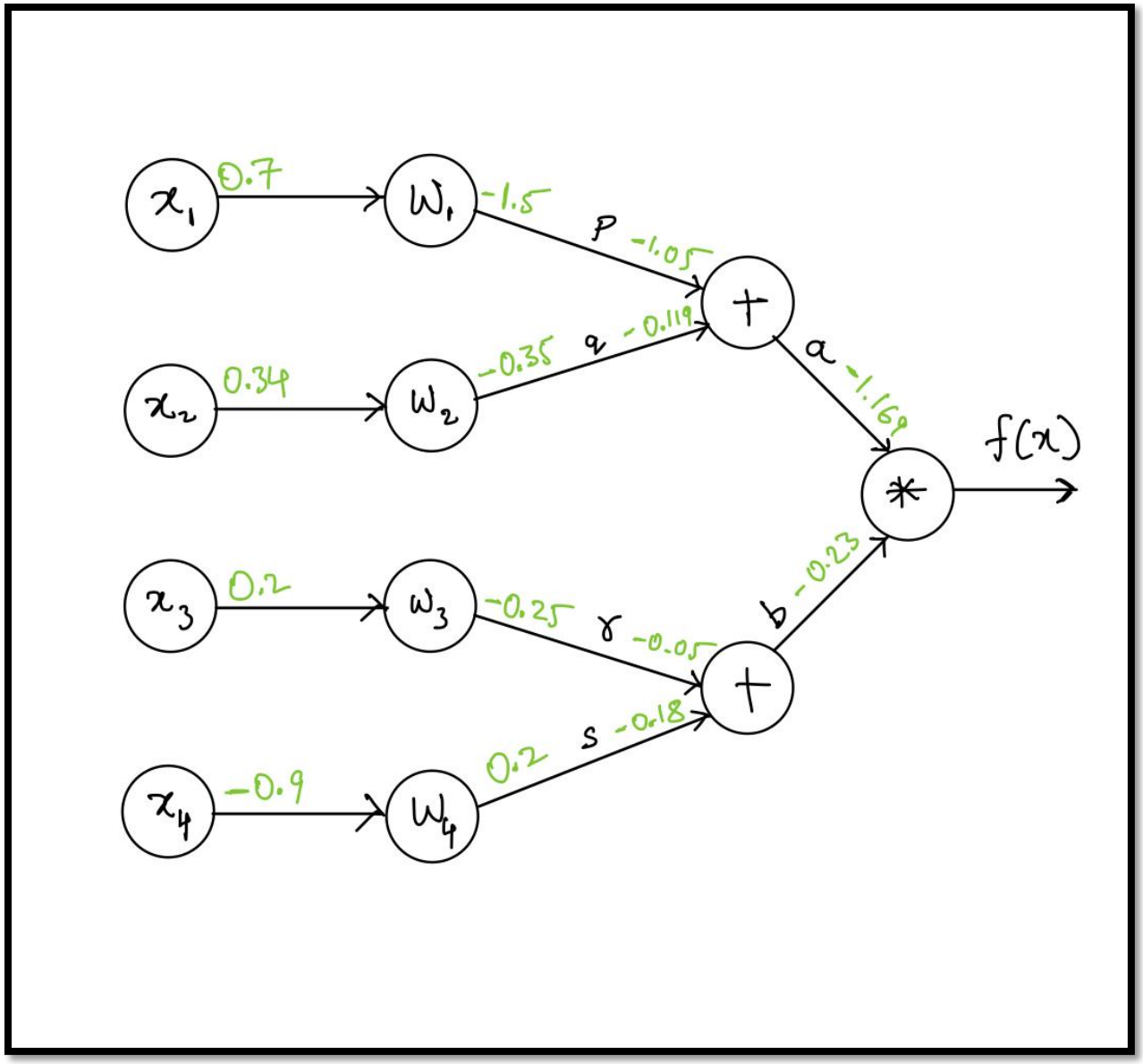
s= $x_4 * w_4$ = (-0.9) ∗ (0.2) = -0.18



Let sums be ‘a’ and ‘b’, then

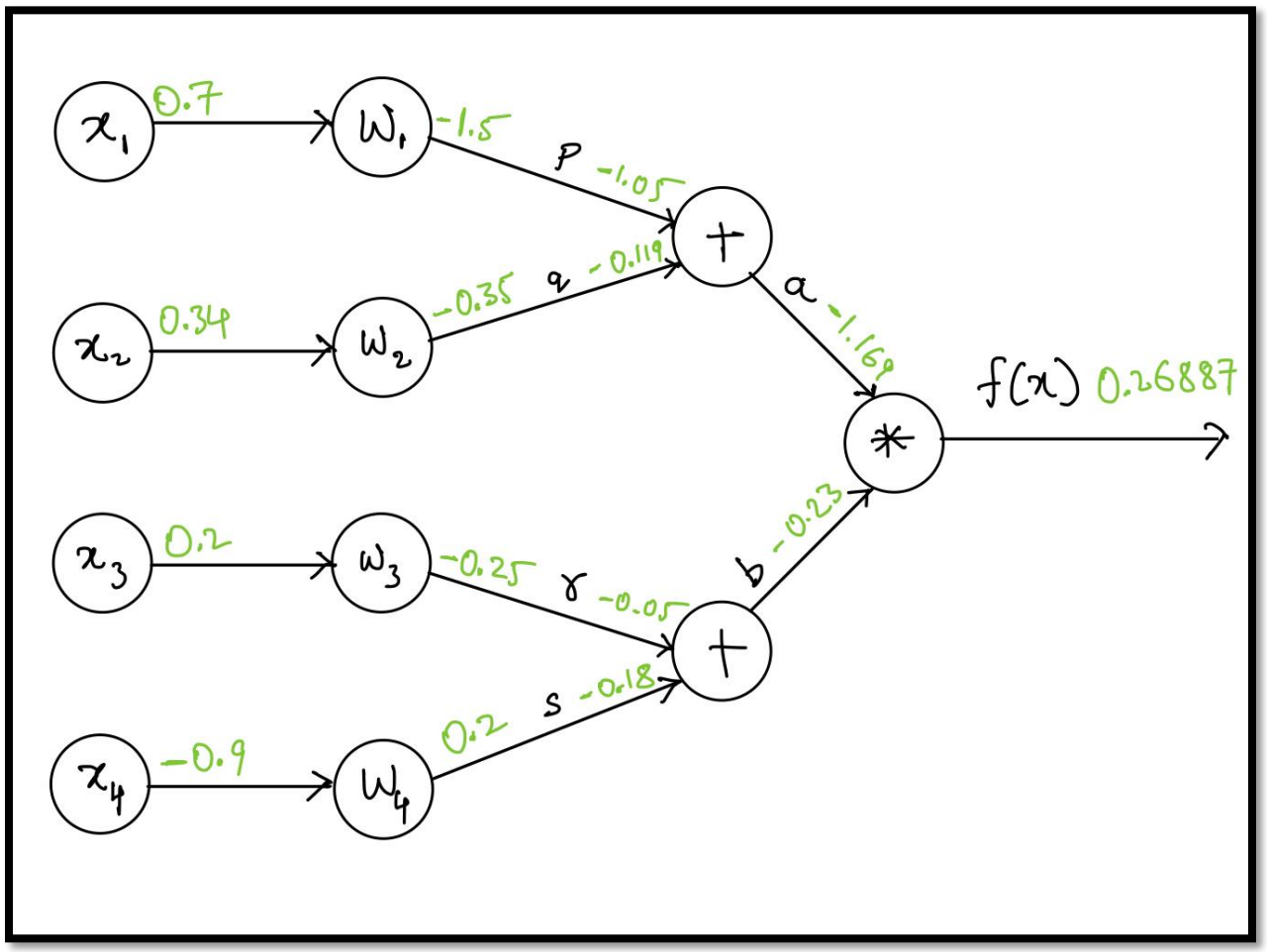
a=p + q = (-1.05) + (-0.119) = -1.169

b=r + s = (-0.05) + (-0.18) = -0.23


x_1
0.7
w_1
-1.5
p -1.05
x_2
0.34
w_2
-0.35
q -0.119
+
a -1.169
*
f(x)
x_3
0.2
w_3
-0.25
r -0.05
b -0.23
+
x_4
-0.9
w_4
0.2
s -0.18

Finally,

$f(x)$ = a $*$ b = (-1.169) $*$ (-0.23) = 0.26887



- <u>Backward pass</u>:

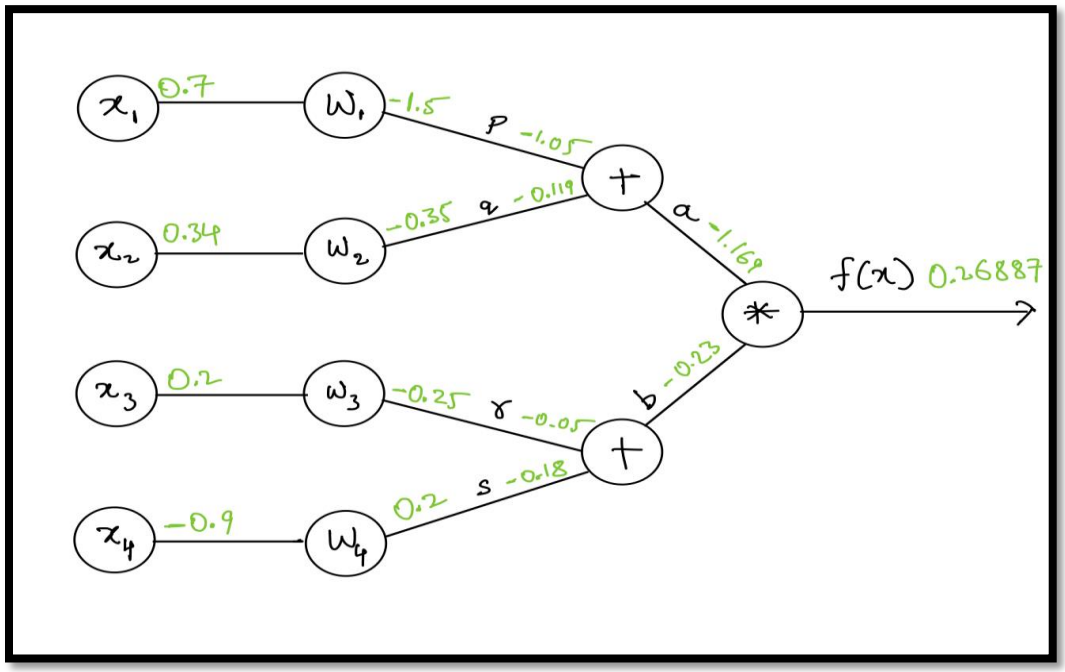
$f(x) = (x_1w_1 + x_2w_2) * (x_3w_3 + x_4w_4)$

a= $(x_1w_1 + x_2w_2)$

b= $(x_3w_3 + x_4w_4)$

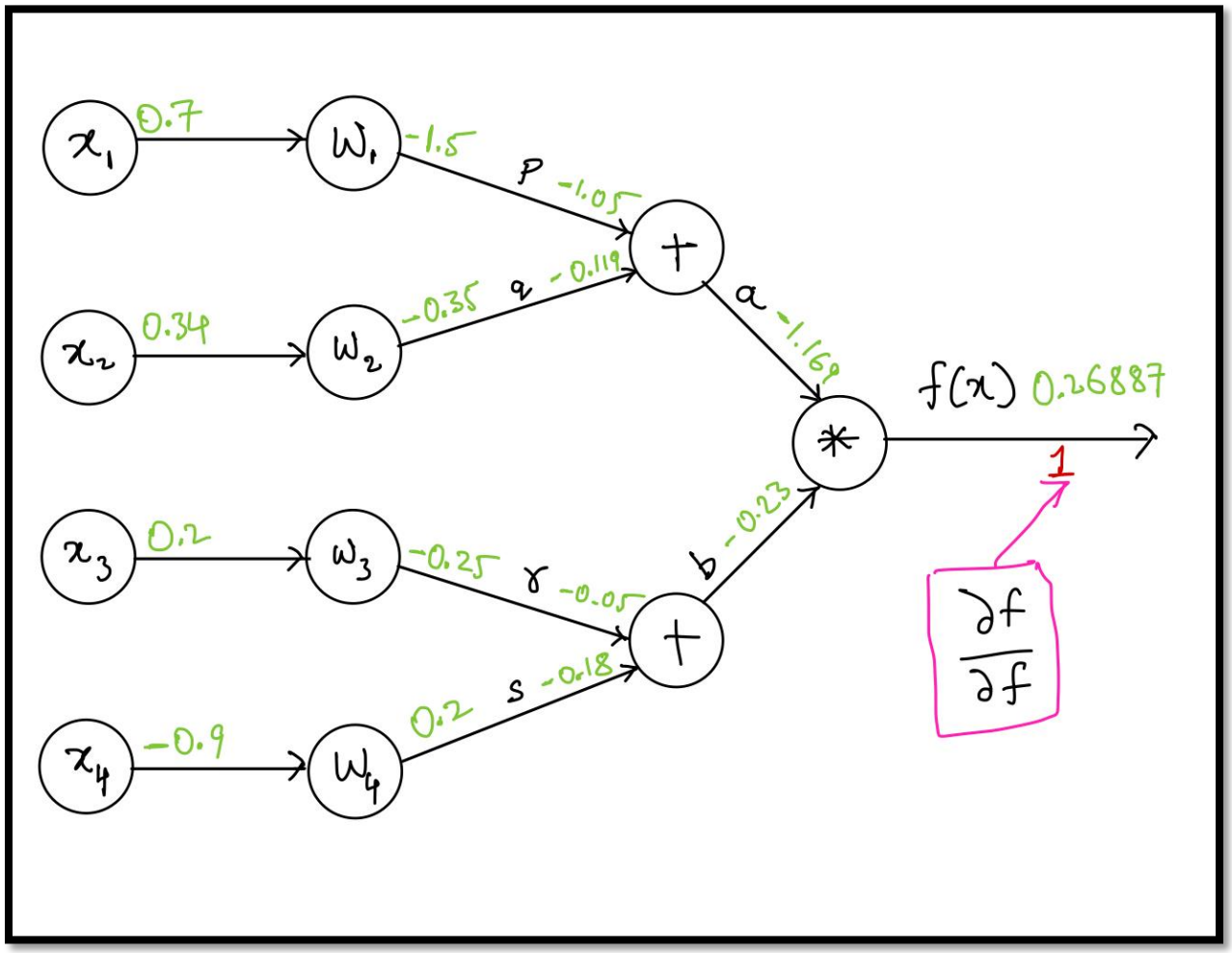
We have to find

$$\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial x_4}, \frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \frac{\partial f}{\partial w_3}, \frac{\partial f}{\partial w_4}$$



We know that,

$\frac{\partial f}{\partial f} = 1$



$f = a.b$

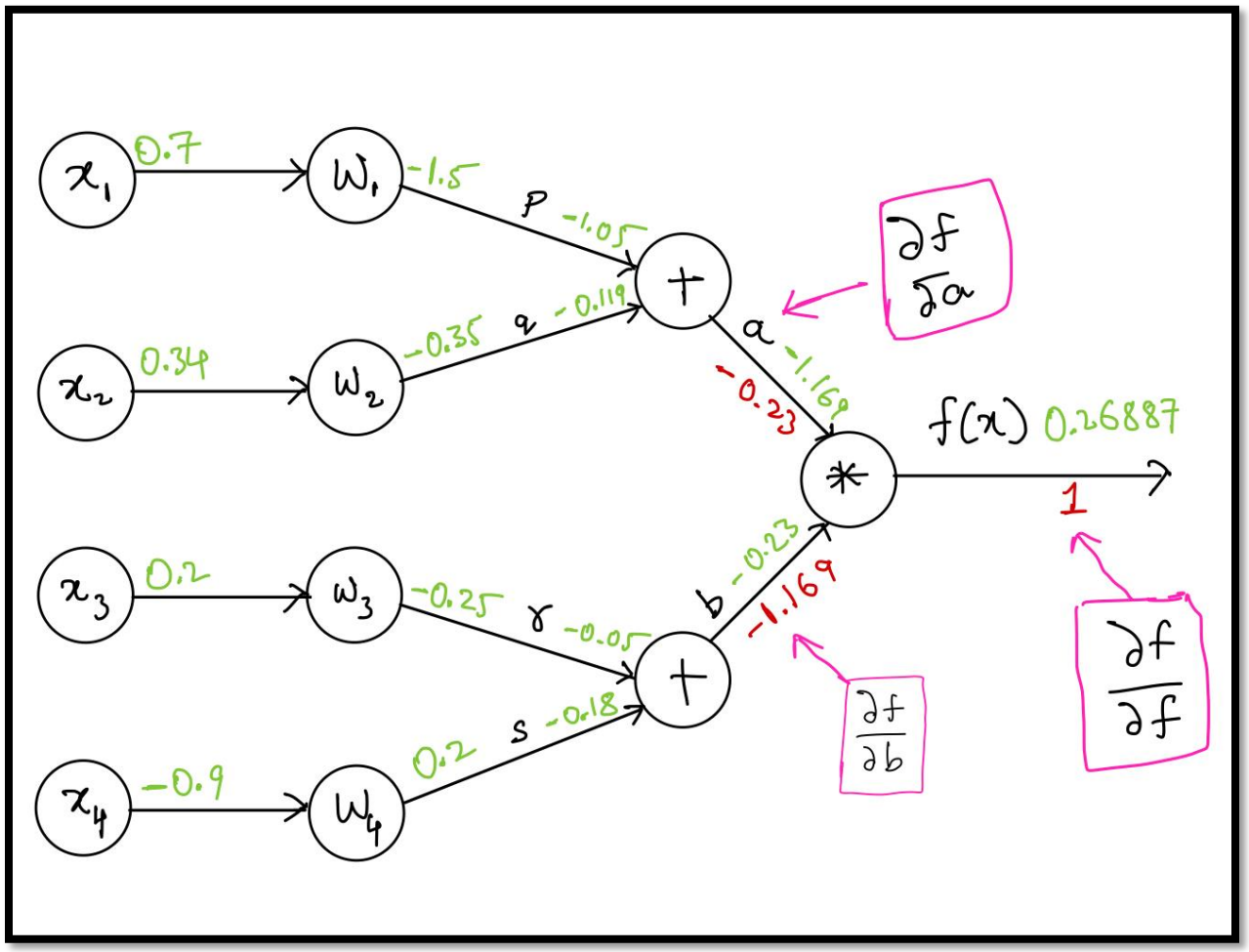
So,

$\frac{\partial f}{\partial a} = b$

$$\frac{\partial f}{\partial a} = -0.23$$

$\frac{\partial f}{\partial b} = a$

$$\frac{\partial f}{\partial b} = -1.169$$



a= p+q

So,

$$\frac{\partial a}{\partial p} = 1$$

$$\frac{\partial a}{\partial q} = 1$$

$$\frac{\partial f}{\partial p} = \frac{\partial f}{\partial a} \cdot \frac{\partial a}{\partial p}$$

So, from previous equations,

$$\frac{\partial f}{\partial p} = (-0.23) \cdot 1$$

$$\frac{\partial f}{\partial p} = \ -0.23$$

$$\frac{\partial f}{\partial q} = \frac{\partial f}{\partial a} \cdot \frac{\partial a}{\partial q}$$

From previous equations,

$$\frac{\partial f}{\partial q} = (-0.23).1$$

$$\frac{\partial f}{\partial q} = \ -0.23$$

b = r + s

$$\frac{\partial b}{\partial r} = \ 1$$

$$\frac{\partial b}{\partial s} = \ 1$$

$$\frac{\partial f}{\partial r} = \frac{\partial f}{\partial b} \cdot \frac{\partial b}{\partial r}$$

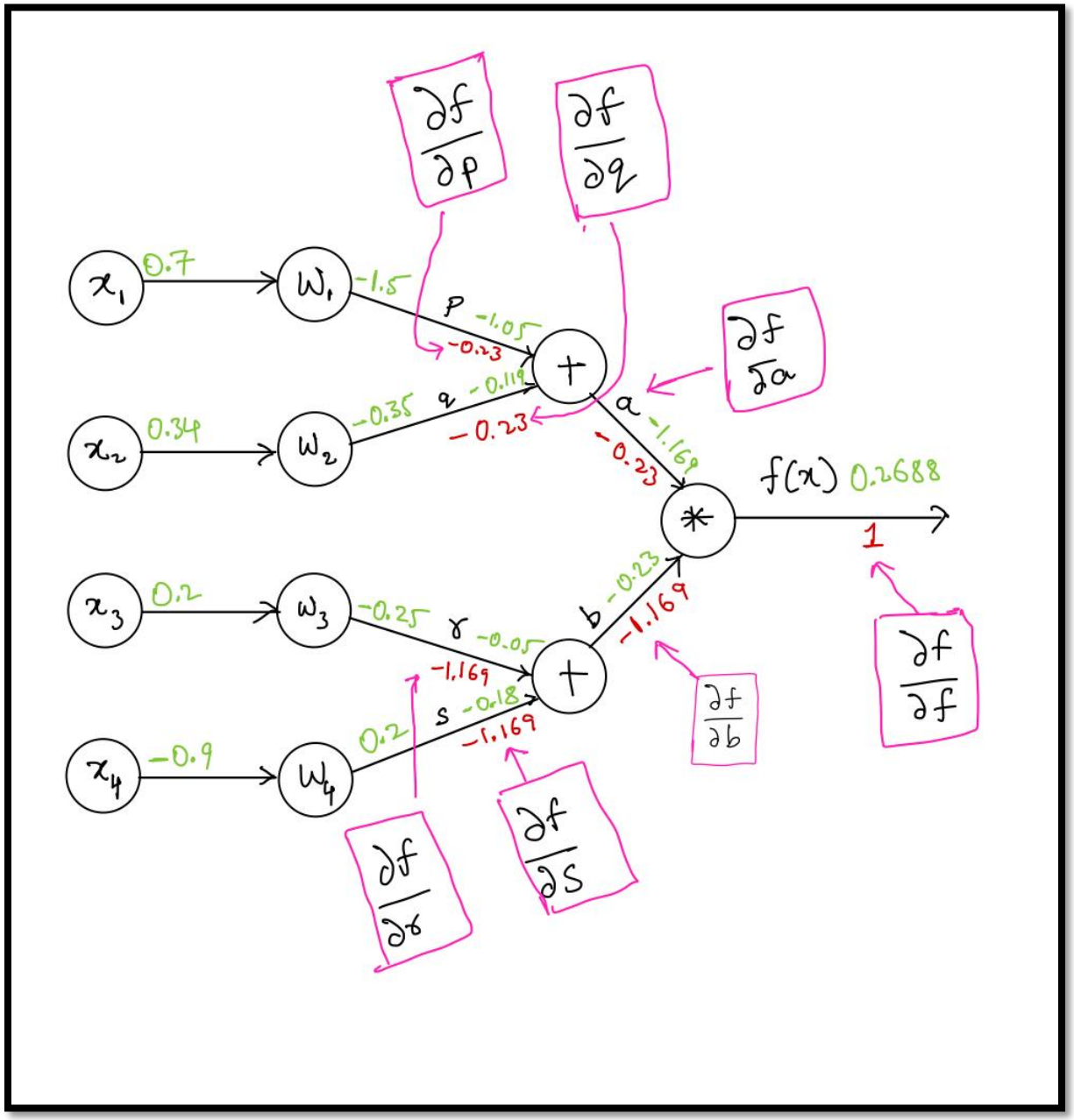
$\frac{\partial f}{\partial r} = (-1.169).1$

$$\frac{\partial f}{\partial r} = \ -1.169$$

$$\frac{\partial f}{\partial s} = \ \frac{\partial f}{\partial b} \cdot \frac{\partial b}{\partial s}$$

$\frac{\partial f}{\partial s} = (-1.169).1$

$$\frac{\partial f}{\partial s} = \ -1.169$$



p = $x_1 w_1$

$\frac{\partial p}{\partial x_1} = w_1$

$$\frac{\partial p}{\partial x_1} = -1.5$$

$$\frac{\partial p}{\partial w_1} = x_1$$

$$\frac{\partial p}{\partial w_1} = 0.7$$

$$\frac{\partial f}{\partial x_1} = \frac{\partial f}{\partial a} \cdot \frac{\partial a}{\partial p} \cdot \frac{\partial p}{\partial x_1}$$

$$\frac{\partial f}{\partial x_1} = (-0.23)(1)(-1.5)$$

$$\boxed{\frac{\partial f}{\partial x_1} = 0.345}$$

$$\frac{\partial f}{\partial w_1} = \frac{\partial f}{\partial a} \cdot \frac{\partial a}{\partial p} \cdot \frac{\partial p}{\partial w_1}$$

$$\frac{\partial f}{\partial w_1} = (-0.23)(1)(0.7)$$

$$\boxed{\frac{\partial f}{\partial w_1} = -0.161}$$

q = $x_2 w_2$

$$\frac{\partial q}{\partial x_2} = w_2 = -0.35$$

$$\frac{\partial q}{\partial w_2} = x_2 = 0.34$$

$$\frac{\partial f}{\partial x_2} = \frac{\partial f}{\partial a} \cdot \frac{\partial a}{\partial q} \cdot \frac{\partial q}{\partial x_2}$$

$$\frac{\partial f}{\partial x_2} = (-0.23).(1).(-0.35)$$

$$\boxed{\frac{\partial f}{\partial x_2} = 0.0805}$$

$$\frac{\partial f}{\partial w_2} = \frac{\partial f}{\partial a} \cdot \frac{\partial a}{\partial q} \cdot \frac{\partial q}{\partial w_2}$$

$$\frac{\partial f}{\partial w_2} = (-0.23).(1).(0.34)$$

$$\boxed{\frac{\partial f}{\partial w_2} = -0.0782}$$

r = $x_3 w_3$

$$\frac{\partial r}{\partial x_3} = w_3 = -0.25$$

$$\frac{\partial r}{\partial w_3} = x_3 = 0.2$$

$$\frac{\partial f}{\partial x_3} = \frac{\partial f}{\partial b} \cdot \frac{\partial b}{\partial r} \cdot \frac{\partial r}{\partial x_3}$$

$$\frac{\partial f}{\partial x_3} = (-1.169)(1)(-0.25)$$

$$\boxed{\frac{\partial f}{\partial x_3} = 0.29225}$$

$$\frac{\partial f}{\partial w_3} = \frac{\partial f}{\partial b} \cdot \frac{\partial b}{\partial r} \cdot \frac{\partial r}{\partial w_3}$$

$$\frac{\partial f}{\partial w_3} = (-1.169).(1).(0.2)$$

$$\frac{\partial f}{\partial w_3} = -0.2338$$

s = $x_4 w_4$

$$\frac{\partial s}{\partial x_4} = w_4 = 0.2$$

$$\frac{\partial s}{\partial w_4} = x_4 = -0.9$$

$$\frac{\partial f}{\partial x_4} = \frac{\partial f}{\partial b} \cdot \frac{\partial b}{\partial s} \cdot \frac{\partial s}{\partial x_4}$$

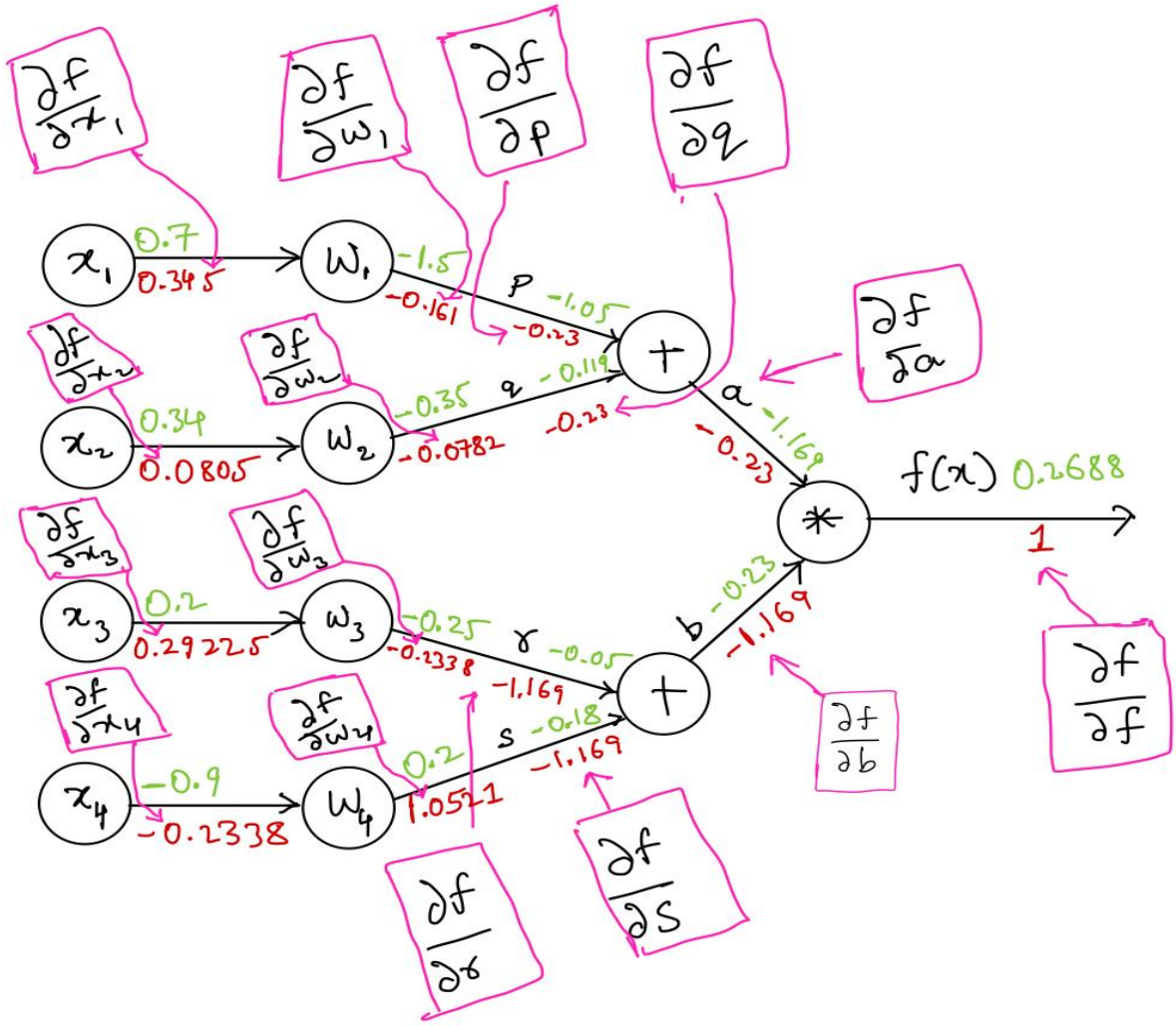
$$\frac{\partial f}{\partial x_4} = (-1.169).(1).(0.2)$$

$$\frac{\partial f}{\partial x_4} = -0.2338$$

$$\frac{\partial f}{\partial w_4} = \frac{\partial f}{\partial b} \cdot \frac{\partial b}{\partial s} \cdot \frac{\partial s}{\partial w_4}$$

$$\frac{\partial f}{\partial w_4} = (-1.169).(1).(-0.9)$$

$$\frac{\partial f}{\partial w_4} = 1.0521$$



Final Computational Graph

Question – 2:

Dataset used: We used MNIST dataset to design the two models Model_a and Model_b .

Input Image: 28 x 28 pixels (Gray scale images)

Number of Training Samples: 60,000

Number of Testing samples: 10,000

Number of Classes: 10 (handwritten digits 0-9)

3. Deep Learning Architecture:

Model 1 Architecture:

- Input Layer: 784 nodes

- Hidden layers:
  First hidden layer: 64 nodes
  Second hidden layer: 128 nodes
  Third hidden layer: 64 nodes
  Activation Function: ReLU

- Output Layer: 10 nodes (one for each class)
  Activation Function: SoftMax for the output layer.

Model 2 Architecture:

- Input Layer: 784 nodes

- Hidden Layers:
  First hidden layer: 256 nodes
  Second hidden layer: 512 nodes
  Third hidden layer: 128 nodes
  Activation Function: ReLU

- Output Layer: 10 nodes
  Activation Function: SoftMax for the output layer.

Other Details:

• Loss Function: Categorical Cross-Entropy

• Optimizer: Choose an optimizer - Adam

• Number of Epochs: 500

• Batch Size: 64

4. Experiment and Results:

- **Accuracy**: The Model with higher accuracy performs better. So, Model_b performs better as it has higher testing accuracy.

```
[10] # To find Training accuracy and Testing accuracy
     print("Training Accuracy - Model_a:", max(history_a.history['accuracy'])) #
     print("Training Accuracy - Model_b:", max(history_b.history['accuracy'])) #

     print("Testing Accuracy - Model_a:", max(history_a.history['val_accuracy']))
     print("Testing Accuracy - Model_b:", max(history_b.history['val_accuracy']))
```

```
Training Accuracy - Model_a: 0.9999499917030334
Training Accuracy - Model_b: 1.0
Testing Accuracy - Model_a: 0.9821000099182129
Testing Accuracy - Model_b: 0.9848999977111816
```

- **Loss**: Lower loss model performs better. So, Model_b is better in this aspect as it has lower loss

```
[11] # To find Training loss and Testing loss
     print("Training Loss - Model_a:", min(history_a.history['loss'])) #
     print("Training Loss - Model_b:", min(history_b.history['loss'])) #

     print("Testing Loss - Model_a:", min(history_a.history['val_loss']))
     print("Testing Loss - Model_b:", min(history_b.history['val_loss']))
```

```
Training Loss - Model_a: 0.00012653382145799696
Training Loss - Model_b: 0.0
Testing Loss - Model_a: 0.08327233791351318
Testing Loss - Model_b: 0.06417370587587357
```

- **Computational efficiency**: The model with a smaller difference between training accuracy and testing accuracy has less overfitting, in this case, Model_b is better

```
[12] #to find the difference between best training accuracy and best validation(test) accuracy
     #to find which model generalizes better
     gap_a = max(history_a.history['accuracy']) - max(history_a.history['val_accuracy']) # for model_a
     gap_b = max(history_b.history['accuracy']) - max(history_b.history['val_accuracy']) # for model_b

     if gap_a < gap_b:
         print("Model_a generalizes better")
     else:
         print("Model_b generalizes better")
```

```
Model_b generalizes better
```

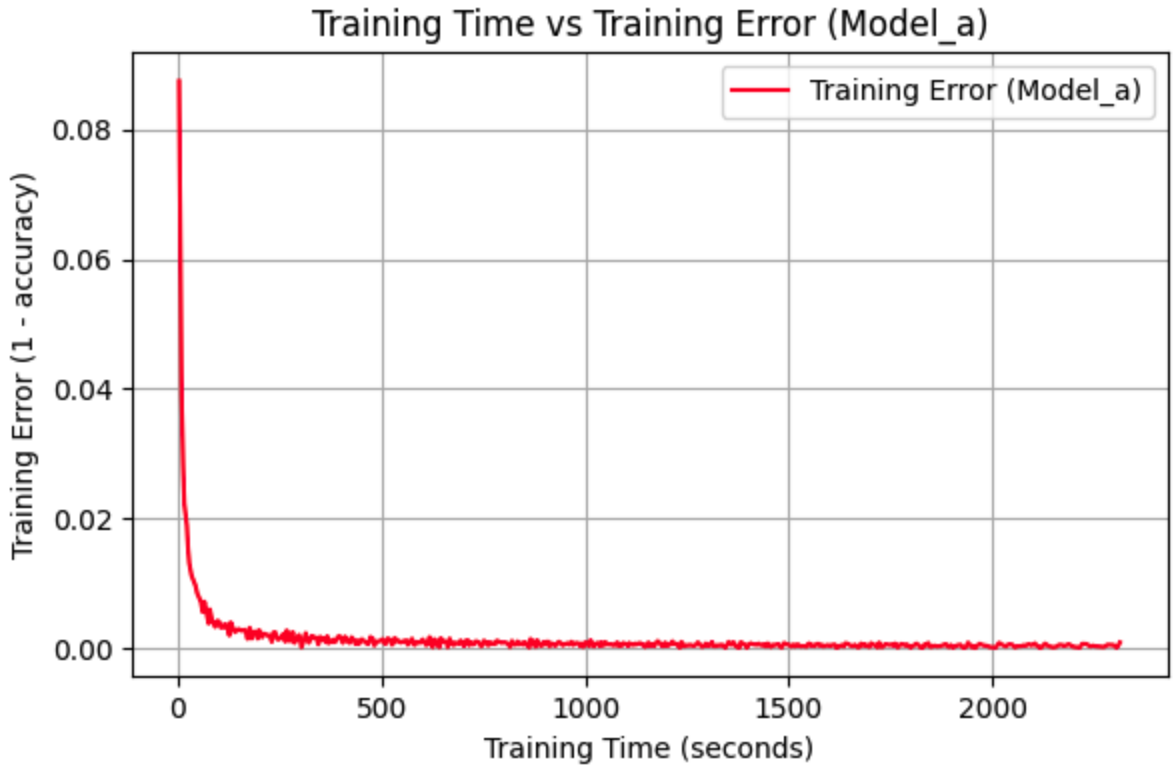
- **Training Time**: For similar testing accuracy, the model with less training time is preferred. So, in this case, Model_a is better

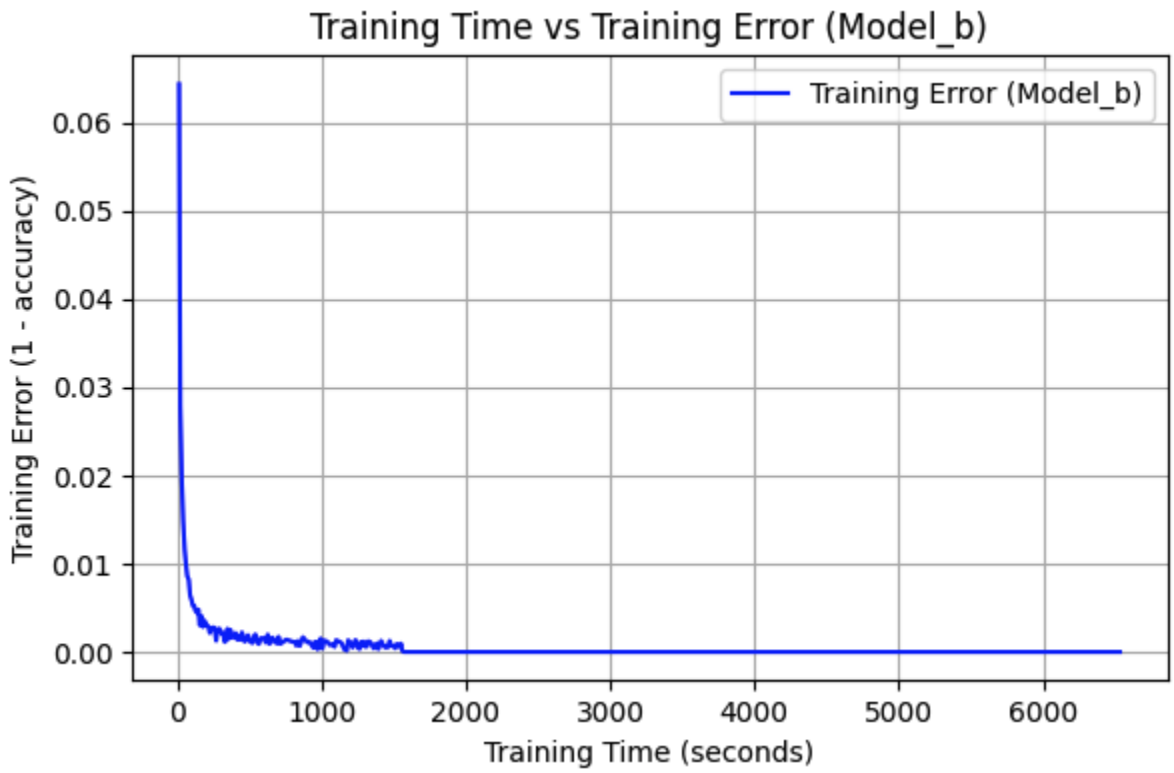
```
#to find the total running time of the models
train_time_a = sum(time_error_callback_a.times) # for model_a
train_time_b = sum(time_error_callback_b.times) # for model_b

print(f"Model_a total training time: {train_time_a}")
print(f"Model_b total training time: {train_time_b}")
```

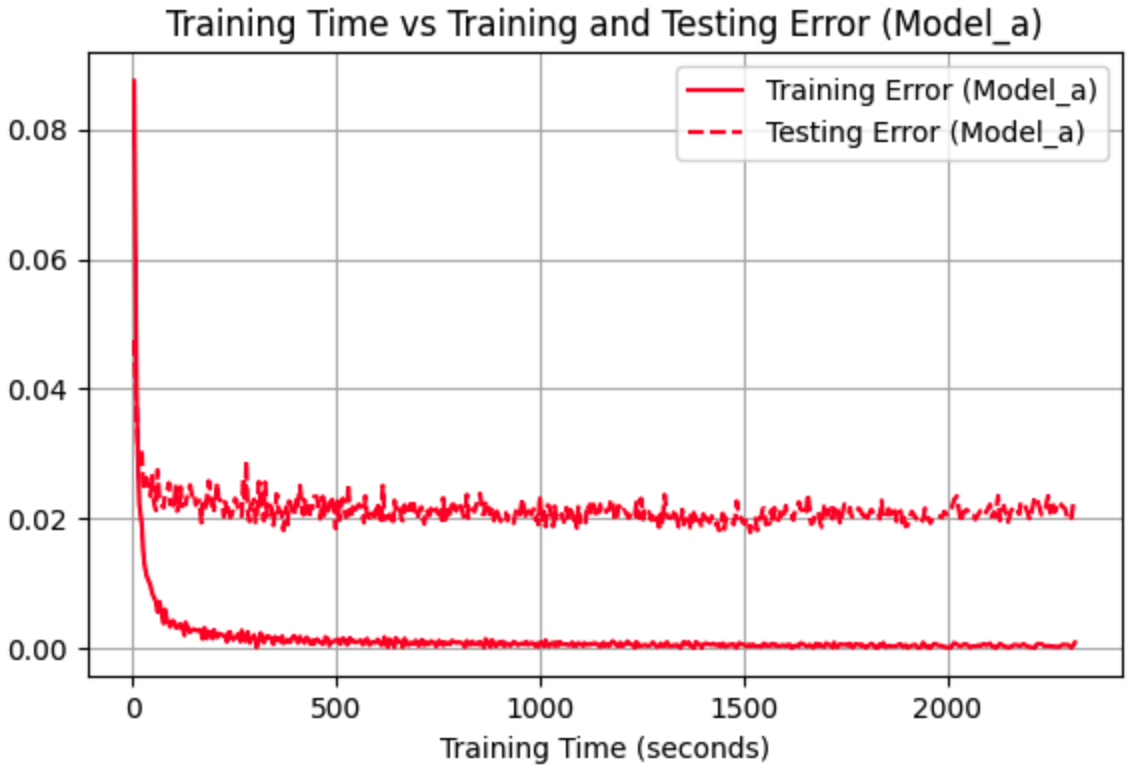
```
Model_a total training time: 2311.5374891757965
Model_b total training time: 6533.145178794861
```

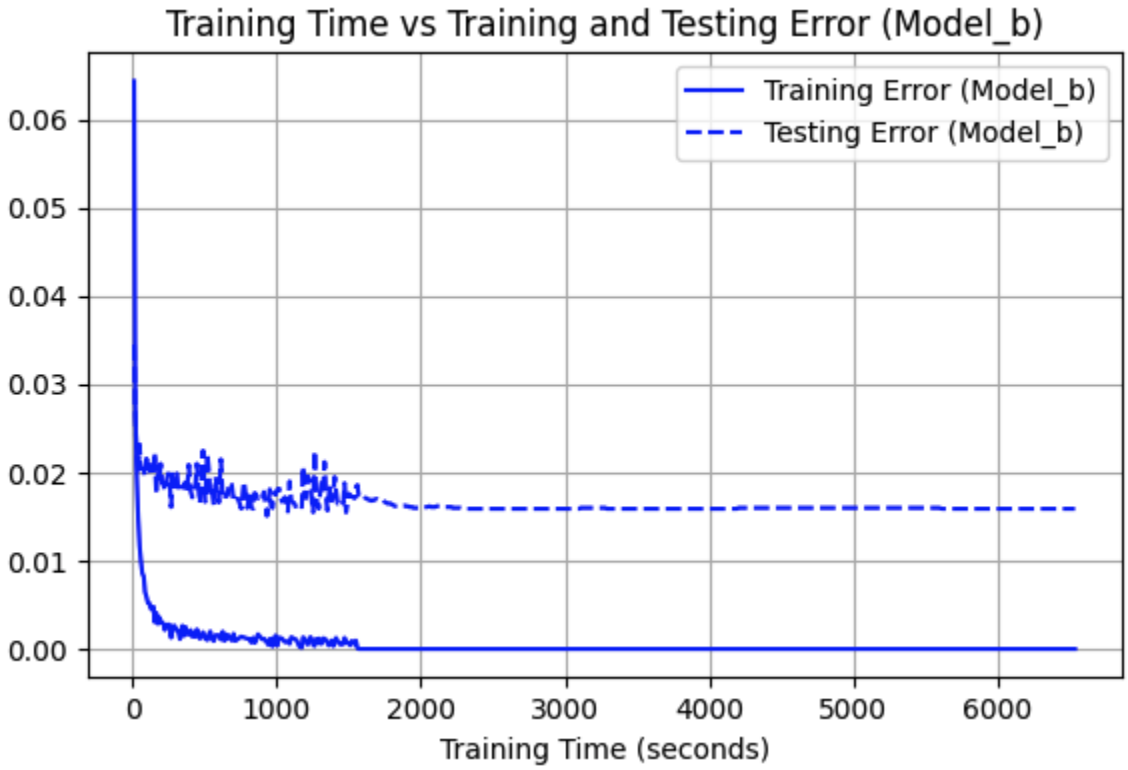
A graph that represents training error (y-axis) and training time (x-axis):

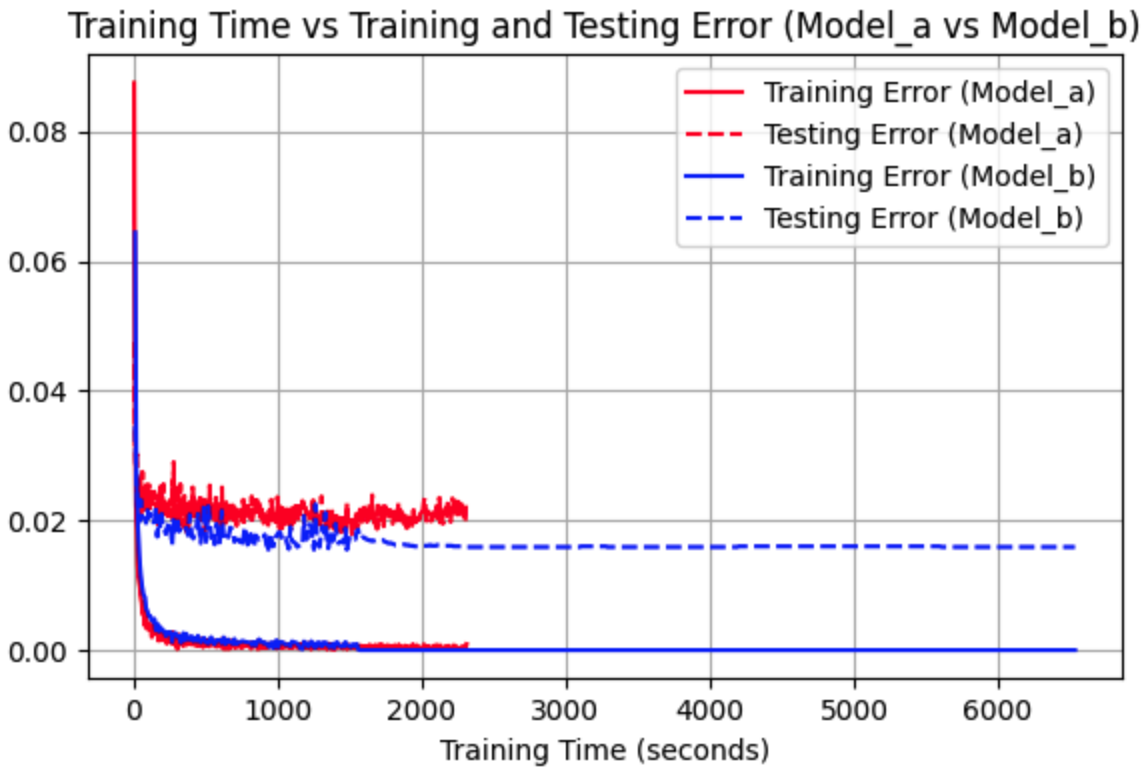

Training Time vs Training Error (Model_a)
Training Error (Model_a)
Training Error (1 - accuracy)
0.08
0.06
0.04
0.02
0.00
0
500
1000
1500
2000
Training Time (seconds)


Training Time vs Training Error (Model_b)
Training Error (Model_b)
Training Error (1 - accuracy)
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0
1000
2000
3000
4000
5000
6000
Training Time (seconds)

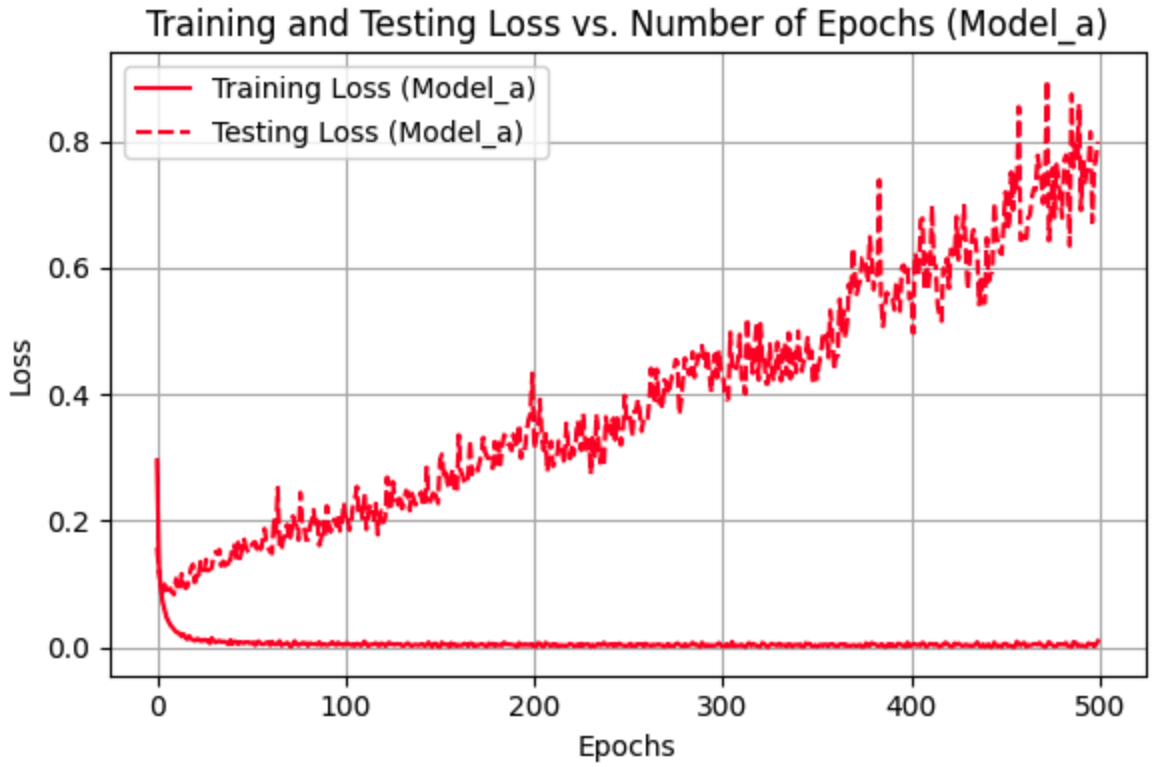
A graph that represents error (i.e., training error and testing error on y-axis) and training time (x-axis):

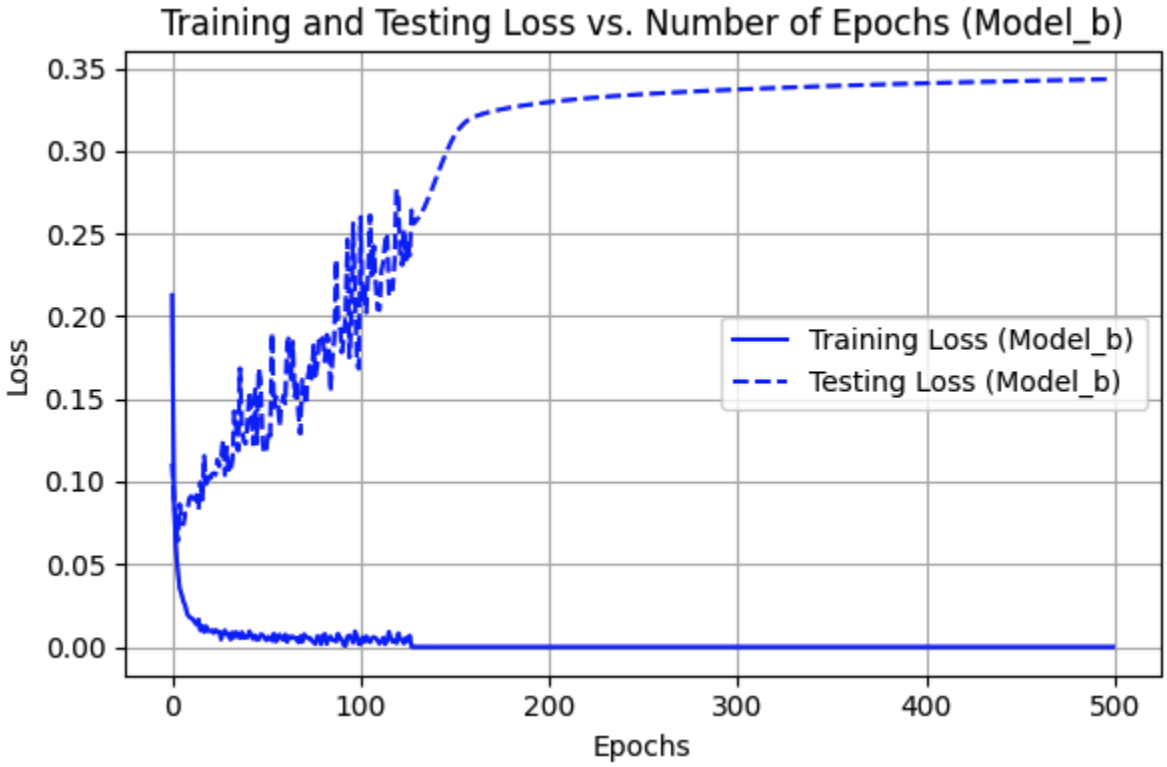

Training Time vs Training and Testing Error (Model_a)
Training Error (Model_a)
Testing Error (Model_a)
0.08
0.06
0.04
0.02
0.00
0
500
1000
1500
2000
Training Time (seconds)

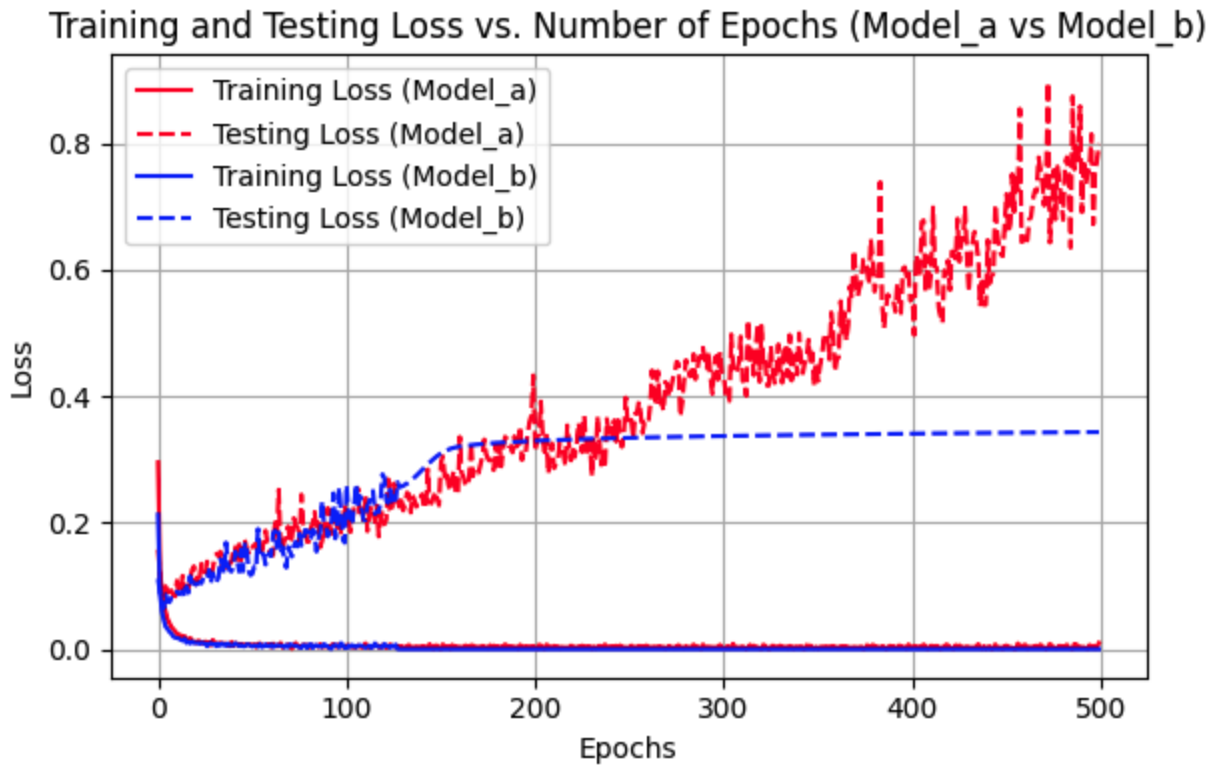

Training Time vs Training and Testing Error (Model_b)
Training Error (Model_b)
Testing Error (Model_b)
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0
1000
2000
3000
4000
5000
6000
Training Time (seconds)



A graph that represents loss (training loss and testing loss) and number of epochs (x-axis):


Training and Testing Loss vs. Number of Epochs (Model_a)
Training Loss (Model_a)
Testing Loss (Model_a)
Loss
Epochs
0.0
0.2
0.4
0.6
0.8
0
100
200
300
400
500


Training and Testing Loss vs. Number of Epochs (Model_b)
Training Loss (Model_b)
Testing Loss (Model_b)
Loss
Epochs
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0
100
200
300
400
500



The state-of-the-art (SOTA) accuracy for the MNIST dataset:

Data augmentation, ensembling, and other sophisticated techniques, along with models like Convolutional Neural Networks (CNNs), have been used to reach the state-of-the-art (SOTA) accuracy for the MNIST dataset. Model accuracies for some widely known models are:

- **LeNet-5**: ~99.2%
- **Ensemble of CNNs**: ~99.8%-99.9%
- **Capsule Networks (CapsNet)**: ~99.75%-99.9%
- **Deep CNN (like ResNet)**: ~99.7%-99.8%

For my Model_a, best testing accuracy = 98.36%
For my Model_b, best testing accuracy = 98.47%

5. Conclusion:

Successfully drew a computational graph for the given function, calculated the forward pass and backward pass values

Successfully wrote and executed the code which provided visual comparisons between **model_a** and **model_b** in terms of training time, training error, testing error, and loss.

We can say that,

- **Better Accuracy**: The Model with higher accuracy performs better. So, Model_b performs better as it has higher testing accuracy.

- **Lower Loss**: Lower loss model performs better. So, Model_b is better in this aspect as it has lower loss

- **Computational efficiency**: The model with a smaller difference between training accuracy and testing accuracy has less overfitting, in this case, Model_b is better

- **Training Time**: For similar testing accuracy, the model with less training time is preferred. So, in this case, Model_a is better

6. References:

1. https://www.kaggle.com/datasets/hojjatk/mnist-dataset

2. https://www.sciencedirect.com/topics/computer-science/deep-neural-network#:~:text=Deep%20neural%20networks%20are%20a,compared%20to%20conventional%20neural%20networks.

3. https://en.wikipedia.org/wiki/Activation_function

4. https://towardsdatascience.com/softmax-activation-function-how-it-actually-works-d292d335bd78#:~:text=Softmax%20is%20an%20activation%20function,all%20possible%20outcomes%20or%20classes.

5. S. Sabour, N. Frosst and G. E. Hinton, "Dynamic Routing Between Capsules," in Advances in Neural Information Processing Systems, 2017. Available: https://arxiv.org/abs/1710.09829.

6. Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

7. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929-1958, 2014. Available: http://jmlr.org/papers/v15/srivastava14a.html.

8. D. C. Ciresan, U. Meier and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, Providence, RI, 2012, pp. 3642-3649, doi: 10.1109/CVPR.2012.6248110.