6CS012 - Artificial Intelligence and Machine Learning. Modeling the Neuron: From MCP to Perceptron, Its Learning Algorithm, and Limitations.

Prepared By: Siman Giri {Module Leader - 6CS012}

March 9, 2025

Worksheet - 3.

1 Instructions

This worksheet contains programming exercises based on the material discussed from the slides.This is a graded exercise and are to be completed on your own and is compulsory to submit.
Please answer the questions below using python in the Jupyter Notebook and follow the guidelines below:

- This worksheet must be completed individually.
- All the solutions must be written in Jupyter Notebook {Preferred}.

Learning Objectives:

- **Understand the Computational Model of the MCP Neuron:** Learn the fundamental structure and operation of the McCulloch-Pitts (MCP) neuron and implement it in Python.
- **Analyze the Limitations of the MCP Neuron:** Gain insight into the limitations of the MCP neuron, and explore the motivation behind advancing to more sophisticated models like the perceptron.
- **Understand the Perceptron Model for Binary Classification:** Develop a deep understanding of the perceptron model and its application to binary classification tasks, with hands-on implementation in Python.
- **Evaluate the Limitations of the Perceptron:** Understand the limitations of the perceptron, particularly in dealing with non-linearly separable data, and explore potential solutions.
- **Strengthen Algorithmic and Coding Skills:** Build confidence in writing algorithms and implementing models from scratch through practical coding exercises, enhancing both theoretical understanding and practical skills.

2 Introduction.

Welcome to Workshop - Three. This workshop is based on the exercise we conducted in Tutorial. This workshop shall serve as a warm up exercise to get you further used to the programming **Neural Network**.

3 MCP Neurons:

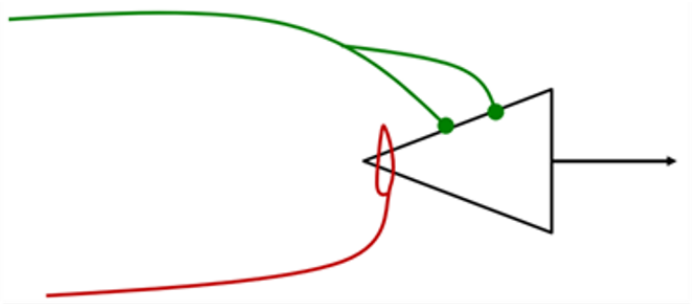

Fig: A Single MCP Neuron proposed as Synaptic Model.

Figure 1: McCulloch - Pits Artificial Neuron

The first computational model of a neuron as proposed by Warren MuCulloch (neuroscientist) and Walter Pitts (logician) in 1943 in their paper titled:

- *"Logical Calculus of Ideas Immanent in Nervous Activity(1943)."*

Based on there thought on what was the fundamentals elements to represent computation in biological neurons they proposed a model also known as **linear threshold gate** or **threshold logic units** or **MCP neurons** most commonly.

It is highly simplified representation, thus we can not make conclusion about neuron based on properties of MCP neurons.Regardless of limitation, MCP is considered a significant first step towards development theory of "Mind and Body" laying the foundations for study of various learning mechanisms and paved the way for advances in artificial intelligence, including the invention of the perceptron and more complex model.

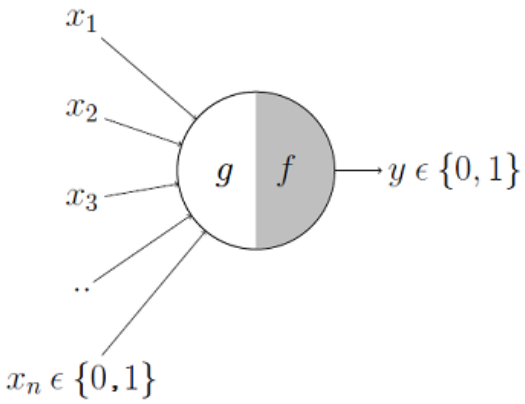

Figure 2: A Symbolic Representation of Neuron. {This is the Representation we will be using further}.

3.1 Mathematical Formalization of the MCP Neuron:

Algorithm 1 MCP Neuron Model

Input: $x_1, x_2, \ldots, x_n$ (n inputs)
Output: y (output of the neuron)
Step 1: Compute the sum of inputs:

$$g(X) = \sum_{i=1}^{n} x_i$$

Step 2: If any input x_i is **inhibitory**, set the output to 0:
if any input x_i is inhibitory **then**
    $y \leftarrow 0$
else
    Step 3: If $g(X) \geq T$, then set the output to 1, else 0:
    if $g(X) \geq T$ **then**
        $y \leftarrow 1$
    else
        $y \leftarrow 0$
    end if
end if

3.2 Task - 1: Implementation of MCP Neurons:

1. Design a MCP Neuron for "AND" Boolean Functions and "OR" Boolean Functions with two inputs:
   Hint:

   - Aggregate all the inputs.
   - Handcraft a Threshold values for "AND" and "OR" Function.
   - Implement a `if else` statement as stated above.

   You may use code base provided below or write your own code from scratch:

For "AND" Operations.

```
def MCP_Neurons_AND(X1, X2, T):
  """
  This functions implements basic AND operations with MCP Neuron for two inputs.
  Arguments:
  Inputs:
  X1 (1 nd array): An array of binary values.
  X2 (1 nd array): An array of binary values.
  Output:
  state_neuron(1D-list): An state of neuron 1 Or 0 for the particular inputs.
  """
  assert len(X1) == len(X2)
  ### YOUR CODE HERE ###
  # Perform an element wise addition of two input arrays stored in a new array(list):
  # Create a new array to put all the prediction let's name that a state_neuron.
  # Append 1 in sate_neuron if sum (element) of above list is above Threshold else append 0.
  return state_neuron
```

Sample Usage for "AND" Function.

```
# Example usage for MCP_Neurons_AND function
X1 = [0, 0, 1, 1]
X2 = [0, 1, 0, 1]
T = 2 # Threshold value
# Call the MCP_Neurons_AND function
result = MCP_Neurons_AND(X1, X2, T)
# Print the result
print(f"Output of AND gate for inputs {X1} and {X2} with threshold {T}: {result}")
```

For "OR" Operations.

```
def MCP_Neurons_OR(X1, X2, T):
    """
    This function implements basic OR operations with MCP Neuron for two inputs.
    Arguments:
    Inputs:
    X1 (1D array): An array of binary values.
    X2 (1D array): An array of binary values.
    Output:
    state_neuron (1D list): The state of the neuron (1 or 0) for the particular inputs.
    """
    assert len(X1) == len(X2)
```

```
    ### YOUR CODE HERE ###
  # Perform an element wise addition of two input arrays stored in a new array(list):
  # Create a new array to put all the prediction let's name that a state_neuron.
  # Append 1 in sate_neuron if sum (element) of above list is above Threshold else append 0.
    return state_neuron
```

Sample Usage for "OR" Function.

```
# Example usage for MCP_Neurons_OR function
X1 = [0, 0, 1, 1]
X2 = [0, 1, 0, 1]
T = 1 # Threshold value for OR gate
# Call the MCP_Neurons_OR function
result_or = MCP_Neurons_OR(X1, X2, T)
# Print the result
print(f"Output of OR gate for inputs {X1} and {X2} with threshold {T}: {result_or}")
```

3.2.1 Answer the Following Question:

- You can use Text cell of your notebook to answer the question.
- **Question - 1:** List out all the limitations of MCP - Neurons.
- **Question - 2:** Think if you can develop a logic to solve for XOR function using MCP Neuron. {Can you devise a `if else` rules.}

4 The Perceptron.

The Perceptron, introduced by Frank Rosenblatt in 1958 and later corrected by Minsky and Papert, extends the McCulloch-Pitts neuron:

- by incorporating learnable numerical weights and
- an adaptive learning process.

It serves as a Linear Binary Classifier, dividing data into two categories based on a linear decision boundary.

A perceptron makes a decision based on a weighted sum of inputs:

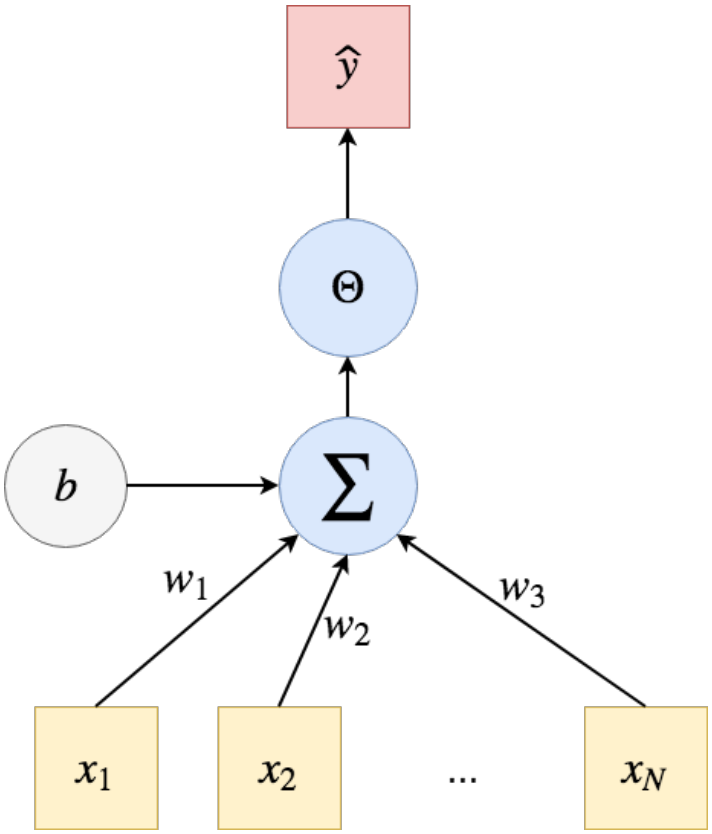

Figure 3: A Symbolic Representation of The Perceptron.

4.1 Mathematical Formulation:

A perceptron takes a set of inputs $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}$, assigns them weights $\mathbf{w_1}, \mathbf{w_2}, \ldots, \mathbf{w_n}$, and computes a weighted sum:

$$z = \sum_{i=1}^{n} w_i x_i + w_o$$

where:

- w_i are the weights learned during training,

- w_o is the bias, also learned during training (adjusts the decision boundary),
- z is the net weighted input.

The perceptron model then applies a threshold activation function (also known as the step function) on z (the net weighted input) given by:

$$f(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

This threshold function determines whether the perceptron activates (outputs 1) or remains inactive (outputs 0).

Algorithm 2 Perceptron Learning Algorithm

Require: Training dataset $D = \{(x_i, y_i)\}$, where:
x_i is the input feature vector (including bias term)
$y_i \in \{0, 1\}$ or $y_i \in \{-1, 1\}$
Require: Learning rate η (small positive value, e.g., 0.01)
Require: Number of epochs (max iterations)
Initialize: Randomly assign small values to $w = (w_1, w_2, \ldots, w_n)$
Randomly initialize bias b (or include it in w)
for each epoch **do**
    ConvergenceFlag = **True**
    for each training sample (x_i, y_i) **do**
        Compute weighted sum:

$$z = \sum w_i x_i + b$$

        Apply activation function (step function):

$$\hat{y} = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{if } z < 0 \end{cases}$$

        if $\hat{y} \neq y$ **then** ▷ Update weights if misclassified
            Update weights:

$$w_i = w_i + \eta(y - \hat{y})x_i$$

            Update bias:

$$b = b + \eta(y - \hat{y})$$

            ConvergenceFlag = **False**
        end if
    end for
    if ConvergenceFlag is **True then**
        Break ▷ Stop if no updates occur (convergence)
    end if
end for
Output: Learned weights w and bias b

4.2 Understanding The Perceptron Learning Algorithm:

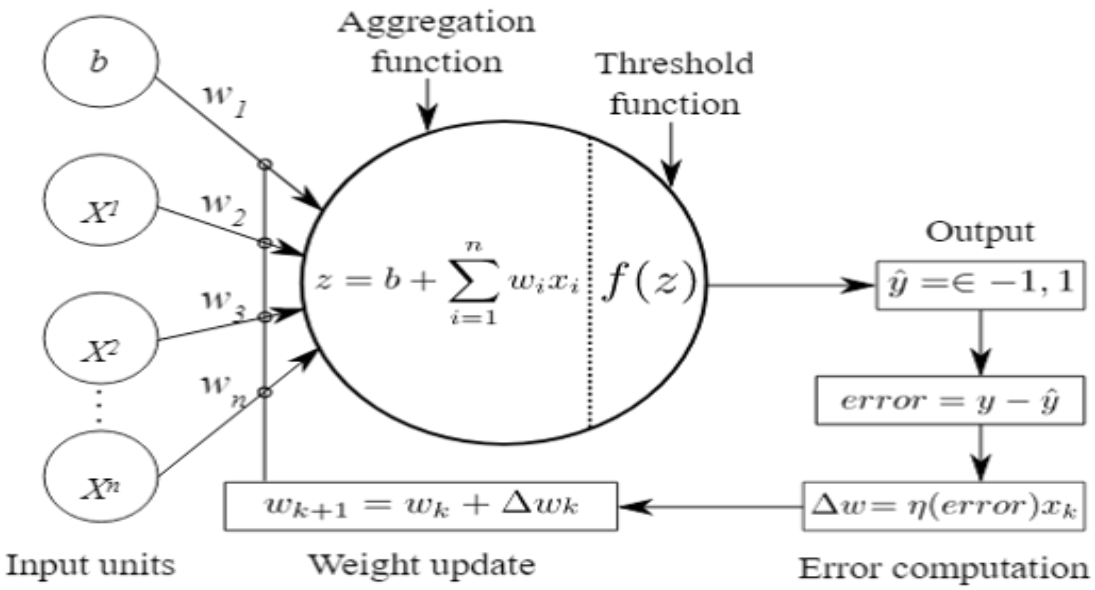

Figure 4: A Pictorial Representation of Perceptron Learning Rule

- **Initialization:** At the beginning, the algorithm initializes the weights (self.weights) to zeros. These weights represent the parameters of the perceptron model.
- **Training (Learning):** The perceptron is trained using a training dataset consisting of input samples (X) and corresponding target labels (y).
  1. The algorithm iterates over the entire training dataset for a fixed number of epochs (nepochs).
  2. For each epoch, the algorithm iterates over each sample in the training dataset:
  3. For each sample, the perceptron makes a prediction based on the current weights.
     - If the prediction is incorrect (i.e., the predicted label does not match the true label),:
       * the algorithm updates the weights to reduce the prediction error.
       * The weight update rule is based on the perceptron learning rule:

$$\Delta w_j = \eta \times (y - \hat{y}) \times x_j$$

where:
· Δw_j is the change in weight for the j-th feature,
· η is the learning rate, controlling the step size of weight updates,
· y is the true label,
· $\hat{y}$ is the predicted label,
· x_j is the j-th feature of the input sample.

       * The bias weight is updated as:

$$\Delta w_o = \eta \times (y - \hat{y})\{x_o = 1.\}$$

- **Prediction:** After training, the perceptron can make predictions on new input samples.
  - Given an input sample (X), the algorithm calculates the aggregated input using the current weights.
  - The perceptron activation function (unit step function) is applied to the aggregated input to determine the predicted class label.
  - If the aggregated input is greater than or equal to zero, the predicted label is assigned as 1; otherwise, it is assigned as 0.

Overall, the perceptron learning algorithm aims to find the decision boundary that separates the input space into two classes. It adjusts the weights based on prediction errors, moving the decision boundary towards correct classification. The algorithm continues iterating until either all samples are correctly classified or until the maximum number of epochs is reached.

4.3 Task 2: Perceptron Algorithm for 0 vs 1 Classification.

1. Objective:

In this exercise, you will implement a Perceptron learning algorithm for binary classification using the MNIST dataset. Specifically, you will classify the digits 0 and 1. After completing the Perceptron algorithm, you will evaluate the model's performance and visualize misclassified images.

`Dataset: mnist_0_and_1.csv`

2. Load the Dataset:

Start by loading the MNIST dataset containing digits 0 and 1.

Loading the Dataset:

```
import pandas as pd
import numpy as np
# Load the dataset
df_0_1 = pd.read_csv("/content/mnist_0_and_1.csv") # Add the correct file path if necessary
# Extract features and labels
X = df_0_1.drop(columns=["label"]).values # 784 pixels
y = df_0_1["label"].values # Labels (0 or 1)
# Check the shape of the features and labels
print("Feature matrix shape:", X.shape)
print("Label vector shape:", y.shape)
```

Answer the Following Question:

1. **Question - 1: What does the shape of X represent?**
2. **Question - 2: What does the shape of X represent?**

Visualize the Dataset:

Visualizing the Dataset.

```
# Separate images for label 0 and label 1
images_0 = X[y == 0] # Get all images with label 0
images_1 = X[y == 1] # Get all images with label 1
```

```
fig, axes = plt.subplots(2, 5, figsize=(10, 5))
# Check if the arrays have the required amount of data
if len(images_0) < 5 or len(images_1) < 5:
    print("Error: Not enough images in images_0 or images_1 to plot 5 images.")
else:
    for i in range(5):
        # Plot digit 0
        axes[0, i].imshow(images_0[i].reshape(28, 28), cmap="gray")
        axes[0, i].set_title("Label: 0")
        axes[0, i].axis("off")
        # Plot digit 1
        axes[1, i].imshow(images_1[i].reshape(28, 28), cmap="gray")
        axes[1, i].set_title("Label: 1")
        axes[1, i].axis("off")
    plt.suptitle("First 5 Images of 0 and 1 from MNIST Subset")
    plt.show()
```

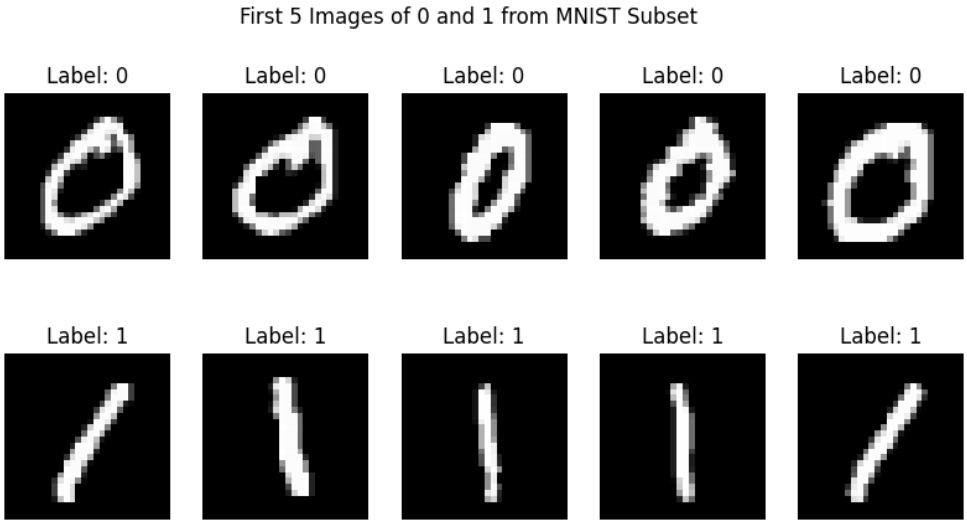

Figure 5: Sample Output

3. Initialize Weights and Bias:

We initialize the weights and bias to zero. This will be used to make predictions for each input.

Initialization of the Weights:

```
# Initialize weights and bias
weights = np.zeros(X.shape[1]) # 784 weights (one for each pixel)
bias = 0
learning_rate = 0.1
epochs = 100
```

Answer the Following Question:

1. **Question - 3: What does the weights array represent in this context?**
2. **Question - 4: Why are we initializing the weights to zero? What effect could this have on the training process?**

4. Implement a Decision Function or Activation Function:

Assuming we have already learn the weights we will implement a step function which serves as an decision function for perceptron.

Implementation Decision Function:

```
import numpy as np
def decision_function(X, weights, bias):
    """
    Compute the predicted labels for the input data.

    Parameters:
    - X: Features (input data) as a numpy array of shape (n_samples, n_features)
    - weights: Updated weights after training
    - bias: Updated bias after training

    Returns:
    - y_pred_all: The predicted labels for the input data
    """
    predictions = np.dot(X, weights) + bias
    #####Your Code Here############# # Activation function (step function)
    return y_pred_all
```

5. Implement the Perceptron Learning Algorithm:

Now, we will implement the main function train_perceptron of the Perceptron algorithm. The algorithm will iterate over the dataset, compute the output, and update the weights and bias if the prediction is incorrect.

Training the Perceptron:

```
import numpy as np
def train_perceptron(X, y, weights, bias, learning_rate=0.1, epochs=100):
    """
    Train the perceptron using the Perceptron Learning Algorithm.
    Parameters:
    - X: Features (input data) as a numpy array of shape (n_samples, n_features)
    - y: Labels (true output) as a numpy array of shape (n_samples,)
    - weights: Initial weights as a numpy array of shape (n_features,)
    - bias: Initial bias value (scalar)
    - learning_rate: Learning rate for weight updates (default is 0.1)
    - epochs: Number of iterations to train the model (default is 100)

    Returns:
    - weights: Updated weights after training
    - bias: Updated bias after training
    - accuracy: Total correct prediction.
    """
    # Step 3: Perceptron Learning Algorithm
    # Your Code here#

    return weights, bias, accuracy
```

Answer the Following Question:

1. **Question - 5: What is the purpose of the output = np.dot(X[i], weights) + bias line?**

2. **Question - 6: What happens when the prediction is wrong? How are the weights and bias updated?**

3. **Question - 7:Why is the final accuracy important, and what do you expect it to be?**

6. Putting it all Together:

Train the perceptron algorithm on whole dataset:

Training the Perceptron Algorithm:

```
# After training the model with the perceptron_learning_algorithm
weights, bias, accuracy = train_perceptron(X, y, weights, bias)
# Evaluate the model using the new function
print("The Final Accuracy is: ", accuracy)
```

7. Visualizing the Misclassified Image:

Finally, let's visualize the images where the model made incorrect predictions. If all images were correctly classified, print a message indicating this.

Visualizing Misclassified Image.

```
# Get predictions for all data points
predictions = np.dot(X, weights) + bias
y_pred = np.where(predictions >= 0, 1, 0)
# Calculate final accuracy
final_accuracy = np.mean(y_pred == y)
print(f"Final Accuracy: {final_accuracy:.4f}")
# Step 5: Visualize Misclassified Images
misclassified_idx = np.where(y_pred != y)[0]
if len(misclassified_idx) > 0:
    fig, axes = plt.subplots(2, 5, figsize=(10, 5))
    for ax, idx in zip(axes.flat, misclassified_idx[:10]): # Show 10 misclassified images
        ax.imshow(X[idx].reshape(28, 28), cmap="gray")
        ax.set_title(f"Pred: {y_pred[idx]}, True: {y[idx]}")
        ax.axis("off")
    plt.suptitle("Misclassified Images")
    plt.show()
else:
    print("All images were correctly classified!")
```

Answer the Following Question:

1. **Question - 8:What does misclassified_idx store, and how is it used in this code?**

2. **Question - 9: How do you interpret the result if the output is "All images were correctly classified!"?**

4.4 Task 3: Perceptron Algorithm for 3 vs 5 Classification.

1. Objective:

In this exercise, you will implement a Perceptron learning algorithm for binary classification using the MNIST dataset. Specifically, you will classify the digits 3 and 5. After completing the Perceptron algorithm, you will evaluate the model's performance and visualize misclassified images.

`Dataset: mnist_3_and_5.csv`

4.5 To - Do:

1. Implement each Step as we implemented above.
2. Visualize the final misclassified images and Provide your conclusion.

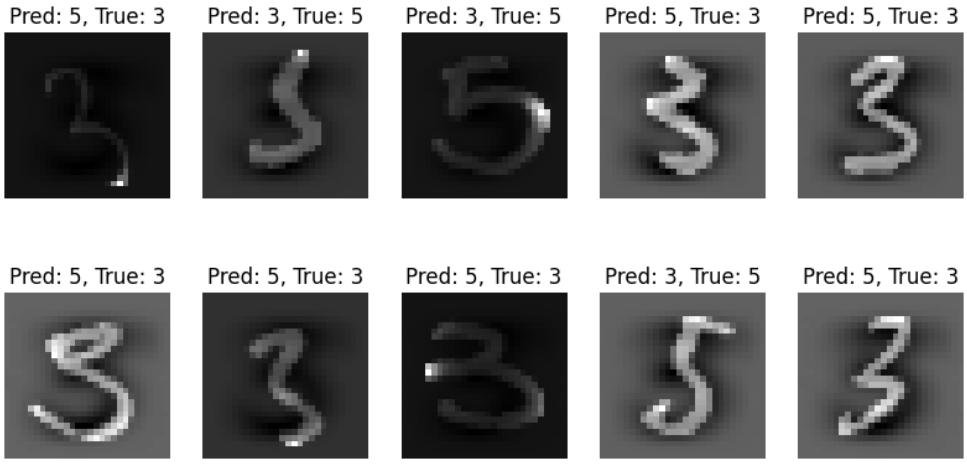

Figure 6: Misclassified Image on 3 or 5 Classification.

Good Luck.