```
[10] 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 0r 0 for the particular inputs.
       """
       assert len(X1) == len(X2)

       state_neuron = []

       for x1, x2 in zip(X1, X2):
           if (x1 + x2) >= T:
               state_neuron.append(1)
           else:
               state_neuron.append(0)

       return state_neuron
```

```
[11] # 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)
```

```
[11] 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}")
```

```
Output of AND gate for inputs [0, 0, 1, 1] and [0, 1, 0, 1] with threshold 2: [0, 0, 0, 1]
```

```
[12] 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)

         state_neuron = []

         for x1, x2 in zip(X1, X2):
             sum_inputs = x1 + x2
             state_neuron.append(1 if sum_inputs >= T else 0)

         return state_neuron
```

```python
# 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}")
```

```
Output of OR gate for inputs [0, 0, 1, 1] and [0, 1, 0, 1] with threshold 1: [0, 1, 1, 1]
```

Implementation for 0 Vs. 1 Classification.

Step 1: Load the Dataset

[1]
```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[2]
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


# Load the dataset
df_0_1 = pd.read_csv("/content/drive/MyDrive/Artificial Intelligence and Machine Learning/Week3Workshop3/mnist_0_and_1.csv

# 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)
```

```
Feature matrix shape: (12665, 784)
Label vector shape: (12665,)
```

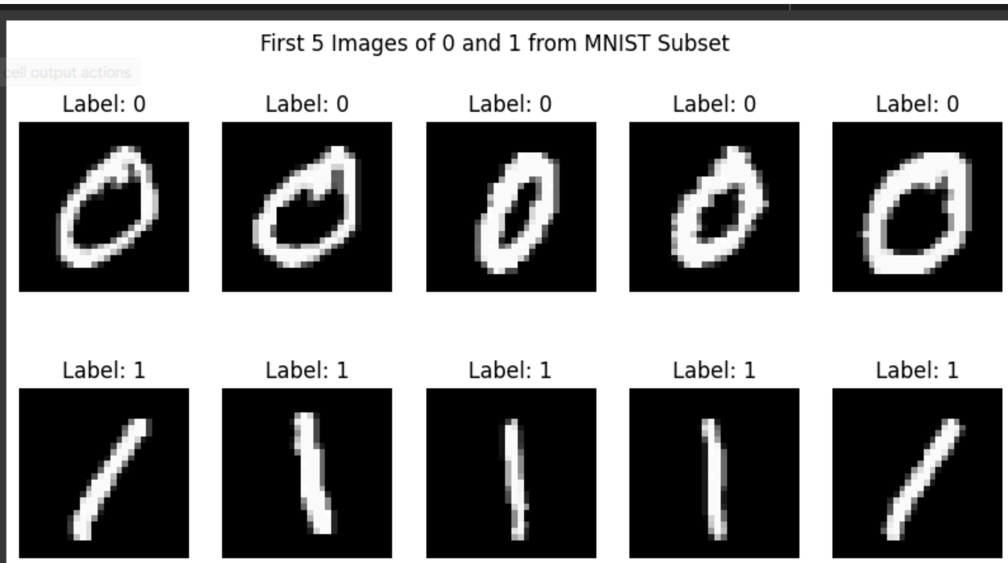
Viewing the Dataset.

[3]
```python
# 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
```

```
[3] # 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()
```



Step - 2 - Initializing the Weights:

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

Step - 3 - Make a Decision function:

Step - 3 - Make a Decision function:

[5]
```
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
    y_pred_all = np.where(predictions >= 0, 1, 0)
    return y_pred_all
```

Step - 3 - Implement the Perceptron Learning Algorithm

[6]
```
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
```

```
[6]       - bias: Updated bias after training
          - accuracy: Total correct prediction.
          """
          # Step 3: Perceptron Learning Algorithm
          n_samples = X.shape[0]

          for epoch in range(epochs):
              correct_predictions = 0

              for i in range(n_samples):
                  linear_output = np.dot(X[i], weights) + bias
                  y_pred = 1 if linear_output >= 0 else 0

                  if y_pred != y[i]:
                      update = learning_rate * (y[i] - y_pred)
                      weights += update * X[i]
                      bias += update
                  else:
                      correct_predictions += 1

              accuracy = correct_predictions / n_samples

          return weights, bias, accuracy
```

Training the Perceptron

```
[7] # 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)
```

```
The Final Accuracy is:  1.0
```

Step 5: Visualize Misclassified Images

Step 5: Visualize Misclassified Images

[8]
```python
# 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!")
```

```
Final Accuracy: 1.0000
All images were correctly classified!
```

[9]
```python
predictions = np.dot(X, weights) + bias
y_pred = np.where(predictions >= 0, 1, 0)

final_accuracy = np.mean(y_pred == y)
print(f"Final Accuracy: {final_accuracy:.4f}")

classified_idx = np.where(y_pred == y)[0]

if len(classified_idx) > 0:
    fig, axes = plt.subplots(2, 5, figsize=(10, 5))
    for ax, idx in zip(axes.flat, classified_idx[:10]):
        ax.imshow(X[idx].reshape(28, 28), cmap="gray")
        ax.set_title(f"Pred: {y_pred[idx]}, True: {y[idx]}")
        ax.axis("off")
```

```
[9]     for ax, idx in zip(axes.flat, classified_idx[:10]):
            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("Correctly Classified Images")
        plt.show()
    else:
        print("No images were correctly classified!")
```

```
Final Accuracy: 1.0000
```