

Output of Worksheet-3

```
[1] return state_neuron

[2] from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

[3] # 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}")

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

[4] 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.
    """

[5] # 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

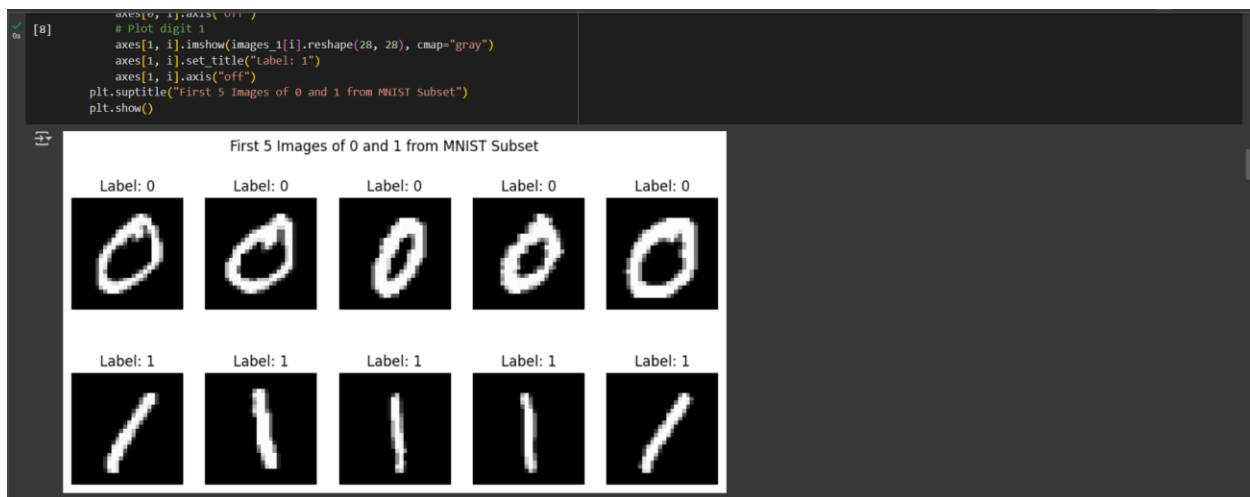
[6] import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Load the dataset
df_0_1 = pd.read_csv("/content/drive/MyDrive/AI and ML/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)

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



```
[9] # 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:

```
[10] 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)
    y_pred_all = np.where(predictions >= 0, 1, 0)
    return y_pred_all
```

Training the Perceptron

```
# 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)
```

Epoch 0: Accuracy = 0.9967
Epoch 10: Accuracy = 0.9995
Epoch 20: Accuracy = 1.0000
Epoch 30: Accuracy = 1.0000
Epoch 40: Accuracy = 1.0000
Epoch 50: Accuracy = 1.0000
Epoch 60: Accuracy = 1.0000
Epoch 70: Accuracy = 1.0000
Epoch 80: Accuracy = 1.0000
Epoch 90: Accuracy = 1.0000
The Final Accuracy is: 1.0

Step 5: Visualize Misclassified Images

```
[13] # Get predictions for all data points
predictions = np.dot(X, weights) + bias
y_pred = np.where(predictions >= 0, 1, 0)
```

```

[13] # 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!

```

Part 2

Part 2

```

[15] # Load the dataset
df_3_5 = pd.read_csv("/content/drive/MyDrive/AI and ML/mnist_0_and_1.csv") # Add the correct file path if necessary

# Extract features and labels
X = df_3_5.drop(columns=["label"]).values # 784 pixels
y = df_3_5["label"].values # Labels (3 or 5)

# 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,)

```

```

[16] # Separate images for label 0 and label 1
images_3 = X[y == 3] # Get all images with label 0
images_5 = X[y == 5] # Get all images with label 1

fig, axes = plt.subplots(2, 5, figsize=(10, 5))

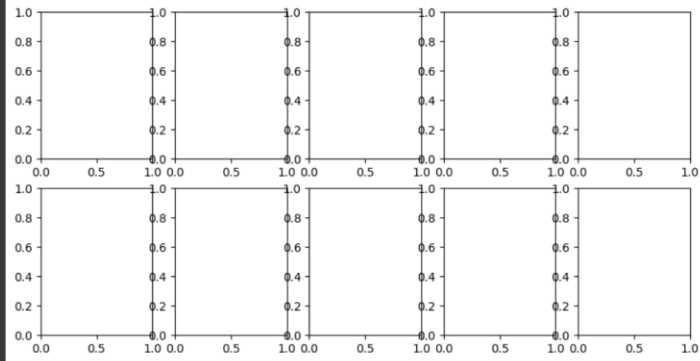
# Check if the arrays have the required amount of data
if len(images_3) < 5 or len(images_5) < 5:
    print("Error: Not enough images in images_3 or images_5 to plot 5 images.")
else:
    for i in range(5):
        # Plot digit 0
        axes[0, i].imshow(images_3[i].reshape(28, 28), cmap="gray")
        axes[0, i].set_title('label: 3')

```

```

Error: Not enough images in images_3 or images_5 to plot 5 images.

```



```

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

```

```

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

[18] 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)
    y_pred_all = np.where(predictions>=0, 5, 3)
    return y_pred_all

[19] 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,)

```

```

weights, bias, accuracy = train_perceptron(X, y, weights, bias)

```

```

# Evaluate the model using the new function
print("The Final Accuracy is: ", accuracy)

```

```

Epoch 0: Accuracy = 0.0000
Epoch 10: Accuracy = 0.0000
Epoch 20: Accuracy = 0.0000
Epoch 30: Accuracy = 0.0000
Epoch 40: Accuracy = 0.0000
Epoch 50: Accuracy = 0.0000
Epoch 60: Accuracy = 0.0000
Epoch 70: Accuracy = 0.0000
Epoch 80: Accuracy = 0.0000
Epoch 90: Accuracy = 0.0000
The Final Accuracy is: 0.0

```

```

# Get predictions for all data points
predictions = np.dot(X, weights) + bias
y_pred = np.where(predictions >= 0, 5, 3)

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

```

```

print("All images were correctly classified!")

```

Final Accuracy: 0.0000

Misclassified Images

