# CS 405: Deep Learning | Lab

## Experiment 1: Design and Implementation of a Single-Layer Perceptron Model

**Student Name:** Nabeel Shan
**Registration No:** 468752

### 1. Objective

To introduce the concept and working of a perceptron neural network by implementing a single-layer perceptron that classifies inputs based on a step activation function using weights and bias.

### 2. Introduction

A **Perceptron** is the fundamental building block of Deep Learning. Originally developed by Frank Rosenblatt, it is a binary classification algorithm that maps a vector of real-valued inputs to a single binary output.

The core operation involves:

1. **Weighted Sum:** Calculating the dot product of inputs ($x$) and weights ($w$), adding a bias ($b$).
$$z = w \cdot x + b$$
2. **Activation:** Applying a step function (Hard Limit) to determine the output class.
$$y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

This experiment implements this logic from scratch to classify a simple object (Apple vs. Not Apple) based on weight and color features.

```
import numpy as np

class Perceptron:
    def __init__(self, weights, bias):
        """
        Initialize the Perceptron with specific weights and bias.

        Args:
            weights (list or np.array): The weights for the inputs.
            bias (float): The bias value.
        """
        self.weights = np.array(weights)
        self.bias = bias

    def activation(self, z):
        """
        Step Activation Function.
        Returns 1 if z >= 0, else 0.
        """
        return 1 if z >= 0 else 0

    def predict(self, inputs):
        """
        Calculate the weighted sum and apply activation.

        Formula: z = (w * x) + b
        """
        inputs = np.array(inputs)

        # Calculate dot product (Weighted Sum)
        weighted_sum = np.dot(self.weights, inputs) + self.bias

        return self.activation(weighted_sum)
```

## Classifying an object

```
input_features = [150, 0.9]

# Model Parameters
```

```
weights = [0.5, 1.0]
bias = 1.5
```

## 1. Initialize the Model

```
p = Perceptron(weights, bias)
```

## 2. Make a Prediction

```
prediction = p.predict(input_features)
```

## 3. Output Results

```
print(f"Input Features: Weight={input_features[0]}g, Color={input_features[1]}")
print(f"Weighted Sum Calculation: ({input_features[0]} * {weights[0]}) + ({input_features[1]} * {weights[1]}) + {bias}")
print("=" * 100)
print("Prediction Result:", "Apple (1)" if prediction == 1 else "Not an Apple (0)")
```

```
Input Features: Weight=150g, Color=0.9
Weighted Sum Calculation: (150 * 0.5) + (0.9 * 1.0) + 1.5
====================================================================================================
Prediction Result: Apple (1)
```

```
Start coding or generate with AI.
```