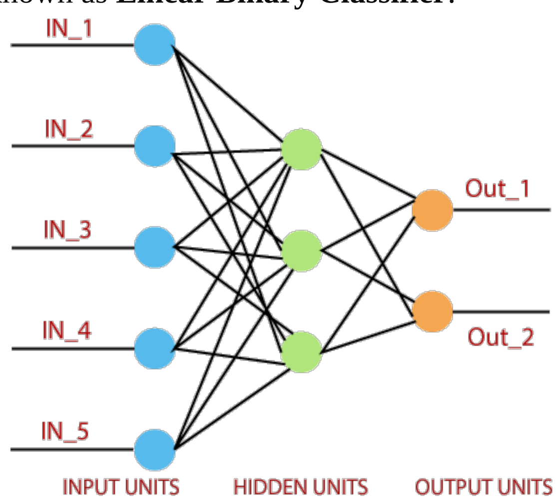
Experiment No. 7

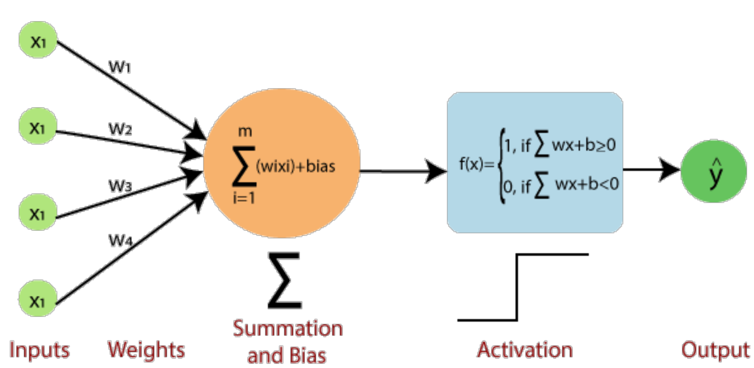
Aim: Implementation of Single Layer Perceptron Learning Algorithm

Objective: Able to implement and understand the aspects of Single Layer Perceptron Learning Algorithm.

Theory:
The perceptron is a single processing unit of any neural network. **Frank Rosenblatt** first proposed in **1958** is a simple neuron which is used to classify its input into one or two categories. Perceptron is a linear classifier, and is used in supervised learning. It helps to organize the given input data.
A perceptron is a neural network unit that does a precise computation to detect features in the input data. Perceptron is mainly used to classify the data into two parts. Therefore, it is also known as **Linear Binary Classifier**.

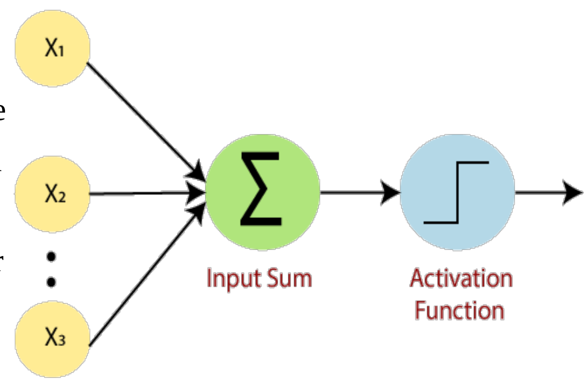


Perceptron uses the step function that returns +1 if the weighted sum of its input 0 and -1.
The activation function is used to map the input between the required value like (0, 1) or (-1, 1).
A regular neural network looks like this:

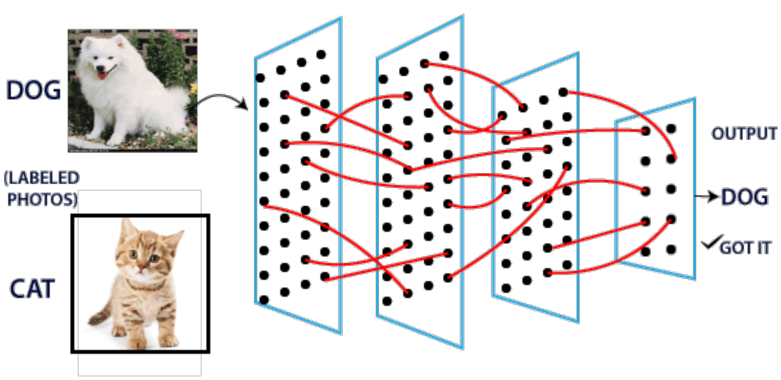


The perceptron consists of 4 parts.

- **Input value or One input layer:** The input layer of the perceptron is made of artificial input neurons and takes the initial data into the system for further processing.
- **Weights and Bias:**
  Weight: It represents the dimension or strength of the connection between units. If the weight to node 1 to node 2 has a higher quantity, then neuron 1 has a more considerable influence on the neuron.
  Bias: It is the same as the intercept added in a linear equation. It is an additional parameter which task is to modify the output along with the weighted sum of the input to the other neuron.
- **Net sum:** It calculates the total sum.
- **Activation Function:** A neuron can be activated or not, is determined by an activation function. The activation function calculates a weighted sum and further adding bias with it to give the result.



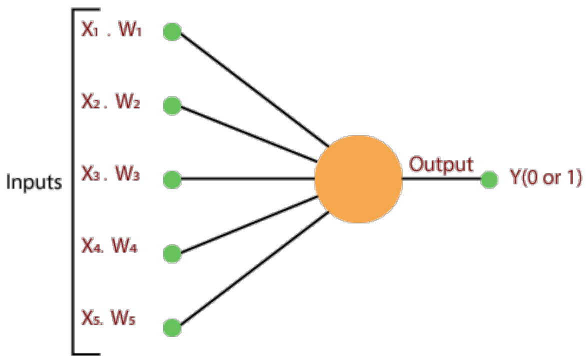
A standard neural network looks like the below diagram.



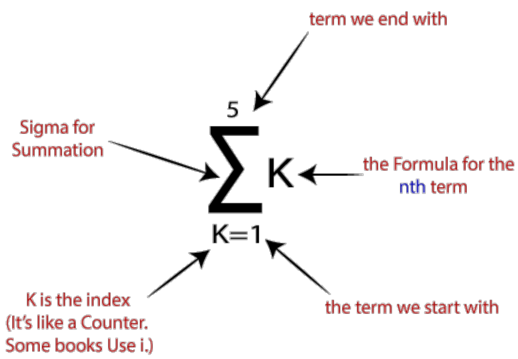
How does it work?

The perceptron works on these simple steps which are given below:

a. In the first step, all the inputs x are multiplied with their weights **w**.



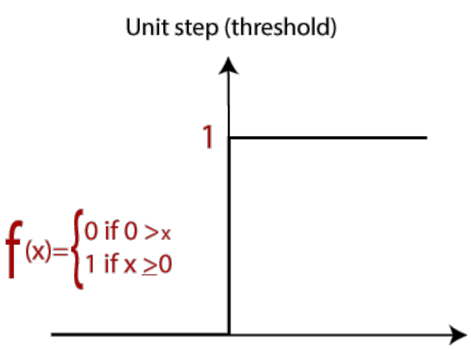
b. In this step, add all the increased values and call them the **Weighted sum**.



c. In our last step, apply the weighted sum to a correct **Activation Function**.

For Example:

A Unit Step Activation Function



Implementation:

Single Layer Perceptron Learning algorithm

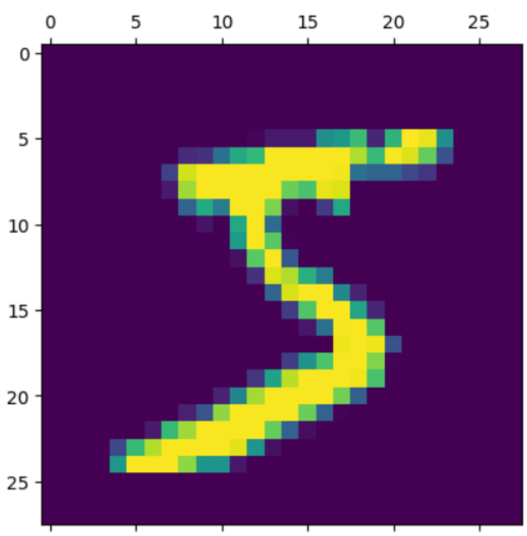
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
%matplotlib inline
```

```
(x_train, y_train),     (x_test, y_test) = keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 1s 0us/step
```

```
len(x_train)
len(x_test)
x_train[0].shape
plt.matshow(x_train[0])
```

```
<matplotlib.image.AxesImage at 0x7818c8b80400>
```



```
# Normalizing the dataset
x_train = x_train/255
x_test = x_test/255

# Flatting the dataset in order
# to compute for model building
x_train_flatten = x_train.reshape(len(x_train), 28*28)
x_test_flatten = x_test.reshape(len(x_test), 28*28)
```

```
model = keras.Sequential([
        keras.layers.Dense(10, input_shape=(784,),
                                        activation='sigmoid')
])
model.compile(
        optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy'])

model.fit(x_train_flatten, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.4644 - accuracy: 0.8774
Epoch 2/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3032 - accuracy: 0.9152
Epoch 3/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2830 - accuracy: 0.9208
Epoch 4/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2727 - accuracy: 0.9238
Epoch 5/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2668 - accuracy: 0.9255
<keras.src.callbacks.History at 0x7818c590aad0>
```


```
# Normalizing the dataset
x_train = x_train/255
x_test = x_test/255

# Flatting the dataset in order
# to compute for model building
x_train_flatten = x_train.reshape(len(x_train), 28*28)
x_test_flatten = x_test.reshape(len(x_test), 28*28)
```

```
model = keras.Sequential([
        keras.layers.Dense(10, input_shape=(784,),
                                        activation='sigmoid')
])
model.compile(
        optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy'])

model.fit(x_train_flatten, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.4644 - accuracy: 0.8774
Epoch 2/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3032 - accuracy: 0.9152
Epoch 3/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2830 - accuracy: 0.9208
Epoch 4/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2727 - accuracy: 0.9238
Epoch 5/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2668 - accuracy: 0.9255
```

```
<keras.src.callbacks.History at 0x7818c590aad0>
```

```
model.evaluate(x_test_flatten, y_test)
```

```
313/313 [==============================] - 1s 1ms/step - loss: 0.2667 - accuracy: 0.9259
```

```
[0.2666851282119751, 0.9258999824523926]
```

Conclusion:

The Single Layer Perceptron Learning Algorithm provides a foundation for understanding neural network training and classification tasks. While limited to linearly separable problems, it serves as the basis for more complex neural network architectures capable of handling non-linear relationships in data.