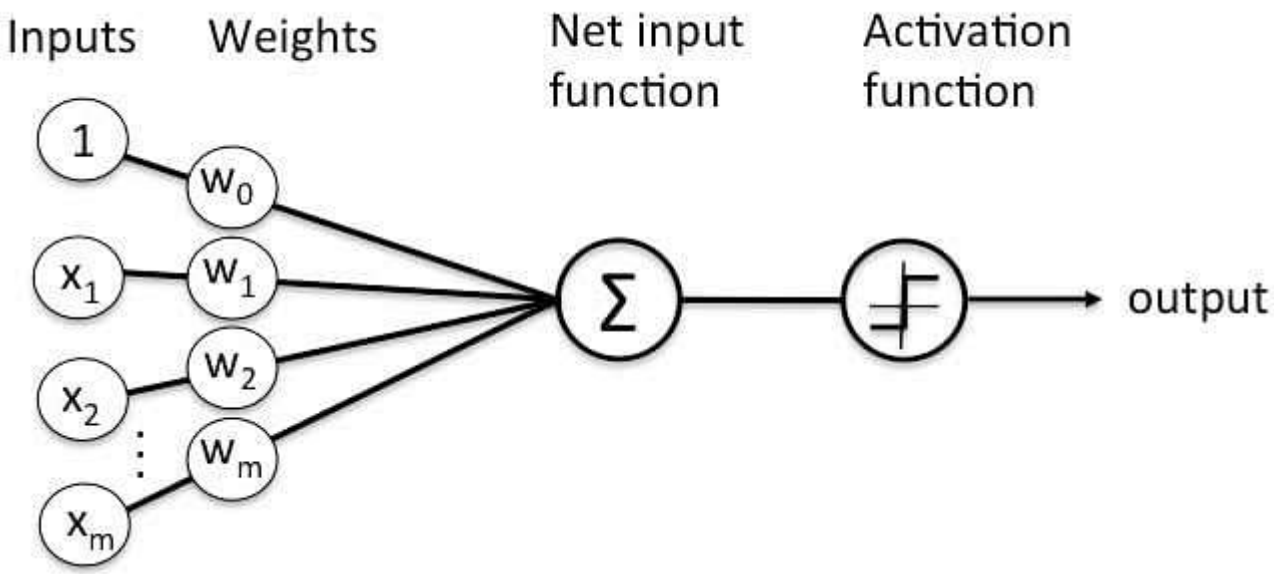
WEEK -7

Aim: .Create a perceptron with appropriate number of inputs and outputs. Train it using a fixed increment learning algorithm until no change in weights is required. Output the final weights.

Description :

Perceptron

Perceptron was introduced by Frank Rosenblatt in 1957. He proposed a Perceptron learning rule based on the original MCP neuron. A Perceptron is an algorithm for supervised learning of binary classifiers. This algorithm enables neurons to learn and processes elements in the training set one at a time.



Types of Perceptron:

1. Single layer: Single layer perceptron can learn only linearly separable patterns.
2. Multilayer: Multilayer perceptrons can learn about two or more layers having a greater processing power.

The Perceptron algorithm learns the weights for the input signals in order to draw a linear decision boundary.

Note: Supervised Learning is a type of Machine Learning used to learn models from labeled training data. It enables output prediction for future or unseen data. Let us focus on the Perceptron Learning Rule in the next section.

Perceptron in Machine Learning

The most commonly used term in Artificial Intelligence and Machine Learning (AIML) is Perceptron. It is the beginning step of learning coding and Deep Learning technologies, which consists of input values, scores, thresholds, and weights implementing logic gates. Perceptron is the nurturing step of an Artificial Neural Link. In 19h century, Mr. Frank Rosenblatt invented the Perceptron to perform specific high-level calculations to detect input data capabilities or business intelligence. However, now it is used for various other purposes.

What is the Perceptron Model in Machine Learning?

A machine-based algorithm used for supervised learning of various binary sorting tasks is called Perceptron. Furthermore, Perceptron also has an essential role as an Artificial Neuron or Neural link in detecting certain input data computations in business intelligence. A perceptron model is also classified as one of the best and most specific types of Artificial Neural networks. Being a supervised learning algorithm of binary classifiers, we can also consider it a single-layer neural network with four main parameters: input values, weights and Bias, net sum, and an activation function.

How Does Perceptron Work?

AS discussed earlier, Perceptron is considered a single-layer neural link with four main parameters. The perceptron model begins with multiplying all input values and their weights, then adds these values to create the weighted sum. Further, this weighted sum is applied to the activation function ‘f’ to obtain the desired output. This activation function is also known as the step function and is represented by ‘f.’

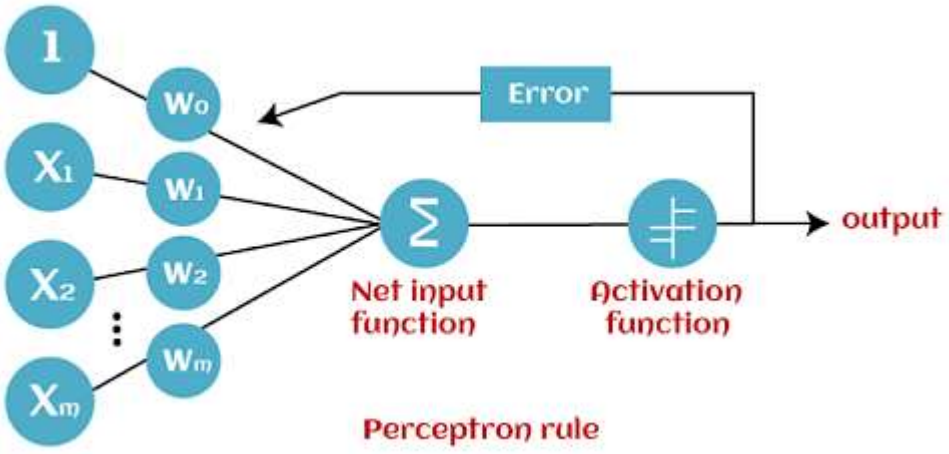


IMAGE COURTESY: javapoint

This step function or Activation function is vital in ensuring that output is mapped between (0,1) or (-1,1). Take note that the weight of input indicates a node's strength. Similarly, an input value gives the ability the shift the activation function curve up or down.

Step 1: Multiply all input values with corresponding weight values and then add to calculate the weighted sum. The following is the mathematical expression of it:

$\sum wi*xi = x1*w1 + x2*w2 + x3*w3+……..x4*w4$

Add a term called bias 'b' to this weighted sum to improve the model's performance.

Step 2: An activation function is applied with the above-mentioned weighted sum giving us an output either in binary form or a continuous value as follows:

$Y=f(\sum wi*xi + b)$

Code:

```
# importing libraies

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt


dataset = [[2.7810836,2.550537003,0],
```

```
        [1.465489372,2.362125076,0],
        [3.396561688,4.400293529,0],
        [1.38807019,1.850220317,0],
        [3.06407232,3.005305973,0],
        [7.627531214,2.759262235,1],
        [5.332441248,2.088626775,1],
        [6.922596716,1.77106367,1],
        [8.675418651,-0.242068655,1],
        [7.673756466,3.508563011,1]]

df = pd.DataFrame(dataset, columns=['X1', 'X2', 'label'])
df
```

	X1	X2	label
0	2.781084	2.550537	0
1	1.465489	2.362125	0
2	3.396562	4.400294	0
3	1.388070	1.850220	0
4	3.064072	3.005306	0
5	7.627531	2.759262	1
6	5.332441	2.088627	1
7	6.922597	1.771064	1
8	8.675419	-0.242069	1
9	7.673756	3.508563	1

```
def train_data(data):
    X = data.drop(['label'], axis=1)
    y = data['label']
    return X, y


def train(X, y):
```

```
# Equation of the model or classifier
# W1.X1 + W2.X2 + b = Y

# initialising the Guassian random weights to model
w1, w2, b = tuple(np.random.normal(size=3))
print("Initial Weights + bias : ", (w1, w2, b))
# w1 = w2 = b = 0.1

# Number of epochs and learning rate
epochs = 5
alpha = 0.01

flag = True
j = 0
while True:
    error = []
    j += 1
    for i in range(len(X)):
        # Predicting target label with current weights
        y_pred = w1*X.iloc[i, 0] + w2*X.iloc[i, 1] + b
        # actual target label
        t = y[i]
        # difference between actual and predicted
        diff = t - y_pred
        # error
        err = diff**2
        error.append(err)
        # --- Updation ---
        # gradients
        delta_w1 = diff*X.iloc[i, 0]
        delta_w2 = diff*X.iloc[i, 1]
```

```
            delta_b = diff
            # New weights
            w1 = w1 + alpha*delta_w1
            w2 = w2 + alpha*delta_w2
            b = b + alpha*delta_b

        # Mean square error
        MSE = np.array(error).mean()
        print("Epoch -", str(j))
        print("Mean Square error :", MSE)
        print("Weights :-")
        print("w1 :", w1)
        print("w2 :", w2)
        print("b :", b)
        if MSE < 0.05:
            break
    return w1, w2, b

X, y = train_data(df)
w1, w2, b = train(X, y)
```

```
Initial Weights + bias :  (0.45383305727291356, 2.921453190325383, -1.507149292129177)
Epoch - 1
Mean Square error : 27.515367427850332
Weights :-
w1 : -0.23856619504981438
w2 : 1.8000603365770242
b : -1.7882868952438176
Epoch - 2
Mean Square error : 8.261620772125706
Weights :-
w1 : -0.11047171277812418
w2 : 1.4214958497359798
b : -1.8343984308743442
Epoch - 3
Mean Square error : 4.831733800777088
Weights :-
w1 : -0.013192103316310877
w2 : 1.141349278784362
b : -1.8641903144771175
Epoch - 4
Mean Square error : 2.916739546056733
Weights :-
w1 : 0.0588086220507394 2
w2 : 0.9325889736332161
b : -1.882492991930284
Epoch - 5
Mean Square error : 1.8359445894846662
Weights :-
w1 : 0.11203217827880266
w2 : 0.7768358686550576
b : -1.8923020626808693
```

```
Epoch - 283
Mean Square error : 0.05013491816237138
Weights :-
w1 : 0.1736292661037689
w2 : 0.04289557433884931
b : -0.468976265941062
Epoch - 284
Mean Square error : 0.050095992112689114
Weights :-
w1 : 0.1735576776845373
w2 : 0.04267375758722747
b : -0.46784547437779683
Epoch - 285
Mean Square error : 0.050057734579379454
Weights :-
w1 : 0.17348675108250222
w2 : 0.04245399147650932
b : -0.46672513670232973
Epoch - 286
Mean Square error : 0.05002013370168125
Weights :-
w1 : 0.17341648017932776
w2 : 0.04223625704902787
b : -0.4656151562710585
Epoch - 287
Mean Square error : 0.04998317783302722
Weights :-
w1 : 0.17334685891324036
w2 : 0.042020535522375044
b : -0.464515437333827
```

```
def predict(x):
    y_pred = w1*x[0] + w2*x[1] + b
    return 1.0 if y_pred > 0.3 else 0.0


for (i, j) in zip(X.values, y):
    y_p = predict(i)
    print("I/P :", i[0], i[1])
    print("Expected :", j, "Predicted :", y_p)
    print("---------------------------------")
```

```
I/P : 2.7810836 2.550537003
Expected : 0 Predicted : 0.0
---------------------------------
I/P : 1.465489372 2.362125076
Expected : 0 Predicted : 0.0
---------------------------------
I/P : 3.396561688 4.400293529
Expected : 0 Predicted : 1.0
---------------------------------
I/P : 1.38807019 1.850220317
Expected : 0 Predicted : 0.0
---------------------------------
I/P : 3.06407232 3.005305973
Expected : 0 Predicted : 0.0
---------------------------------
I/P : 7.627531214 2.759262235
Expected : 1 Predicted : 1.0
---------------------------------
I/P : 5.332441248 2.088626775
Expected : 1 Predicted : 1.0
---------------------------------
I/P : 6.922596716 1.77106367
Expected : 1 Predicted : 1.0
---------------------------------
I/P : 8.675418651 -0.242068655
Expected : 1 Predicted : 1.0
---------------------------------
I/P : 7.673756466 3.508563011
Expected : 1 Predicted : 1.0
---------------------------------
```