# Introduction to Perceptron

Aman Kumar Nirala, 19030121010

Fri Jun 4 18:33:55 2021

## Introduction

A perceptron is a single layer NN that is used as a linear classifier. A perceptron is a digital form of a neuron which is the building block of our nervous system.

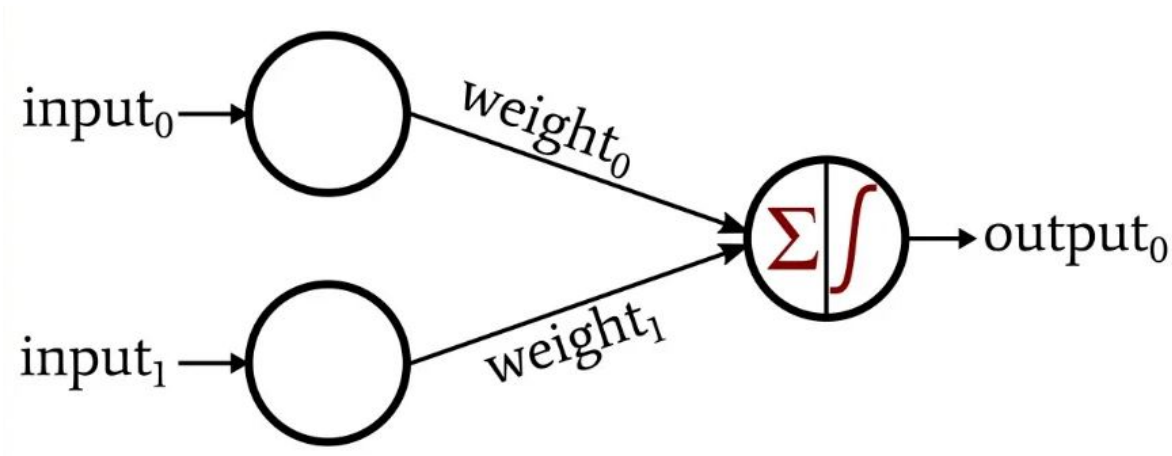


Figure 1: Perceptron Image

## How does it work?

There are four major components of perceptrons that affects the output of the model,

- Input
- Weight
- Bias
- Activation function

First an input is provided to the model which is then changes according to the weight and the bias for the respective input. This can be represented as:

$$f(w, b) = x^T w + b$$

where w = weight tensor, x = input tensor, b = bias Once we get the output from this linear function we sum them up and pass it to the activation function which then gives the output. The mathematical relationship is shown below:

$$\hat{y} = c(x^T w + b)$$

where

$$\hat{y} = output$$

and

$$c = \begin{cases} 1 & if\, x \geq 0 \\ 0 & else \end{cases}$$

While training, depending upon the output from the perceptron we calculate the error and adjust the weights and bias after every iteration. This can be represented as:

$$w = w + \Delta w$$

$$\Delta w = lr.x_i(y_i - \hat{y_i})$$

$$b = b + lr(y_i - \hat{y_i})$$

where lr = learning rate which ranges between 0 to 1

Now lets implement it in python and test our implementation using a sample dataset for classification of breast cancer.

### Importing Libraries

```
'''
@author: Aman Kumar Nirala (github.com/amannirala13)
Created at: Fri Jun  4 18:33:55 2021
'''

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split as tts

from sklearn.metrics import classification_report, confusion_matrix
```

### Perceptron Class

```
class perceptron:

    '''
    DESCRIPTION:
        This is the primary constructor for the perceptron class. It takes in learning rate
        and interations as arguments and initializes other variables

    ARGUMENTS:
        # learning_rate -> Defines the learning rate i.e. lr of the model. Default = 0.01
        # iterations -> Defines the number of time the model should train itself.
        Default = 1000
    '''
    def __init__(self, learning_rate = 0.01, iterations = 1000):
        self.learning_rate = learning_rate
        self.iterations = iterations
        self.weight = None
        self.bias = None
        self.activation_func = self.step_func

    '''
```

```
    DESCRIPTION:
        This function takes in training data set and trains the perceptron.

    ARGUMENTS:
        # X -> Independent variable set
        # y -> Dependent variable set
    '''
    def fit(self, X, y):
        n_sample, n_features = X.shape
        self.weight = np.zeros(n_features)
        self.bias = 0

        y = np.array([1 if i > 0 else 0 for i in y])

        for _ in range(self.iterations):
            for index, x_i in enumerate(X):
                y_predicted = self.predict(x_i)
                delta = self.learning_rate * (y[index] - y_predicted)
                self.weight += delta * x_i
                self.bias = delta

    '''
    DESCRIPTION:
        This function returns the prediction set from the preceptron.

    ARGUMENTS:
        # X -> Independent variable set
    '''
    def predict(self, X):
        l_output = np.dot(X,self.weight) + self.bias
        return self.activation_func(l_output)

    '''
    DESCRIPTION:
        The activation function which determins the final output of the perceptron depending
        upon the input provided by the linear function.

    ARGUMENTS:
        # X -> Input set from linear function
    '''
    def step_func(self, X):
        return np.where(X >= 0, 1, 0)
```

### Cleaning and Preparing data

```
def clean_data(data):
    data = data.iloc[:,:-1]
    data = data.iloc[:, 1:]
    data["diagnosis"] = data["diagnosis"].map({'M':1, 'B':0})
    return data
```

### Dividing the data into Testing and Training sets

```
def get_features_and_outputs(data):
    X = data.iloc[:, 2:]
    y = data.loc[:, "diagnosis"]
    X = pd.DataFrame(MinMaxScaler().fit_transform(X.values))
```

```
    return X, y
```

### Main function

```
def main():
    data = pd.read_csv("data.csv")
    data = clean_data(data)
    X, y = get_features_and_outputs(data)

    X_train, X_test, y_train, y_test = tts(X, y, test_size = 0.2, random_state = 13)

    neuron = perceptron()
    neuron.fit(np.array(X_train),np.array(y_train))

    y_pred = neuron.predict(np.array(X_test))
    validity = np.where(y_pred == np.array(y_test), 1, 0)
    accuracy = (np.sum(validity) / np.size(validity)) * 100
    print('-----------------------------')
    print("Accuracy = ",accuracy,"%")
    print('-----------------------------')
    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test, y_pred))
```

### Entry point of the program

```
if __name__ == "__main__":
    main()
```

```
-----------------------------
Accuracy =  95.6140350877193 %
-----------------------------
[[74  4]
 [ 1 35]]
              precision    recall  f1-score   support

           0       0.99      0.95      0.97        78
           1       0.90      0.97      0.93        36

    accuracy                           0.96       114
   macro avg       0.94      0.96      0.95       114
weighted avg       0.96      0.96      0.96       114
```

---

## Conclusion

As we can see the model with lr = 0.01 and after 1k iterations gave us an accuracy of about 96%. Thus we have successfully created a single level perceptron and used it for classifying a linear dataset.