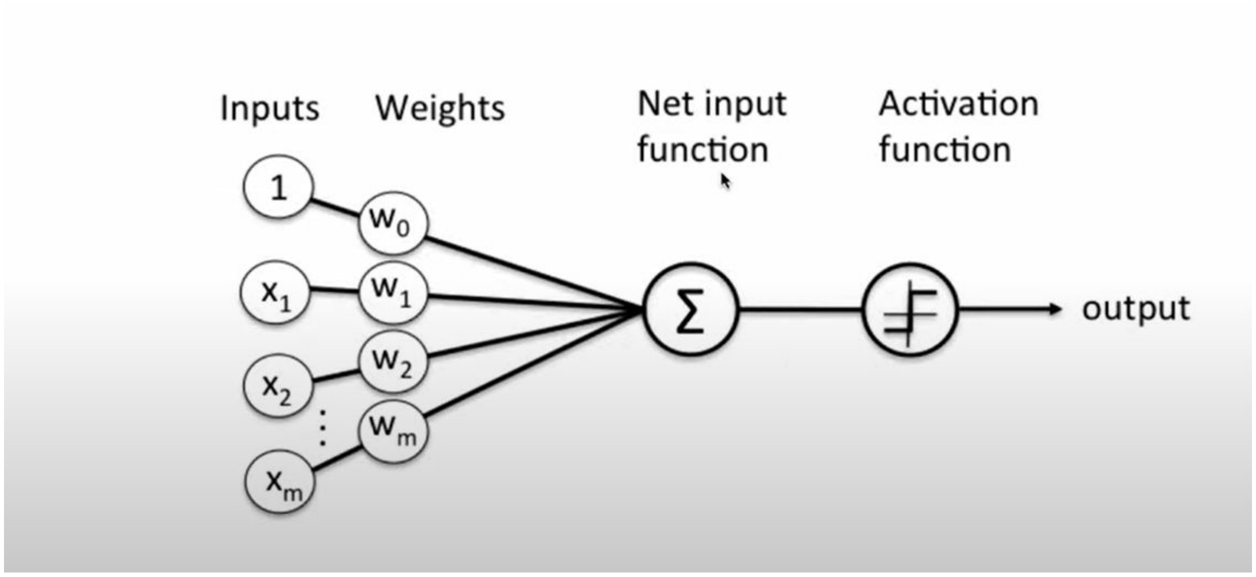
# Assignment-1C: Linear Perceptron

In this problem, you will implement a linear perceptron as discussed in class. You have two different datasets for this assignment on which you need to train and test your model independently. Create 70:30 train-test splits on both the datasets and train the model for a maximum of 106 iterations in case the model does not converge on the given dataset.

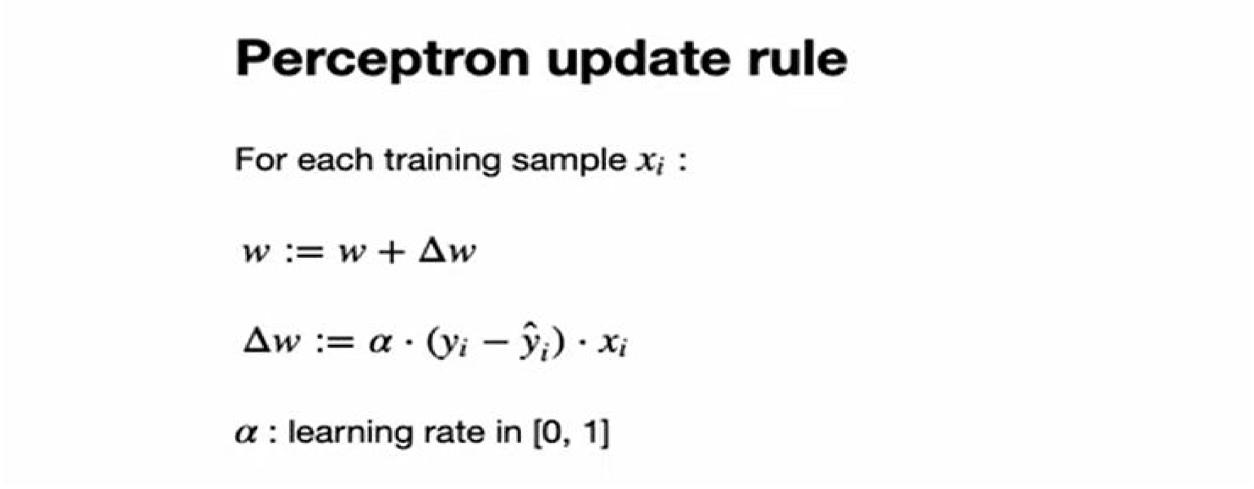
1)Mathematical representation:



Linear function will be
f(w,b) = wTx + b

We will be using unit function as activating function, so if the input reaches threshold it will be 1 , otherwise it will be zero

We will update the weights in case of misclassification



**Perceptron update rule**

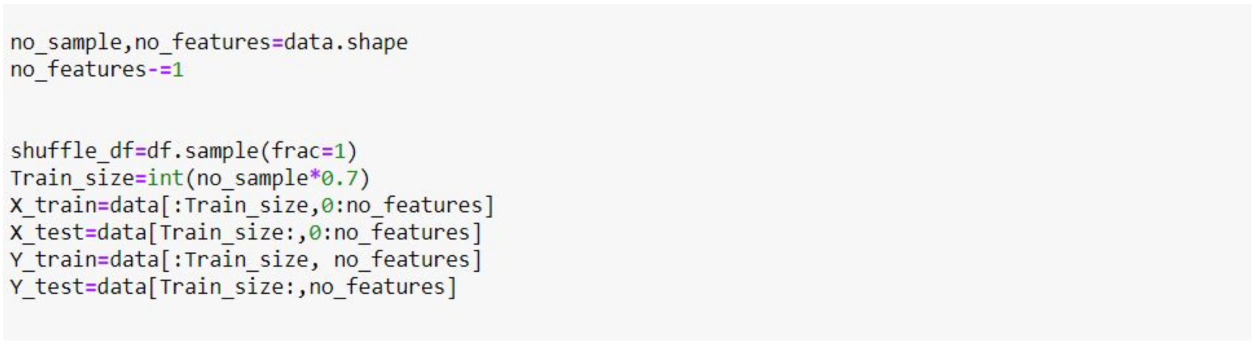
For each training sample $x_i$ :

$$w := w + \Delta w$$

$$\Delta w := \alpha \cdot (y_i - \hat{y}_i) \cdot x_i$$

$\alpha$ : learning rate in [0, 1]

Implementation:
Since we want a test train split in 70:30 ratio we divide the dataset accordingly

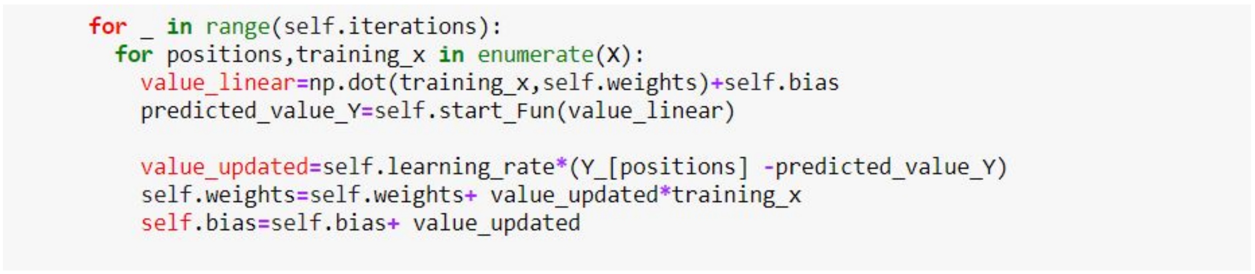


```
no_sample,no_features=data.shape
no_features-=1


shuffle_df=df.sample(frac=1)
Train_size=int(no_sample*0.7)
X_train=data[:Train_size,0:no_features]
X_test=data[Train_size:,0:no_features]
Y_train=data[:Train_size, no_features]
Y_test=data[Train_size:,no_features]
```

We use the unit function as activating function

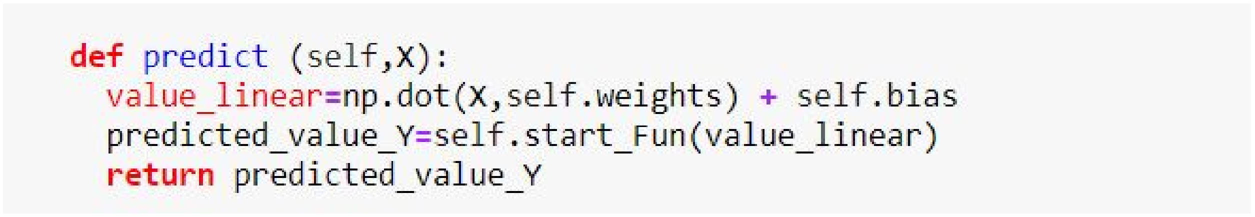
The loop will keep on updating the value of the weights for the given no of iterations for each of the training points. The change in the value depends on the learning rate and the difference between the predicted value and the linear value.



```
for _ in range(self.iterations):
  for positions,training_x in enumerate(X):
    value_linear=np.dot(training_x,self.weights)+self.bias
    predicted_value_Y=self.start_Fun(value_linear)

    value_updated=self.learning_rate*(Y_[positions] -predicted_value_Y)
    self.weights=self.weights+ value_updated*training_x
    self.bias=self.bias+ value_updated
```

We use the Predict method to get an approximation of our output
q'=k(f(w,b)) where f(w,b) is the linear function and k(x) is the activation function



```
def predict (self,X):
  value_linear=np.dot(X,self.weights) + self.bias
  predicted_value_Y=self.start_Fun(value_linear)
  return predicted_value_Y
```

| | |
|---|---|
| 2) Dataset- dataset_LP_1 | Accuracy- 98.8 percent |
| Dataset- dataset_LP_2 | Accuracy- 99.67 percent |

3)The second dataset is more linearly separable as the misclassifications are the least among the two datasets

4) Limitations:
 Perceptron is not the best algorithm to use in case of non-linearly separable problems..
a)The perceptron model cannot approximate the functions for non-linearly separable datasets. This limitation is apparent in case of functions like XOR which cannot be modelled by any single perceptron
b) Another shortcoming of Perceptron is the time required for large datasets over large number of iterations