What is Supervised learning??

* Supervised learning, in the context of artificial intelligence (AI) and machine learning, is a type of system in which both input and desired output data are provided. Input and output data are labelled for classification to provide a learning basis for future data processing.
* Supervised machine learning systems provide the learning algorithms with known quantities to support future judgments. Chatbots, self-driving cars, facial recognition programs, expert systems and robots are among the systems that may use either supervised or unsupervised learning. Supervised learning systems are mostly associated with retrieval-based AI but they may also be capable of using a generative learning model.
* Training data for supervised learning includes a set of examples with paired input subjects and desired output (which is also referred to as the supervisory signal).
* In supervised learning for image processing, for example, an AI system might be provided with labelled pictures of vehicles in categories such as cars and trucks. After a sufficient amount of observation, the system should be able to distinguish between and categorize unlabeled images, at which time training can be said to be complete.

Advantages:

* Supervised learning models have some advantages over the unsupervised approach, but they also have limitations. The systems are more likely to make judgments that humans can relate to, for example, because humans have provided the basis for decisions. However, in the case of a retrieval-based method, supervised learning systems have trouble dealing with new information. If a system with categories for cars and trucks is presented with a bicycle, for example, it would have to be incorrectly lumped in one category or the other. If the AI system was generative, however, it may not know what the bicycle is but would be able to recognize it as belonging to a separate category.

Supervised learning Algorithm:(Single layer Perceptrons):

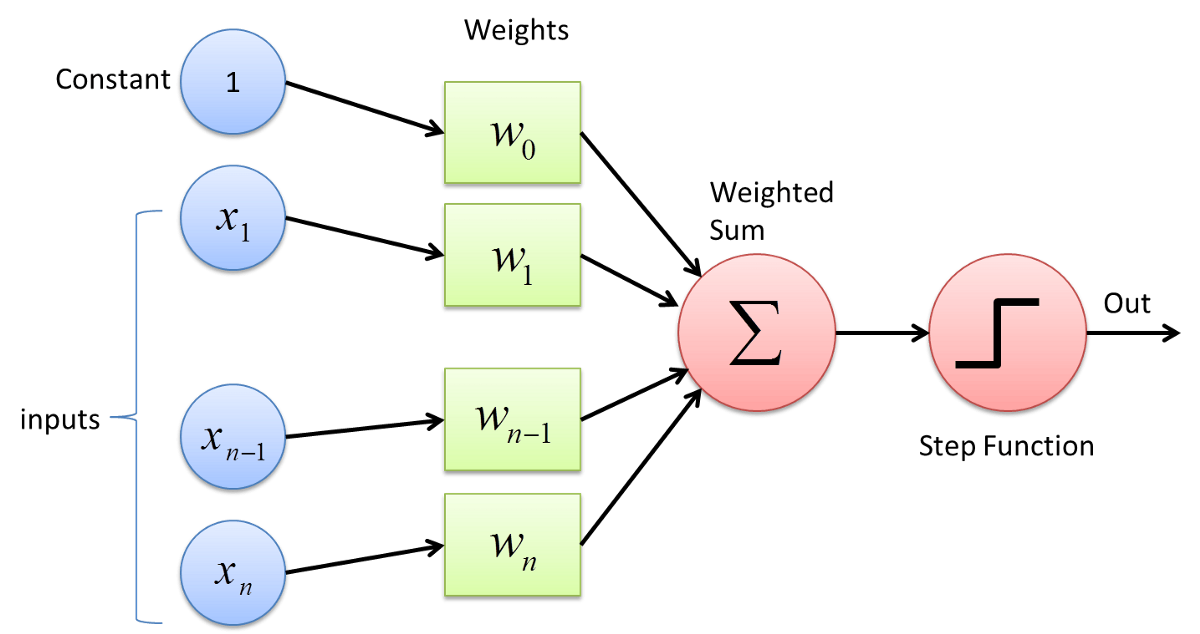
**What is Single layer Perceptrons??**

Perceptron is a single layer neural network and a multi-layer perceptron is called Neural Networks.

Perceptron is a linear classifier (binary). Also, it is used in supervised learning. It helps to classify the given input data

The perceptron consists of 4 parts .

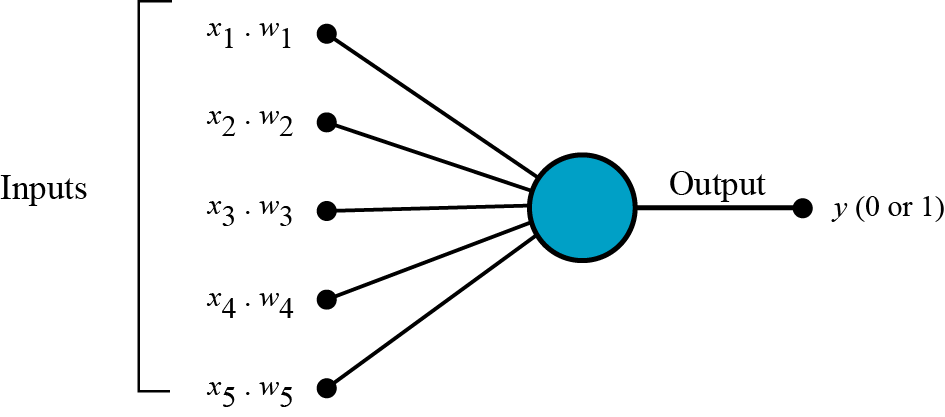
1. Input values or One input layer
2. Weights and Bias
3. Net sum
4. Activation Function



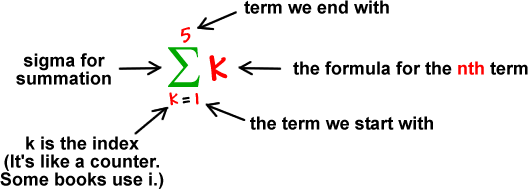
#### **how does it work?**

The perceptron works on these simple steps

1. All the inputs **x** are multiplied with their weights **w**.Let’s call it **k.**

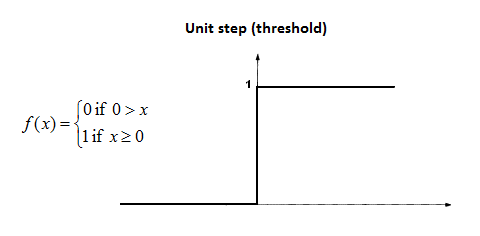


b. **Add** all the multiplied values and call them **Weighted Sum.**



c. **Apply** that weighted sum to the correct [activation function.](https://medium.com/towards-data-science/activation-functions-neural-networks-1cbd9f8d91d6" \t "https://towardsdatascience.com/_blank)

For Example : Unit Step Activation Function.



#### **Where we use Perceptron?**

*Perceptron is usually used to classify the data into two parts.*

**Learning Algorithm:**

1. Initialize weights at random
2. For each training pair/pattern ( x, ytarget )

- Compute output y

- Compute error, δ=(ytarget – y)

- Use the error to update weights as follows:

∆w = w – wold = η \* δ\*x or wnew = wold + η \* δ\*x

where η is called the learning rate or step size and it determines how smoothly the learning process is taking place.

1. Repeat 2 until convergence (i.e. error δ is zero)
2. The Perceptron Learning Rule is then given by wnew = wold + η \* δ\*x where δ=(ytarget – y)

-: **IMPLEMENTED CODE** :-

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import random

from sklearn.metrics import confusion\_matrix,accuracy\_score,precision\_score,recall\_score,f1\_score

from mlxtend.plotting import plot\_decision\_regions

from sklearn.decomposition import PCA

w = []

data = pd.read\_csv('slp.csv')

b = []

y\_acc = []

x\_epoc = []

data = data.iloc[:,[2,3,4]]

for i in range(len(data.index)):

b.append(-1)

data.insert(loc=0,column='bais',value=b)

def initilize\_weights(data):

no\_of\_weigths = len(data.columns)

for i in range( no\_of\_weigths-1):

w.append( round(random.uniform(-0.05,0.05),2))

def accuracy(predvalues,target,i):

print('Accuracy score: {}'.format(accuracy\_score(target, predvalues)))

#for early stop

y\_acc.append(accuracy\_score(target, predvalues))

x\_epoc.append(i)

plt.subplot(2,1,1)

plt.plot(x\_epoc,y\_acc)

plt.title("early stop")

plt.xlabel("epocs")

plt.ylabel("accuracy")

# print('Precision score: {}'.format(precision\_score(target, predvalues)))

# print('Recall score: {}'.format(recall\_score(target, predvalues)))

#print('F1 score: {}'.format(f1\_score(target, predvalues)))

def update\_weigths(w,n,ypred,t,dataarray,j):

for i in range(len(w)):

w[i] = w[i]- (n\*(ypred-t)\*dataarray[j][i])

def confusionmatrix(predvalues,t):

print("confusion matrix")

cm = confusion\_matrix(t,predvalues)

print(cm)

print("\n")

def plot\_decision(X,predvalues,target):

#dimensionality reduction for plotting

pca = PCA(n\_components=1)

principalComponents = pca.fit\_transform(X)

principalDf = pd.DataFrame(data = principalComponents

, columns = ['principal component 1'])

print(principalDf)

plt.subplot(2,1,2)

from matplotlib.colors import ListedColormap

X\_set, y\_set = X,target

X1, X2 = np.meshgrid(np.arange(X\_set[:, 0].min() - 1, X\_set[:, 0].max() + 1,step=0.1),

np.arange(X\_set[:, 1].min() - 1, X\_set[:, 1].max() + 1,step=0.1))

plt.contourf(X1, X2, slp.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

predvalues = []

class slp:

def train\_model(epocs,learning\_rate):

y=0

target = np.array(data['Purchased'])

dataarray= np.array(data)

for i in range(epocs):

print("starting with epocs ---->"+" "+str(i+1) )

print("\n")

predvalues = []

for k in range(len(data.index)):

for j in range( len(data.columns)-1):

y = y + (dataarray[k][j]\*w[j])

#print("activation value:" +" "+str(y))

if y>0:

ypred = 1

else:

ypred = 0

predvalues.append(ypred)

if ypred != target[k]:

update\_weigths(w,learning\_rate,ypred,target[k],dataarray,k)

print("updated weightd are :")

print(w)

#i is the nth epoc

print("predicted values for epoc"+" "+ str(i+1)+" is:")

print(predvalues)

accuracy(predvalues,target,i)

confusionmatrix(predvalues,target)

#plot\_decision(dataarray[:,1:3],predvalues,target)

def predict(x):

predictedvalues = []

for i in x:

y=0

y = y+(-1\*w[0])

for j in range(len(i)):

y = y + (i[j]\*w[j+1])

if y>0:

predictedvalues.append(1)

else:

predictedvalues.append(0)

predictedvalues = np.array(predictedvalues)

return predictedvalues

initilize\_weights(data)

slp.train\_model(epocs=13,learning\_rate=0.25)

#testing

ypredicted = slp.predict(np.array([[27,84000]]))

print(ypredicted)

**Result:**

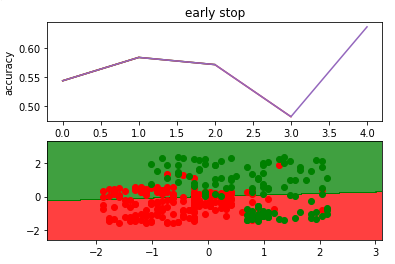
**For train data:**

After 5 epochs the accuracy is 65%

confusion matrix

[[198 11]

[105 6]]



**For test data:**

The accuracy is 78%

confusion matrix

[[32 16]

[ 2 30]]

