We are building a simple neural network by training one perceptron. We will see how to make predictions for all data points and also visualise the hypothesis generated by our algo. We will run this perceptron over a non linear dataset and will see the results.

Perceptron Implementation - Part II

- Make Predictions
- Visualise Decision Surface
- Linear vs Non-linear Classification

```
def getPredictions(x_Test, weights, labels=True):

    if x_Test.shape[1] != weights.shape[0]:
        ones = np.ones((x_Test.shape[0], 1))
        x_Test = np.hstack((ones, x_Test))

    probs = predict(x_Test, weights)

    if not labels:
        return probs

    else:
        labels = np.zeros(probs.shape)
        labels[probs >= 0.5] = 1
        return labels
```

if test data doesnot have column of 1's appended then we are going to add 0.

$(m \times n)$ $(n+1)$

$w_0, w_1, \ldots, w_n$

$X \cdot W$

$m \times (n+1)$ $(n+1) \times 1$

output of algo is going to be a vector

$$\begin{bmatrix} \\ \\ x \end{bmatrix} = \begin{bmatrix} \text{pred} \\ 0.8 \\ 0.7 \\ 0.6 \\ \vdots \end{bmatrix}$$

Vectorization is being used.

All the places where probability is >0.5 make label as 1

a = np.zeros((5,5))

a[2,3] = 10

print(a)

a[a>0] = 20

print(a)

Perceptron Implementation - Part II

- Make Predictions
- Visualise Decision Surface
- Linear vs Non-Linear Classification

```
def getPredictions(X_Test, weights, labels=True) :

    if X_Test.shape[1] != weights.shape[0] :
        ones = np.ones((X_Test.shape[0], 1))
        X_Test = np.hstack((ones, X_Test))

    probs = predict(X_Test, weights)

    if not labels :
        return probs
    else :
        labels = np.zeros(probs.shape)
        labels[probs >= 0.5] = 1
        return labels
```

```
a = np.zeros((5,5))
a[2,3] = 10
print(a)

a[a>0] = 20
print(a)
```

```
[[ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0. 10.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]]
[[ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0. 20.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]]
```

weights

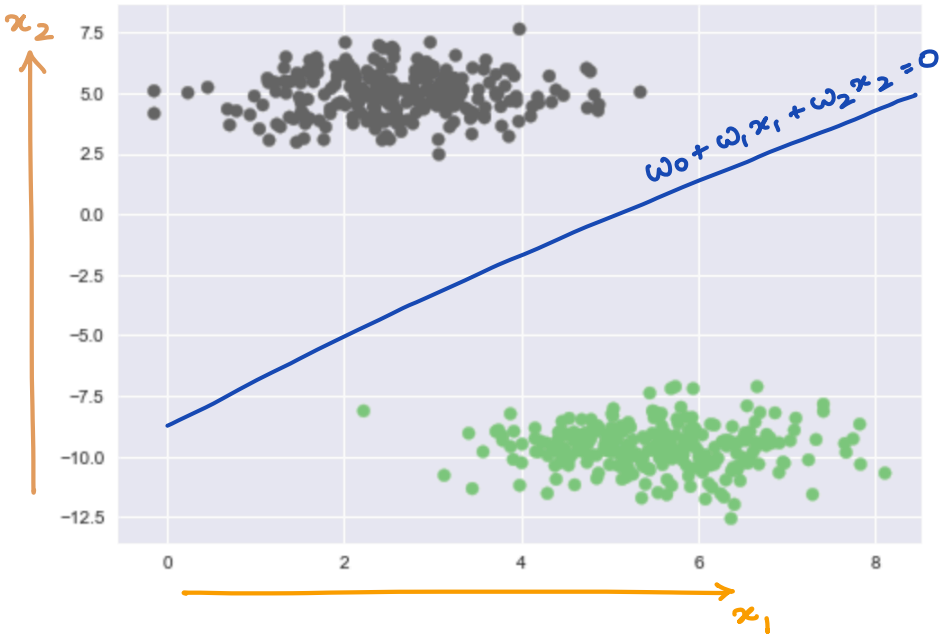
plt.scatter(X[:,0], X[:,1], c=Y, cmap=plt.cm.Accent)

plt.show()

```
weights
```
w_0 w_1 w_2

```
array([ 0.0656045 , -0.19220497,  2.05528664])
```

```
plt.scatter(X[:,0], X[:,1], c=Y, cmap=plt.cm.Accent)
plt.show()
```



$$x_2 = -\frac{(w_0 + w_1 x_1)}{w_2}$$

If we plot x_2 v/s x_1 using this formula we will get the line.

```
x1 = np.linspace(-8, 2, 10)
x2 = -(weights[0] + weights[1] * x1) / weights[2]

plt.scatter(x[:,0], x[:,1], c=y, cmap=plt.cm.Accent)
plt.plot(x1, x2, c='red')
plt.show()
```

```
x1 = np.linspace(-1,10,10)
x2 = -(weights[0] + weights[1]*x1) / weights[2]
```

```
plt.scatter(X[:,0], X[:,1], c=Y, cmap=plt.cm.Accent)
plt.plot(x1, x2, c='red')
plt.show()
```

→ we are able to separate the data

Lets try with different value of random_state

X, Y = make_blobs (n_samples = 500, centers = 2, n_features = 2, random_state = 1)

```
X,Y = make_blobs(n_samples=500, centers=2, n_features=2, random_state=1)
print(X.shape, Y.shape)
```

```
x1 = np.linspace(-15,5,10)
x2 = -(weights[0] + weights[1]*x1) / weights[2]
```

```
plt.scatter(X[:,0], X[:,1], c=Y, cmap=plt.cm.Accent)
plt.plot(x1, x2, c='red')
plt.show()
```

find the accuracy

Y_ = getPredictions (X, weights, labels = True)

print (Y_)

```python
#Find the accuracy
Y_ = getPredictions(X, weights, labels=True)
print(Y_)
```

```
[1. 0. 1. 0. 1. 0. 0. 1. 0. 1. 1. 1. 1. 0. 0. 1. 0. 1. 0. 1. 1. 1. 0. 0.
 1. 1. 0. 0. 0. 1. 0. 0. 1. 1. 1. 1. 0. 0. 1. 0. 1. 1. 0. 1. 1. 1. 0. 1.
 0. 0. 0. 1. 1. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 1. 0.
 1. 1. 1. 0. 1. 1. 1. 0. 1. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 1. 0. 1.
 0. 1. 1. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 1.
 1. 0. 1. 1. 1. 0. 0. 1. 1. 0. 0. 0. 0. 1. 1. 1. 0. 1. 0. 0. 1. 0. 1. 0.
 0. 0. 1. 0. 1. 1. 1. 0. 0. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 1.
 0. 1. 0. 0. 0. 0. 0. 1. 0. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 0. 1. 0. 0. 1.
 1. 1. 0. 1. 0. 1. 0. 1. 0. 1. 1. 1. 1. 0. 1. 1. 0. 1. 0. 1. 1. 1. 1. 0.
 0. 0. 1. 1. 0. 1. 0. 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 1. 0. 1. 1. 0.
 1. 1. 1. 1. 1. 0. 1. 0. 1. 0. 0. 1. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0.
 1. 1. 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 1. 0. 1. 1. 0. 1. 0. 0. 1. 0.
 0. 0. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 0. 1. 1. 0. 1. 1. 0. 0. 1. 1.
 1. 0. 0. 1. 1. 0. 0. 1. 1. 0. 0. 1. 0. 0. 0. 1. 1. 1. 0. 0. 1. 1. 0. 0.
 1. 1. 0. 1. 1. 0. 0. 0. 0. 0. 0. 0. 1. 1. 1. 0. 0. 1. 1. 0. 0. 0. 1. 0.
 0. 1. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 0. 0. 0. 0. 1. 0.
 1. 1. 0. 0. 0. 1. 1. 0. 1. 0. 0. 0. 0. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 0.
 0. 1. 0. 1. 0. 0. 1. 0. 0. 1. 0. 0. 1. 0. 1. 1. 0. 0. 0. 1. 0. 1. 1. 0.
 1. 1. 0. 1. 1. 1. 0. 0. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.
 1. 1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 1. 0. 1. 0. 0. 1. 1. 0. 1. 0. 1. 1. 0.
 0. 1. 0. 1. 1. 0. 0. 0. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 0. 1.]
```

```python
Y_ == Y
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
```

element wise comparison

500

```
training_acc = np.sum(Y_==Y)/Y.shape[0]
print(training_acc*100)
```

```
100.0
```

Training Accuracy is 100%
All predictions are correct

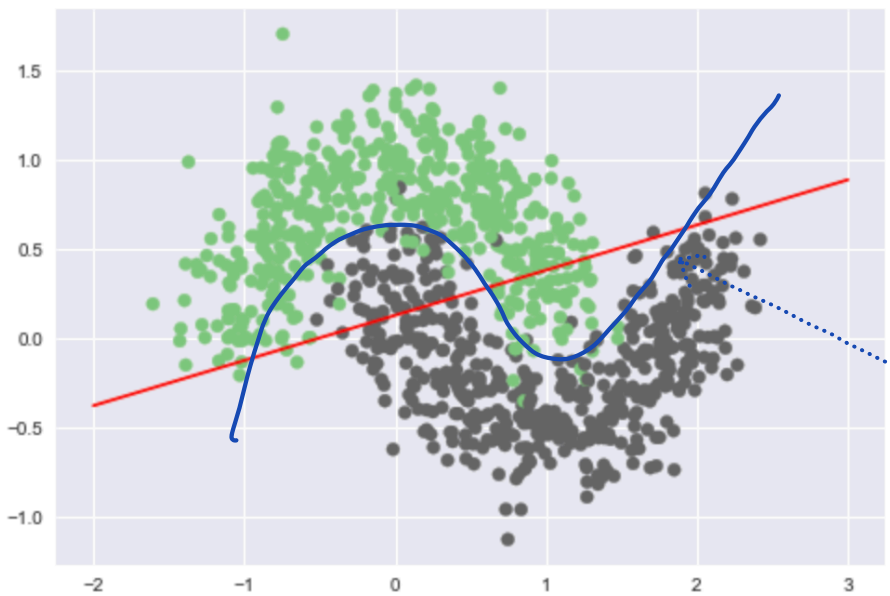
what if we use non-linear data ?

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs, make_moons
```

```
# X,Y = make_blobs(n_samples=500, centers=2, n_features=2, random_state=1)
X,Y = make_moons(n_samples=1000, shuffle=True, noise=0.2, random_state=1)
print(X.shape, Y.shape)
```

```
x1 = np.linspace(-2,3,10)
x2 = -(weights[0] + weights[1]*x1) / weights[2]
```

```
plt.scatter(X[:,0], X[:,1], c=Y, cmap=plt.cm.Accent)
plt.plot(x1, x2, c='red')
plt.show()
```



Now, not able to learn a very good boundary bcz data was not linear.

We need to learn some other hypothesis which should be able to separate data like this.
This kind of boundary can be learnt by MLP (Multi layer Perceptron)

```
training_acc = np.sum(Y_==Y)/Y.shape[0]
print(training_acc*100)
```

```
86.8
```

now accuracy is only 86.8%
Some points are not classified correctly.