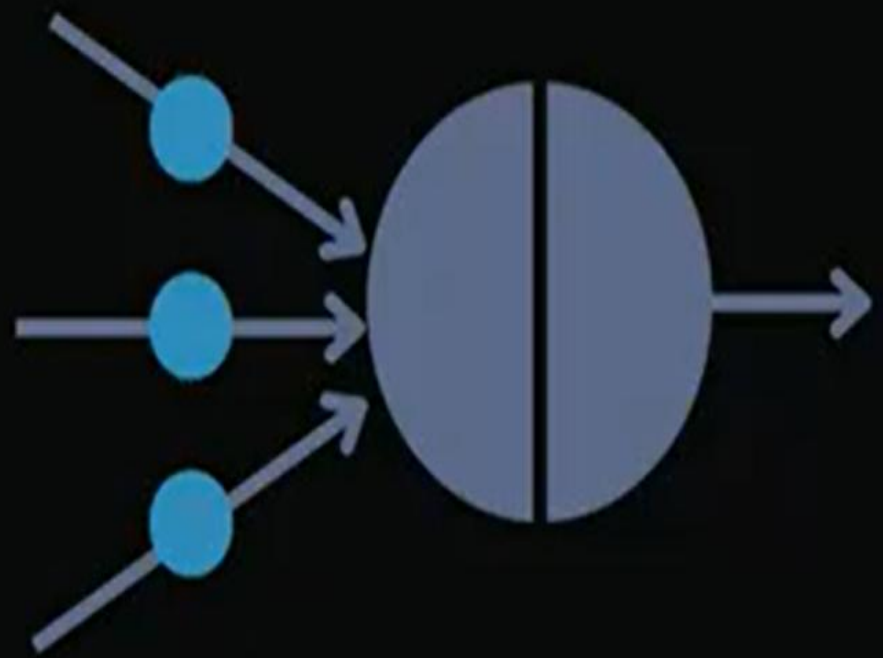
Artificial Neural Network

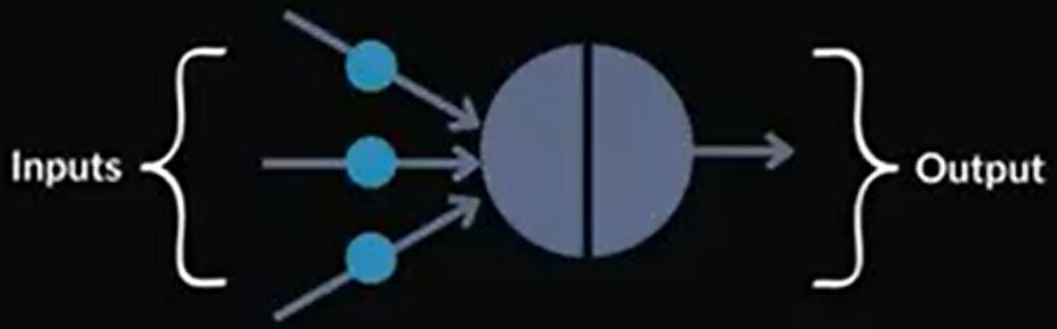
in

Python

Perceptron



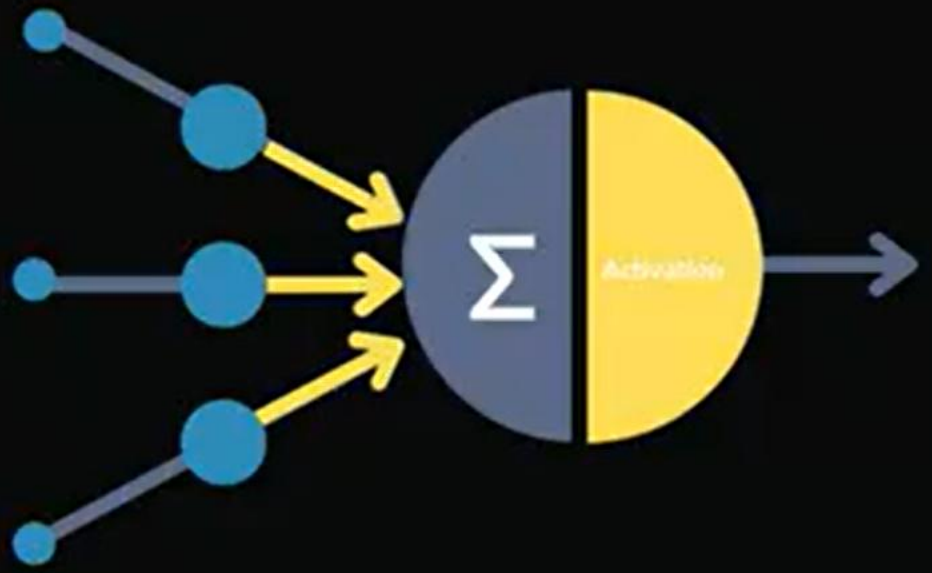
How Perceptron works?



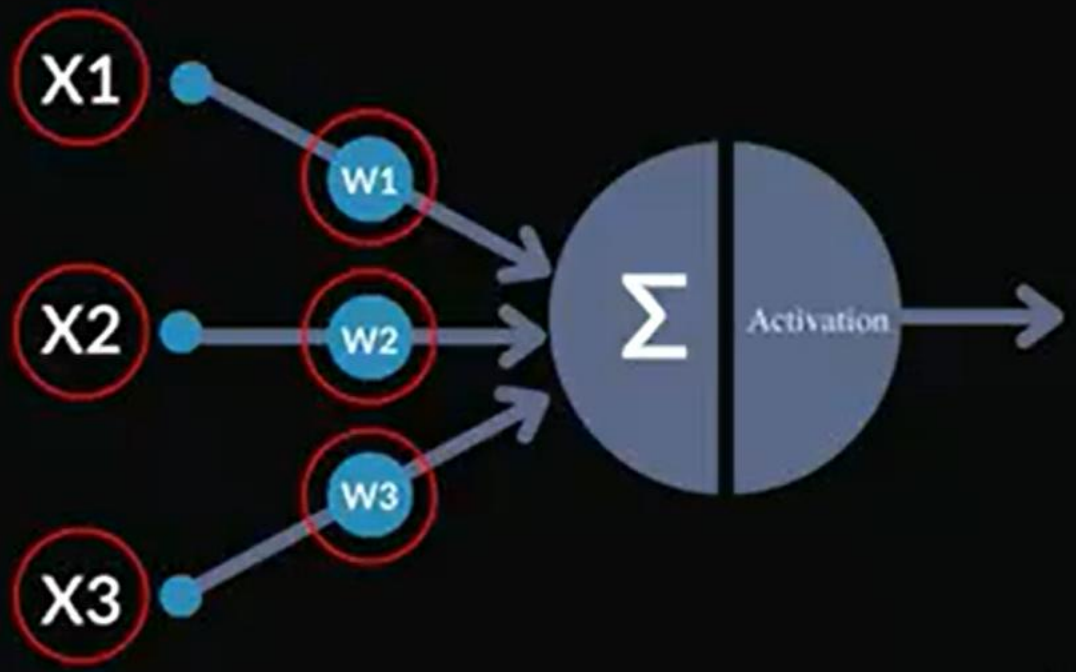
How Perceptron works?

How Perceptron works?

activation(X1.W1 + X2.W2 + X3.W3)



How Perceptron works?



Input Vector

$$\begin{bmatrix} X1 \\ X2 \\ X3 \\ \vdots \\ Xn \end{bmatrix}$$

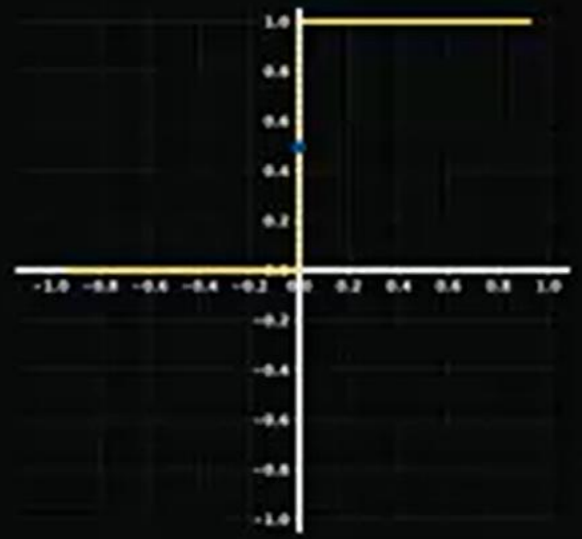
Weight Vector

$$\begin{bmatrix} W1 & W2 & W3 & \cdots & Wn \end{bmatrix}$$

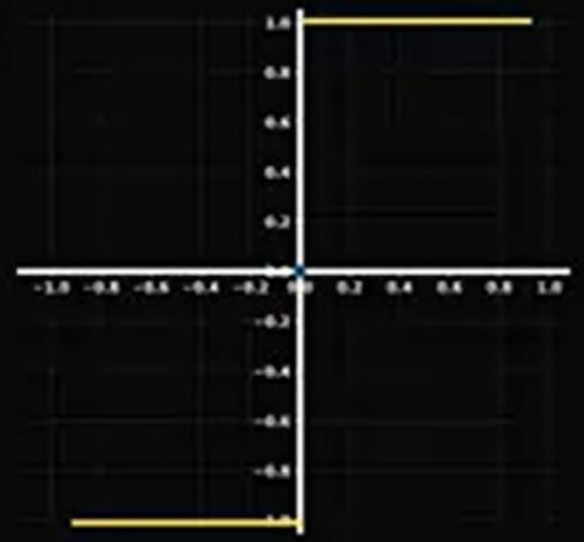
$$Z = X1W1 + X2W2 + X3W3 \cdots XnWn$$

$$\text{Output} = \text{Activation}(Z)$$

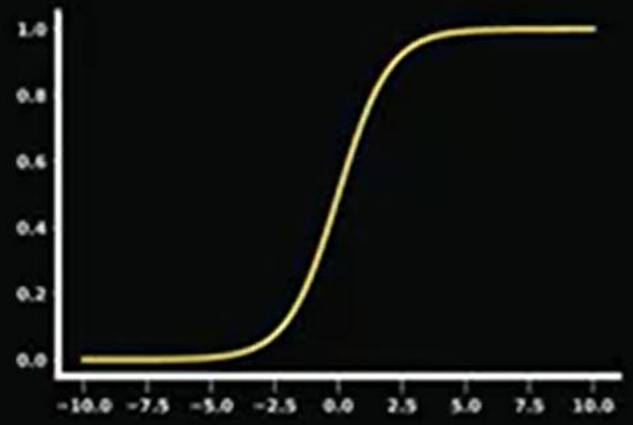
ReLU



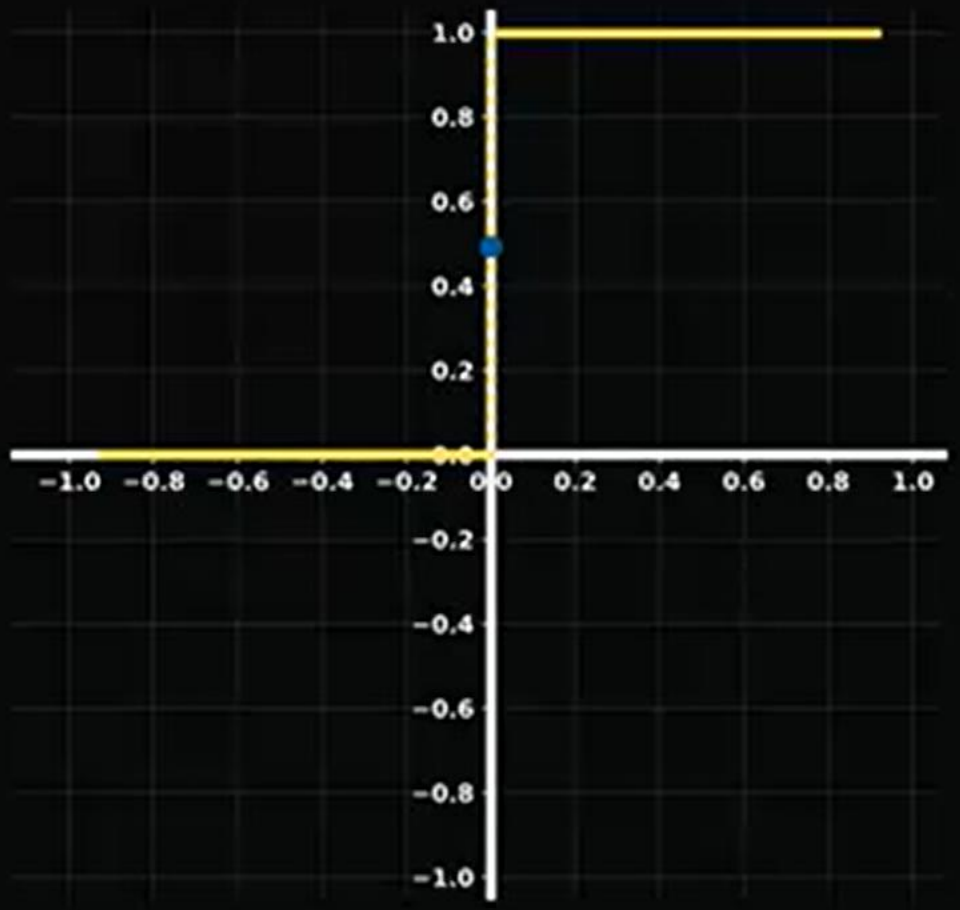
Heaviside



Sign



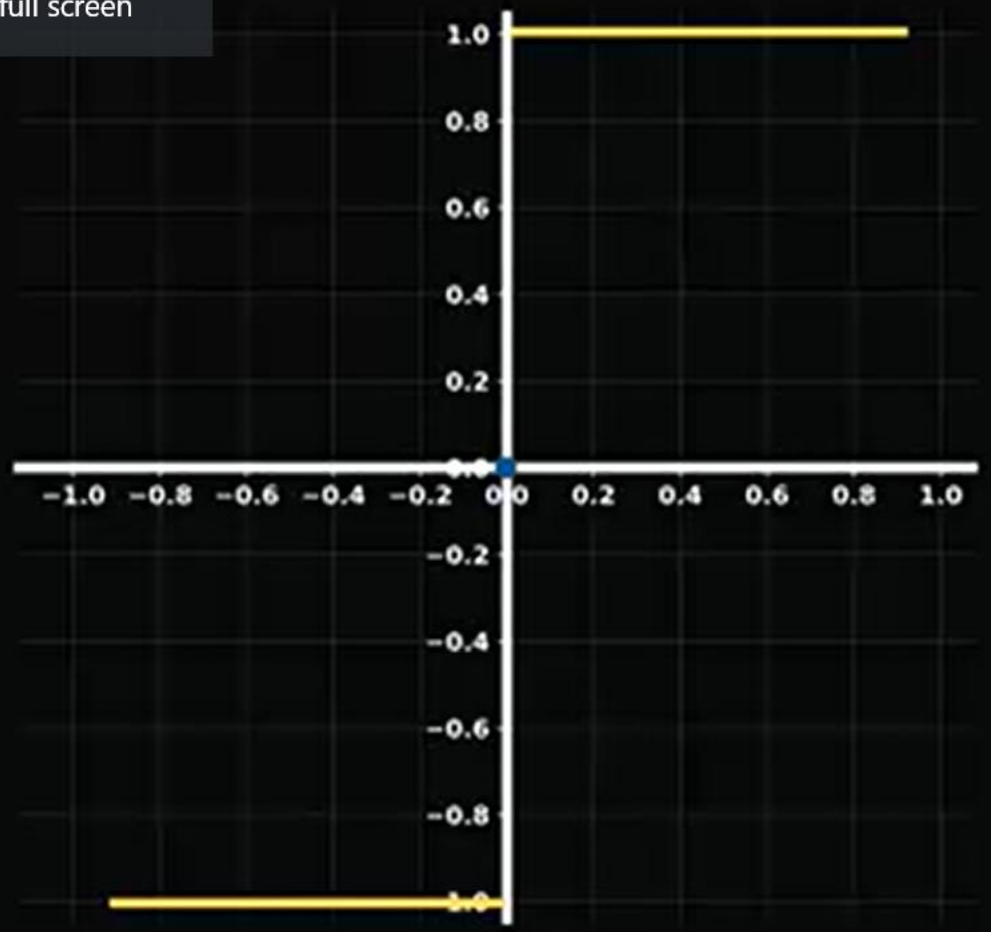
Sigmoid



$$\text{heaviside}(z) = \begin{cases} 0 \text{ if } z < 0 \\ 1 \text{ if } z >= 0 \end{cases}$$

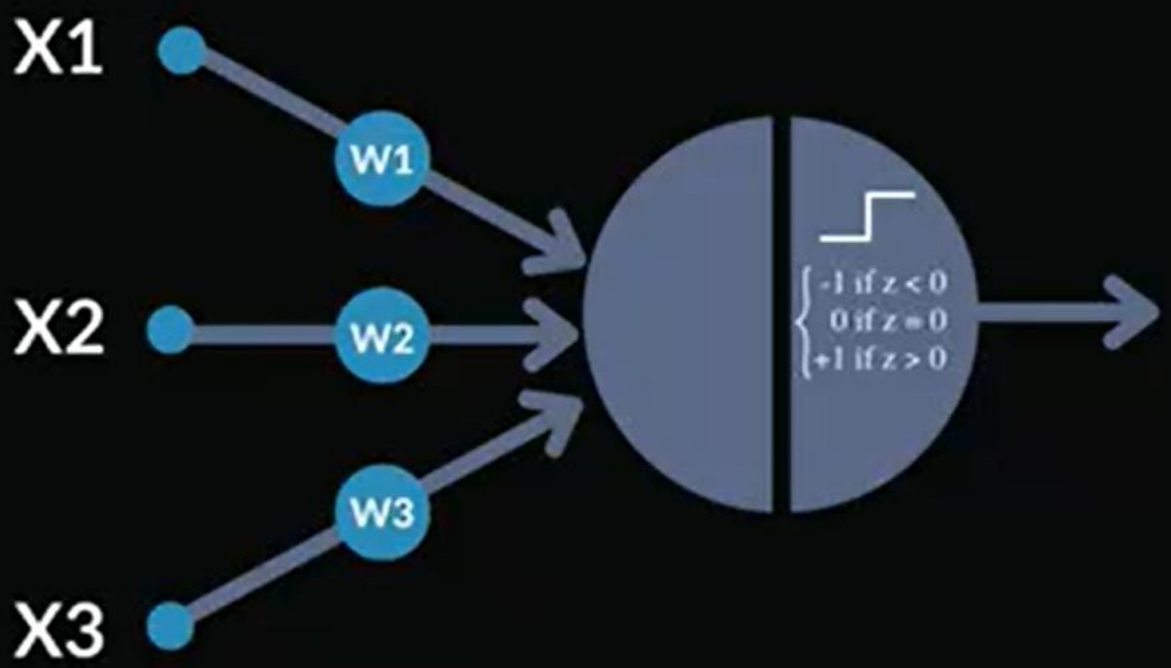
Heaviside

$$\text{sign}(z) = \begin{cases} -1 \text{ if } z < 0 \\ 0 \text{ if } z = 0 \\ +1 \text{ if } z > 0 \end{cases}$$

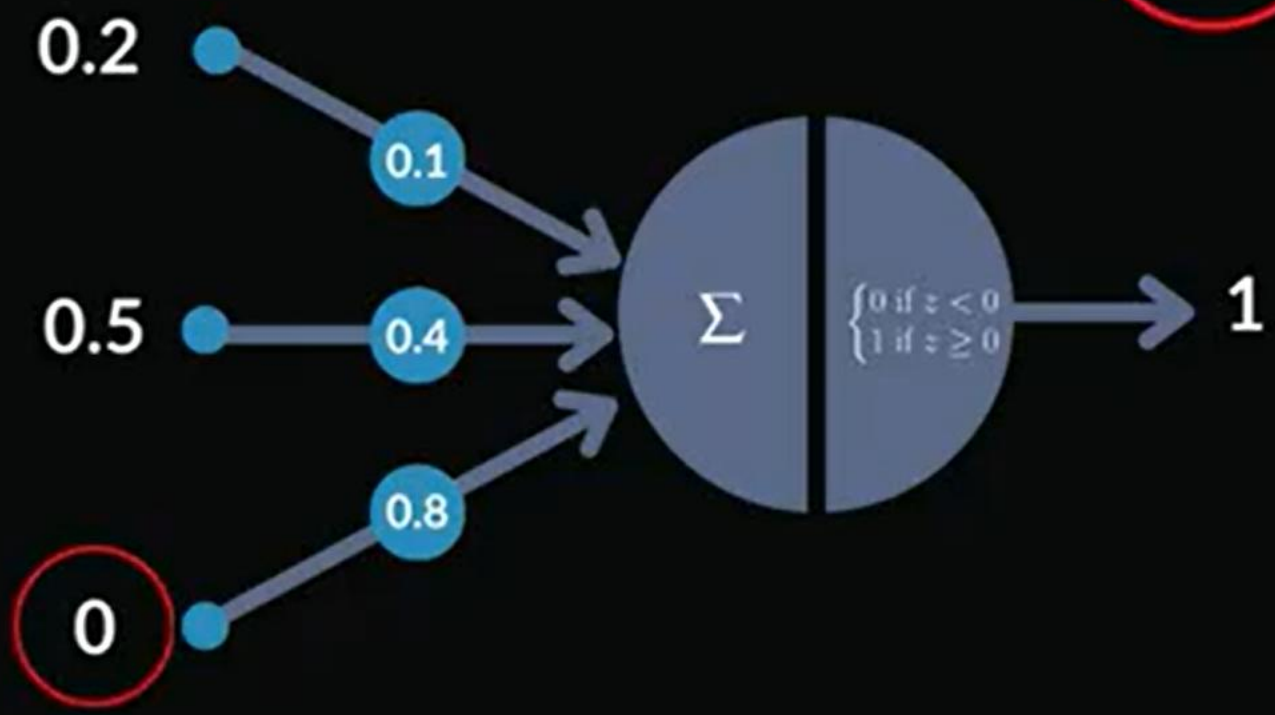


Sign

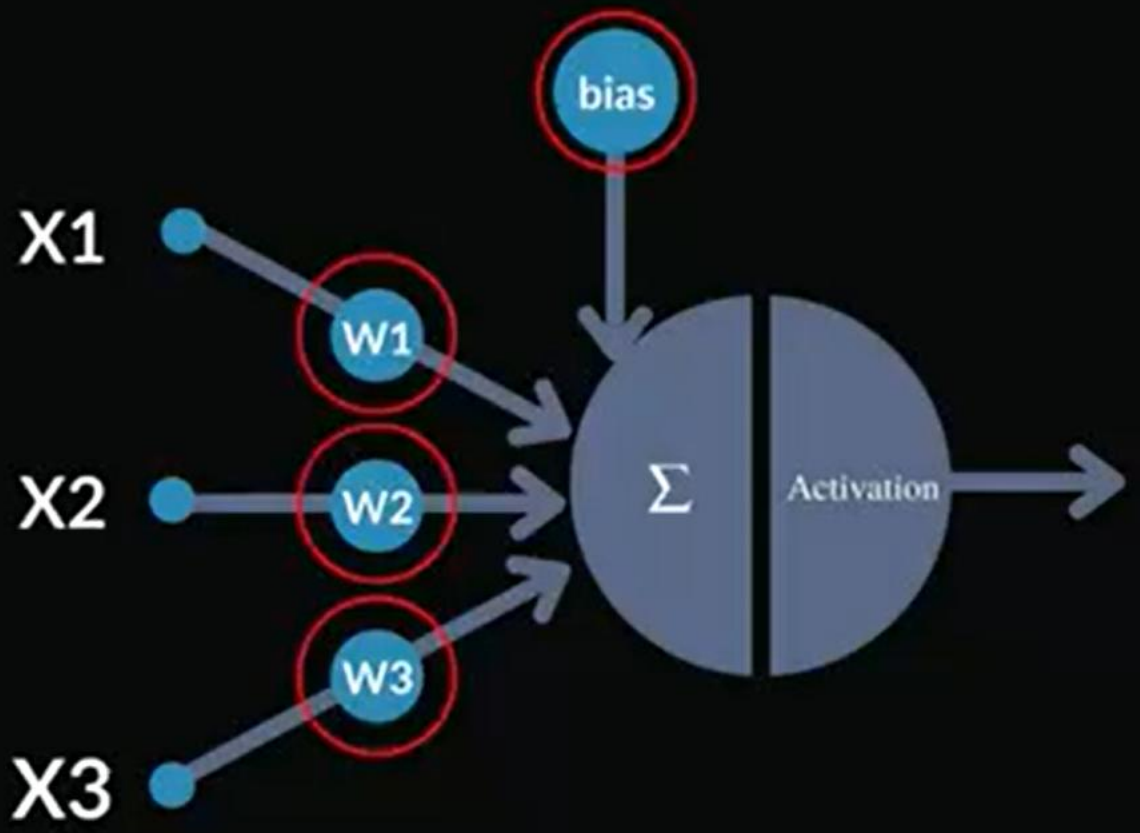
Image pixels as input



$$z = 0.2 * 0.1 + 0.5 * 0.4 + 0 * 0.8 = 0.22 > 0$$



Learning in Perceptron

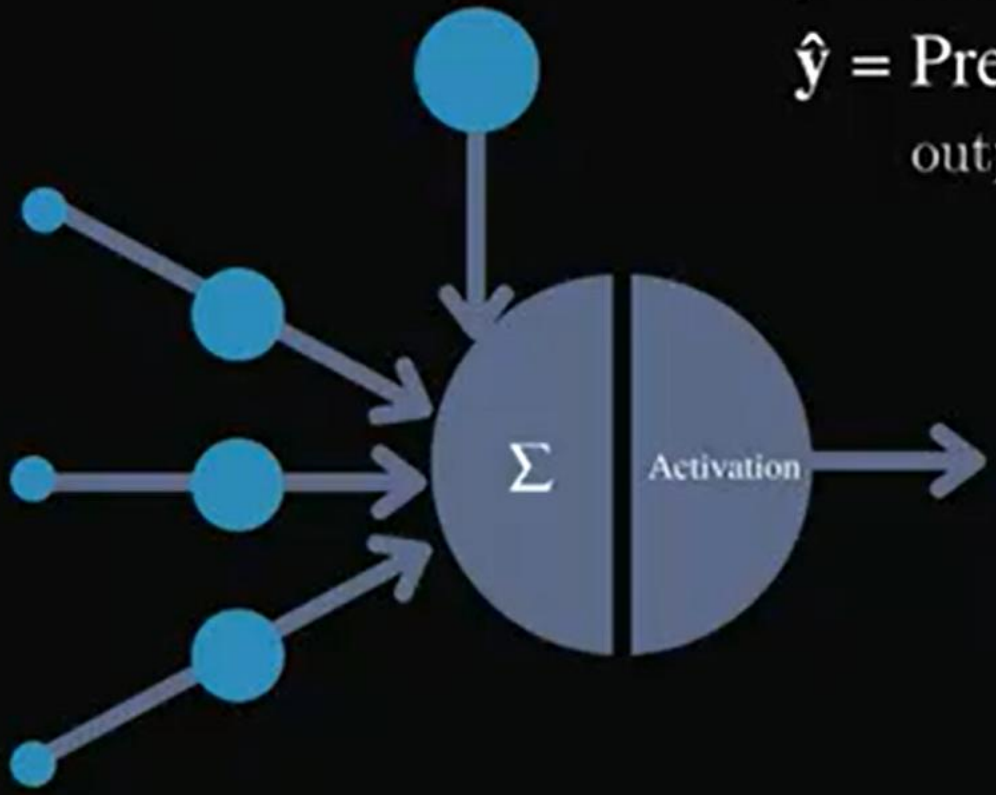


Learning in Perceptron

y = Actual value

$\hat{y}$ = Predicted value

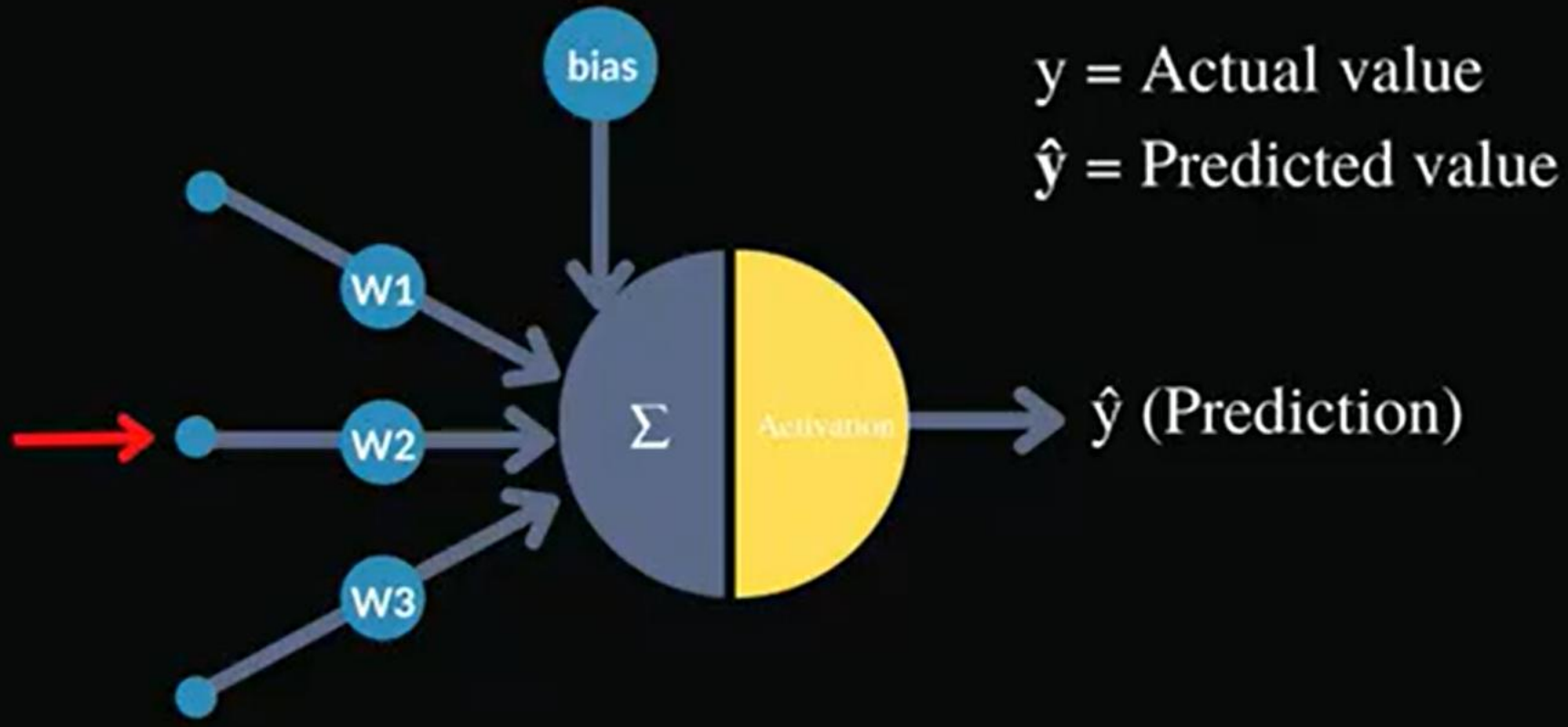
$$\text{output} = \Phi(WX + b)$$



$\hat{y}$ (Prediction)

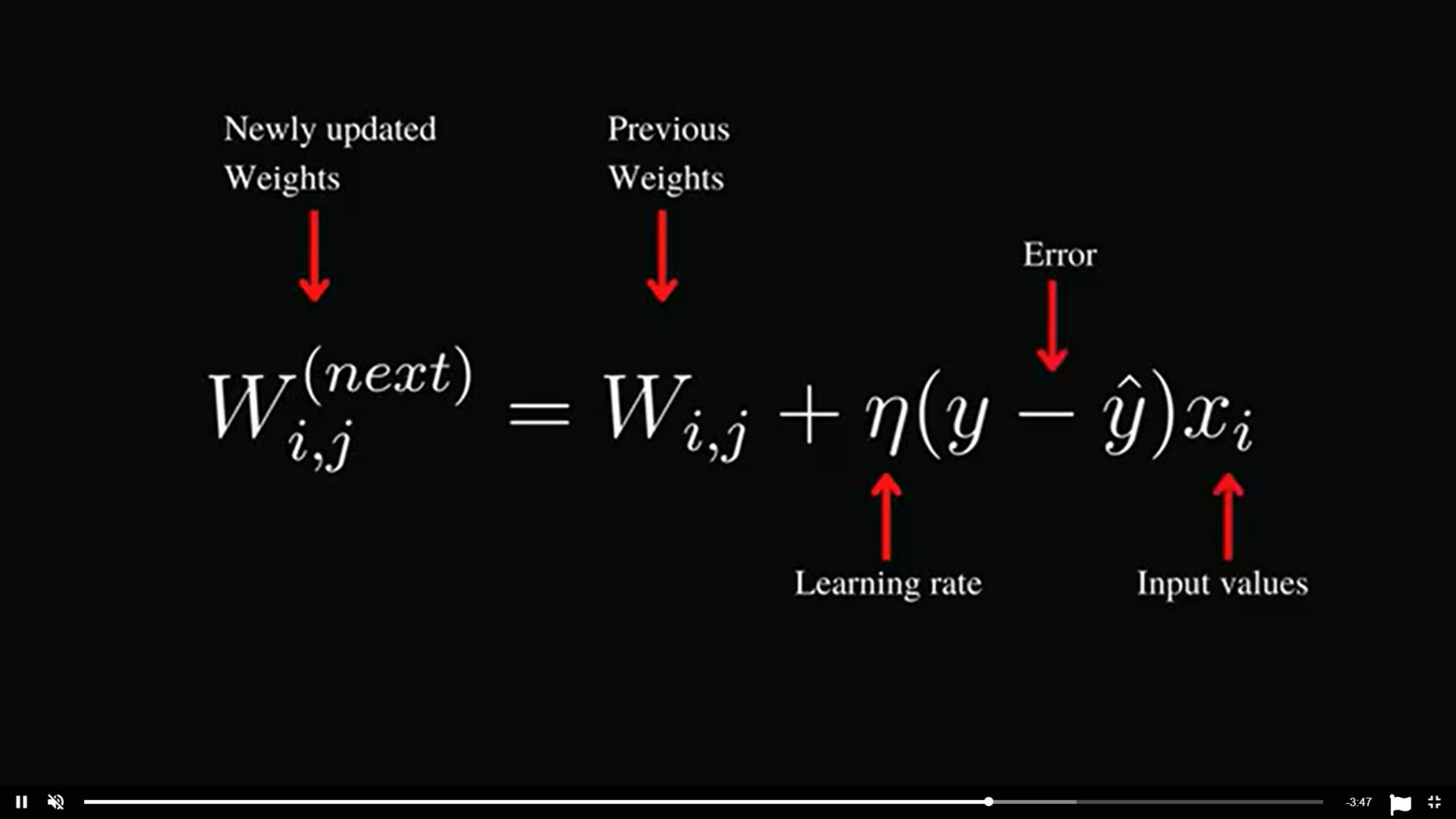
Error = $(y - \hat{y})$

Learning in Perceptron



$$\hat{y} = \Phi(WX + b)$$

$$W_{i,j}^{(next)} = W_{i,j} + \eta(y - \hat{y})x_i$$


Newly updated
Weights
Previous
Weights
Error
$W_{i,j}^{(next)} = W_{i,j} + \eta(y - \hat{y})x_i$
Learning rate
Input values

$$W_{i,j}^{(next)} = W_{i,j} + \eta(y - \hat{y})x_i$$

$$b_{i,j}^{(next)} = b_{i,j} + \eta(y - \hat{y})$$

Implementing Perceptron in Python

```
import numpy as np

class Perceptron:

    def __init__(self, learning_rate, epochs):
        self.weights = None
        self.bias = None
        self.learning_rate = learning_rate
        self.epochs = epochs
```

```python
# heaviside activation function
def activation(self, z):
    return np.heaviside(z, 0) # haviside(z) heaviside -> activation
```

```
    def fit(self, X, y):
        n_features = X.shape[1]

        # Initializing weights and bias
        self.weights = np.zeros((n_features))
        self.bias = 0

        # Iterating until the number of epochs
        for epoch in range(self.epochs):

            # Traversing through the entire training set
            for i in range(len(X)):
                z = np.dot(X, self.weights) + self.bias # Finding the dot product and adding the bi
                y_pred = self.activation(z) # Passing through an activation function

                #Updating weights and bias
                self.weights = self.weights + self.learning_rate * (y[i] - y_pred[i]) * X[i]
                self.bias = self.bias + self.learning_rate * (y[i] - y_pred[i])

        return self.weights, self.bias
```

```
def predict(self, X):
    z = np.dot(X, self.weights) + self.bias
    return self.activation(z)
```

Classifying Iris dataset using Perceptron

Loading the dataset

```
from sklearn.datasets import load_iris

iris = load_iris()
```

Splitting the dataset

```
from sklearn.model_selection import train_test_split

X = iris.data[:, (0, 1)] # petal length, petal width
y = (iris.target == 0).astype(np.int)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=42)
```

Training and making predictions

Alright, now let's train our Perceptron algorithm,

```
perceptron = Perceptron(0.001, 100)

perceptron.fit(X_train, y_train)

pred = perceptron.predict(X_test)
```

Classifying Iris dataset using Scikit-learn Perceptron class

```
from sklearn.linear_model import Perceptron


sk_perceptron = Perceptron()
sk_perceptron.fit(X_train, y_train)
sk_perceptron_pred = sk_perceptron.predict(X_test)

# Accuracy

accuracy_score(sk_perceptron_pred, y_test)

-----

0.88
```