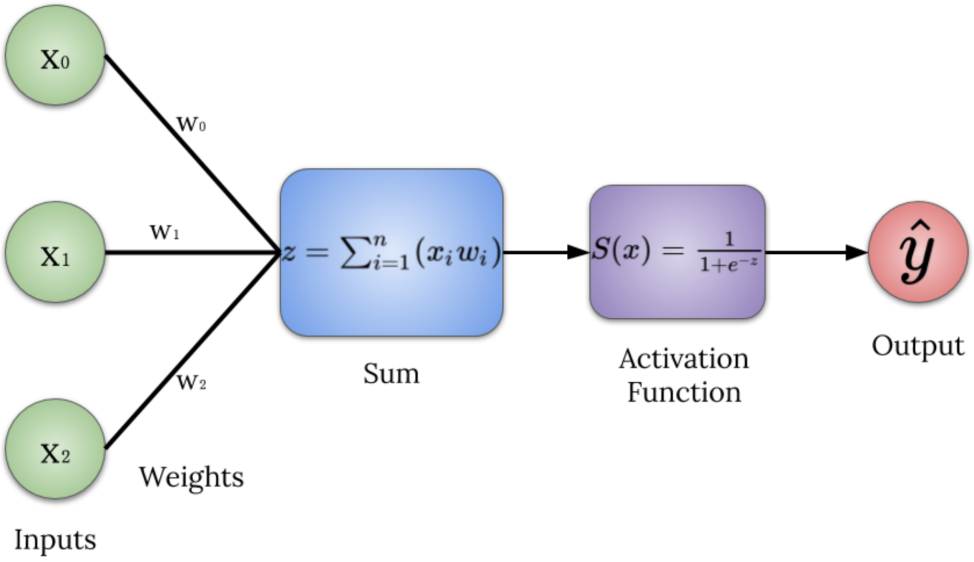
In [ ]:

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

#Create the DataFrame

#Input values: x1 = 0.05, x2 = 0.10
#Output values: y1 = 0.01, y2 = 0.99

df=pd.DataFrame([[0.05, 0.1, 0.01, 0.99]], columns=['x1', 'x2', 'y1', 'y2'])
df

target=df.iloc[:, 2:]
target=np.array(target)
print('Actual output:',target)

inputs=df.iloc[:, :2]
inputs=np.array(inputs)
print('Input values:', inputs)
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Build the Model
model = Sequential()
model.add(Dense(units = 2, activation = 'sigmoid', use_bias=True, bias_initializer="one
s", weights =[np.array([[0.15, 0.20], [0.25, 0.30]]), np.array([0.40, 0.35])]))
model.add(Dense(units =2, activation = 'sigmoid', use_bias=True, bias_initializer="one
s", weights =[np.array([[0.40, 0.45], [0.50, 0.55]]), np.array([0.25, 0.60])]))

# Compiling
model.compile(optimizer = 'SGD', loss ='mean_squared_error')
# observe the effects of epochs ,10,100,1000
classifier = model.fit(inputs, target, epochs=10)
```




| Input | | | Output |
|---|---|---|---|
| 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 |

# A one layer hidden Layer network training using BP without using Keras

# NN training using BP

# Forward propagation

is when data goes into the neural network and out pops a prediction. It is straightforward part, that involves matrix multiplications, add the biases at each stage, calculate the outputs through an activation function, and get to our final result.

# Backpropagation :

The process of adjusting the weights by looking at the difference between prediction and the actual result. The back propagation algorithm uses gradient descent method to minimize the error function with respect to modified network weights. A learning rate or a step size ranging from 0 to 1 is usually specified which determines the magnitude of the weights changes. Error = difference between actual & desired output Change weight relative to error size

# Calculate output layer error , then propagate back to previous layer

# Update the weights in the back-propagation network propagating backward the errors associated with output neurons.

(a) Calculate the error gradient for the neurons in the output layer

Calculate the weight corrections:

$$\Delta w_{jk}(p) = \alpha \times y_j(p) \times \delta_k(p)$$

Update the weights at the output neurons:

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p)$$

In [8]:

```
import numpy as np # helps with the math
import matplotlib.pyplot as plt # to plot error during training

# input data
inputs = np.array([[0, 1, 0],
                   [0, 1, 1],
                   [0, 0, 0],
                   [1, 0, 0],
                   [1, 1, 1],
                   [1, 0, 1]])
# output data
outputs = np.array([[0], [0], [0], [1], [1], [1]])

# create NeuralNetwork class
class NeuralNetwork:

    # intialize variables in class
    def __init__(self, inputs, outputs):
        self.inputs  = inputs
        self.outputs = outputs
        # initialize weights as .50 for simplicity
        self.weights = np.array([[.50], [.50], [.50]])
        self.error_history = []
        self.epoch_list = []

    #activation function ==> S(x) = 1/1+e^(-x)
    def sigmoid(self, x, deriv=False):
        if deriv == True:
            return x * (1 - x)
        return 1 / (1 + np.exp(-x))

    # data will flow through the neural network.
    def feed_forward(self):
        self.hidden = self.sigmoid(np.dot(self.inputs, self.weights))

    # going backwards through the network to update weights
    def backpropagation(self):
        self.error  = self.outputs - self.hidden
        delta = self.error * self.sigmoid(self.hidden, deriv=True)
        self.weights += np.dot(self.inputs.T, delta)

    # train the neural net
    def train(self, epochs=25000):
        for epoch in range(epochs):
            # flow forward and produce an output
            self.feed_forward()
            # go back though the network to make corrections based on the output
            self.backpropagation()
            # keep track of the error history over each epoch
            self.error_history.append(np.average(np.abs(self.error)))
            self.epoch_list.append(epoch)

    # function to predict output on new and unseen input data
    def predict(self, new_input):
        prediction = self.sigmoid(np.dot(new_input, self.weights))
        return prediction

# create neural network
NN = NeuralNetwork(inputs, outputs)
```

```
# train neural network
NN.train()

# create two new examples to predict
example = np.array([[1, 1, 0]])
example_2 = np.array([[0, 1, 1]])
print (example)
# print the predictions for both examples
print('Predicted:',NN.predict(example), ' - Actual: ', example[0][0])
print('Predcited',NN.predict(example_2), ' - Actual: ', example_2[0][0])

# plot the error over the entire training duration
plt.figure(figsize=(15,5))
plt.plot(NN.epoch_list, NN.error_history)
plt.xlabel('Epoch')
plt.ylabel('Error')
plt.show()
```

```
[[1 1 0]]
Predicted: [[0.99089925]]  - Actual:  1
Predcited [[0.006409]]  - Actual:  0
```