import statements

In [2]:
```
import numpy as np
```

function defination

In [3]:
```
def sigmoid (x):
    return 1/(1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)
```

Input datasets

In [4]:
```
inputs = np.array([[0,0],[0,1],[1,0],[1,1]])
expected_output = np.array([[0],[1],[1],[0]])

epochs = 10000
lr = 0.1
inputLayerNeurons, hiddenLayerNeurons, outputLayerNeurons = 2,2,1
```

Random weights and bias initialization

In [5]:
```
hidden_weights = np.random.uniform(size=(inputLayerNeurons,hiddenLayerNe
urons))
hidden_bias =np.random.uniform(size=(1,hiddenLayerNeurons))
output_weights = np.random.uniform(size=(hiddenLayerNeurons,outputLayerN
eurons))
output_bias = np.random.uniform(size=(1,outputLayerNeurons))

print("Initial hidden weights: ",end='')
print(*hidden_weights)
print("Initial hidden biases: ",end='')
print(*hidden_bias)
print("Initial output weights: ",end='')
print(*output_weights)
print("Initial output biases: ",end='')
print(*output_bias)
```

```
Initial hidden weights: [0.82052933 0.39428938] [0.47936364 0.97718061]
Initial hidden biases: [0.93732222 0.66840377]
Initial output weights: [0.85877789] [0.81728556]
Initial output biases: [0.89547486]
```

Training algorithm

In [6]:
```
for _ in range(epochs):
        hidden_layer_activation = np.dot(inputs,hidden_weights)
        hidden_layer_activation += hidden_bias
        hidden_layer_output = sigmoid(hidden_layer_activation)

        output_layer_activation = np.dot(hidden_layer_output,output_weig
hts)
        output_layer_activation += output_bias
        predicted_output = sigmoid(output_layer_activation)

        #Backpropagation
        error = expected_output - predicted_output
        d_predicted_output = error * sigmoid_derivative(predicted_output
)

        error_hidden_layer = d_predicted_output.dot(output_weights.T)
        d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_
layer_output)

        #Updating Weights and Biases
        output_weights += hidden_layer_output.T.dot(d_predicted_output)
* lr
        output_bias += np.sum(d_predicted_output,axis=0,keepdims=True) *
lr
        hidden_weights += inputs.T.dot(d_hidden_layer) * lr
        hidden_bias += np.sum(d_hidden_layer,axis=0,keepdims=True) * lr
```

In [7]:
```
print("Final hidden weights: ",end='')
print(*hidden_weights)
print("Final hidden bias: ",end='')
print(*hidden_bias)
print("Final output weights: ",end='')
print(*output_weights)
print("Final output bias: ",end='')
print(*output_bias)

print("\nOutput from neural network after 10,000 epochs: ",end='')
print(*predicted_output)
```

```
Final hidden weights: [2.98033689 6.02213327] [2.97996777 6.01739985]
Final hidden bias: [-4.41556636 -1.87560029]
Final output weights: [-6.28166562] [6.45155088]
Final output bias: [-3.09350009]

Output from neural network after 10,000 epochs: [0.09022419] [0.885862
8] [0.88587383] [0.13958549]
```

References

https://towardsdatascience.com/implementing-the-xor-gate-using-backpropagation-in-neural-networks-c1f255b4f20d (https://towardsdatascience.com/implementing-the-xor-gate-using-backpropagation-in-neural-networks-c1f255b4f20d), An Introduction to Neural Networks by Kevin Gurney, https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/ (https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/)

In [ ]: