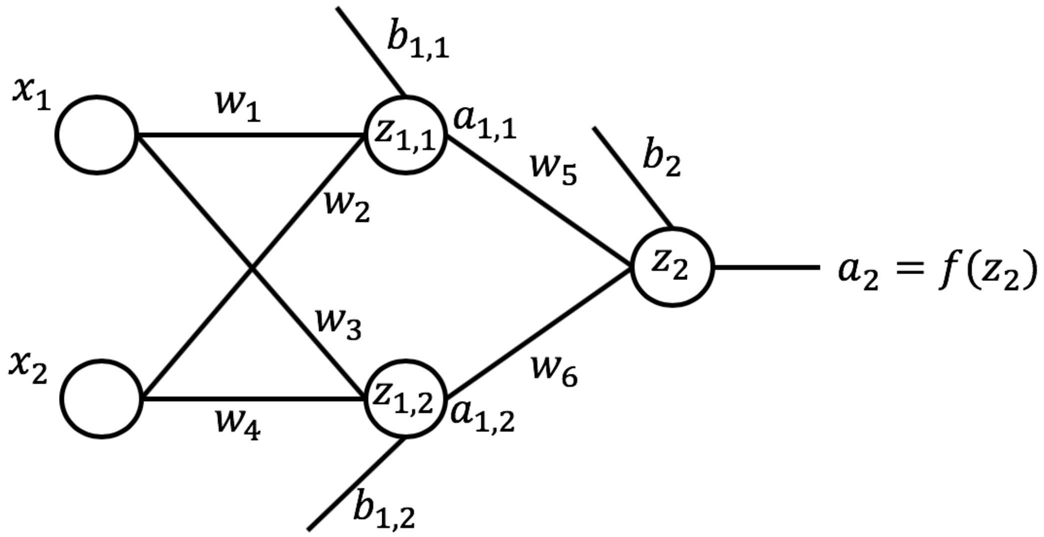
Artificial Neural Network

Creating a Neural Network



In [ ]:
```
import numpy as np

#initializing the weights
weights = np.around(np.random.uniform(size=6), decimals=2)
#initializing the biases
biases = np.around(np.random.uniform(size=3), decimals=2)
```

In [ ]:
```
print('weights: ', weights)
print('biases: ', biases)
```

```
weights:  [0.92 0.9  0.03 0.96 0.14 0.28]
biases:  [0.61 0.94 0.85]
```

Now, we have weights and bias, we can compute outpute for x_1 and x_2 inputs

In [ ]:
```
x_1 = 0.5
x_2 = 0.85
```

Computing the weighted sum of the inputs, $z_{1,1}$, at the first node of the hidden layer.

In [ ]:
```
z_11 = x_1 * weights[0] + x_2 * weights[1] + biases[0]
print('The weighted sum of the inputs at the first node in the hidden layer is {
```

```
The weighted sum of the inputs at the first node in the hidden layer is 1.835
```

Computing the weighted sum of the inputs, $z_{1,2}$, at the first node of the hidden layer.

In [ ]:
```
z_12 = x_1 * weights[2] + x_2 * weights[3] + biases[1]
print('The weighted sum of the inputs at the second node in the hidden layer is 
```

```
The weighted sum of the inputs at the second node in the hidden layer is 1.771
```

Assuming a sigmoid activation function, let's compute the activation of the first node, $a_{1,1}$, in the hidden layer.

In [ ]:
```
a_11 = 1.0 / (1.0 + np.exp(-z_11))
print('The activation of the first node in the hidden layer is {}'.format(np.arc
```

```
The activation of the first node in the hidden layer is 0.8624
```

Computing the activation of the second node, $a_{1,2}$, in the hidden layer. Assigning the value to **a_12**.

In [ ]:
```
a_12 = 1.0 / (1.0 + np.exp(-z_12))
print('The activation of the second node in the hidden layer is {}'.format(np.ar
```

```
The activation of the second node in the hidden layer is 0.8546
```

Now these activations will serve as the inputs to the output layer. So, let's compute the weighted sum of these inputs to the node in the output layer. Assign the value to **z_2**.

In [ ]:
```
z_2 = a_11 * weights[4] + a_12 * weights[5] + biases[2]
print('The weighted sum of the inputs at the node in the output layer is {}'.for
```

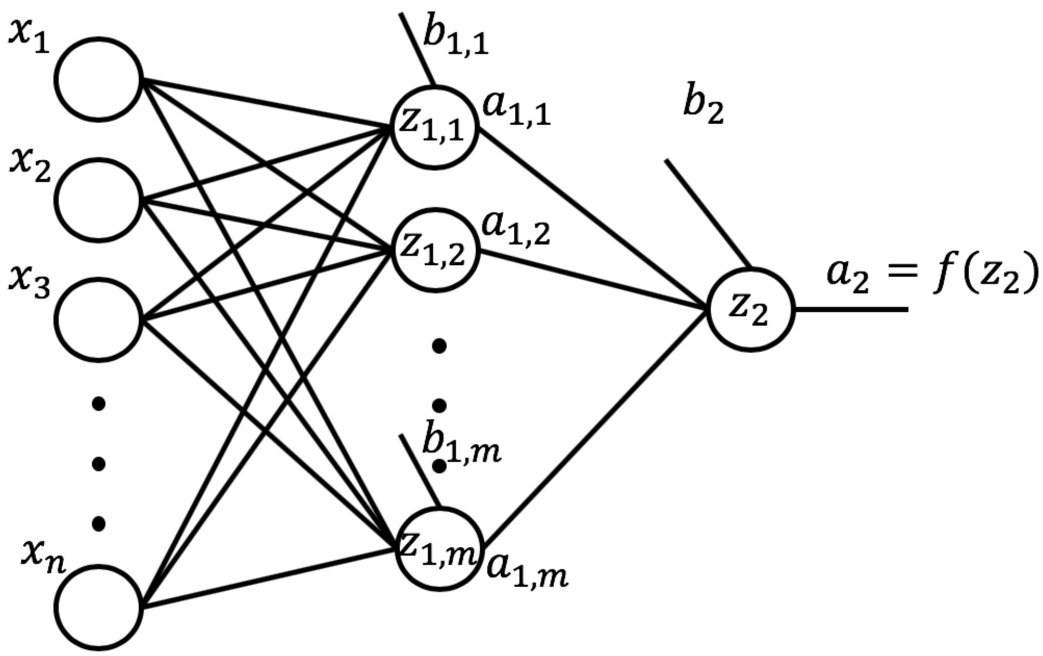
```
The weighted sum of the inputs at the node in the output layer is 1.21
```

Computing the output of the network as the activation of the node in the output layer. Assign the value to **a_2**.

In [ ]:
```
a_2 = 1.0 / (1.0 + np.exp(-z_2))
print('The activation of the node in the output layer is {}'.format(np.around(a_
```

```
The activation of the node in the output layer is 0.7703
```

Creating complex Neural Network



Intializing a Network

In [ ]:
```
num_inputs = 2 # number of inputs
num_hidden_layers = 2 # number of hidden layers
num_nodes_hidden = [2, 2] # number of nodes in each hidden layer
num_nodes_output = 1 # number of nodes in the output layer
```

Initializing the weights and biases in a network

In [ ]:
```
def initialize_network(num_inputs, num_hidden_layers, num_nodes_hidden, num_node

    num_nodes_previous = num_inputs # number of nodes in the previous layer

    network = {}

    # loop through each layer and randomly initialize the weights and biases ass
    for layer in range(num_hidden_layers + 1):

        if layer == num_hidden_layers:
            layer_name = 'output' # name last layer in the network output
            num_nodes = num_nodes_output
        else:
            layer_name = 'layer_{}'.format(layer + 1) # otherwise give the layer
            num_nodes = num_nodes_hidden[layer]

        # initialize weights and bias for each node
        network[layer_name] = {}
        for node in range(num_nodes):
            node_name = 'node_{}'.format(node+1)
            network[layer_name][node_name] = {
                'weights': np.around(np.random.uniform(size=num_nodes_previous),
                'bias': np.around(np.random.uniform(size=1), decimals=2),
            }

        num_nodes_previous = num_nodes

    return network # return the network
print(initialize_network(num_inputs, num_hidden_layers, num_nodes_hidden, num_nc
```

```
{'layer_1': {'node_1': {'weights': array([0.  , 0.52]), 'bias': array([0.55])},
'node_2': {'weights': array([0.49, 0.77]), 'bias': array([0.16])}}, 'layer_2':
{'node_1': {'weights': array([0.76, 0.02]), 'bias': array([0.14])}, 'node_2': {'w
eights': array([0.12, 0.31]), 'bias': array([0.67])}}, 'output': {'node_1': {'wei
ghts': array([0.47, 0.82]), 'bias': array([0.29])}}}
```

Using *initialize_network* function to create a network with following structure

- 5 inputs nodes
- 3 hiden layers with 3 nodes each
- 1 output layer

In [ ]:
```
my_network = initialize_network(5, 3, [3, 3, 3], 1)
print(my_network)
```

{'layer_1': {'node_1': {'weights': array([0.73, 0.7 , 0.33, 0.33, 0.98]), 'bias': array([0.62])}, 'node_2': {'weights': array([0.95, 0.77, 0.83, 0.41, 0.45]), 'bias': array([0.4])}, 'node_3': {'weights': array([1. , 0.18, 0.96, 0.42, 0.42]), 'bias': array([0.46])}}, 'layer_2': {'node_1': {'weights': array([0.37, 0.47, 0.04]), 'bias': array([0.08])}, 'node_2': {'weights': array([0.73, 0.64, 0.03]), 'bias': array([0.3])}, 'node_3': {'weights': array([0.22, 0.06, 0.52]), 'bias': array([0.42])}}, 'layer_3': {'node_1': {'weights': array([0.05, 0.57, 0.8]), 'bias': array([0.11])}, 'node_2': {'weights': array([0.28, 0.64, 0.49]), 'bias': array([0.51])}, 'node_3': {'weights': array([0.46, 0.89, 0.61]), 'bias': array([0.6])}}, 'output': {'node_1': {'weights': array([0.44, 0.48, 0.89]), 'bias': array([0.21])}}}

Computing the Weighted Sum at a Node

In [ ]:
```
def compute_weighted_sum(inputs, weights, bias):
    return np.sum(inputs * weights) + bias
```

In [ ]:
```
node_weights = my_network['layer_1']['node_1']['weights']
node_bias = my_network['layer_1']['node_1']['bias']
weighted_sum = compute_weighted_sum(num_inputs, node_weights, node_bias)
print('The weighted sum at the first node in the hidden layer is {}'.format(np.a
```

```
The weighted sum at the first node in the hidden layer is 6.76
```

Similarly we can compute the weighted sum at each node of different layers.

Computing Activation of a Node

In [ ]:
```
def node_activation(weighted_sum):
    return 1.0 / (1.0 + np.exp(-1 * weighted_sum))
```

In [ ]:
```
node_output  = node_activation(compute_weighted_sum(num_inputs, node_weights, nc
print('The output of the first node in the hidden layer is {}'.format(np.around(
```

```
The output of the first node in the hidden layer is 0.9988
```

Forward Propagation

The final piece of building a neural network that can perform predictions is to put everything together. So creating a function that applies the *compute_weighted_sum* and *node_activation* functions to each node in the network and propagates the data all the way to the output layer and outputs a prediction for each node in the output layer.

The way we are going to accomplish this is through the following procedure:

1. Start with the input layer as the input to the first hidden layer.
2. Compute the weighted sum at the nodes of the current layer.
3. Compute the output of the nodes of the current layer.
4. Set the output of the current layer to be the input to the next layer.
5. Move to the next layer in the network.
6. Repeat steps 2 - 4 until we compute the output of the output layer.

In [ ]:
```
def forward_propagate(network, inputs):
    
    layer_inputs = list(inputs) # start with the input layer as the input to the
    
    for layer in network:
        
        layer_data = network[layer]
        
        layer_outputs = []
        for layer_node in layer_data:
        
            node_data = layer_data[layer_node]
        
            # compute the weighted sum and the output of each node at the same t
            node_output = node_activation(compute_weighted_sum(layer_inputs, noc
            layer_outputs.append(np.around(node_output[0], decimals=4))
        
        if layer != 'output':
            print('The outputs of the nodes in hidden layer number {} is {}'.for
        
        layer_inputs = layer_outputs # set the output of this layer to be the ir
    
    network_predictions = layer_outputs
    return network_predictions
```

5 inputs that we can feed to **my_network**.

In [ ]:
```
from random import seed
import numpy as np

np.random.seed(12)
inputs = np.around(np.random.uniform(size=5), decimals=2)

print('The inputs to the network are {}'.format(inputs))
```

```
The inputs to the network are [0.15 0.74 0.26 0.53 0.01]
```

Using the *forward_propagate* function to compute the prediction of my_network

In [ ]:
```
predictions = forward_propagate(my_network, inputs)
print('The predicted value by the network for the given input is {}'.format(np.a
```

```
The outputs of the nodes in hidden layer number 1 is [0.8202, 0.8249, 0.772]
The outputs of the nodes in hidden layer number 2 is [0.6904, 0.81, 0.741]
The outputs of the nodes in hidden layer number 3 is [0.7684, 0.8299, 0.89]
The predicted value by the network for the given input is 0.8505
```

Creating the another network and computing the prediction

In [ ]:
```
#Network Structure
new_network = initialize_network(5, 4, [2, 3, 2, 3], 3)

#inputs to the network
inputs = np.around(np.random.uniform(size=5), decimals=2)

#Getting the prediction
predictions = forward_propagate(new_network, inputs)
```

```
The outputs of the nodes in hidden layer number 1 is [0.7236, 0.7701]
The outputs of the nodes in hidden layer number 2 is [0.7517, 0.6694, 0.7302]
The outputs of the nodes in hidden layer number 3 is [0.8634, 0.8774]
The outputs of the nodes in hidden layer number 4 is [0.8946, 0.8215, 0.8026]
```