In [66]:

```
import math
import random
from random import seed
```

In [67]:

```
def initialize_network(n_inputs, n_hidden, n_outputs):
    network = []

    hidden_layer = [{'weights': [random.random() for _ in range(n_inputs + 1)]} for _ in range(n_hidden)]
    network.append(hidden_layer)

    output_layer = [{'weights': [random.random() for _ in range(n_hidden + 1)]} for _ in range(n_outputs)]
    network.append(output_layer)

    return network
```

In [68]:

```
def activate(weights, inputs):
    return weights[-1] + sum(w * x for w, x in zip(weights[:-1], inputs))
```

In [69]:

```
def transfer(activation):
    return 1.0 / (1.0 + math.exp(-activation))
```

In [70]:

```
def forward_propagate(network, row):
    inputs = row

    for layer in network:
        new_inputs = []
        for neuron in layer:
            activation = activate(neuron['weights'], inputs)
            neuron['output'] = transfer(activation)
            new_inputs.append(neuron['output'])
        inputs = new_inputs

    return inputs
```

In [71]:

```
def transfer_derivative(output):
    # Calculate the derivative of the sigmoid activation function
    return output * (1.0 - output)
```

In [72]:

```
def backward_propagate_error(network, expected):
    for i in reversed(range(len(network))):
        layer = network[i]
        errors = []

        if i != len(network) - 1:
            for j in range(len(layer)):
                error = 0.0
                for neuron in network[i + 1]:
                    error += (neuron['weights'][j] * neuron['delta'])
                errors.append(error)
        else:
            for j in range(len(layer)):
                neuron = layer[j]
                errors.append(expected[j] - neuron['output'])

        for j in range(len(layer)):
            neuron = layer[j]
            # Calculate and store the delta value
            neuron['delta'] = errors[j] * transfer_derivative(neuron['output'])
```

In [73]:

```
def update_weights(network, row, l_rate):
    for i in range(len(network)):
        inputs = row[:-1] if i == 0 else [neuron['output'] for neuron in network[i - 1]]

        for neuron in network[i]:
            for j in range(len(inputs)):
                neuron['weights'][j] += l_rate * neuron['delta'] * inputs[j]

            neuron['weights'][-1] += l_rate * neuron['delta']
```

In [74]:

```
def train_network(network, train, l_rate, n_epoch, n_outputs):
    for epoch in range(n_epoch):
        sum_error = 0

        for row in train:
            # Forward pass
            outputs = forward_propagate(network, row)

            # Prepare the expected output (one-hot encoding)
            expected = [0] * n_outputs
            expected[int(row[-1])] = 1

            # Calculate and accumulate the squared error
            sum_error += sum((expected[i] - outputs[i]) ** 2 for i in range(n_outputs))

            # Backpropagate and update weights
            backward_propagate_error(network, expected)
            update_weights(network, row, l_rate)

        # Print epoch details
        print(f'> epoch={epoch}, lrate={l_rate:.3f}, error={sum_error:.3f}')
```

In [75]:

```
seed(1)

# Define the dataset
dataset = [
    [2.7810836, 2.550537003, 0],
    [1.465489372, 2.362125076, 0],
    [3.396561688, 4.400293529, 0],
    [1.38807019, 1.850220317, 0],
    [3.06407232, 3.005305973, 0],
    [7.627531214, 2.759262235, 1],
    [5.332441248, 2.088626775, 1],
    [6.922596716, 1.77106367, 1],
    [8.675418651, -0.242068655, 1],
    [7.673756466, 3.508563011, 1]
]

# Define network parameters
n_inputs = len(dataset[0]) - 1
n_outputs = len(set([row[-1] for row in dataset]))

# Initialize the neural network
network = initialize_network(n_inputs, 2, n_outputs)

# Train the network
train_network(network, dataset, 0.5, 20, n_outputs)

# Print the trained network
for layer in network:
    print(layer)
```

```
> epoch=0, lrate=0.500, error=6.350
> epoch=1, lrate=0.500, error=5.531
> epoch=2, lrate=0.500, error=5.221
> epoch=3, lrate=0.500, error=4.951
> epoch=4, lrate=0.500, error=4.519
> epoch=5, lrate=0.500, error=4.173
> epoch=6, lrate=0.500, error=3.835
> epoch=7, lrate=0.500, error=3.506
> epoch=8, lrate=0.500, error=3.192
> epoch=9, lrate=0.500, error=2.898
> epoch=10, lrate=0.500, error=2.626
> epoch=11, lrate=0.500, error=2.377
> epoch=12, lrate=0.500, error=2.153
> epoch=13, lrate=0.500, error=1.953
> epoch=14, lrate=0.500, error=1.774
> epoch=15, lrate=0.500, error=1.614
> epoch=16, lrate=0.500, error=1.472
> epoch=17, lrate=0.500, error=1.346
> epoch=18, lrate=0.500, error=1.233
> epoch=19, lrate=0.500, error=1.132
[{'weights': [-1.4688375095432327, 1.8508873254395142, 1.0858178629550297], 'output': 0.029980305604442621, 'delta': -0.0059546604162323368}, {'weights': [0.37711098142462174, -0.0625909894552995, 0.2765123702642716], 'output': 0.9456229000211323, 'delta': 0.002627965285086386}]
[{'weights': [2.515394649397849, -0.3391927502445991, -0.9671565426390271], 'output': 0.23648794202357587, 'delta': -0.04270059278364587}, {'weights': [-2.5584149848484263, 1.0036422106209208, 0.42383086467582687], 'output': 0.7790535202438367, 'delta': 0.03803132596437354}]
```

In [76]:

```
from math import exp

# Calculate neuron activation for an input
def activate(weights, inputs):
    activation = weights[-1]
    for i in range(len(weights) - 1):
        activation += weights[i] * inputs[i]
    return activation
```

In [77]:

```
# Transfer neuron activation
def transfer(activation):
    return 1.0 / (1.0 + exp(-activation))
```

In [78]:

```
# Forward propagate input to a network output
def forward_propagate(network, row):
    inputs = row
    for layer in network:
        new_inputs = []
        for neuron in layer:
            activation = activate(neuron['weights'], inputs)
            neuron['output'] = transfer(activation)
            new_inputs.append(neuron['output'])
        inputs = new_inputs
    return inputs
```

In [79]:

```
# Make a prediction with a network
def predict(network, row):
    outputs = forward_propagate(network, row)
    return outputs.index(max(outputs))

# Test making predictions with the network
dataset = [
    [2.7810836, 2.550537003, 0],
    [1.465489372, 2.362125076, 0],
    [3.396561688, 4.400293529, 0],
    [1.38807019, 1.850220317, 0],
    [3.06407232, 3.005305973, 0],
    [7.627531214, 2.759262235, 1],
    [5.332441248, 2.088626775, 1],
    [6.922596716, 1.77106367, 1],
    [8.675418651, -0.242068655, 1],
    [7.673756466, 3.508563011, 1]
]

network = [
    [{'weights': [-1.482313569067226, 1.8308790073202204, 1.078381922048799]},
     {'weights': [0.23244990332399884, 0.3621998343835864, 0.40289821191094327]}],
    [{'weights': [2.5001872433501404, 0.7887233511355132, -1.1026649757805829]},
     {'weights': [-2.429350576245497, 0.8357651039198697, 1.0699217181280656]}]
]

for row in dataset:
    prediction = predict(network, row)
    print('Expected=%d, Got=%d' % (row[-1], prediction))
```

```
Expected=0, Got=0
Expected=0, Got=0
Expected=0, Got=0
Expected=0, Got=0
Expected=0, Got=0
Expected=1, Got=1
Expected=1, Got=1
Expected=1, Got=1
Expected=1, Got=1
Expected=1, Got=1
```

In [ ]: