CODE

```
import pandas as pd
import math
from random import random, seed

network = list() #This represents the whole network which is an array of layers
eta = 0.5 #learning rate

def initialize_network(n_inputs, n_hiddens, n_outputs):
#   seed(1)
    for k in range(len(n_hiddens)):
        hidden_bias = random()
        if(k == 0):
            hidden_layer = [{'id':j, 'weights':[random() for i in 
range(n_inputs)], 'bias':hidden_bias, 'output':0, 'net':0, 'inputs':[]} for j in 
range(n_hiddens[k])]
        else:
            hidden_layer = [{'id':j, 'weights':[random() for i in 
range(n_hiddens[k-1])], 'bias':hidden_bias, 'output':0, 'net':0, 'inputs':[]} 
for j in range(n_hiddens[k])]
        network.append(hidden_layer)
    output_bias = random()
    output_layer = [{'id':j, 'weights':[random() for i in range(n_hiddens[-1])], 
'bias':output_bias, 'output':0, 'net':0, 'inputs':[]} for j in range(n_outputs)]
    network.append(output_layer)


def weighted_sum(weights, bias, inputs):
    net = bias
    for i in range(len(weights)):
        net += weights[i] * inputs[i]
    return net

def activate(net):
    return 1.0/(1.0 + math.exp(-net))

def forward_propogate(network, inputs):
    theInputs = inputs
    for layer in network:
        new_inputs = []
        for node in layer:
            node['inputs'] = theInputs
            node['net'] = weighted_sum(node['weights'], node['bias'], theInputs)
#           print("node[Net]:{}".format(node['net']))
            node['output'] = activate(node['net'])
            new_inputs.append(node['output'])
        theInputs = new_inputs
    return theInputs

def activation_derivative(value):
    return value * (1.0 - value)

def backward_propogate_error(network, target_output):
    for i in reversed(range(len(network))):
        layer = network[i]
        errors = list()
        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(target_output[j] - neuron['output'])

        for j in range(len(layer)):
            neuron = layer[j]
            neuron['delta'] = errors[j] * 
activation_derivative(neuron['output'])

def update_weights(network, inputs, eta):
    for layer in network:
        for neuron in layer:
            for j in range(len(neuron['weights'])):
                neuron['weights'][j] += eta * neuron['delta'] * neuron['inputs']
[j]
            neuron['bias'] += eta * neuron['delta']

def train_network(network, inputs, eta, n_epoch, n_outputs):
    for epoch in range(n_epoch):
        sum_error = 0.0
        for row in inputs:
            if (row[-1] == 1):
                target_output[0] = 0.99
                target_output[1] = 0.01
            if (row[-1] == -1):
                target_output[0] = 0.01
                target_output[1] = 0.99
            row = row[:-1]
            outputs = forward_propogate(network, row)
            sum_error += sum([(target_output[i] - outputs[i]) ** 2 for i in 
range(len(outputs))])
            backward_propogate_error(network, target_output)
            update_weights(network, row, eta)
        print('epoch=%d, error=%.3f' %(epoch, sum_error) )

def test_output(network, inputs):
    print("The Test Output is\n{}\n\n".format(forward_propogate(network, 
inputs)))
```

```
#main
#The network contains an input layer, output, layer and multiple hidden layers
initialize_network(4,[4],2)

data = pd.read_csv('data.csv')

train_network(network, data.values, eta, 500, 2)
print('\n\n')

inputs = [3,1,1,1]#answer:No(-1) [will activate second output neuron]
test_output(network, inputs)
```

DATA

Outlook	Temperature	Humidity	Wind	PlayTennis
1	1	1	2	-1
1	1	1	1	-1
2	1	1	2	1
3	2	1	2	1
3	3	2	2	1
3	3	2	1	-1
2	3	2	1	1
1	2	1	2	-1
1	3	2	2	1
3	2	2	2	1
1	2	2	1	1
2	2	1	1	1
2	1	2	2	1
3	2	1	1	-1

OUTPUT

```
epoch=0, error=9.189
epoch=1, error=6.918
epoch=2, error=6.814
epoch=3, error=6.813
epoch=4, error=6.811
epoch=5, error=6.808
epoch=6, error=6.805
epoch=7, error=6.801
epoch=8, error=6.798
epoch=9, error=6.794
epoch=10, error=6.790
.
.
.
.
.
epoch=490, error=0.077
epoch=491, error=0.077
epoch=492, error=0.077
epoch=493, error=0.076
epoch=494, error=0.076
epoch=495, error=0.076
epoch=496, error=0.075
epoch=497, error=0.075
epoch=498, error=0.074
epoch=499, error=0.074


The Test Output is
[0.05838935064180531, 0.9414509060304482]
```