4. Write a program for 3-2-2 (3 nodes in input layes, 2 nodes in hidden layer and 2 node in output layer) ANN. You may use numpy and panda only. [3 marks].

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import OneHotEncoder

def NeuralNetwork(X_train, Y_train, X_val=None, Y_val=None, epochs=10, nodes=[], lr=0.15):
    hidden_layers = len(nodes) - 1
    weights = InitializeWeights(nodes)

    for epoch in range(1, epochs+1):
        weights = Train(X_train, Y_train, lr, weights)

        if(epoch % 20 == 0):
            print("Epoch {}".format(epoch))
            print("Training Accuracy:{}".format(Accuracy(X_train, Y_train, weights)))
            if X_val.any():
                print("Validation Accuracy:{}".format(Accuracy(X_val, Y_val, weights)))

    return weights


def InitializeWeights(nodes):
  """Initialize weights with random values in [-1, 1] (including bias)"""
  layers, weights = len(nodes), []

  for i in range(1, layers):
      w = [[np.random.uniform(-1, 1) for k in range(nodes[i-1] + 1)]
            for j in range(nodes[i])]
      weights.append(np.matrix(w))

  return weights

def ForwardPropagation(x, weights, layers):
    activations, layer_input = [x], x
    for j in range(layers):
        activation = Sigmoid(np.dot(layer_input, weights[j].T))
        activations.append(activation)
        layer_input = np.append(1, activation) # Augment with bias

    return activations

def BackPropagation(y, activations, weights, layers):
  outputFinal = activations[-1]
  error = np.matrix(y - outputFinal) # Error at output

  for j in range(layers, 0, -1):
      currActivation = activations[j]

      if(j > 1):
          # Augment previous activation
          prevActivation = np.append(1, activations[j-1])
      else:
          # First hidden layer, prevActivation is input (without bias)
          prevActivation = activations[0]

      delta = np.multiply(error, SigmoidDerivative(currActivation))
      weights[j-1] += lr * np.multiply(delta.T, prevActivation)

      w = np.delete(weights[j-1], [0], axis=1) # Remove bias from weights
      error = np.dot(delta, w) # Calculate error for current layer

  return weights

def Train(X, Y, lr, weights):
    layers = len(weights)
    for i in range(len(X)):
        x, y = X[i], Y[i]
        x = np.matrix(np.append(1, x)) # Augment feature vector

        activations = ForwardPropagation(x, weights, layers)
        weights = BackPropagation(y, activations, weights, layers)

    return weights

def Sigmoid(x):
    return 1 / (1 + np.exp(-x))

def SigmoidDerivative(x):
    return np.multiply(x, 1-x)

def Predict(item, weights):
    layers = len(weights)
    item = np.append(1, item) # Augment feature vector

    ##_Forward Propagation_##
    activations = ForwardPropagation(item, weights, layers)
    outputFinal = activations[-1].A1
    index = FindMaxActivation(outputFinal)

    # Initialize prediction vector to zeros
    y = [0 for i in range(len(outputFinal))]
    y[index] = 1  # Set guessed class to 1

    return y # Return prediction vector


def FindMaxActivation(output):
    """Find max activation in output"""
    m, index = output[0], 0
    for i in range(1, len(output)):
        if(output[i] > m):
            m, index = output[i], i

    return index

def Accuracy(X, Y, weights):
    """Run set through network, find overall accuracy"""
    correct = 0

    for i in range(len(X)):
        x, y = X[i], list(Y[i])
        guess = Predict(x, weights)

        if(y == guess):
            # Guessed correctly
            correct += 1

    return correct / len(X)

layers = [3, 2, 2]
lr, epochs = 0.15, 100

X_train = [[1, 2, 4], [1, 2, 6], [5, 6, 9]]
Y_train = [[0, 1], [1, 1], [1, 1]]
X_val = [[3, 2, 5], [2, 5, 6], [5, 6, 9]]
Y_val = [[0, 1], [1, 1], [1, 1]]

X_test = [[5, 8, 9], [3, 5, 1], [5, 6, 9]]
Y_test = [[0, 1], [0, 1], [1, 1]]
```

```
X_train = np.array(X_train)
Y_train = np.array(Y_train)
X_val = np.array(X_val)
Y_val = np.array(Y_val)
weights = NeuralNetwork(X_train, Y_train, X_val, Y_val, epochs=epochs, nodes=layers, lr=lr);

print("Testing Accuracy: {}".format(Accuracy(X_test, Y_test, weights)))
```