SOFTWARE ENGINEERING FOR WEB APPLICATIONS

HOMEWORK – 5

1. Source Code:

```
import math
import random

random.seed(0)


def sigmoid(x):
    """
    computes the sigmoid function 1/(1+e^(-x))
    """
    return 1.0 / (1.0 + math.exp(-x))


def d_sigmoid(y):
    """
    derivatve of the sigmoid function
    :return:
    """
    return y * (1 - y)


def matrix(a, b, element=0.0):
    """
    generate a matrix
    """
    mat = []
    for i in range(a):
        mat.append([element] * b)
    return mat


def shuffle_matrix(mat, a, b):
    """
   shuffle the elements of the matrix
    """
    for i in range(len(mat)):
        for j in range(len(mat[0])):
            mat[i][j] = random.uniform(a, b)

# class definition for the neural network
class NeuralNetwork:
    def __init__(self, n_i, n_h, n_o):
        # initialize the number of input, hidden, and output nodes
        """
```

```python
        n_i - no of input nodes
        n_h - no of hidden layer nodes
        n_o - no of output nodes
        """
        self.n_i = n_i + 1
        self.n_h = n_h
        self.n_o = n_o

        #
        self.a_i = [1.0] * self.n_i
        self.a_h = [1.0] * self.n_h
        self.a_o = [1.0] * self.n_o

        # weight matrix
        self.w_i = matrix(self.n_i, self.n_h)  # W1
        self.w_o = matrix(self.n_h, self.n_o)  # Theta2

        shuffle_matrix(self.w_i, -1, 1)
        shuffle_matrix(self.w_o, -1, 1)
        print("\nInitial weights:")
        print("W1: ")
        for i in range(self.n_i):
            print(self.w_i[i])
        print("W2: ")
        for j in range(self.n_h):
            print(self.w_o[j])

        self.c_i = matrix(self.n_i, self.n_h)
        self.c_o = matrix(self.n_h, self.n_o)

    def forward_propagation(self, inputs):
        """
        function to perform forward propagation
        """
        if len(inputs) != self.n_i - 1:
            print('incorrect number of inputs')

        for i in range(self.n_i - 1):
            self.a_i[i] = inputs[i]

        for j in range(self.n_h):
            sum = 0.0
            for i in range(self.n_i):
                sum += (self.a_i[i] * self.w_i[i][j])
            self.a_h[j] = sigmoid(sum)

        for k in range(self.n_o):
            sum = 0.0
            for j in range(self.n_h):
                sum += (self.a_h[j] * self.w_o[j][k])
            self.a_o[k] = sigmoid(sum)

        return self.a_o

    def back_propagation(self, t, N, M):
```

```
        """
        function to perform backpropagation
        """

        # calculate the delta for output layer
        out = [0.0] * self.n_o
        for k in range(self.n_o):
            error = t[k] - self.a_o[k]
            out[k] = error * d_sigmoid(self.a_o[k])

        # update weight W2
        for j in range(self.n_h):
            for k in range(self.n_o):
                c = out[k] * self.a_h[j]
                self.w_o[j][k] += N * c + M * self.c_o[j][k]
                self.c_o[j][k] = c

        # calculate the delta for the hidden layer
        hid = [0.0] * self.n_h
        for j in range(self.n_h):
            error = 0.0
            for k in range(self.n_o):
                error += out[k] * self.w_o[j][k]
            hid[j] = error * d_sigmoid(self.a_h[j])

        # update W1
        for i in range(self.n_i):
            for j in range(self.n_h):
                c = hid[j] * self.a_i[i]
                self.w_i[i][j] += N * c + M * self.c_i[i][j]
                self.c_i[i][j] = c

        error = 0.0
        for k in range(len(t)):
            error = 0.5 * (t[k] - self.a_o[k]) ** 2
        return error

    def print_weights(self):
        """
        print(the weights
        """
        print("\nFinal weights: ")
        print('W1: ')
        for i in range(self.n_i):
            print(self.w_i[i])
        print('W2: ')
        for j in range(self.n_h):
            print(self.w_o[j])

    def test(self, patterns):
        """
        testing the model
        """
        print("\n")
        for p in patterns:
```

```
            inputs = p[0]
            # print('Inputs:', p[0], '-->', self.forward_propagation(inputs),
'\tTarget', p[1]
            print('Inputs:', p[0], '-->', self.forward_propagation(inputs),
'Target'.rjust(10), p[1])

    def train(self, patterns, max_iterations=1000, N=0.5, M=0.5):
        """
        training the model
        """

        N = learning_rate
        M = N / 2
        error = 0.0
        for i in range(max_iterations):
            for p in patterns:
                inp = p[0]
                t = p[1]
                self.forward_propagation(inp)
                error = self.back_propagation(t, N, M)

            if i == 0:
                print("\nFirst-batch error: ", error)

            if error < expected_error:
                print("\nFinal error: ", error)
                print("\nTotal number of batches trained: ", i + 1)
                break
        # self.test(patterns)


def main():
    X = [
        [[0, 0], [0]],
        [[0, 1], [1]],
        [[1, 0], [1]],
        [[1, 1], [0]]
    ]
    global expected_error
    global learning_rate
    expected_error = float(input('Enter the expected error: '))
    learning_rate = float(input('Enter the learning rate: '))
    NN = NeuralNetwork(2, 2, 1)
    NN.train(X)
    NN.print_weights()


if __name__ == "__main__":
    main()
```

Output:

a. For learning rate = 0.5 and expected error = 0.1:

```
Enter the expected error: 0.1
Enter the learning rate: 0.5

Initial weights:
W1:
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.022549442737217704, -0.19013172509917142]
W2:
[0.5675971780695452]
[-0.3933745478421451]

First-batch error:  0.1528562731414498

Final error:  0.09990708012694793

Total number of batches trained:  596

Final weights:
W1:
[0.8205497350676921, 2.1576600041387506]
[-1.0459614445587375, -1.8469025970949688]
[-0.24824546094051156, 1.1440557337712816]
W2:
[1.4658862263160384]
[-0.9890290929247303]
```

b. For learning rate = 1.0 and expected error = 0.1:

```
Enter the expected error: 0.1
Enter the learning rate: 1.0

Initial weights:
W1:
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142]
W2:
[0.5675971780695452]
[-0.3933745478421451]

First-batch error:  0.1579000632765516

Final error:  0.09977700044780606

Total number of batches trained:  272

Final weights:
W1:
[1.0279513934896458, 1.831900908252926]
[-1.7694748268661464, -1.8224832559110051]
[-0.429960038457029, 1.1054533011429586]
W2:
[1.3172902132462327]
[-0.7359047098099407]
```

c. For learning rate = 2.0 and expected error = 0.1:

```
Enter the expected error: 0.1
Enter the learning rate: 2.0

Initial weights:
W1:
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142]
W2:
[0.5675971780695452]
[-0.3933745478421451]

First-batch error:  0.16951077410382095

Final error:  0.09967403104786529

Total number of batches trained:  138

Final weights:
W1:
[1.2413452725216187, 1.9692078011385545]
[-2.2690624414490372, -3.2516769863220323]
[1.164664000401361, -0.9888216916507425]
W2:
[-0.8021891943374395]
[1.9558173576458016]
```

d. For learning rate = 1.6 and expected error = 0.1

```
Enter the expected error: 0.1
Enter the learning rate: 1.6

Initial weights:
W1:
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142]
W2:
[0.5675971780695452]
[-0.3933745478421451]

First-batch error:  0.16461730806750727

Final error:  0.09963021120336203

Total number of batches trained:  219

Final weights:
W1:
[1.9871586987511383, 1.1686923258236765]
[-3.1815689014128523, -2.3532652355233097]
[-1.066010779839323, 1.211354611547206]
W2:
[1.982575468865376]
[-0.8470010283819751]
```

The best value for the learning rate as obtained through the results is alpha = 1.5, due to the minimum time needed for training the batches.

e. For learning rate = 0.5 and expected error = 0.02

```
Enter the expected error: 0.02
Enter the learning rate: 0.5

Initial weights:
W1:
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142]
W2:
[0.5675971780695452]
[-0.3933745478421451]

First-batch error:  0.1528562731414498

Final error:  0.01997390852653138

Total number of batches trained:  769

Final weights:
W1:
[2.9845689133076436, 3.987516883454604]
[-3.2540991141710474, -3.9920204687248226]
[-1.6497741244311088, 1.5500994890864117]
W2:
[4.733032345426558]
[-2.4343824970285364]
```

f. For learning rate = 1.0 and expected error = 0.02

```
Enter the expected error: 0.02
Enter the learning rate: 1.0

Initial weights:
W1:
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142]
W2:
[0.5675971780695452]
[-0.3933745478421451]

First-batch error:  0.1579000632765516

Final error:  0.01979110735661421

Total number of batches trained:  356

Final weights:
W1:
[3.414936124335939, 3.6437575914914486]
[-3.8309918653148713, -3.7774808004538225]
[-1.9220994910353069, 1.4115421531262577]
W2:
[4.6435107168102014]
[-2.3365249580116805]
```

g. For learning rate = 2.0 and expected error = 0.02

```
Enter the expected error: 0.02
Enter the learning rate: 2.0

Initial weights:
W1:
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142]
W2:
[0.5675971780695452]
[-0.3933745478421451]

First-batch error:  0.16951077410382095

Final error:  0.0192508826156375

Total number of batches trained:  174

Final weights:
W1:
[3.342375866182082, 3.866582564265442]
[-3.7607724913443277, -4.5862234383649625]
[1.382685565058697, -2.1552231536577606]
W2:
[-2.381887120259488]
[4.776829117726538]
```

h. For learning rate = 1.6 and expected error = 0.02:

```
Enter the expected error: 0.02
Enter the learning rate: 1.6

Initial weights:
W1:
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142]
W2:
[0.5675971780695452]
[-0.3933745478421451]

First-batch error:  0.16461730806750727

Final error:  0.019996036034704935

Total number of batches trained:  263

Final weights:
W1:
[3.80488328755284, 3.2665611474808594]
[-4.510344178692863, -3.728950215412691]
[-2.135174772979505, 1.4011109918942433]
W2:
[4.729971257346958]
[-2.3597084580508705]
```

The best value for the learning rate is alpha = 1.5 due to the minimum time required for training the batches.

2. Source Code:

```
<!DOCTYPE html>
<html>
<head>
  <title>Volume</title>
  <style type="text/css">
    table, th, tr, td {
      border: 1px solid black;
    }
```

```
  </style>
</head>
<body>
  <h1>This Website will find the Volume for a Cylinder, Sphere or Cone</h1>

  <p>Select the units (English or SI)</p>
  <form id="HW5">
      <input type="radio" id="English" value="English" name="unit"
onclick="unitClick(this)" checked>
      <label for="English">English</label>
      <input type="radio" id="SI" name="unit" value="SI" onclick="unitClick(this)">
      <label for="SI">SI</label>
      <br>

      <label>Select the shape</label>
      <select id="Shape" onChange="shapeClick(this)">
        <option value="cylinder">Cylinder</option>
        <option value="sphere">Sphere</option>
        <option value="cone">Cone</option>
      </select>
      <br><br>

      <label>Enter the radius</label>
      <input type="text" name="radius" id= "r" oninput="radiusInput(this)">
      <br><br>

      <label>For the cylinder and cone, enter the height</label>
      <input type="text" name="height" id="h" oninput="heightInput(this)">
      <br><br>

      <button onclick="reset()">Reset the form</button>

  </form>


  <h1>Results</h1>

  <p>You chose to use <span id="unit_show">English</span> units <br>
    and to find the volume of a <span id="type_show">cylinder</span><br></p>

  <table>
```

```
      <tr>
         <th>Shape</th>
         <th>Radius</th>
         <th>Height</th>
         <th>Volume</th>
      </tr>
      <tr>
         <td> </td>
          <td>(<span id="cal_unit1">ft</span>)</td>
          <td>(<span id="cal_unit2">ft</span>)</td>
          <td>(<span id="cal_unit3">ft</span>^3)</td>
      </tr>
      <tr>
         <td><span id="type_show1">cylinder</span></td>
         <td><span id="radius"></span></td>
         <td><span id="height"></span></td>
         <td><span id="vol"></span></td>
      </tr>
   </table>

   <button onclick="calculate()">Click to calculate results</button>
</body>
</html>



<script type="text/javascript">
function reset() {
   var x = document.forms["HW5"];
   x.r.value = "";
   x.h.value = "";
   obj.selectedIndex = 0;

   document.getElementById('vol').innerHTML = "";
   document.getElementById('radius').innerHTML = "";
   document.getElementById('height').innerHTML = "";
   document.getElementById('cal_unit1').innerHTML = "ft";
   document.getElementById('cal.untt2').innerHTML = "ft";
   document.getElementById('cal_unit3').innerHTML = "ft";
   document.getElementById('type_show').innerHTML = "cylinder"
   document.getElementById('type_show1').innerHTML = "cylinder";
```

```
}

function radiusInput(obj) {
  document.getElementById('radius').innerHTML = obj.value;
}

function heightInput(obj) {
  document.getElementById('height').innerHTML = obj.value;
}

function shapeClick(obj) {
  var shape = obj.value;
  document.getElementById('type_show').innerHTML = shape;
  document.getElementById('type_show1').innerHTML = shape;
}

function unitClick(obj) {
  var unit = obj.value;
  document.getElementById('unit_show').innerHTML = unit;
  if (unit == "English") {
    document.getElementById('cal_unit1').innerHTML = "ft";
    document.getElementById('cal_unit2').innerHTML = "ft";
    document.getElementById('cal_unit3').innerHTML = "ft";
  } else {
    document.getElementById('cal_unit1').innerHTML = "m";
    document.getElementById('cal_unit2').innerHTML = "m";
    document.getElementById('cal_unit3').innerHTML = "m";
  }
}

function typeClick(obj) {
  var index = obj.selectedIndex;
  var type = obj.options[index].value;
  document.getElementById('type_show').innerHTML = type;
  document.getElementById('type_show1').innerHTML = type;
}

function calculate() {
  var radius = document.getElementById('r').value;
  var mySelect = document.getElementById('Shape');
  var index = mySelect.selectedIndex;
```

```
   var type = mySelect.options[index].text;
   var height = document.getElementById('h').value;
   var v;
   if(radius == "") {
      window.alert("Please input radius");
      return;
   }
   if(type == "Cylinder") {
      if (height == "") {
         window.alert("Please enter height")
         return;
      }

      v = 3.14159 * radius * radius * height;
   } else if (type == "Sphere") {
      v = 4/3 * 3.14159 * radius * radius * radius;
   } else if (type == "Cone") {
      if (height == "") {
         window.alert("Please enter height");
         return;
      }
      v = 1/3 * 3.14159 * radius * radius * height;
   }

   document.getElementById('vol').innerHTML = v;
   document.getElementById('radius').innerHTML = radius;
   document.getElementById('height').innerHTML = height;
}
   </script>
```

The result is shown below:

This Website will find the Volume for a Cylinder, Sphere or Cone

Select the units (English or SI)

English SI

Select the shape Cone

Enter the radius 4

For the cylinder and cone, enter the height 5

Reset the form

Results

You chose to use SI units
and to find the volume of a cone

Shape	Radius	Height	Volume
	(m)	(m)	(m^3)
cone	4	5	83.77573333333332

Click to calculate results