HW5

Topics: ANN & JS

Course: SEII

Xinyu Lyu(xl422)

Email: xl422@scarletmail.rutgers.edu

1.Use backprogragation.py to test

(1) source code:

```
# -*- coding: utf-8 -*-
import math
import random

random.seed(0)

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

def dsigmoid(y):
    """
    derivative of sigmoid
    """
    return y * (1 - y)

def rand(x, y):
    return (y - x) * random.random() + x

def generateMatrix(I, J):
    a = []
    for i in range(I):
        a.append([0.0] * J)
    return a

def randomizeMatrix(matrix, x, y):
    for i in range(len(matrix)):
        for j in range(len(matrix[0])):
            matrix[i][j] = random.uniform(x, y)

class neural_network:
    def __init__(self, ni, nh, no):
        """
        :param ni:number of input nodes
        :param nh:number of hidden nodes
        :param no:number of output nodes
        """
        self.ni = ni + 1   # +1 for offset node
        self.nh = nh
```

```
        self.no = no

        self.ai = [1.0] * self.ni
        self.ah = [1.0] * self.nh
        self.ao = [1.0] * self.no

        self.wi = generateMatrix(self.ni, self.nh)    # input layers to hidden layers weights
        self.wo = generateMatrix(self.nh, self.no)    # hidden layers to output layers weights

        randomizeMatrix(self.wi, -1, 1)
        randomizeMatrix(self.wo, -1, 1)

        print ' '
        print 'Initial weights:'
        print '(input layers to hidden layers weights:)'
        for i in range(self.ni):
            if i==self.nh:
                print self.wi[i], '(Offset node)'
            else:
                print self.wi[i]
        print '(hidden layers to output layers weights:)'
        for j in range(self.nh):
            print self.wo[j]
        print ' '

        self.ci = generateMatrix(self.ni, self.nh)
        self.co = generateMatrix(self.nh, self.no)


    def run(self, inputs):
        if len(inputs) != self.ni - 1:
            print 'incorrect number of inputs'

        for i in range(self.ni - 1):
            self.ai[i] = inputs[i]

        for j in range(self.nh):
            sum = 0.0
            for i in range(self.ni):
                sum += ( self.ai[i] * self.wi[i][j] )
            self.ah[j] = sigmoid(sum)

        for k in range(self.no):
            sum = 0.0
```

```
            for j in range(self.nh):
                sum += ( self.ah[j] * self.wo[j][k] )
            self.ao[k] = sigmoid(sum)

        return self.ao

    def backPropagate(self, targets, N, M):
        # calculate deltas in output layers
        # dE/dw[j][k] = (t[k] - ao[k]) * s'( SUM( w[j][k]*ah[j] ) ) * ah[j]
        output_deltas = [0.0] * self.no
        for k in range(self.no):
            error = targets[k] - self.ao[k]
            output_deltas[k] = error * dsigmoid(self.ao[k])

        # update weights in output layers
        for j in range(self.nh):
            for k in range(self.no):
                # output_deltas[k] * self.ah[j] is dError/dweight[j][k]
                change = output_deltas[k] * self.ah[j]
                self.wo[j][k] += N * change + M * self.co[j][k]
                self.co[j][k] = change

        # calculate deltas in hidden layers
        hidden_deltas = [0.0] * self.nh
        for j in range(self.nh):
            error = 0.0
            for k in range(self.no):
                error += output_deltas[k] * self.wo[j][k]
            hidden_deltas[j] = error * dsigmoid(self.ah[j])

        # update weights in input layers
        for i in range(self.ni):
            for j in range(self.nh):
                change = hidden_deltas[j] * self.ai[i]
                # print 'activation',self.ai[i],'synapse',i,j,'change',change
                self.wi[i][j] += N * change + M * self.ci[i][j]
                self.ci[i][j] = change

        # Calculate the sum of squares of error.
        error = 0.0
        for k in range(len(targets)):
            error = 0.5 * (targets[k] - self.ao[k]) ** 2
        return error
    def final_weights(self):
```

```
            print ' '
            print 'Final weights:'
            print '(input layers to hidden layers weights:)'
            for i in range(self.ni):
                if i==self.nh:
                    print self.wi[i], '(Offset node)'
                else:
                    print self.wi[i]
            print '(hidden layers to output layers weights:)'
            for j in range(self.nh):
                print self.wo[j]
            print ''

        def train(self, training_set,N,target_error,max_iterations=1000,M=0.5):
            """
            :param training_set:tranining set
            :param max_iterations:max number of iterations
            :param N:learning rate
            :param M:learning for last time (algorithm optimization)
            :param target_error
            """
            M=N / 2
            for i in range(max_iterations):
                for p in training_set:
                    inputs = p[0]
                    targets = p[1]
                    self.run(inputs)
                    error = self.backPropagate(targets, N, M)
                if i==0:
                    print 'The first-batch error          --> ', error
                if i==max_iterations-1:
                        print 'Can not achieve the target error:',target_error,', please change the
learning rate.'
                        return 0;
                if error<target_error:
                    print 'The final error                --> ', error,'<',target_error
                    print ' '
                    print 'Total number of batches run through in training is:',i+1,'times.'
                    break

def main():
    training_set = [[[0, 0], [0]], [[0, 1], [1]],[[1, 0], [1]],[[1, 1], [0]]]
    target_error = float(raw_input('Please input a float target_error:'))
    learning_rate = float(raw_input('Please input a float learning_rate:'))
```

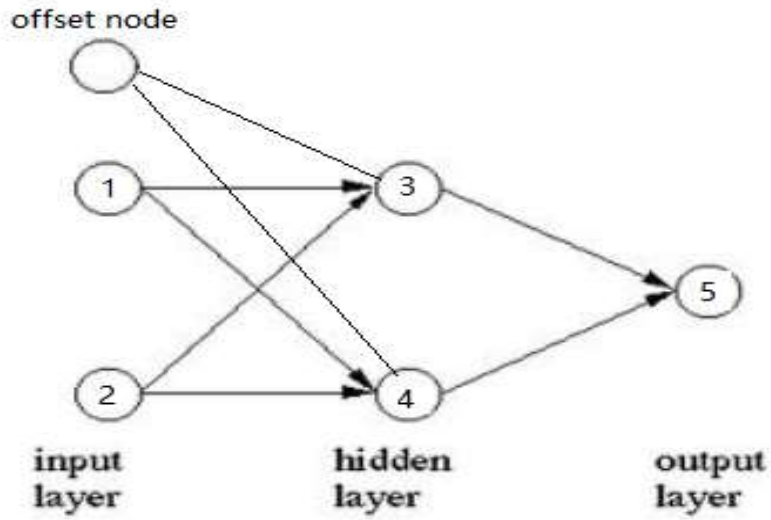
```
    nn = neural_network(2, 2, 1)
    if nn.train(training_set,learning_rate,target_error)!=0:
        nn.final_weights()

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

tips:

In the program, for convenience, I add a separate offset node as the offset placed in the input layer, its value (output, no input) is fixed at 1 and its weight is automatically included in the above weight adjustment.

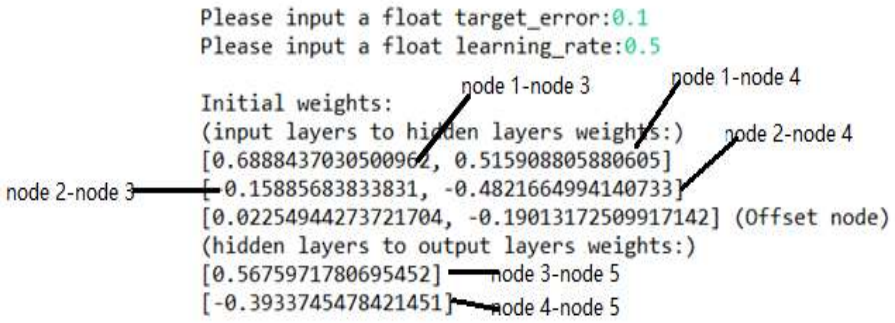
However, if the bias is used as the values that are bound to all neurons, respectively, then offset adjustments are needed without the need for weight adjustments (there are no bias nodes now). It is not as convenient as the first method. Therefore, I adopt the first method.



(2) Screen-shots of the running results

(i) Target Error=0.1

Learning rate=0.5



Learning rate=1

```
Please input a float target_error:0.1
Please input a float learning_rate:1

Initial weights:
(input layers to hidden layers weights:)
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142] (Offset node)
(hidden layers to output layers weights:)
[0.5675971780695452]
[-0.3933745478421451]

The first-batch error      -->  0.157900063277
The final error            -->  0.0997770004478 < 0.1

Total number of batches run through in training is: 272 times.

Final weights:
(input layers to hidden layers weights:)
[1.0279513934896456, 1.831900908252926]
[-1.7694748268661467, -1.8224832559110051]
[-0.4299600384570291, 1.1054533011429584] (Offset node)
(hidden layers to output layers weights:)
[1.3172902132462323]
[-0.7359047098099406]
```

Learning rate=1.3

```
Please input a float target_error:0.1
Please input a float learning_rate:1.3

Initial weights:
(input layers to hidden layers weights:)
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142] (Offset node)
(hidden layers to output layers weights:)
[0.5675971780695452]
[-0.3933745478421451]

The first-batch error      -->  0.161166620301
The final error            -->  0.0988475928189 < 0.1

Total number of batches run through in training is: 213 times.

Final weights:
(input layers to hidden layers weights:)
[1.2328943277073197, 1.6919350024021764]
[-2.224245237055146, -1.961795211598559]
[-0.533709065966133, 1.153500108766297] (Offset node)
(hidden layers to output layers weights:)
[1.41428539672471]
[-0.7110798139521709]
```

Learning rate=1.4

```
Please input a float target_error:0.1
Please input a float learning_rate:1.4

Initial weights:
(input layers to hidden layers weights:)
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142] (Offset node)
(hidden layers to output layers weights:)
[0.5675971780695452]
[-0.3933745478421451]

The first-batch error      -->  0.162296221927
The final error            -->  0.0998313678161 < 0.1

Total number of batches run through in training is: 203 times.

Final weights:
(input layers to hidden layers weights:)
[1.3085130526092132, 1.5912225106899347]
[-2.3908472961472853, -2.0178641913566295]
[-0.5733163554243719, 1.1642026346450909] (Offset node)
(hidden layers to output layers weights:)
[1.4405021623711887]
[-0.6960738063258363]
```

Learning rate=1.5 (best)

```
Please input a float target_error:0.1
Please input a float learning_rate:1.5

Initial weights:
(input layers to hidden layers weights:)
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142] (Offset node)
(hidden layers to output layers weights:)
[0.5675971780695452]
[-0.3933745478421451]

The first-batch error      -->  0.16344640766
The final error            -->  0.0990224192709 < 0.1

Total number of batches run through in training is: 202 times.

Final weights:
(input layers to hidden layers weights:)
[1.5119923321530733, 1.4729624629523952]
[-2.6719897383903684, -2.1335864532965108]
[-0.7108626555720723, 1.1968388219131723] (Offset node)
(hidden layers to output layers weights:)
[1.594222634048278]
[-0.7275419220140864]
```

Learning rate=1.6

```
Please input a float target_error:0.1
Please input a float learning_rate:1.6

Initial weights:
(input layers to hidden layers weights:)
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142] (Offset node)
(hidden layers to output layers weights:)
[0.5675971780695452]
[-0.3933745478421451]

The first-batch error      -->  0.164617308068
The final error            -->  0.0996302112034 < 0.1

Total number of batches run through in training is: 219 times.

Final weights:
(input layers to hidden layers weights:)
[1.9871586987511123, 1.1686923258236515]
[-3.181568901412836, -2.3532652355233075]
[-1.0660107798393081, 1.211354611547192] (Offset node)
(hidden layers to output layers weights:)
[1.9825754688653414]
[-0.8470010283819575]
```

(ii) Target Error=0.02

Learning rate=0.5

```
Please input a float target_error:0.02
Please input a float learning_rate:0.5

Initial weights:
(input layers to hidden layers weights:)
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142] (Offset node)
(hidden layers to output layers weights:)
[0.5675971780695452]
[-0.3933745478421451]

The first-batch error      -->  0.152856273141
The final error            -->  0.0199739085265 < 0.02

Total number of batches run through in training is: 769 times.

Final weights:
(input layers to hidden layers weights:)
[2.9845689133076454, 3.987516883454604]
[-3.2540991141710505, -3.992020468724824]
[-1.649774124431111, 1.5500994890864115] (Offset node)
(hidden layers to output layers weights:)
[4.733032345426561]
[-2.4343824970285364]
```

Learning rate=1

```
Please input a float target_error:0.02
Please input a float learning_rate:1

Initial weights:
(input layers to hidden layers weights:)
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142] (Offset node)
(hidden layers to output layers weights:)
[0.5675971780695452]
[-0.3933745478421451]

The first-batch error      -->  0.157900063277
The final error            -->  0.0197911073566 < 0.02

Total number of batches run through in training is: 356 times.

Final weights:
(input layers to hidden layers weights:)
[3.414936124335938, 3.643757591491449]
[-3.8309918653148705, -3.7774808004538225]
[-1.9220994910353064, 1.4115421531262577] (Offset node)
(hidden layers to output layers weights:)
[4.6435107168102014]
[-2.3365249580116805]
```

Learning rate=2

```
Please input a float target_error:0.02
Please input a float learning_rate:2

Initial weights:
(input layers to hidden layers weights:)
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142] (Offset node)
(hidden layers to output layers weights:)
[0.5675971780695452]
[-0.3933745478421451]

The first-batch error      -->  0.169510774104
The final error            -->  0.0192508826156 < 0.02

Total number of batches run through in training is: 174 times.

Final weights:
(input layers to hidden layers weights:)
[3.3423758661820853, 3.8665825642654434]
[-3.76077249134433, -4.586223438364962]
[1.3826855650586967, -2.1552231536577606] (Offset node)
(hidden layers to output layers weights:)
[-2.381887120259491]
[4.776829117726542]
```

Learning rate=5

```
Please input a float target_error:0.02
Please input a float learning_rate:5

Initial weights:
(input layers to hidden layers weights:)
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142] (Offset node)
(hidden layers to output layers weights:)
[0.5675971780695452]
[-0.3933745478421451]

The first-batch error      -->  0.217809458711
The final error            -->  0.0187425137916 < 0.02

Total number of batches run through in training is: 57 times.

Final weights:
(input layers to hidden layers weights:)
[3.2991636201727443, 4.461965412009957]
[-4.317439215097667, -5.791108522567948]
[1.7422040720332599, -2.4872187461585296] (Offset node)
(hidden layers to output layers weights:)
[-2.5818335138074238]
[5.293744758497689]
```

Learning rate=9,(best)

```
Please input a float target_error:0.02
Please input a float learning_rate:9

Initial weights:
(input layers to hidden layers weights:)
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142] (Offset node)
(hidden layers to output layers weights:)
[0.5675971780695452]
[-0.3933745478421451]

The first-batch error      -->  0.316207796076
The final error            -->  0.0163669882026 < 0.02

Total number of batches run through in training is: 31 times.

Final weights:
(input layers to hidden layers weights:)
[3.7310477017795676, 4.959864039612585]
[-4.87967014894296, -6.6333247407073905]
[2.01678068741914, -2.8109561103369884] (Offset node)
(hidden layers to output layers weights:)
[-2.7344201206709706]
[5.670635191696419]
```

Learning rate=10

```
Please input a float target_error:0.02
Please input a float learning_rate:10

Initial weights:
(input layers to hidden layers weights:)
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142] (Offset node)
(hidden layers to output layers weights:)
[0.5675971780695452]
[-0.3933745478421451]

The first-batch error      -->  0.345130900605
The final error            -->  0.0145670404083 < 0.02

Total number of batches run through in training is: 105 times.

Final weights:
(input layers to hidden layers weights:)
[3.924779783274931, 6.120482370830499]
[-4.062233115747465, -7.4568931576600335]
[1.1789494760093553, -4.366272379982039] (Offset node)
(hidden layers to output layers weights:)
[-2.586885975694859]
[6.016100097624181]
```

2. JavaScript

Just open the html file to test or use Nodepad++ to find the codes.

(1) Codes:

```
<html>
<head>
<title> New Document </title>
<meta charset="utf-8">
<style type="text/css">
      *{
          margin:0px;
          padding:0px;
      }
      body table{
            border:1px solid black;
      }
      th{
            border:1px solid black;
            text-align:center;
            line-height:center;
      }
      td{
            border:1px solid black;
            text-align:center;
            line-height:center;
      }
   </style>
<script type="text/javascript">
function volume()
 {
   var volume;
   var radius = document.getElementById('r').value;
   var height = document.getElementById('h').value;
   radius = Math.abs(radius);
   if(document.getElementById('MyForm').shape.value=="cylinder")
   {volume = height * Math.PI * Math.pow(radius, 2);}
   if(document.getElementById('MyForm').shape.value=="cone")
   {volume=height * Math.PI * Math.pow(radius, 2)/3;}
   if(document.getElementById('MyForm').shape.value=="sphere")
   {volume = (4/3) * Math.PI * Math.pow(radius, 3);}
   volume = volume.toFixed(15);
   document.getElementById('vol').innerHTML =volume;
   document.getElementById('height').innerHTML =height;
   document.getElementById('radius').innerHTML =radius;
```

```
}
  function print1()
  {
    if(document.getElementById('MyForm').units[0].checked)
    { document.getElementById("p1").innerHTML="English";
        document.getElementById("cal_unit1").innerHTML="ft";
        document.getElementById("cal_unit2").innerHTML="ft";
        document.getElementById("cal_unit3").innerHTML="ft";
    }
    if(document.getElementById('MyForm').units[1].checked)
    {    document.getElementById("p1").innerHTML="SI";
      document.getElementById("cal_unit1").innerHTML="m";
        document.getElementById("cal_unit2").innerHTML="m";
        document.getElementById("cal_unit3").innerHTML="m";
        }
  }
  function print2()
  {
    if(document.getElementById('MyForm').shape.value=="cylinder")
    { document.getElementById("p2").innerHTML="Cylinder";
        document.getElementById("type_show1").innerHTML="cylinder"}
    if(document.getElementById('MyForm').shape.value=="cone")
    {    document.getElementById("p2").innerHTML="Cone";
    document.getElementById("type_show1").innerHTML="cone"}
    if(document.getElementById('MyForm').shape.value=="sphere")
    {    document.getElementById("p2").innerHTML="Sphere";
         document.getElementById("type_show1").innerHTML="sphere"}
  }

  function reset() {
              var x = document.forms["MyForm"];
              x.r.value = "";
              x.h.value = "";
              document.getElementById('type_show1').innerHTML = "";
              document.getElementById('vol').innerHTML = "";
              document.getElementById('radius').innerHTML = "";
              document.getElementById('height').innerHTML = "";
              document.getElementById("cal_unit1").innerHTML = "ft";
              document.getElementById("cal_unit2").innerHTML = "ft";
              document.getElementById("cal_unit3").innerHTML = "ft";
              document.getElementById('type_show1').innerHTML = "cylinder";
      }
      function shapeClick() {
```

```
   var type = document.getElementById('MyForm').shape.value;
   document.getElementById('type_show1').innerHTML =type;
   document.getElementById('p2').innerHTML =type;
}
       function unitsClick() {
   if(document.getElementById('MyForm').units[0].checked)
    { document.getElementById("p1").innerHTML="English";
        document.getElementById("cal_unit1").innerHTML="ft";
        document.getElementById("cal_unit2").innerHTML="ft";
        document.getElementById("cal_unit3").innerHTML="ft";
    }
    if(document.getElementById('MyForm').units[1].checked)
    {   document.getElementById("p1").innerHTML="SI";
      document.getElementById("cal_unit1").innerHTML="m";
        document.getElementById("cal_unit2").innerHTML="m";
        document.getElementById("cal_unit3").innerHTML="m";
        }
}
</script>
</head>
<body>
<h1>This web site will find the volume<br>
for a Cylinder, Sphere or Cone</h1>
<br>
<form action="javascript:return false;" id="MyForm">
Select the units(English or SI)<br>
<input type="radio" name="units" value="English" onChange = "unitsClick()" checked/>English
<input type="radio" name="units" value="SI" onChange = "unitsClick()" />SI<br><br>
Select the shape
<select id="Shape" name="shape" onChange = "shapeClick()">
                    <option value="cylinder" id="cylinder" selected>cylinder</option>
                    <option value="cone" id="cone" >cone</option>
                    <option value="sphere" id="sphere" >sphere</option>
</select><br><br>
Enter the radius:<input id="r" name="radius"style="text-align:right"></input><br><br>
For the cylinder and cone, Enter the height:<input id="h" name="height" style="text-
align:right"></input><br><br>
<!--<input type="reset" value="reset the form" /><br><br>-->
</form>
<button onclick="reset()">reset the form</button><br><br>
<h1>Results</h1><br>
You selected to use <span id="p1">English</span> units<br><br>
You selected to find the value for a <span id="p2">cylinder</span> shape<br> <br>
```

```
<table border="1">
    <tr>
        <td>Shape</td>
        <td>Radius</td>
        <td>Height</td>
        <td>Volume</td>
    </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">shape</span></td>
        <td><span id="radius"></span></td>
        <td><span id="height"></span></td>
        <td><span id="vol"></span></td>
    </tr>
</table>

<button onclick="volume();print1();print2()">Calculate</button><br><br>
</body>
</html>
```

(2) Test Results:

This web site will find the volume for a Cylinder, Sphere or Cone

Select the units(English or SI)
◉English ○SI

Select the shape cylinder ▾

Enter the radius: 1

For the cylinder and cone, Enter the height: 2

reset the form

Results

You selected to use English units

You selected to find the value for a Cylinder shape

Shape	Radius	Height	Volume
	(ft)	(ft)	(ft^3)
cylinder	1	2	6.283185307179586

Calculate

This web site 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: 2

For the cylinder and cone, Enter the height: 3

reset the form

Results

You selected to use English units

You selected to find the value for a Cone shape

Shape	Radius	Height	Volume
	(ft)	(ft)	(ft^3)
cone	2	3	12.566370614359172

Calculate

This web site will find the volume for a Cylinder, Sphere or Cone

Select the units(English or SI)
○English ◉SI

Select the shape sphere ▾

Enter the radius: 4

For the cylinder and cone, Enter the height: 0

reset the form

Results

You selected to use SI units

You selected to find the value for a Sphere shape

Shape	Radius	Height	Volume
	(m)	(m)	(m^3)
sphere	4	0	268.082573106328994

Calculate