**HW4**

**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 j 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()

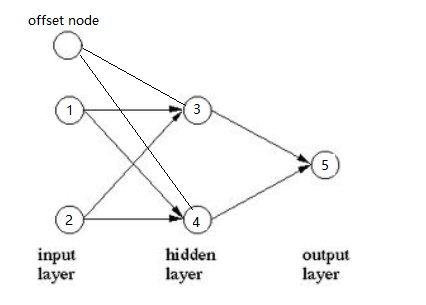
if \_\_name\_\_ == *"\_\_main\_\_"*:

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.

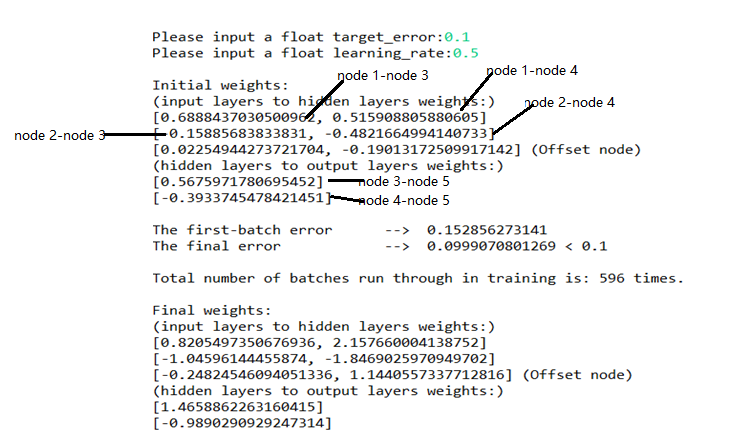
However, if the bias is used as the values that are bound to all neurons, respectively, then offset adjustments are needed without the need for weight adjustments (there are no bias nodes now). It is not as convenient as the first method. Therefore, I adopt the first method.



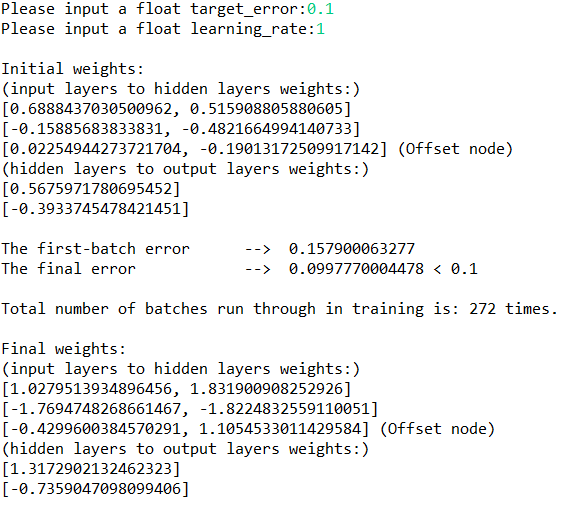
(2) Screen-shots of the running results

**(i) Target Error=0.1**

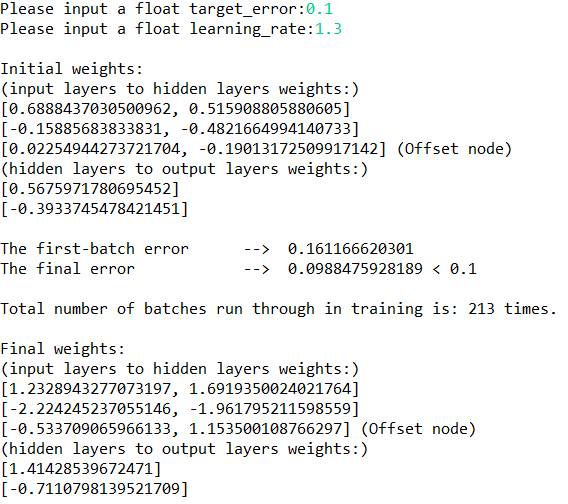
Learning rate=0.5



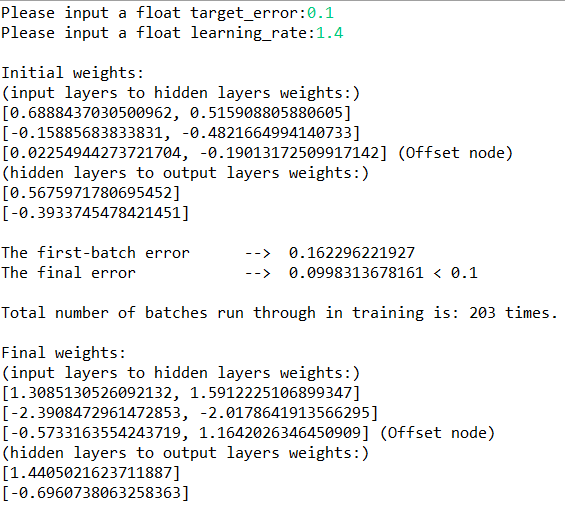
Learning rate=1



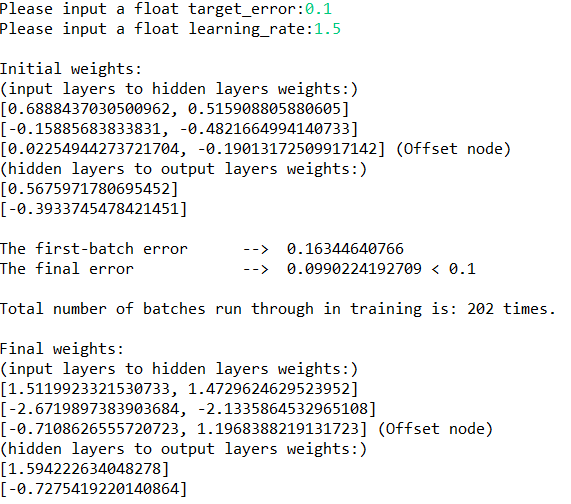
Learning rate=1.3

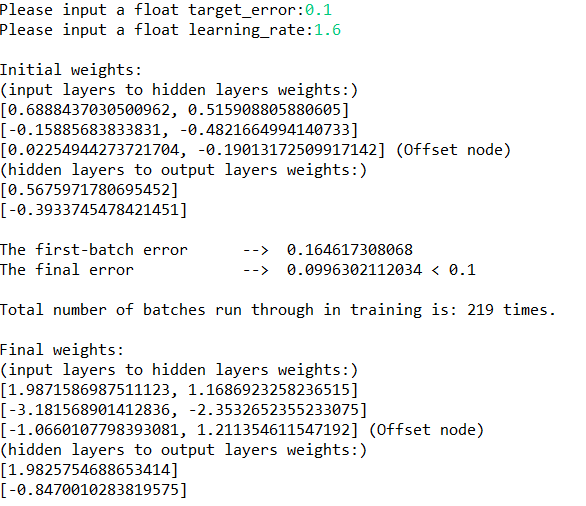


Learning rate=1.4



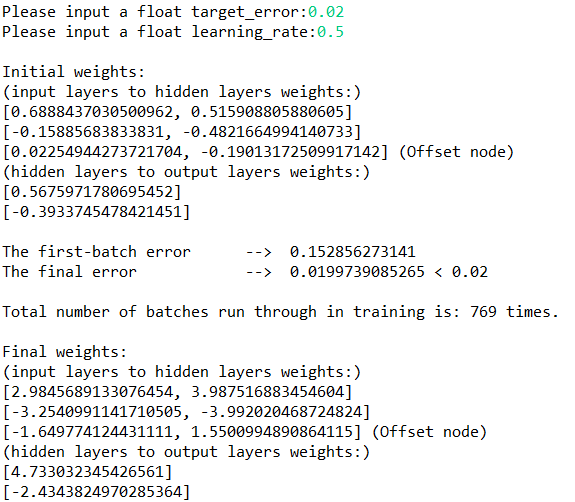
Learning rate=1.5 (best)

  
Learning rate=1.6

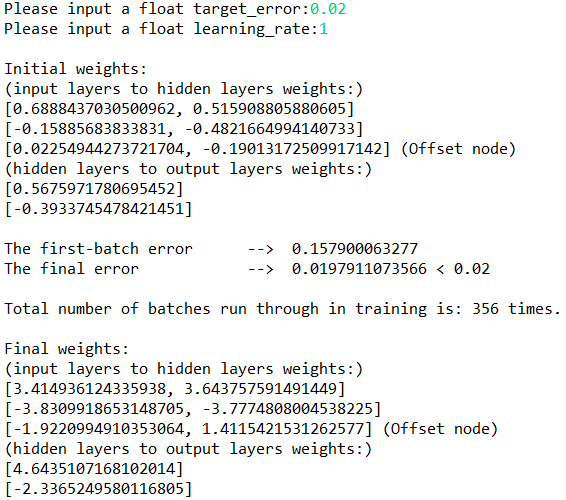


**(ii) Target Error=0.02**

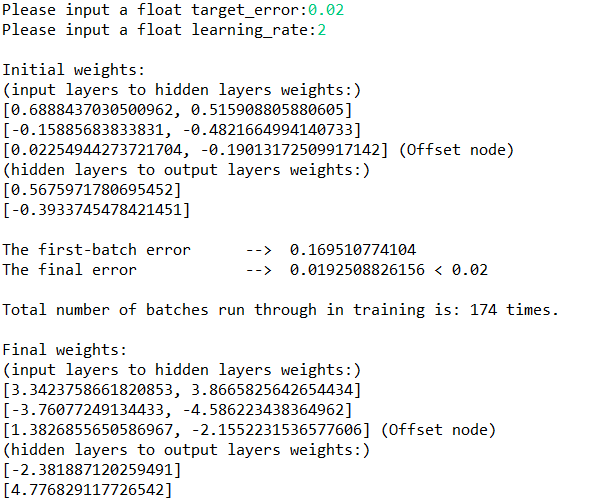
Learning rate=0.5



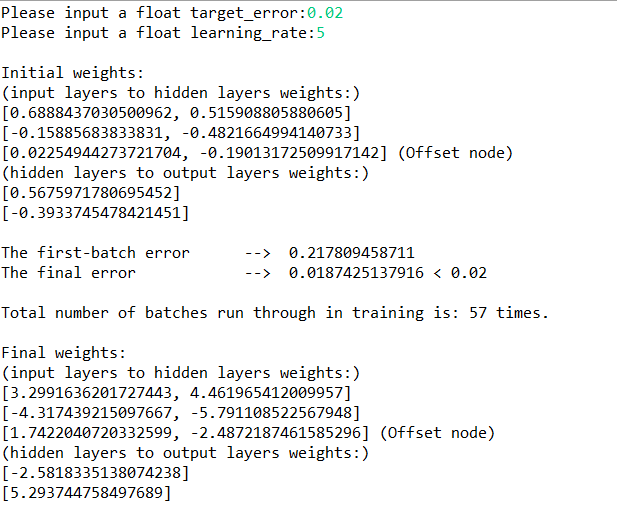
Learning rate=1



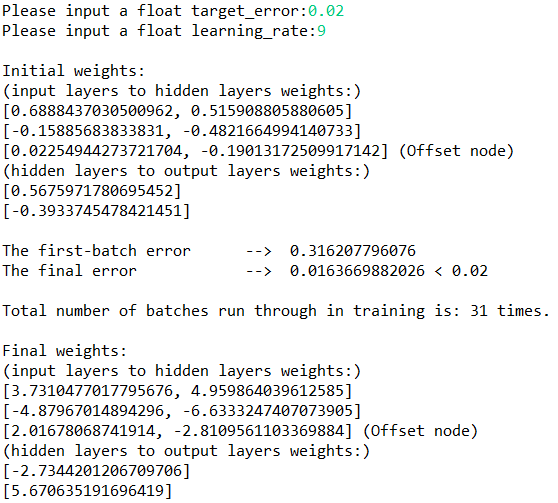
Learning rate=2



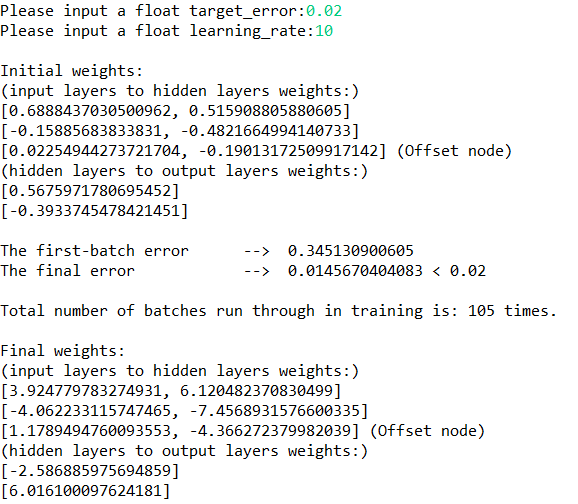
Learning rate=5



Learning rate=9,(best)



Learning rate=10



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>

1. Test Results:

