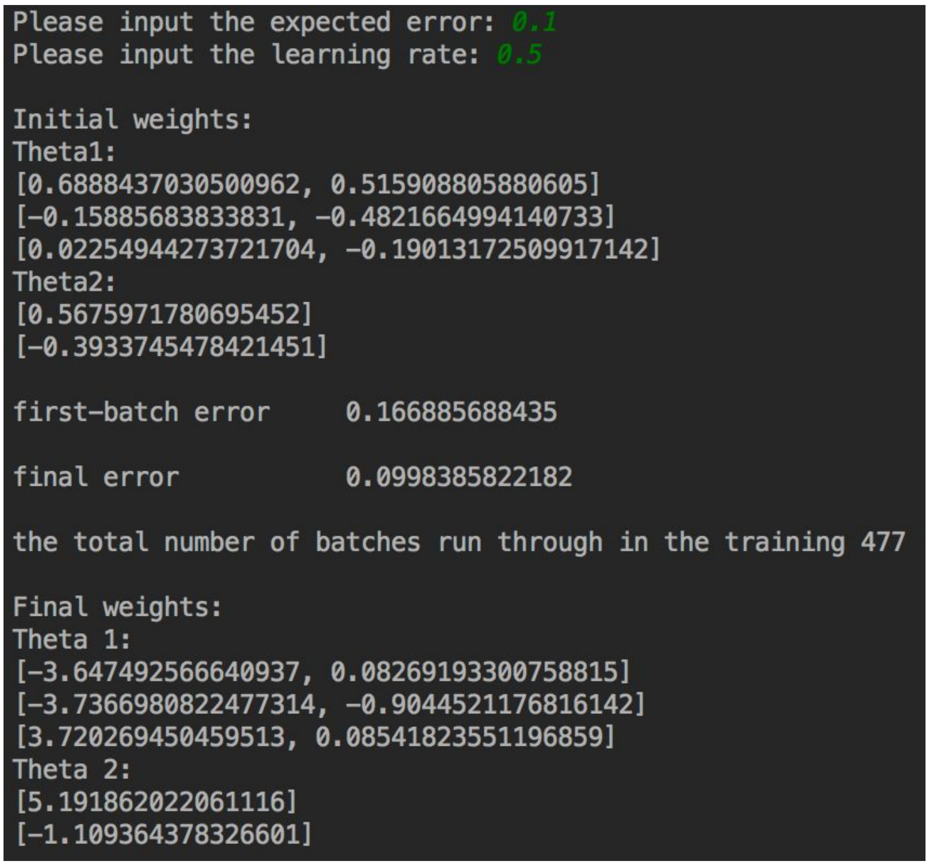
Q1:

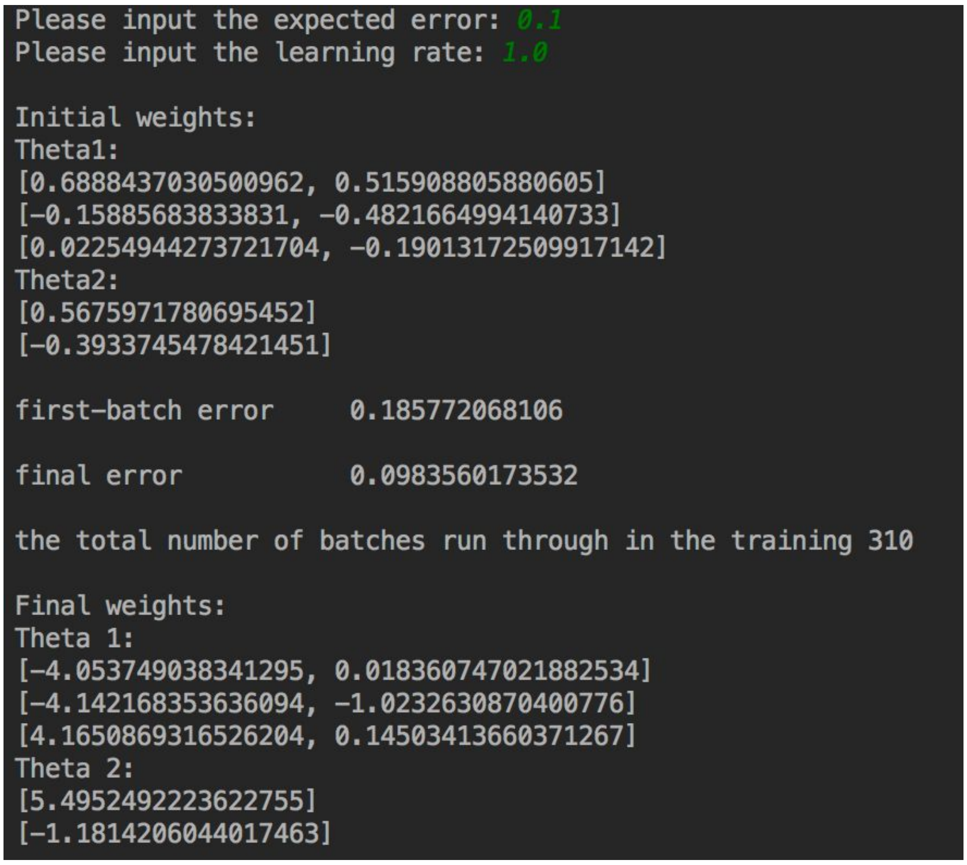
(1) Source Code:

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

random.seed(0)

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


def dsigmoid(y):
    """
    sigmoid function
    :param y:
    :return:
    """
    return y * (1 - y)

def randomNum(a, b):
    """
    create a random number between a and b
    :param a:
    :param b:
    :return:
    """
    return (b - a) * random.random() + a

def constructMatrix(I, J, fill=0.0):
    """
    create the matrix
    :param I: number of row
    :param J: number of column
    :param fill: value of element
    :return: the matrix
    """
    m = []
    for i in range(I):
        m.append([fill] * J)
    return m


def randomizeMatrix(matrix, a, b):
    """
    randomize the matrix
    :param matrix:
    :param a:
    :param b:
    """
    for i in range(len(matrix)):
        for j in range(len(matrix[0])):
            matrix[i][j] = random.uniform(a, b)


class NN:
    def __init__(self, ni, nh, no):
        # number of input, hidden, and output nodes
        """
        construct the neural network
        :param ni:number of input unit
        :param nh:number of hidden layer unit
        :param no:number of output unit
        """
        self.ni = ni + 1
        self.nh = nh
        self.no = no

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

        # weight matrix
        self.wi = constructMatrix(self.ni, self.nh)  # Theta1
        self.wo = constructMatrix(self.nh, self.no)  # Theta2

        randomizeMatrix(self.wi, -1, 1)
        randomizeMatrix(self.wo, -1, 1)
        print "\n" + 'Initial weights:'
        print 'Theta1: '
        for i in range(self.ni):
            print self.wi[i]
        print 'Theta2: '
        for j in range(self.nh):
            print self.wo[j]

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

    def runNN(self, inputs):
        """
        forward propagation
        :param inputs:
        :return:
        """
        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):
        """
        backpropagation
        :param targets:
        :param N: learning rate
        :param M: old learning rate
        :return:
        """

        # calculate the delta for output layer
        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 the Theta2
        for j in range(self.nh):
            for k in range(self.no):
                # output_deltas[k] * self.ah[j] 才是 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 the delta for hidden layer
        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 the Theta1
        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

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

    def weights(self):
        """
        print the weight
        """
        print "\n" + 'Final weights:'
        print 'Theta 1: '
        for i in range(self.ni):
            print self.wi[i]
        print 'Theta 2: '
        for j in range(self.nh):
            print self.wo[j]
        print ''
    def test(self, patterns):
        """
        test
        :param patterns:test
        """
        print "\n"
        for p in patterns:
            inputs = p[0]
            # print 'Inputs:', p[0], '-->', self.runNN(inputs),
'\tTarget', p[1]
            print 'Inputs:', p[0], '-->', self.runNN(inputs),
'Target'.rjust(10), p[1]

    def train(self, patterns, max_iterations=1000, N=0.5, M=0.1):
        """
        train
        :param patterns: the batch
        :param max_iterations:
        :param N: learning rate
        :param M: last time learning rate
        """
        N = learningRate
        for i in range(max_iterations):
            for p in patterns:
                inputs = p[0]
                targets = p[1]
                self.runNN(inputs)
                error = self.backPropagate(targets, N, M)

            if i == 0:
                print "\n" + 'first-batch error    ', error

            if error < expectedError:
                print "\n" + 'final error          ', error
                print "\n" + 'the total number of batches run through in
the training', i + 1
                break
        self.test(patterns)


def main():
    pat = [
        [[0, 0], [1]],
        [[0, 1], [1]],
        [[1, 0], [1]],
        [[1, 1], [0]]
    ]
    global expectedError
    global learningRate
    expectedError = input('Please input the expected error: ')
    learningRate = input('Please input the learning rate: ')
    myNN = NN(2, 2, 1)
    myNN.train(pat)
    myNN.weights()


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

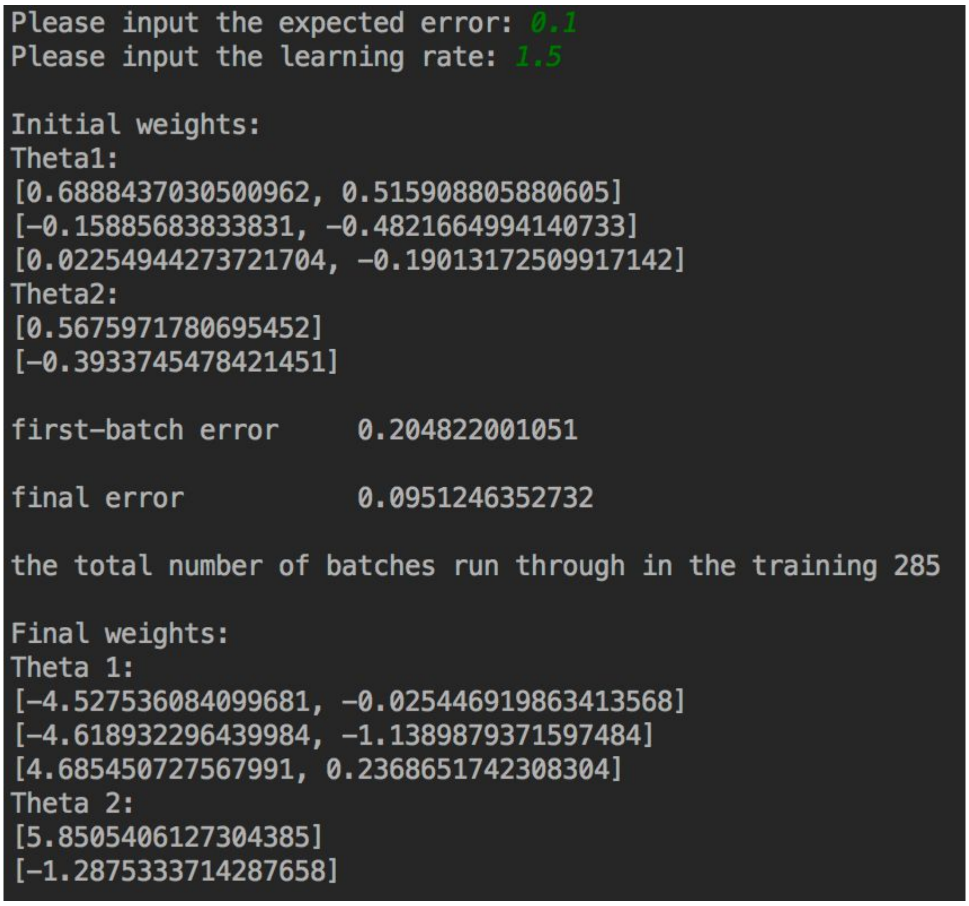
(2)
(a)



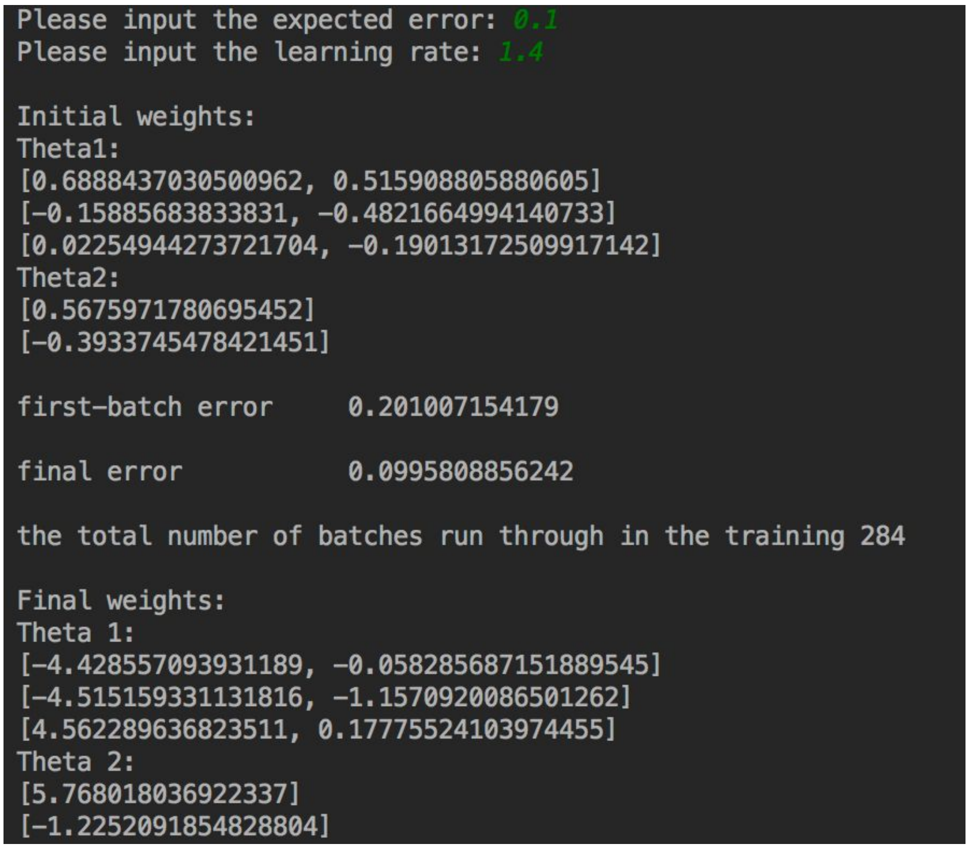
```
Please input the expected error: 0.1
Please input the learning rate: 0.5

Initial weights:
Theta1:
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142]
Theta2:
[0.5675971780695452]
[-0.3933745478421451]

first-batch error     0.166885688435

final error           0.0998385822182

the total number of batches run through in the training 477

Final weights:
Theta 1:
[-3.647492566640937, 0.08269193300758815]
[-3.7366980822477314, -0.9044521176816142]
[3.720269450459513, 0.08541823551196859]
Theta 2:
[5.191862022061116]
[-1.109364378326601]
```



```
Please input the expected error: 0.1
Please input the learning rate: 1.0

Initial weights:
Theta1:
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142]
Theta2:
[0.5675971780695452]
[-0.3933745478421451]

first-batch error     0.185772068106

final error           0.0983560173532

the total number of batches run through in the training 310

Final weights:
Theta 1:
[-4.053749038341295, 0.018360747021882534]
[-4.142168353636094, -1.0232630870400776]
[4.1650869316526204, 0.14503413660371267]
Theta 2:
[5.4952492223622755]
[-1.1814206044017463]
```



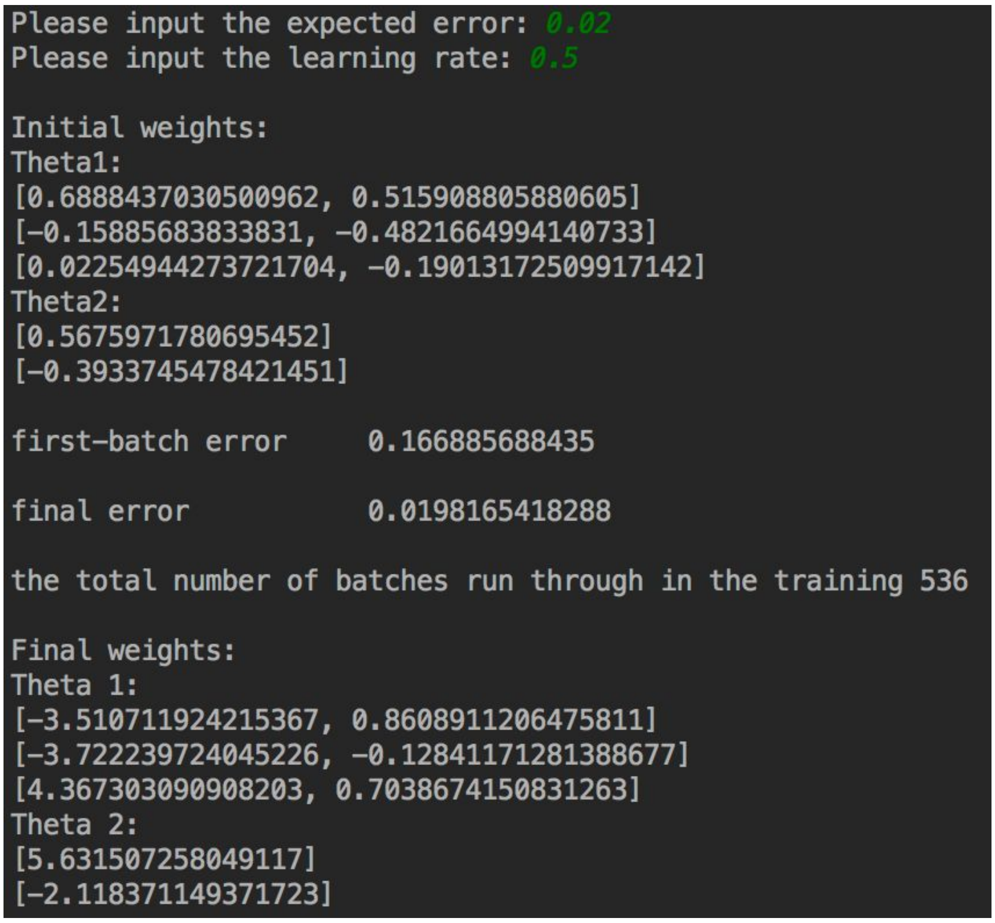
```
Please input the expected error: 0.1
Please input the learning rate: 1.5

Initial weights:
Theta1:
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142]
Theta2:
[0.5675971780695452]
[-0.3933745478421451]

first-batch error     0.204822001051

final error           0.0951246352732

the total number of batches run through in the training 285

Final weights:
Theta 1:
[-4.527536084099681, -0.025446919863413568]
[-4.618932296439984, -1.1389879371597484]
[4.685450727567991, 0.2368651742308304]
Theta 2:
[5.8505406127304385]
[-1.2875333714287658]
```



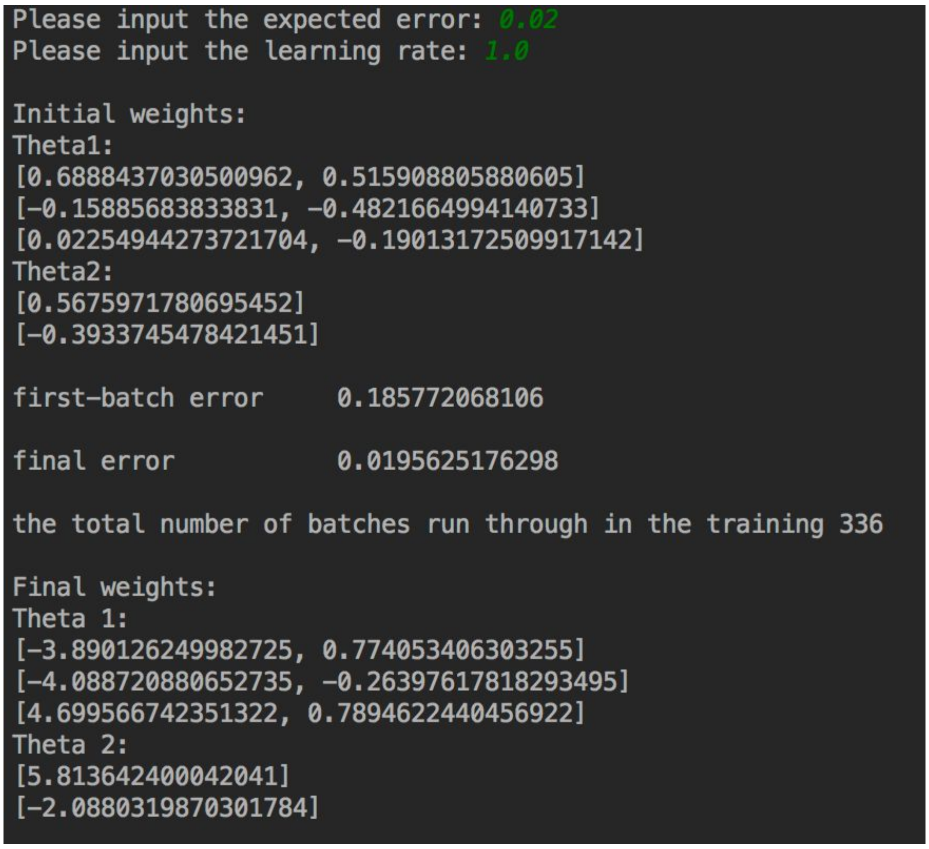
```
Please input the expected error: 0.1
Please input the learning rate: 1.4

Initial weights:
Theta1:
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142]
Theta2:
[0.5675971780695452]
[-0.3933745478421451]

first-batch error     0.201007154179

final error           0.0995808856242

the total number of batches run through in the training 284

Final weights:
Theta 1:
[-4.428557093931189, -0.058285687151889545]
[-4.515159331131816, -1.1570920086501262]
[4.562289636823511, 0.17775524103974455]
Theta 2:
[5.768018036922337]
[-1.2252091854828804]
```

After several tries, when learning rate equals to 1.4, I come up with the best result(minimum training time)

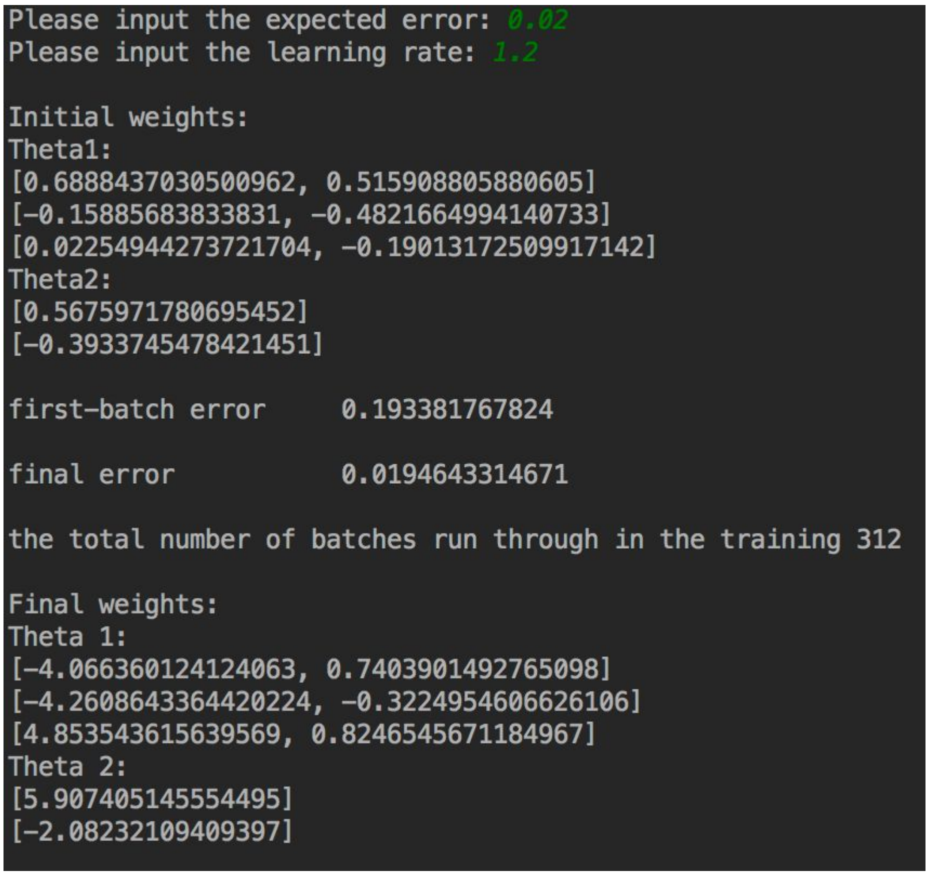
(b)



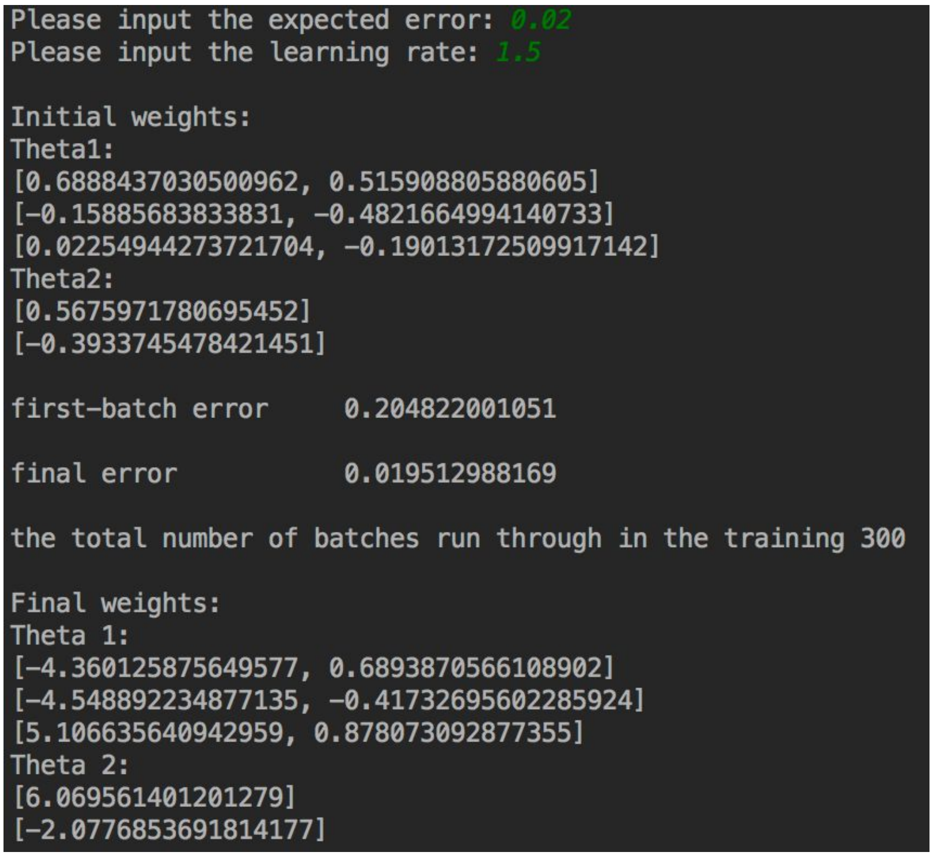
```
Please input the expected error: 0.02
Please input the learning rate: 0.5

Initial weights:
Theta1:
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142]
Theta2:
[0.5675971780695452]
[-0.3933745478421451]

first-batch error      0.166885688435

final error            0.0198165418288

the total number of batches run through in the training 536

Final weights:
Theta 1:
[-3.510711924215367, 0.8608911206475811]
[-3.722239724045226, -0.12841171281388677]
[4.367303090908203, 0.7038674150831263]
Theta 2:
[5.631507258049117]
[-2.118371149371723]
```



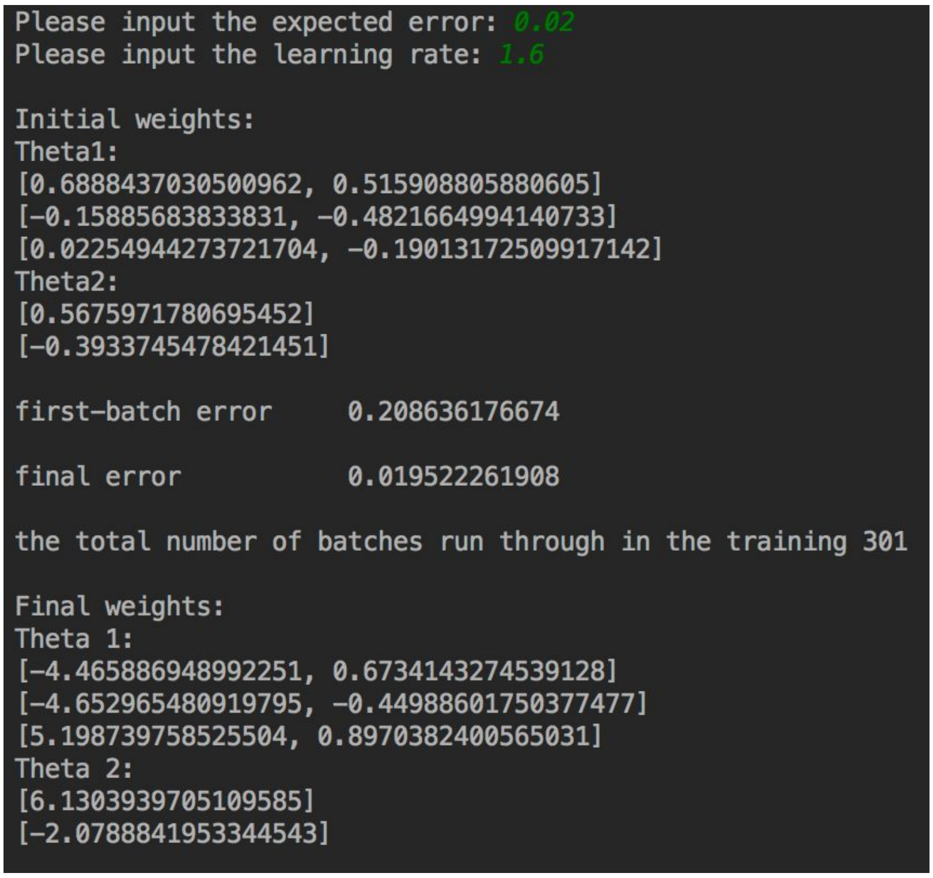
```
Please input the expected error: 0.02
Please input the learning rate: 1.0

Initial weights:
Theta1:
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142]
Theta2:
[0.5675971780695452]
[-0.3933745478421451]

first-batch error     0.185772068106

final error           0.0195625176298

the total number of batches run through in the training 336

Final weights:
Theta 1:
[-3.890126249982725, 0.774053406303255]
[-4.088720880652735, -0.26397617818293495]
[4.699566742351322, 0.7894622440456922]
Theta 2:
[5.813642400042041]
[-2.0880319870301784]
```



```
Please input the expected error: 0.02
Please input the learning rate: 1.2

Initial weights:
Theta1:
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142]
Theta2:
[0.5675971780695452]
[-0.3933745478421451]

first-batch error     0.193381767824

final error           0.0194643314671

the total number of batches run through in the training 312

Final weights:
Theta 1:
[-4.066360124124063, 0.7403901492765098]
[-4.2608643364420224, -0.3224954606626106]
[4.853543615639569, 0.8246545671184967]
Theta 2:
[5.907405145554495]
[-2.08232109409397]
```



```
Please input the expected error: 0.02
Please input the learning rate: 1.5

Initial weights:
Theta1:
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142]
Theta2:
[0.5675971780695452]
[-0.3933745478421451]

first-batch error     0.204822001051

final error           0.019512988169

the total number of batches run through in the training 300

Final weights:
Theta 1:
[-4.360125875649577, 0.6893870566108902]
[-4.548892234877135, -0.41732695602285924]
[5.106635640942959, 0.878073092877355]
Theta 2:
[6.069561401201279]
[-2.0776853691814177]
```



```
Please input the expected error: 0.02
Please input the learning rate: 1.6

Initial weights:
Theta1:
[0.6888437030500962, 0.515908805880605]
[-0.15885683833831, -0.4821664994140733]
[0.02254944273721704, -0.19013172509917142]
Theta2:
[0.5675971780695452]
[-0.3933745478421451]

first-batch error     0.208636176674

final error           0.019522261908

the total number of batches run through in the training 301

Final weights:
Theta 1:
[-4.465886948992251, 0.6734143274539128]
[-4.652965480919795, -0.44988601750377477]
[5.198739758525504, 0.8970382400565031]
Theta 2:
[6.1303939705109585]
[-2.0788841953344543]
```

After several tries, when learning rate equals to 1.5, I come up with the best result(minimum training time)

Q2 :

```
<!DOCTYPE html>
<html>
<head>
  <style>
    table,th,td {
      border: 1px solid black;
    }
  </style>
<title>Homework5</title>
</head>
<body>

<h1>This web site will find the volume for a Cylinder,Sphere or
Cone</h1>

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

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

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

    For the cylinder and cone, Enter the height
    <input id = "h" name = "Height" oninput =
"heightInput(this)"></input>
  </form>

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

<h1>Results</h1>

You selected to use <span id="unit_show">English</span> units<br>
You selected to find the value for a <span
id="type_show">cylinder</span> shape<br>

<table style = "width:30%">
  <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 the 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_unit2").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 input height!");
      return;
    }
    v = 3.1415 * radius * radius * height;
  } else if (type == "Sphere") {
    v = 4/3 * radius * radius * radius * 3.1415;
  } else if (type == "Cone") {
    if (height == "") {
      window.alert("Please input height!");
      return;
    }
    v = 1/3 * radius * radius * 3.1415 * height;
  }

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