机器学习实验报告

实验内容：神经网络手写字符识别

班级：计科1402班

姓名：张崇

学号：0902140229

目录

一、实验内容

神经网络识别手写字符

二、实验原理

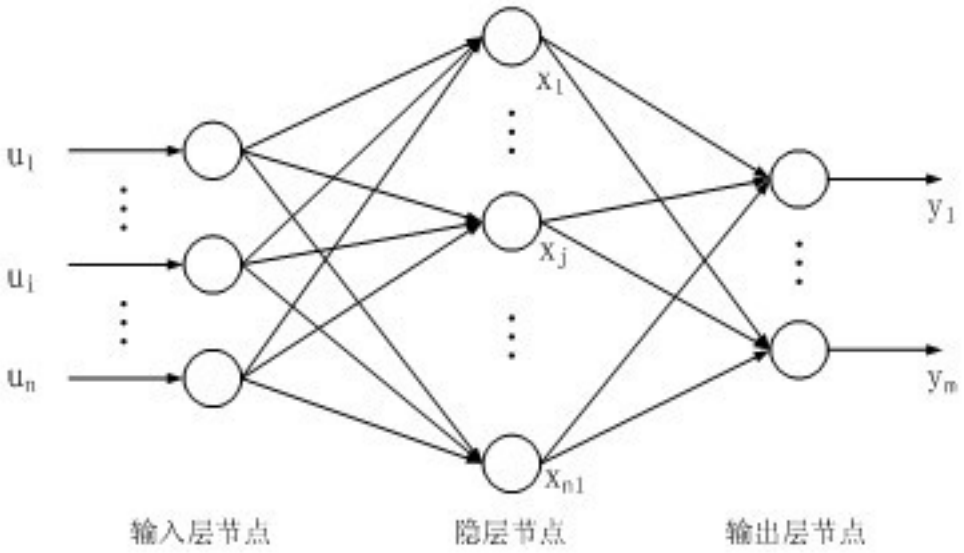
1.神经网络

神经网络由能够互相通信的节点构成，赫布理论解释了人体的神经网络是如何通过改变自身的结构和神经连接的强度来记忆某种模式的。而人工智能中的神经网络与此类似。请看下图，最左一列蓝色节点是输入节点，最右列节点是输出节点，中间节点是隐藏节点。该图结构是分层的，隐藏的部分有时候也会分为多个隐藏层。如果使用的层数非常多就会变成我们平常说的深度学习了。

每一层（除了输入层）的节点由前一层的节点加权加相加加偏置向量并经过激活函数得到，公式如下：

$$a_2^{(2)} = f(w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + b_2^{(1)})$$

其中f是激活函数，b是偏置向量。



这一类拓扑结构的神经网络称作前馈神经网络，因为该结构中不存在回路。有输出反馈给输入的神经网络称作递归神经网络（RNN）。在本实验中使用前馈神经网络中经典的BP神经网络来实现手写识别系统。

2.如何使用神经网络

神经网络属于监督学习，那么多半就三件事，决定模型参数，通过数据集训练学习，训练好后就能到分类工具/识别系统用了。数据集可以分为2部分（训练集，验证集）
系统通过对比训练集的正确答案和自己的解答来不断学习改良自己，提升验证集中的准确率

3.项目结构

```
├── data.csv
├── dataLabels.csv
├── neural_network_design.py
├── nn.json
├── ocr.html
├── ocr.js
├── ocr.py
└── server.py
```

其中，
客户端：（ocr.js）
服务器：（server.py）
用户接口：（ocr.html）
神经网络：(ocr.py)
神经网络设计脚本：(neural_network_design.py)

用户接口(ocr.html)是一个html页面，用户在canvans上写数字，之后点击选择训练或是预测。客户端(ocr.js)将收集到的手写数字组合成一个数组发送给服务器端(server.py)处理，服务器调用神经网络模块(ocr.py)，它会在初始化时通过已有的数据集训练一个神经网络，神经网络的信息会被保存在文件中，等之后再一次启动时使用。最后，神经网络设计脚本(neural_network_design.py)是用来测试不同隐藏节点数下的性能，决定隐藏节点数用的。

三、实验过程

1.实验程序入口

```
<!DOCTYPE html>
<html>
<head>
    <script src="ocr.js"></script>
</head>
<body onload="ocrDemo.onLoadFunction()">
    <div id="main-container" style="text-align: center;">
        <h1>OCR Demo</h1>
        <canvas id="canvas" width="200" height="200"></canvas>
        <form name="input">
            <p>Digit: <input id="digit" type="text"> </p>
            <input type="button" value="Train" onclick="ocrDemo.train()">
            <input type="button" value="Test" onclick="ocrDemo.test()">
            <input type="button" value="Reset"
onclick="ocrDemo.resetCanvas();"/>
        </form>
    </div>
</body>
</html>
```

2.实现客服端

```
var ocrDemo = {
    CANVAS_WIDTH: 200,
    TRANSLATED_WIDTH: 20,
    PIXEL_WIDTH: 10, // TRANSLATED_WIDTH = CANVAS_WIDTH / PIXEL_WIDTH
    BATCH_SIZE: 1,

    PORT: "8000",
    HOST: "http://localhost",

    BLACK: "#000000",
    BLUE: "#0000ff",

    trainArray: [],
    trainingRequestCount: 0,

    onLoadFunction: function() {
        this.resetCanvas();
    },

    resetCanvas: function() {
        var canvas = document.getElementById('canvas');
        var ctx = canvas.getContext('2d');
```

```
        this.data = [];
        ctx.fillStyle = this.BLACK;
        ctx.fillRect(0, 0, this.CANVAS_WIDTH, this.CANVAS_WIDTH);
        var matrixSize = 400;
        while (matrixSize--) this.data.push(0);
        this.drawGrid(ctx);

        canvas.onmousemove = function(e) { this.onMouseMove(e, ctx,
canvas) }.bind(this);
        canvas.onmousedown = function(e) { this.onMouseDown(e, ctx,
canvas) }.bind(this);
        canvas.onmouseup = function(e) { this.onMouseUp(e,
ctx) }.bind(this);
    },

    drawGrid: function(ctx) {
        for (var x = this.PIXEL_WIDTH, y = this.PIXEL_WIDTH; x <
this.CANVAS_WIDTH; x += this.PIXEL_WIDTH, y += this.PIXEL_WIDTH) {
            ctx.strokeStyle = this.BLUE;
            ctx.beginPath();
            ctx.moveTo(x, 0);
            ctx.lineTo(x, this.CANVAS_WIDTH);
            ctx.stroke();

            ctx.beginPath();
            ctx.moveTo(0, y);
            ctx.lineTo(this.CANVAS_WIDTH, y);
            ctx.stroke();
        }
    },

    onMouseMove: function(e, ctx, canvas) {
        if (!canvas.isDrawing) {
            return;
        }
        this.fillSquare(ctx, e.clientX - canvas.offsetLeft, e.clientY -
canvas.offsetTop);
    },

    onMouseDown: function(e, ctx, canvas) {
        canvas.isDrawing = true;
        this.fillSquare(ctx, e.clientX - canvas.offsetLeft, e.clientY -
canvas.offsetTop);
    },

    onMouseUp: function(e) {
        canvas.isDrawing = false;
    },

    fillSquare: function(ctx, x, y) {
        var xPixel = Math.floor(x / this.PIXEL_WIDTH);
        var yPixel = Math.floor(y / this.PIXEL_WIDTH);
```

```
        this.data[((xPixel - 1)  * this.TRANSLATED_WIDTH + yPixel) - 1] =
1;

        ctx.fillStyle = '#ffffff';
        ctx.fillRect(xPixel * this.PIXEL_WIDTH, yPixel *
this.PIXEL_WIDTH, this.PIXEL_WIDTH, this.PIXEL_WIDTH);
    },

    train: function() {
        var digitVal = document.getElementById("digit").value;
        if (!digitVal || this.data.indexOf(1) < 0) {
            alert("Please type and draw a digit value in order to train
the network");
            return;
        }
        this.trainArray.push({"y0": this.data, "label":
parseInt(digitVal)});
        this.trainingRequestCount++;

        // Time to send a training batch to the server.
        if (this.trainingRequestCount == this.BATCH_SIZE) {
            alert("Sending training data to server...");
            var json = {
                trainArray: this.trainArray,
                train: true
            };

            this.sendData(json);
            this.trainingRequestCount = 0;
            this.trainArray = [];
        }
    },

    test: function() {
        if (this.data.indexOf(1) < 0) {
            alert("Please draw a digit in order to test the network");
            return;
        }
        var json = {
            image: this.data,
            predict: true
        };
        this.sendData(json);
    },

    receiveResponse: function(xmlHttp) {
        if (xmlHttp.status != 200) {
            alert("Server returned status " + xmlHttp.status);
            return;
        }
        var responseJSON = JSON.parse(xmlHttp.responseText);
        if (xmlHttp.responseText && responseJSON.type == "test") {
```

```
            alert("The neural network predicts you wrote a \'" +
responseJSON.result + '\'');
        }
    },

    onError: function(e) {
        alert("Error occurred while connecting to server: " +
e.target.statusText);
    },

    sendData: function(json) {
        var xmlHttp = new XMLHttpRequest();
        xmlHttp.open('POST', this.HOST + ":" + this.PORT, false);
        xmlHttp.onload = function()
{ this.receiveResponse(xmlHttp); }.bind(this);
        xmlHttp.onerror = function()
{ this.onError(xmlHttp) }.bind(this);
        var msg = JSON.stringify(json);
        xmlHttp.setRequestHeader('Content-length', msg.length);
        xmlHttp.setRequestHeader("Connection", "close");
        xmlHttp.send(msg);
    }
}
```

3.实现服务端

```
import BaseHTTPServer
import json
from ocr import OCRNeuralNetwork
import numpy as np

HOST_NAME = 'localhost'
PORT_NUMBER = 8000
HIDDEN_NODE_COUNT = 15

data_matrix = np.loadtxt(open('data.csv', 'rb'), delimiter = ',')
data_labels = np.loadtxt(open('dataLabels.csv', 'rb'))

data_matrix = data_matrix.tolist()
data_labels = data_labels.tolist()

nn = OCRNeuralNetwork(HIDDEN_NODE_COUNT, data_matrix, data_labels,
list(range(5000)));

class JSONHandler(BaseHTTPServer.BaseHTTPRequestHandler):
    def do_POST(s):
        response_code = 200
        response = ""
        var_len = int(s.headers.get('Content-Length'))
        content = s.rfile.read(var_len);
        payload = json.loads(content);

        if payload.get('train'):
```

```
            nn.train(payload['trainArray'])
            nn.save()
        elif payload.get('predict'):
            try:
                response = {"type":"test",
"result":nn.predict(str(payload['image']))}
            except:
                response_code = 500
        else:
            response_code = 400

        s.send_response(response_code)
        s.send_header("Content-type", "application/json")
        s.send_header("Access-Control-Allow-Origin", "*")
        s.end_headers()
        if response:
            s.wfile.write(json.dumps(response))
        return

if __name__ == '__main__':
    server_class = BaseHTTPServer.HTTPServer;
    httpd = server_class((HOST_NAME, PORT_NUMBER), JSONHandler)

    try:
        httpd.serve_forever()
    except KeyboardInterrupt:
        pass
    else:
        print "Unexpected server exception occurred."
    finally:
        httpd.server_close()
```

4.实现神经网络

我们使用反向传播算法（Backpropagation）来训练神经网络。

算法主要分为三个步骤：

第一步：初始化神经网络

一般将所有权值与偏置量置为(-1,1)范围内的随机数，在我们这个例子中，使用(-0.06,0.06)这个范围，输入层到隐藏层的权值存储在矩阵theta1中，偏置量存在input_layer_bias中，隐藏层到输出层则分别存在theta2与hidden_layer_bias中。

创建随机矩阵的代码如下，注意输出的矩阵是以size_out为行，size_in为列。可能你会想为什么不是size_in在左边。你可以这么想，一般都是待处理的输入放在右边，处理操作（矩阵）放在左边。

这里说明一下会用到的每一个矩阵/向量及其形状：

变量名	描述	形状
y0	输入层	1 * 400
theta1	输入-隐藏层权值矩阵	隐藏层节点数 * 400
input_layer_bias	输入-隐藏层偏置向量	隐藏层节点数 * 1
y1	隐藏层	隐藏层节点数 * 1
theta2	隐藏-输出层权值矩阵	10 * 隐藏层节点数
hidden_layer_bias	隐藏-输出层偏置向量	10 * 1
y2	输出层	10 * 1

第二步：前向传播

前向传播是指输入数据通过一层一层计算到达输出层得到输出结果，输出层会有10个节点分别代表0~9，哪一个节点的输出值最大就作为我们的预测结果。一般用sigmoid函数作为激发函数。可以将实数范围的数字映射到(0, 1)，S型的形状也很理想，最关键是导数可直接得到。使用numpy的vectorize能得到标量函数的向量化版本，直接处理向量。

第三步：反向传播

它需要通过计算误差率然后系统根据误差改变网络的权值矩阵和偏置向量。通过训练数据的标签我们得到actual_vals用来和输出层相减得到误差率output_errors，输出层的误差只能用来改进上一层，想要改进上上一层就需要计算上一层的输出误差，

```
import csv
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np
from numpy import matrix
from math import pow
from collections import namedtuple
```

```
import math
import random
import os
import json

class OCRNeuralNetwork:
    LEARNING_RATE = 0.1
    WIDTH_IN_PIXELS = 20
    NN_FILE_PATH = 'nn.json'

    def __init__(self, num_hidden_nodes, data_matrix, data_labels,
training_indices, use_file=True):
        self.sigmoid = np.vectorize(self._sigmoid_scalar)
        self.sigmoid_prime = np.vectorize(self._sigmoid_prime_scalar)
        self._use_file = use_file
        self.data_matrix = data_matrix
        self.data_labels = data_labels

        if (not os.path.isfile(OCRNeuralNetwork.NN_FILE_PATH) or not
use_file):
            # Step 1: Initialize weights to small numbers
            self.theta1 = self._rand_initialize_weights(400,
num_hidden_nodes)
            self.theta2 = self._rand_initialize_weights(num_hidden_nodes,
10)
            self.input_layer_bias = self._rand_initialize_weights(1,
num_hidden_nodes)
            self.hidden_layer_bias = self._rand_initialize_weights(1, 10)

            # Train using sample data
            TrainData = namedtuple('TrainData', ['y0', 'label'])
            self.train([TrainData(self.data_matrix[i],
int(self.data_labels[i])) for i in training_indices])
            self.save()
        else:
            self._load()

    def _rand_initialize_weights(self, size_in, size_out):
        return [((x * 0.12) - 0.06) for x in np.random.rand(size_out,
size_in)]

    # The sigmoid activation function. Operates on scalars.
    def _sigmoid_scalar(self, z):
        return 1 / (1 + math.e ** -z)

    def _sigmoid_prime_scalar(self, z):
        return self.sigmoid(z) * (1 - self.sigmoid(z))

    def _draw(self, sample):
        pixelArray = [sample[j:j+self.WIDTH_IN_PIXELS] for j in xrange(0,
len(sample), self.WIDTH_IN_PIXELS)]
        plt.imshow(zip(*pixelArray), cmap = cm.Greys_r,
interpolation="nearest")
```

```
        plt.show()

    def train(self, training_data_array):
        for data in training_data_array:
            # Step 2: Forward propagation
            y1 = np.dot(np.mat(self.theta1), np.mat(data['y0']).T)
            sum1 =  y1 + np.mat(self.input_layer_bias) # Add the bias
            y1 = self.sigmoid(sum1)

            y2 = np.dot(np.array(self.theta2), y1)
            y2 = np.add(y2, self.hidden_layer_bias) # Add the bias
            y2 = self.sigmoid(y2)

            # Step 3: Back propagation
            actual_vals = [0] * 10 # actual_vals is a python list for
easy initialization and is later turned into an np matrix (2 lines down).
            actual_vals[data['label']] = 1
            output_errors = np.mat(actual_vals).T - np.mat(y2)
            hidden_errors = np.multiply(np.dot(np.mat(self.theta2).T,
output_errors), self.sigmoid_prime(sum1))

            # Step 4: Update weights
            self.theta1 += self.LEARNING_RATE *
np.dot(np.mat(hidden_errors), np.mat(data['y0']))
            self.theta2 += self.LEARNING_RATE *
np.dot(np.mat(output_errors), np.mat(y1).T)
            self.hidden_layer_bias += self.LEARNING_RATE * output_errors
            self.input_layer_bias += self.LEARNING_RATE * hidden_errors

    def predict(self, test):
        y1 = np.dot(np.mat(self.theta1), np.mat(test).T)
        y1 =  y1 + np.mat(self.input_layer_bias) # Add the bias
        y1 = self.sigmoid(y1)

        y2 = np.dot(np.array(self.theta2), y1)
        y2 = np.add(y2, self.hidden_layer_bias) # Add the bias
        y2 = self.sigmoid(y2)

        results = y2.T.tolist()[0]
        return results.index(max(results))

    def save(self):
        if not self._use_file:
            return

        json_neural_network = {
            "theta1":[np_mat.tolist()[0] for np_mat in self.theta1],
            "theta2":[np_mat.tolist()[0] for np_mat in self.theta2],
            "b1":self.input_layer_bias[0].tolist()[0],
            "b2":self.hidden_layer_bias[0].tolist()[0]
        };
        with open(OCRNeuralNetwork.NN_FILE_PATH,'w') as nnFile:
            json.dump(json_neural_network, nnFile)
```

```
    def _load(self):
        if not self._use_file:
            return

        with open(OCRNeuralNetwork.NN_FILE_PATH) as nnFile:
            nn = json.load(nnFile)
        self.theta1 = [np.array(li) for li in nn['theta1']]
        self.theta2 = [np.array(li) for li in nn['theta2']]
        self.input_layer_bias = [np.array(nn['b1'][0])]
        self.hidden_layer_bias = [np.array(nn['b2'][0])]
```

5.运行测试

python server.py打开服务器，书写数字查看结果，测试证明只要偏差不大可以准确识别出内容。

四、实验总结

我们基于BP神经网络实现了一个简单的手写字符识别系统。虽然它只能识别数字，虽然性能也非常一般，但这是神经网络的一个起点，从这里出发可以看到更多相关内容。