人工智能 Project1

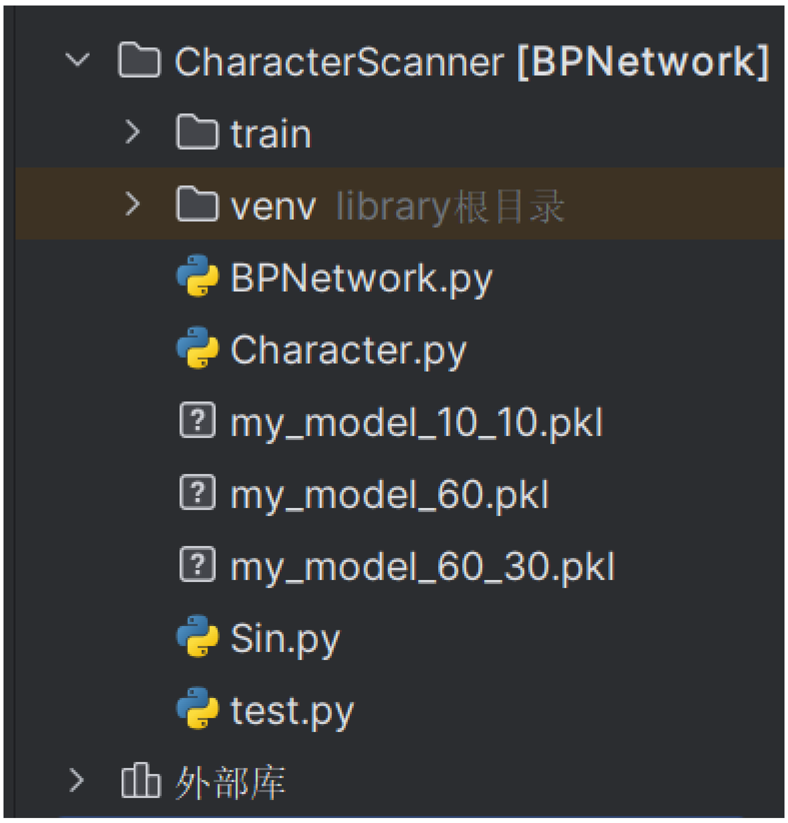
任务一：反向传播

20302010075叶柯玲

项目简介

通过反向传播算法拟合sin(x)函数和实现12个手写字分类

文件结构说明



```
CharacterScanner [BPNetwork]
    train
    venv library根目录
    BPNetwork.py
    Character.py
    my_model_10_10.pkl
    my_model_60.pkl
    my_model_60_30.pkl
    Sin.py
    test.py
外部库
```

train：保存手写字训练集

BPNetwork.py：BP神经网络类

Character.py：12个手写字分类的训练和测试

Sin.py：sin(x)函数拟合的训练和测试

test.py：对已训练模型的单独测试

实现过程

反向传播神经网络的搭建

（1）网络结构

BPnetwork主要分为输入层，隐藏层和输出层，各层神经元个数和隐藏层层数可自行设定，在每层神经元个数的基础上+1设为bias，w由已确定的相邻两层神经元个数决定，delta为反向传播错误时求出的对应w的偏导数

```
    def __init__(self):
        # 输入层神经元个数
        # 拟合sin(x)需要input_n = 2
        # 一个是输入数据，另一个用于调节bias
        self.input_n = 0
        # 输入层神经元输入的数据
        # 包括输入数据和bias（默认为1）
        self.input_cells = []
        # 输入层到隐藏层第一层的weight
        self.input_w = []
        # 输出层神经元个数
        self.output_n = 0
        # 输出层神经元输出值
        self.output_cells = []
        # 隐藏层最后一层到输出层的weight
        self.output_w = []
        # 输出层的bias
        self.output_b = []
        # 输出层的error关于w的偏导
        self.output_deltas = []
        # 隐藏层设置
        # 长度代表隐藏层个数，每个元素代表该层神经元个数
        self.hidden_ns = []
        # 隐藏层weight
        self.hidden_ws = []
        # 每个元素为该层设置的bias值
        self.hidden_bs = []
        # 每层隐藏层的输出值
        self.hidden_results = []
        # error关于i到i+1层weight的偏导
        self.hidden_deltases = []
```

（2）数据初始化

根据输入变量数量和输出变量数量和隐藏层设置初始化网络，其中hidden_set[]的维度代表隐藏层个数，每个维度上的数值代表该层神经元个数（不包括bias）

```
    def setup(self, input_n, output_n, hidden_set):
        # input_n是输入参数的个数，不等同于神经元的个数
        # +1是新增一个神经元来调节bias
        # 且这个神经元的输入值记为1
        self.input_n = input_n + 1
        self.output_n = output_n
        self.hidden_ns = [0.0]*len(hidden_set)
        for i in range(len(hidden_set)):
            self.hidden_ns[i] = hidden_set[i] + 1
        # 初始化神经元个数列表
        self.input_cells = [1.0]*self.input_n
        self.output_cells = [1.0]*self.output_n
        # 初始化输入层和隐藏层第一层之前的weight
        self.input_w = generate_w(self.input_n, self.hidden_ns[0])
        # 初始化隐藏层之间的weight
        self.hidden_ws = [0.0]*(len(self.hidden_ns)-1)
        for i in range(len(self.hidden_ns)-1):
            self.hidden_ws[i] = generate_w(self.hidden_ns[i],
                                           self.hidden_ns[i+1])
        # 初始化隐藏层最后一层到输出层weight
        self.output_w =
        generate_w(self.hidden_ns[len(self.hidden_ns)-1], self.output_n)
        self.output_b = generate_b(self.output_n)
        self.hidden_bs = [0.0]*len(self.hidden_ns)
        for i in range(len(self.hidden_ns)):
            self.hidden_bs[i] = generate_b(self.hidden_ns[i])
        self.hidden_results = [0.0]*(len(self.hidden_ns))
```

（3）向前传播

输入数据值经过网络运算前向传播得到各层神经元的输出值，即每一层神经元的值是前一层神经元数值的加权和再加上bias，最终得到输出层的数值。这里的fit_function为激活函数，在BPnetwork任务中选择了tanh函数

```
        def forward_propagate(self, input_):
            # 向前传播
            for i in range(len(input_)):
                self.input_cells[i] = input_[i]
            # 输入层
            self.hidden_results[0] = [0.0]*self.hidden_ns[0]
            for h in range(self.hidden_ns[0]):
                total = 0.0
                for i in range(self.input_n):
                    total += self.input_w[i][h] * self.input_cells[i]
                self.hidden_results[0][h] = fit_function(total+self.hidden_bs[0][h])
            # 隐藏层
            for k in range(len(self.hidden_ns)-1):
                self.hidden_results[k+1] = [0.0]*self.hidden_ns[k+1]
                for h in range(self.hidden_ns[k+1]):
                    total = 0.0
                    for i in range(self.hidden_ns[k]):
                        total += self.hidden_ws[k][i][h] * self.hidden_results[k][i]

                    self.hidden_results[k+1][h] =
    fit_function(total+self.hidden_bs[k+1][h])
            # 输出层
            for h in range(self.output_n):
                total = 0.0
                for i in range(self.hidden_ns[len(self.hidden_ns)-1]):
                    total += self.output_w[i][h] *
    self.hidden_results[len(self.hidden_ns)-1][i]
                    self.output_cells[h] = fit_function(total+self.output_b[h])
        return self.output_cells[:]
```

（4）获得误差

我认为这是实现BPnetwork最重要的部分，通过将前向传播输出值与目标值进行对比，得到误差，再通过网络链式求导法则，得出各层weight对应的误差偏导数

```
    def get_deltas(self, label):
        # 反向传输错误
        self.output_deltas = [0.0]*self.output_n
        # 输出层deltas
        for o in range(self.output_n):
            error = label[o] - self.output_cells[o]
            self.output_deltas[o] = fit_function(self.output_cells[o], True) *
error
        # 隐层deltas
        tmp_deltas = self.output_deltas
        tmp_w = self.output_w
        self.hidden_deltases = [0.0]*(len(self.hidden_ns))
        k = len(self.hidden_ns) - 1
        while k >= 0:
            self.hidden_deltases[k] = [0.0]*(self.hidden_ns[k])
            for o in range(self.hidden_ns[k]):
                error = 0.0
                for i in range(len(tmp_deltas)):
                    error += tmp_deltas[i] * tmp_w[o][i]
                self.hidden_deltases[k][o] = fit_function(self.hidden_results[k]
[o], True) * error
            k = k - 1
            if k >= 0:
                tmp_w = self.hidden_ws[k]
                tmp_deltas = self.hidden_deltases[k+1]
            else:
                break
```

（5）更新weight和bias

通过上一步得到的偏导数，根据设置的学习率反向修正各层weight和bias的数值

```
    def renew_w(self, learn):
        # 更新隐藏层到输出层权重
        k = len(self.hidden_ns) - 1
        for i in range(self.hidden_ns[k]):
            for o in range(self.output_n):
                change = self.output_deltas[o] * self.hidden_results[k][i]
                self.output_w[i][o] += change * learn
        # 更新隐层权重
        while k > 0:
            for i in range(self.hidden_ns[k-1]):
                for o in range(self.hidden_ns[k]):
                    change = self.hidden_deltases[k][o] * self.hidden_results[k-
1][i]
                    self.hidden_ws[k-1][i][o] += change * learn
            k = k - 1
        # 更新输入层到隐层权重
        for i in range(self.input_n):
            for o in range(self.hidden_ns[0]):
                change = self.hidden_deltases[0][o] * self.input_cells[i]
                self.input_w[i][o] += change * learn

    def renew_b(self, learn):
        # 更新隐藏层bias
        k = len(self.hidden_bs)-1
        while k >= 0:
            for i in range(self.hidden_ns[k]):
                self.hidden_bs[k][i] = self.hidden_bs[k][i] + learn *
self.hidden_deltases[k][i]
            k = k - 1
        # 更新输出层bias
        for o in range(self.output_n):
            self.output_b[o] += self.output_deltas[o] * learn
```

(6) 反向传播

综合以上过程，得到最终的反向传播函数，back_propagate对应sin(x)函数的拟合，因为是一元神经网络，最终设置的误差是输出值和目标值的差值，而back_propagate_c的输出值取决于汉字分类正确与否，正确为1，反之为0

```
    def back_propagate_c(self, input_, label, learn):
        self.forward_propagate(input_)
        self.get_deltas(label)
        self.renew_w(learn)
        self.renew_b(learn)
        return self.get_rightness(label, self.output_cells)

    def back_propagate(self, input_, label, learn):
        self.forward_propagate(input_)
        self.get_deltas(label)
        self.renew_w(learn)
        self.renew_b(learn)
        return self.get_loss(label, self.output_cells)
```

拟合sin(x)函数的训练

(1) 网络设置

初始化神经网络，设置中间层，学习率，epoch次数，因为sin(x)的训练比较简单，耗时很短，中间层可以任意尝试，最终设置的是[10,10,10]的三层中间层

```
    t = BPNetwork()
    # 初始化BP网络，输入输出一个神经元，中间层为三层
    t.setup(1, 1, [10, 10, 10])
    # 获得数据
    # threshold为学习率
    # max_steps为epoch次数
    threshold = 0.05
    max_steps = 3000
```

（2）设置输入值和目标值

```
    # 生成正弦函数目标值
        x = np.linspace(-np.pi, np.pi, 10)
        y = np.sin(x)
```

（3）训练得到错误率

```
    # 开始训练
    for k in range(max_steps):
        epoch = 0.0
        for o in range(len(input_data)):
            epoch += t.back_propagate(input_data[o], labels[o], threshold)
        error = epoch / max_steps
        print("第 %d 次迭代: " % (k + 1))
        print(error)
```

手写汉字分类

（1）网络设置

初始化神经网络，分类手写字的输入文件格式为bmp，故需要28*28个输入神经元，共有12个汉字，故输出神经元个数为12

```
    character = BPNetwork()
    character.setup(28*28, 12, [60])
```

或者通过载入已训练神经网络完成初始化，下方代码载入的是中间层为[60,30]的已训练网络

```
# 载入已训练网络
    with open('my_model_60_30.pkl', 'rb') as f:
        character = pickle.load(f)
```

（2）设置input和label

载入train文件夹下的手写体图片，取前400个为训练集，后220个为测试集，索引 j 即可确定训练图片为何字

```
    sample_size = 400
    test_size = 620 - sample_size
    # 导入图片
    for i in range(0, sample_size):
        for j in range(0, 12):
            input_data[i][j] = list(np.array(imageio.imread("train/" + str(j+1) +
"/" + str(i+1) + ".bmp")).flatten())
```

设置label，此处用二维列表来表示12个汉字的分类，即若输入的是1中的字（博），目标输出应该是[1,0,0,0,0,0,0,0,0,0,0,0]

```
    for i in range(0, 12):
        output_data[i][i] = 1
```

（3）训练得到正确率

目标输出为[1,0,0,0,0,0,0,0,0,0,0,0]，实际训练时几乎不可能得到这样的整数，故在back_propagate_c函数中取最大值的索引作为前向传播的结果与目标函数值1的索引对比得到正确率

```
    # 开始训练
    for i in range(0, max_steps):
        epoch = 0.0
        # if i % 10 == 1:
        #     threshold = 0.05
        # threshold /= 2
        for j in range(0, sample_size):
            for k in range(0, 12):
                epoch += character.back_propagate_c(input_data[j][k],
output_data[k], threshold)
        right = epoch/12/sample_size
        print("epoch:", i + 1, "times", "training rightness:", right)
```

（4）保存训练网络

```
        # 存入网络
        with open('my_model_60_30.pkl', 'wb') as f:
            pickle.dump(character, f)
        if right > 0.85:
            break
```

（5）结果

下图分别为隐藏层为[10,10]和隐藏层为[60]以及隐藏层为[20,15]的训练正确率和测试正确率（截至说明文档编写当天，后续会接着训练）

[10,10]

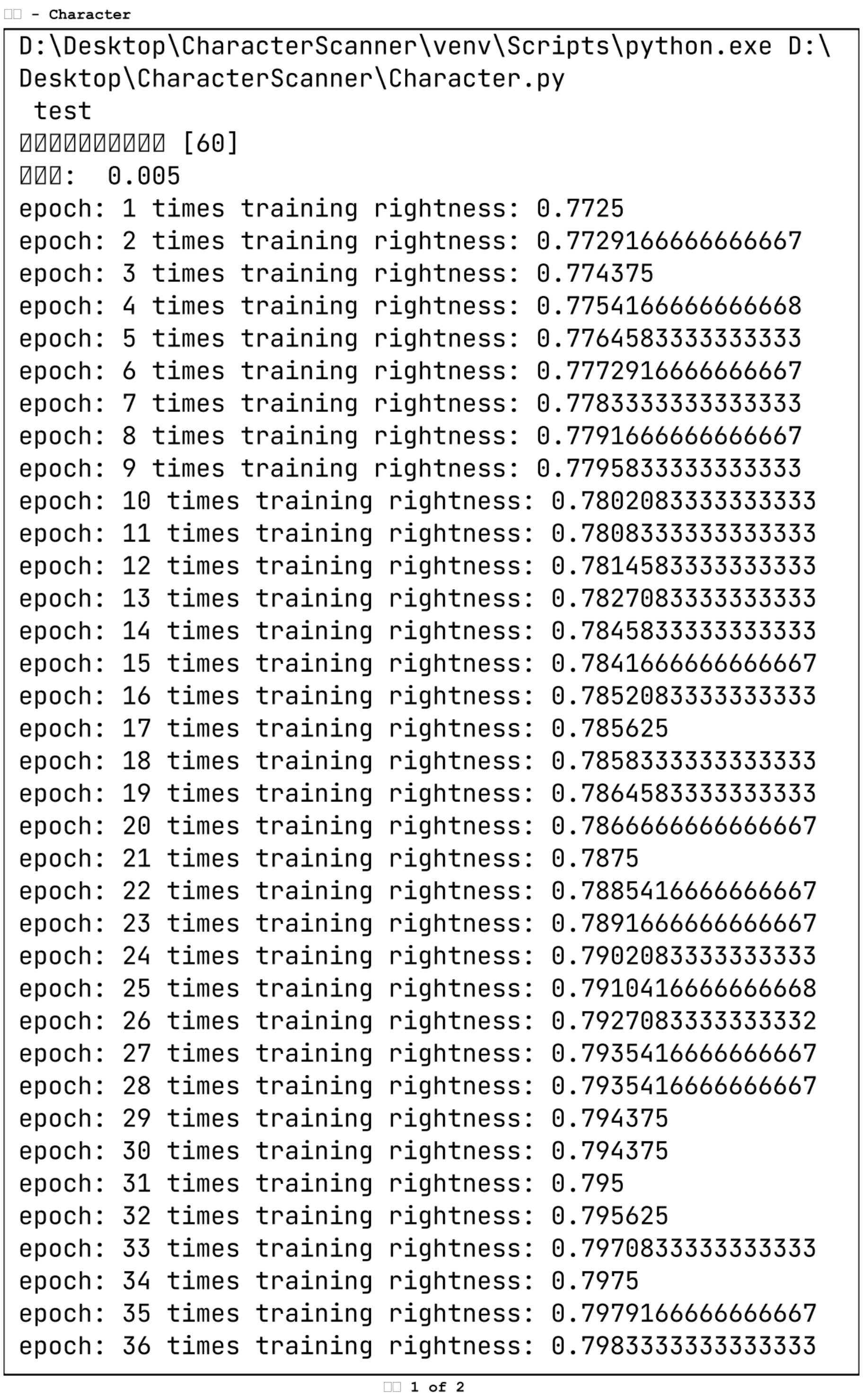
□□ - Character

```
D:\Desktop\CharacterScanner\venv\Scripts\python.exe D:\
Desktop\CharacterScanner\Character.py
 test
□□□□□□□□□□ [10, 10]
□□□:  0.005
epoch: 1 times training rightness: 0.83625
epoch: 2 times training rightness: 0.8375
epoch: 3 times training rightness: 0.8397916666666667
epoch: 4 times training rightness: 0.840625
epoch: 5 times training rightness: 0.8420833333333333
epoch: 6 times training rightness: 0.8427083333333333
epoch: 7 times training rightness: 0.8435416666666667
epoch: 8 times training rightness: 0.84375
epoch: 9 times training rightness: 0.8454166666666667
epoch: 10 times training rightness: 0.8464583333333333
epoch: 11 times training rightness: 0.84625
epoch: 12 times training rightness: 0.8470833333333333
epoch: 13 times training rightness: 0.8483333333333333
epoch: 14 times training rightness: 0.8485416666666667
epoch: 15 times training rightness: 0.8489583333333333
epoch: 16 times training rightness: 0.8495833333333332
epoch: 17 times training rightness: 0.84875
epoch: 18 times training rightness: 0.8495833333333332
epoch: 19 times training rightness: 0.84875
epoch: 20 times training rightness: 0.8491666666666667
epoch: 21 times training rightness: 0.8497916666666667
epoch: 22 times training rightness: 0.849375
epoch: 23 times training rightness: 0.8491666666666667
epoch: 24 times training rightness: 0.8491666666666667
epoch: 25 times training rightness: 0.84875
epoch: 26 times training rightness: 0.8485416666666667
epoch: 27 times training rightness: 0.848125
epoch: 28 times training rightness: 0.8485416666666667
epoch: 29 times training rightness: 0.8491666666666667
epoch: 30 times training rightness: 0.85
epoch: 31 times training rightness: 0.8502083333333332
testing rightness: 0.7375

□□□□□□□□□□□ 0
```

□□ 1 of 1

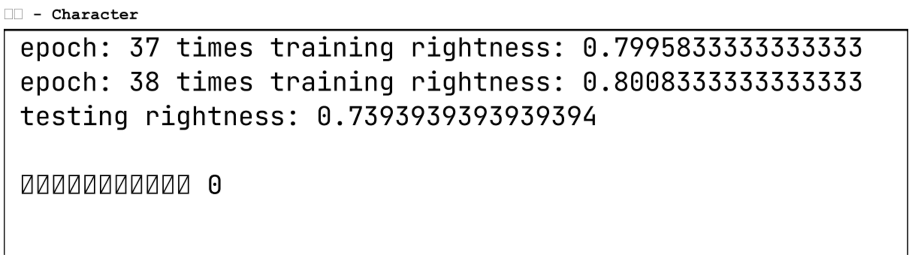
[60,60]



□□ - Character

```
D:\Desktop\CharacterScanner\venv\Scripts\python.exe D:\
Desktop\CharacterScanner\Character.py
 test
□□□□□□□□□□ [60]
□□□:  0.005
epoch: 1 times training rightness: 0.7725
epoch: 2 times training rightness: 0.7729166666666667
epoch: 3 times training rightness: 0.774375
epoch: 4 times training rightness: 0.7754166666666668
epoch: 5 times training rightness: 0.7764583333333333
epoch: 6 times training rightness: 0.7772916666666667
epoch: 7 times training rightness: 0.7783333333333333
epoch: 8 times training rightness: 0.7791666666666667
epoch: 9 times training rightness: 0.7795833333333333
epoch: 10 times training rightness: 0.7802083333333333
epoch: 11 times training rightness: 0.7808333333333333
epoch: 12 times training rightness: 0.7814583333333333
epoch: 13 times training rightness: 0.7827083333333333
epoch: 14 times training rightness: 0.7845833333333333
epoch: 15 times training rightness: 0.7841666666666667
epoch: 16 times training rightness: 0.7852083333333333
epoch: 17 times training rightness: 0.785625
epoch: 18 times training rightness: 0.7858333333333333
epoch: 19 times training rightness: 0.7864583333333333
epoch: 20 times training rightness: 0.7866666666666667
epoch: 21 times training rightness: 0.7875
epoch: 22 times training rightness: 0.7885416666666667
epoch: 23 times training rightness: 0.7891666666666667
epoch: 24 times training rightness: 0.7902083333333333
epoch: 25 times training rightness: 0.7910416666666668
epoch: 26 times training rightness: 0.7927083333333332
epoch: 27 times training rightness: 0.7935416666666667
epoch: 28 times training rightness: 0.7935416666666667
epoch: 29 times training rightness: 0.794375
epoch: 30 times training rightness: 0.794375
epoch: 31 times training rightness: 0.795
epoch: 32 times training rightness: 0.795625
epoch: 33 times training rightness: 0.7970833333333333
epoch: 34 times training rightness: 0.7975
epoch: 35 times training rightness: 0.7979166666666667
epoch: 36 times training rightness: 0.7983333333333333
```

□□ 1 of 2



□□ - Character

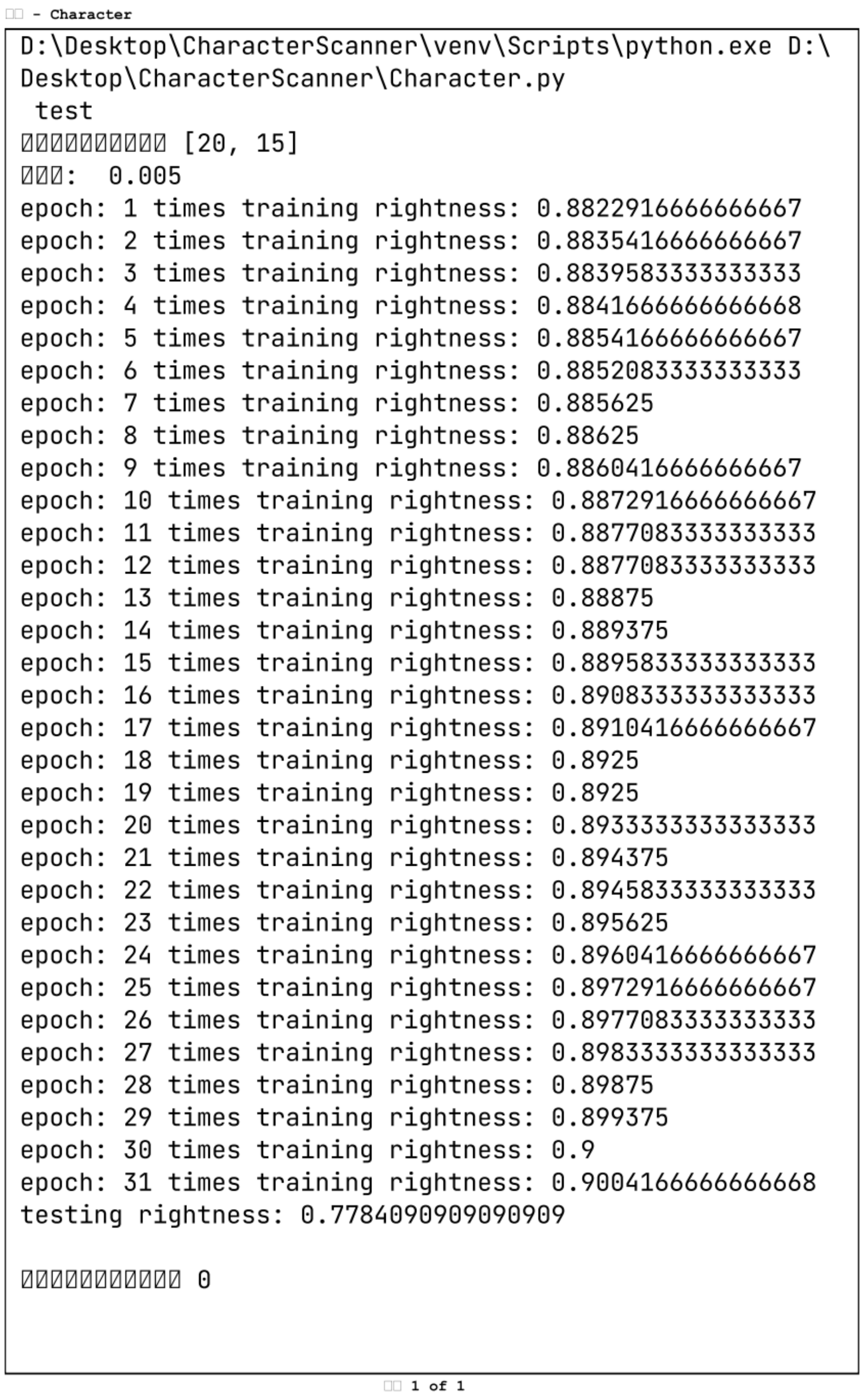
```
epoch: 37 times training rightness: 0.7995833333333333
epoch: 38 times training rightness: 0.8008333333333333
testing rightness: 0.7393939393939394

□□□□□□□□□□□ 0
```

[20,15]

```
D:\Desktop\CharacterScanner\venv\Scripts\python.exe D:\
Desktop\CharacterScanner\Character.py
 test
□□□□□□□□□□ [20, 15]
□□□:  0.05
epoch: 1 times training rightness: 0.17291666666666667
epoch: 2 times training rightness: 0.36645833333333333
epoch: 3 times training rightness: 0.43895833333333334
epoch: 4 times training rightness: 0.500625
epoch: 5 times training rightness: 0.5377083333333333
epoch: 6 times training rightness: 0.5725
epoch: 7 times training rightness: 0.5964583333333333
epoch: 8 times training rightness: 0.6220833333333333
epoch: 9 times training rightness: 0.63875
epoch: 10 times training rightness: 0.6575
epoch: 11 times training rightness: 0.67125
epoch: 12 times training rightness: 0.6872916666666667
epoch: 13 times training rightness: 0.6935416666666667
epoch: 14 times training rightness: 0.7064583333333333
epoch: 15 times training rightness: 0.715625
epoch: 16 times training rightness: 0.7254166666666667
epoch: 17 times training rightness: 0.729375
epoch: 18 times training rightness: 0.736875
epoch: 19 times training rightness: 0.7464583333333333
epoch: 20 times training rightness: 0.753125
epoch: 21 times training rightness: 0.759375
epoch: 22 times training rightness: 0.7583333333333333
epoch: 23 times training rightness: 0.7664583333333332
epoch: 24 times training rightness: 0.7722916666666667
epoch: 25 times training rightness: 0.7716666666666667
epoch: 26 times training rightness: 0.78375
epoch: 27 times training rightness: 0.78375
epoch: 28 times training rightness: 0.7885416666666667
epoch: 29 times training rightness: 0.7945833333333333
epoch: 30 times training rightness: 0.798125
epoch: 31 times training rightness: 0.79625
epoch: 32 times training rightness: 0.8025
epoch: 33 times training rightness: 0.8085416666666667
epoch: 34 times training rightness: 0.8102083333333333
epoch: 35 times training rightness: 0.8079166666666667
epoch: 36 times training rightness: 0.8127083333333333
```

```
epoch: 37 times training rightness: 0.8129166666666667
epoch: 38 times training rightness: 0.81625
epoch: 39 times training rightness: 0.8204166666666667
epoch: 40 times training rightness: 0.8185416666666667
epoch: 41 times training rightness: 0.825625
epoch: 42 times training rightness: 0.82625
epoch: 43 times training rightness: 0.8322916666666668
epoch: 44 times training rightness: 0.8304166666666667
epoch: 45 times training rightness: 0.8347916666666667
```



```
□□ - Character
D:\Desktop\CharacterScanner\venv\Scripts\python.exe D:\
Desktop\CharacterScanner\Character.py
 test
□□□□□□□□□□ [20, 15]
□□□:  0.005
epoch: 1 times training rightness: 0.8822916666666667
epoch: 2 times training rightness: 0.8835416666666667
epoch: 3 times training rightness: 0.8839583333333333
epoch: 4 times training rightness: 0.8841666666666668
epoch: 5 times training rightness: 0.8854166666666667
epoch: 6 times training rightness: 0.8852083333333333
epoch: 7 times training rightness: 0.885625
epoch: 8 times training rightness: 0.88625
epoch: 9 times training rightness: 0.8860416666666667
epoch: 10 times training rightness: 0.8872916666666667
epoch: 11 times training rightness: 0.8877083333333333
epoch: 12 times training rightness: 0.8877083333333333
epoch: 13 times training rightness: 0.88875
epoch: 14 times training rightness: 0.889375
epoch: 15 times training rightness: 0.8895833333333333
epoch: 16 times training rightness: 0.8908333333333333
epoch: 17 times training rightness: 0.8910416666666667
epoch: 18 times training rightness: 0.8925
epoch: 19 times training rightness: 0.8925
epoch: 20 times training rightness: 0.8933333333333333
epoch: 21 times training rightness: 0.894375
epoch: 22 times training rightness: 0.8945833333333333
epoch: 23 times training rightness: 0.895625
epoch: 24 times training rightness: 0.8960416666666667
epoch: 25 times training rightness: 0.8972916666666667
epoch: 26 times training rightness: 0.8977083333333333
epoch: 27 times training rightness: 0.8983333333333333
epoch: 28 times training rightness: 0.89875
epoch: 29 times training rightness: 0.899375
epoch: 30 times training rightness: 0.9
epoch: 31 times training rightness: 0.9004166666666668
testing rightness: 0.7784090909090909

□□□□□□□□□□□ 0
□□ 1 of 1
```

[10,10]总共训练了51次，最终正确率为0.86

[20,15]共训练81次，最终正确率为0.90

[60]训练了近100次，最终正确率为0.80

[60,30]因为训练时间过长暂时放弃

困难点和解决方案

（1）weight的更新

在构建神经网络的过程中，实现前向传播及之前的工作都进行的比较顺利，但是在weight和bias的更新问题上很难流畅地把每条线搭对，拟合sin(x)这一个小任务能够帮助搭建网络和检验正确性，方便后续手写字任务的完成

（2）手写字分类输出值的确定

在刚开始这一任务的时候，我陷入了“识别”的误区，误以为程序是真的认出了这个字，因此在选择输出神经元的时候做了很多无用功，在查阅了资料后意识到给出任务的“分类”这一词并不是我所理解的为了简单化本次project，而是人工智能网络对图像所做出的识别实际就是打标签分类（应该是这样理解的吧）然后就很自然地想到了编码，给每个字设置他独有的标签再训练，但是这样在对比输出值和标签值的时候会比较麻烦，因此最后选择了12个神经元表示输出信息，二维列表的形式储存标签信息

（3）迭代正确率的选择

在有了sin(x)的实验经验的基础上，我继续使用了其对于结果正确率地判断，但是在最后的训练过程中上蹿下跳的迭代正确率让我大跌眼镜，在反复debug的过程中我意识到了拟合实验和分类实验对于正确率的判断是不同的，在拟合过程中正确和错误没有很明确的概念，所以更多的用到的是误差，而回归任务中这12个神经元的误差就不能代表正确率了，所以才会出现训练正确率不是朝着提高趋势走的状况

（4）中间层数和学习率的确定

对手写字分类的识别，我一开始认为复杂的问题就应该把神经网络做的越复杂，学习率设置的越低模拟效果越好，但在实验的过程中发现，太过复杂的中间层实现起来时间不允许，学习率也不是越小越好，中间我尝试过随着迭代次数的增加减小学习率，效果也没有很好，最终选择的方案是在达到收敛前分别用0.05和0.005多次训练（各自大概50次），在接近收敛时，学习率每次自除2直到收敛

对于BP神经网络的理解

以我对反向传播的初步理解，我认为这一算法很符合我对学习这一过程的认知：实践----前向传播、获得结果----获得输出值、与目标对比----获得误差偏导树、修正----反向传播。学习中比较困难的就是目标的设置和修正过程，就像实验过程中比较困难的是手写字分类输出值的确定和weight的更新，而测试结果总是达不到训练结果反映现实中学习过程和最终成果的关系。我们需要在欠拟合和过拟合中找到平衡。