浙江大学

课程名称 人工智能（AI2016）

姓名 应辉

学号 3130102386

所在学院 计算机学院

Project

实验题目及要求

手写数字识别：

使用神经网络算法来对MINIST的测试数据进行分类，并且提供准确率。

要求：

神经网络必须有至少三层

你需要提供测试数据的分类准确率

对于编程语言没有限制，推荐使用PYTHON或者MATLAB

实验方案设计（求解思路）

完整的实验过程分为以下几个步骤：

1. 数据文件以及标签文件的读取
2. 数据的预处理
3. BP神经网络算法的编写
4. 根据训练数据进行训练
5. 测试训练结果

下面具体展示每个步骤的思路：

1. 数据文件以及标签文件的读取

MINIST所给的数据文件是按照以下格式进行存放数据的——

图片文件：

4 BYTE	4 BYTE	4 BYTE	4 BYTE	
2051	图片数量	行数	列数	图像信息

注：图像信息中存放的是每个像素点的灰度信息

标签文件：

4 BYTE	4 BYTE	4 BYTE
2049	图片数量	标签信息

注： 标签信息中存放的是每张图片代表的数字

所以图像文件以及标签文件的读取过程十分简单，只需要先将文件头的部分读取并确认，然后就可以对剩余部分的实际内容进行读取。其中图片文件每（行数*列数）BYTE为一张图片信息，标签文件每1BYTE为一张图片的标签。

2. 数据的预处理

由于MINIST所给的图片大小为28*28的256灰度的数据，也就意味着如果直接采用原始数据，我们输入将是一个784维的空间，这将对计算速度造成比较大的负担。

所以我用最简单的方法对这个数据进行降维处理。首先，我将图像进行二值化，因为区分一个数字我们只需要确认某个点是黑或者白就足够了，过于精细的灰度值是没有必要的。然后我将图像中四边相邻的每4个点合并为1个输入点，这个输入点的值是四个点中黑色点的个数。

3. BP神经网络算法的编写

这部分的内容即为老师上课讲解的BP神经网络的算法实现。基本思路是这样的。

首先根据当前各层的权重信息，通过正向传播，计算出每一层的输出。然后再根据已知的正确分类的信息，反向传播，更新各层的权重信息。其具体计算公式，将在代码中展现

4. 根据训练数据进行训练

在这一步，我们需要做的就是应用第1，2部中得到的数据，使用BP神经网络算法，对权重值进行训练。

在这里，神经网络的输入时2中得到的数据，而输出值是一个10维的向量，取最大值得下标作为实际识别出来的数字。所以，目标值的设置是首先生成一个10维的值都为0的向量，然后将下标为真实数字的值设置为1。

其中还有一系列的参数需要设置，其中包括学习率，训练次数，神经网络的层数，每一层的维度，每一层的初始权重。在这次试验中，我的学习率设置为0.01，训练次数为100。经过试验，发现3层网络已经足够解决数字识别，并且收敛速度也比4层网络更快，因而选择了3层网络。而隐含层的维度设置为图片的行数和列数之和。初始权重设置为0到0.01的随机值。

5. 测试训练结果

当我们得到一个权重信息的时候，就可以用测试数据或者训练数据对其进行测试。测试方法是这样的，对于每一张图片，通过2的预处理后，将其输入神经网络，正向传播得到一个输出，然后去最大输出的下标作为其实际结果，然后将这个实际结果与正确数字进行比较，如果相同这说明分类正确。从而获得该权重信息下的正确率。
在实际代码中，为了跟踪每一步迭代的表现，我在每次迭代之后都会进行一次测试。

实现代码

BP神经网络的实现

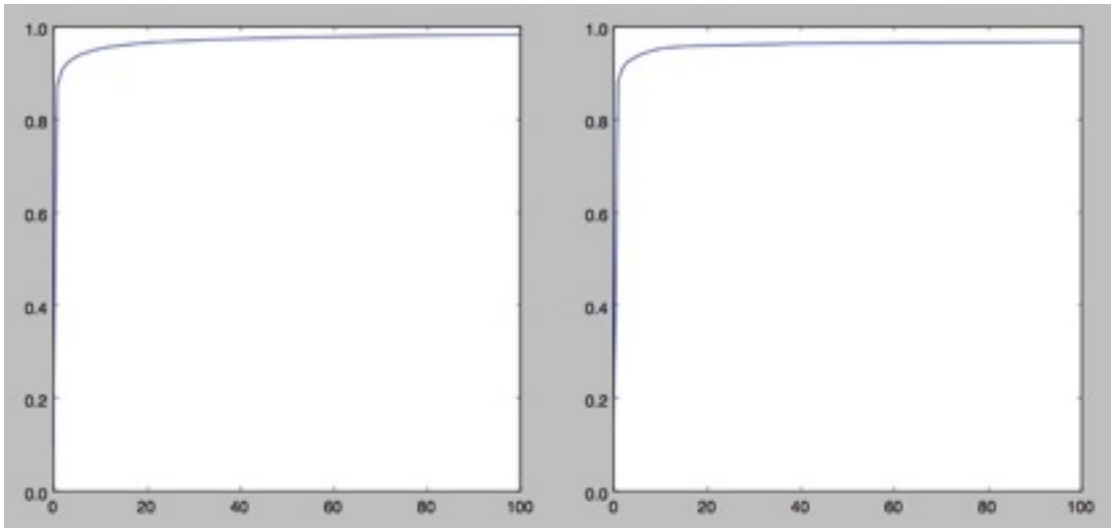
```
# sigmoid函数
def sigmoid(inX):
    return 1.0 / (1 + exp(-inX))

# 计算每一层的输出情况
def cal_each_layer_out(d_input, each_layer_w, layer_num):
    # 由于前一层的输出就等于后一层的输入,这里的e_input代表两个含义
    e_input = d_input
    # 输出结果数组
    e_output = []
    e_output.append(e_input)
    for i in range(layer_num-1):
        # 将前一层的输出矩阵与层之间的权重矩阵进行相乘,并通过sigmoid得到结果
        e_input = sigmoid(dot(e_input,each_layer_w[i]))
        # 将结果保存到返回数组中
        e_output.append(e_input)
    return e_output

#对一个数据进行一次训练,得到新的W
def one_train_for_BP(d_input,d_label,each_layer_dim,each_layer_w,layer_num,neta):
    # 计算当前W下的每一层的输出
    each_layer_output = cal_each_layer_out(d_input,each_layer_w,layer_num)
    # 计算输出层的theta值
    theta = each_layer_output[layer_num-1]*(ones((1,each_layer_dim[layer_num-1]))-
each_layer_output[layer_num-1])\
            *(d_label-each_layer_output[layer_num-1])
    # 进行反向传播
    for i in range(layer_num-1):
        # 保存未更改之前的W
        old_w = each_layer_w[layer_num-2-i]
        # 更新W
        each_layer_w[layer_num-2-i] = each_layer_w[layer_num-2-i] +
dot(each_layer_output[layer_num-2-i].T,theta)*neta
        # 计算前一层新的theta
        theta = each_layer_output[layer_num-2-
i]*(ones((1,each_layer_dim[layer_num-2-i]))-each_layer_output[layer_num-2-i])\
                *dot(theta,old_w.T)
    # 返回结果
    return each_layer_w
```

进行训练的代码段：

```
# 进行N次迭代
for i in range(training_times):
      print "    loop "+str(i)+" ..."
      # 对于每张图片进行一次训练，即使用Incremental mode
      for j in range(image_num):
            # 获得一个输入
            d_input = array(input_data[j])
            # 获得一个目标输出
            d_label = zeros(10)/1.0
            d_label[label_data[j]] = 1.0
            # 进行一次训练
            e a c h _ l a y e r _ w                                                    =
one_train_for_BP(d_input,d_label,each_layer_dim,each_layer_w, 3, neta)
```

实验结果及分析

下图为每次迭代之后的正确率，其中左端测试的是训练数据，右端测试的是测试数据：



从图中，我们可以发现，该次实验的收敛速度非常快，训练样本的量非常大，有60000组。所以在第5次迭代的时候就基本达到了收敛。并且从图中我们还可以看出，训练的数据的正确率是略高于测试数据的。同时，由于数据量比较大的原因，这里也没有出现过拟合的现象。
最终，实验得到的训练数据的正确率，与测试数据的正确率分别为：
训练数据： 98.43% 测试数据： 96.85%

在实验中，我也对于神经网络的其它参数进行了测试。得到了一下经验：

1. 隐含层层数。在本实验的一开始，我曾经尝试使用4层神经网络来进行训练，后来发现在相同的迭代次数下，其实4层的神经网络的效果并不比3层神经网络好。应该是由于迭代次数还不多的原因。所以处于时间上的考虑，我就使用了3层网络，而放弃了4层网络。
2. 隐含层节点个数。在这个问题上，我曾经使用行数加列数，行数，行数的一般，来进行训练。从测试结果上来看，隐含层的节点数量越多，取得的效果确实也会越好，但是效果并不是非常明显。所以这个问题需要综合考虑，如果时间并不紧张，那么为了追求最好的效果，可以稍微多取一些节点。但是如果考虑效率因素的话，需要找到在效果不差的情况下所需要的最少的节点。自本次实验中，行数的一半的隐含层节点个数还是足够的。
3. 学习率的选择。在这上面我也进行了实验，我发现，如果学习率设置的比较高，初始收敛速度会比较快，但是到了后期测试正确率会上不去，并且会一直有一定幅度的上下波动。如果学习率较低的话，收敛速度会比较慢，但是到了后期会比较稳定。所以，在这次试验中，我还是选择了一个比较小的学习率。当然，这也是因情况而定的，在做这个实验之前，我还进行了一个比较简单的神经网络的实验，该实验比较简单，并且训练样本也比较少。在该实验中，使用较大的学习率会更加合适。所以，如果一个模型，如果它比较简单，并且训练样本比较少的话，会适合选择一个较大的学习率。反之，选择小一点的学习率会比较合适。