華中科技大学

课 程 实 验 报 告

课程名称： 人工智能导论

专业班级： 计科 2011 班

学　　号： U202015084

姓　　名： 张文浩

指导教师： 金燕

报告日期： 2021.12.31

计算机科学与技术学院

目 录

实验概述

1.1 实验名称

基于 BP 神经网络的手写数字识别

1.2 实验内容

掌握 BP 神经网络的基本原理，学会用简单的 BP 神经网络实现手写数字识别功能，并对算法和结果进行分析，得出自己的看法和感悟。

1.3 实验要求

1.掌握 BP 神经网络的基本原理，手写实现 BP 神经网络（可以使用 numpy，pandas 等库）。

2.采用实现的 BP 神经网络模型解决手写数字识别问题。

3.分析 BP 神经网络的优缺点。

1.4 需求分析

我们需要用 BP 神经网络来解决手写数字识别的问题。自然，神经网络的输入便是训练或者测试图像，输出为一个长度为 10 的数组，用来记录本次训练或预测得出的数字是 0-9 中哪一个的比例。

本次实验需要从 kaggle 网站下载相关训练集和测试集，训练集中有 42000 张图像，每张图像均为 28 像素*28 像素的灰度图像，各个像素的取值在 0-255 之间，网站中下载的 train.csv 是训练样本集，大小 42001*785，第一行是文字描述，所以实际的样本数据大小是 42000*785，其中第一列的每一个数字是它对应行的数字，可以将第一列单独取出来（可以将数字转换成独热标签，这样更容易进行归类），得到 42000*1 的向量 t_train，，剩下的就是 42000*784 的特征向量集 x_train，所以从 train.csv 可以获取两个矩阵。test.csv 里的数据大小是 28001*784，第一行是文字描述，因此实际的测试数据样本是 28000*784，与 train.csv 不同，没有 label，28000*784 即 28000 个测试样本。

我们要做的，就是用训练集来训练我们的神经网络，在预测测试集的时候有更好的输出。

实验设计

2.1 前置知识

1.微积分：导数、梯度等

2.线性代数：矩阵的运算等

2.2 理论知识与算法原理

（1）感知机：

感知机是接收多个信号，输出一个信号的人工神经元。

对于一个接受两个输入信号的感知机，x1、x2 是输入信号，y 是输出信号，w1、w2 是权重。当输入信号被送往感知机时，被分别乘以相应的权重并求和，只有当和值超过神经元阈值θ时才会被激活。

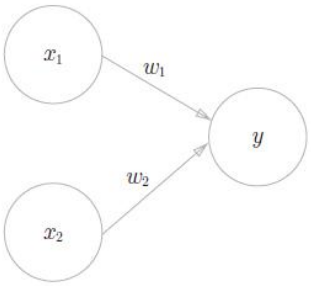

图 2-1 感知机

其中 y 为：

$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leqslant \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$$

如果将θ移项至左侧，并令$-\theta = b$，可得加入偏置 b 之后的神经元输出公式：

$$y = \begin{cases} 0 & (b + w_1x_1 + w_2x_2 \leqslant 0) \\ 1 & (b + w_1x_1 + w_2x_2 > 0) \end{cases}$$

（2）神经网络：

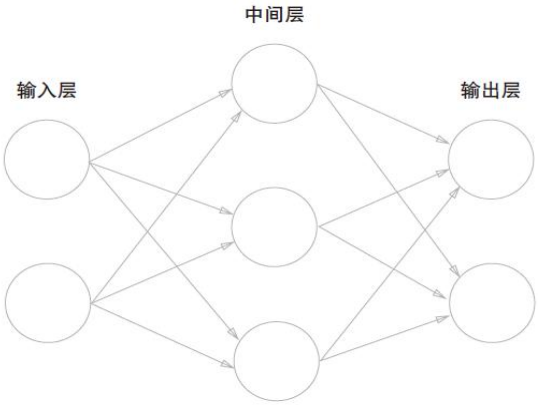

图 2-2 神经网络

我们可以将神经网络分成输入层、中间层（隐藏层）、输出层。

（3）激活函数：

对于加入偏置之后的神经元输出公式，我们可以做如下改进：

$$y = h(b + w_1x_1 + w_2x_2)$$

其中，h（x）为：

$$h(x) = \begin{cases} 0 & (x \leqslant 0) \\ 1 & (x > 0) \end{cases}$$

函数 h（x）可以将输入信号的加权与偏置的和转换成输出函数，这种函数就称为激活函数，h（x）通常被称为阶跃函数，阶跃函数是非线性函数，这也是被选为激活函数的一个条件。

为什么不能用线性函数作为激活函数呢？因为线性函数会使得不论经过多少层，最后的输出都会被局限成为一个线性数据，但很多问题并不能简单地用线性标准来进行分类，而且这样的输出我们总能找到一个没有隐藏层的神经网络来代替，这样就失去了多层网络的优势。因此我们引入一个非线性的连续函数，使得我们的输出变得“多样化”，这样，我们用线性标准无法分类的问题，可以通过曲线来进行分类，从而达到我们的目的。

神经网络中比较常见的一个激活函数就是 sigmoid 函数：

$$h(x) = \frac{1}{1 + \exp(-x)}$$

此外还有 ReLU 函数：

$$h(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leqslant 0) \end{cases}$$

以及 tanh 函数等：

$$y = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

而输出层的激活函数我们也可以不选用上述几种，比较常用的用于多分类问题的输出函数为 softmax 函数：

$$\begin{aligned} y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)} &= \frac{\mathrm{C}\exp(a_k)}{\mathrm{C}\sum_{i=1}^{n} \exp(a_i)} \\ &= \frac{\exp(a_k + \log \mathrm{C})}{\sum_{i=1}^{n} \exp(a_i + \log \mathrm{C})} \\ &= \frac{\exp(a_k + \mathrm{C}')}{\sum_{i=1}^{n} \exp(a_i + \mathrm{C}')} \end{aligned}$$

这里添加常数 C 的作用是防止指数值过大造成溢出。

（4）前向传播：

数据（信息、信号）从输入端输入后，沿着网络的指向，乘以对应的权重后再加和，再将结果作为输入在激活函数中计算，将计算的结果作为输入传递给下一个节点。依次计算，直到得到最终结果。通过每一层的感知器，层层计算，得到输出，每个节点的输出作为下一个节点的输入。这个过程就是正向传播。

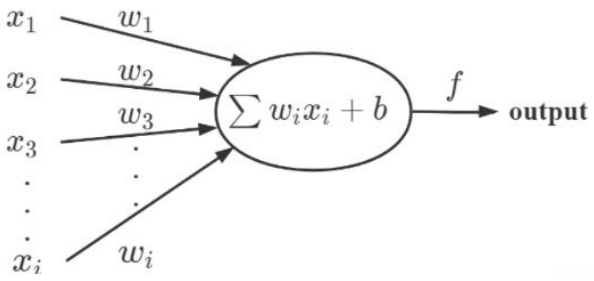

图 2-3 正向传播

（5）反向传播：

将输出的结果与期望的输出结果进行比较，将比较产生的误差利用网络进行反向传播，本质是一个“负反馈”的过程。通过多次迭代，不断地对网络上的各个节点间的权重和偏置进行调整，权重和偏置的调整采用梯度下降法。

（6）梯度下降法：

在正向传播的最后，我们会得到一个预测输出，而预测输出和目标输出之间总会存在误差，这个误差可以用一个函数来表示，这就是损失函数，我们要做的，就是要让损失函数达到最小。

损失函数我们一般选用如下函数，这个函数的形式比较简单，意义也很直观。

$$E = \frac{1}{2}\sum_{k}(y_k - t_k)^2$$

这里的 yk 是神经网络的输出，tk 是目标输出，k 是数据个数。

由微积分知识可以知道，函数梯度的方向是函数值变化最大的方向，因此我们可以沿梯度方向（未必同向）进行数据的修正。

我们要使 E 达到最小值，而 E 又是权值和偏置的函数，因此我们可以对权值和偏置沿对 E 的函数的负梯度方向进行调整，调整量正比于误差 E 对权值或者偏置的导数：

$$\Delta\omega_{j\kappa} = -\eta\frac{\partial E}{\partial\omega_{j\kappa}}$$

应用链式求导法则，我们可以得到权值与偏置的调整量为：

$$\frac{\partial E(i)}{\partial W_{ji}^{(l)}} = \delta_j^{(l)} h_i^{(l-1)}$$

$$\frac{\partial E(i)}{\partial b_j^{(l)}} = \delta_j^{(l)}$$

其中：

$$\delta_j^{(l)} = \sum_{k=1}^{sl+1} W_{kj}^{(l+1)} \delta_k^{(l+1)} f(x)'|_{x=net_j^{(l)}}$$

（7）独热标签：one_hot_label/one-hot 表示是仅正确解标签为 1，其余皆为 0 的数组，就像[0,0,1,0,0,0,0,0,0,0]这样，转换成这种向量之后，可以更方便地进行归类。

2.3 算法步骤

（1）初始化每层之间的权值和偏置，确定激活函数，输出函数，损失函数等，输入训练集。

（2）信号正向传播，确定隐藏层，输出层的输出：加权求和并通过函数。

（3）利用损失函数计算误差，误差反向传播，调整各层之间的权值和偏置。

（4）利用调整后的参数进行新一轮训练。

（5）判断是否达到训练结束条件，如果训练次数达到要求或者正确率达到要求，就停止训练，并画出准确率图像。

（6）利用训练得到的参数去预测测试集，如果达到要求可以应用，没有达到要求需要继续训练。

2.4 具体设计

本次实验要用自己实现的 BP 神经网络进行手写数字的识别。代码分成三个模块，分别是算法模块，训练模块与文件载入模块，下面分别对三个模块进行分析。

（1）算法模块：

①激活函数：sigmoid 函数（上文已经提到过），输出函数：softmax 函数（上文已经提到过），损失函数：上文提到的 E 函数。

②神经网络类：本次实验中使用的神经网络有一层输入层、一层隐藏层、一层输出层。

A. 使用的变量：

params：字典型变量，用于保存神经网络各层之间的权值和偏置，内含 W1，W2（权值），b1、b2（偏置）。

grads：字典型变量，用于保存各个参数的梯度。

B. 包含的方法：

a. __init__(self, input_size,hidden_size, output_size,weight_init_std)

功能：初始化权重和偏置。

参数：各层的结点数，权重初始值。

分析：生成数组长度为隐藏层节点数和输出层节点数的两个 0 数组赋给 b1,b2 两个偏置，作为初始值；生成两个维数分别为（输入层节点数，隐藏层节点数），（隐藏层节点数，输出层节点数）的二维矩阵，并乘以权重初始值，赋给 W1，W2 两个权重。维数的选择要保证矩阵可以相乘。

b. predict(self, x)

功能：输入测试集，返回测试结果

参数：输入的数据

返回值：预测数据

分析：根据 BP 神经网络前向传播的过程可以直接写出，隐藏层输出 z1 等于输入矩阵 x 与权重矩阵 W1 相乘并与偏置 b1 相加之后通过 sigmoid 函数的输出，输出层输出 z2 等于隐藏层输出矩阵 z1 与权重矩阵 W2 相乘并与偏置 b2 相加之后通过 softmax 函数的输出。最后返回输出层输出矩阵即可。

c. gradient(self, x, t)

功能：得到权重和偏置的梯度

参数：输入数据，目标输出

返回值：一个储存 W1，W2，b1， b2 梯度值的字典

分析：首先得到输出层输出 z2，算出输出误差(z2-t)/输入数据数，那么根据之前的分析，W2 的梯度就为隐藏层输出 z1 的转置乘以输出误差，b2 的梯度就是对输出误差求和。在求输出层输出的时候，我们可以求得 a 等于输入矩阵 x 与权重矩阵 W1 的乘积再与偏置 b1 相乘，之后可算出 a 的误差量 da，da 等于输出误差与 W2 转置的乘积，dz1 就等于当自变量等于 a 时，sigmoid 的导数的值乘以 a 的误差量，那么 W1 的梯度就是 x 的转置乘以隐藏层输出 z1 的误差量 dz1，b1 的梯度就是对 dz1 求和。最后返回储存梯度的字典变量。

d. train(self,x,t,lr)

功能：对神经网络进行训练，目的是为了调整神经网络的中的参数。

参数：输入数据，目标输出，学习率

返回值：准确率

分析：首先前向传播得到输出层输出，并与目标输出对比，算出准确率，之后对每个偏置和权重，都在原始值的基础之上减去学习率乘以参数梯度，进行参数的更新。最后返回准确率。

（2）训练模块：

核心思想：

①mini-batch 学习方法，从训练数据中随机选取一部分数据，以这些

mini-batch 为对象进行训练，这样可以提升训练的效率。

②epoch：可以将 epoch 理解成一个单位。一个 epoch 表示学习中所有训练数据均被使用过一次时的更新次数。我们在训练的时候需要定期对训练数据和测试数据记录识别准确率，那么我们可以在每经过一个 epoch 时记录一次训练准确率。

流程：

①读入数据：读入训练集、测试集

②设置初始数据：创建一个神经网络类的对象，其中输入层节点数设为 784（因为输入的图像为 28*28=784），隐藏层节点数为 50（根据隐藏层节点数经验公式 hidden_size = sqrt(input_size+output_size)+a)，输出层节点数 10（一共有 0-9 共 10 种可能的数字），初始化训练次数、训练集大小、batch 大小、学习率、每个 epoch 所重复的次数等。

③开始训练：首先随机选取 batch，并对 batch 进行训练，如果当前训练次数等于每个 epoch 重复次数的整数倍时，就保存训练准确率并输出。如果到达设定的训练次数就停止训练。

④画出准确率随 epoch 的曲线图。

⑤进行测试：因为一共有 28000 个测试图像，所以要重复 28000 次测试，每次测试都要把输出层输出数组中最大元素的下标读取出来，下标就对应预测的数字，保存并写入测试结果文件。

⑥调整：如果测试结果不理想，需要调整数据初始值，重新进行训练和测试。

（3）文件载入模块：

①loadTrainData()

功能：读取训练集数据，获取其中的图像和正确解标签。

返回值：归一化和转换成整型数组之后的图像矩阵和正确解标签向量。

流程：首先一行行地将数据读入一个空表，由于训练集第一行是文字标签，所有先将第一行 remove，然后将目前的表转换成 np 数组，之后利用切片将第一列和其余列分开，实现了正确解标签和图像数据的分离。

②loadTestData()

与上面类似，只不过测试集中没有正确解标签，不需要分离。

③nomalizing(array)

功能：归一化，因为 train.csv 里面提供的表示图像的数据是 0～255 的，为了简化运算，我们可以将其转化为二值图像，因此将所有非 0 的数字，即 1～255 都归一化为 1。

参数：需要进行归一化的数组

返回值：归一化之后的数组

流程：遍历数组，如果某个位置数组元素不为 0，就将其置为 1，最后返回归一化之后的数组。

④toInt(array)

功能：将字符串转换为整数，因为从 csv 文件读取出来的是字符串类型，比如‘253’，而我们接下来运算需要的是整数类型的，因此要转换。

参数：需要进行转换的数组

返回值：转换之后的数组

流程：创建一个新的数组，遍历数组，新数组的对应元素应该为原来的数组元素经过 int 强制类型转换之后的值，最后返回转换之后的数组。

⑤one_hot_label(X)

功能：输入正确解标签向量，将其转换为独热标签

参数：需要进行转换的正确解标签

流程：先创建一个行列数分别为 X 的长度，10 的数组 T，之后遍历 T 的每一行，令 T 每一行中等于 X 的下标处的元素赋为 1 即可，最后返回独热标签 T。

结果分析

由准确率图像可知，每个 epoch 之后所得准确率的波动较大，此时超参数为：

```
iters_num = 10000
batch_size = 100
learning_rate = 0.1
iter_per_epoch = max(train_size / batch_size, 1)
```

图 3-1 原始超参数

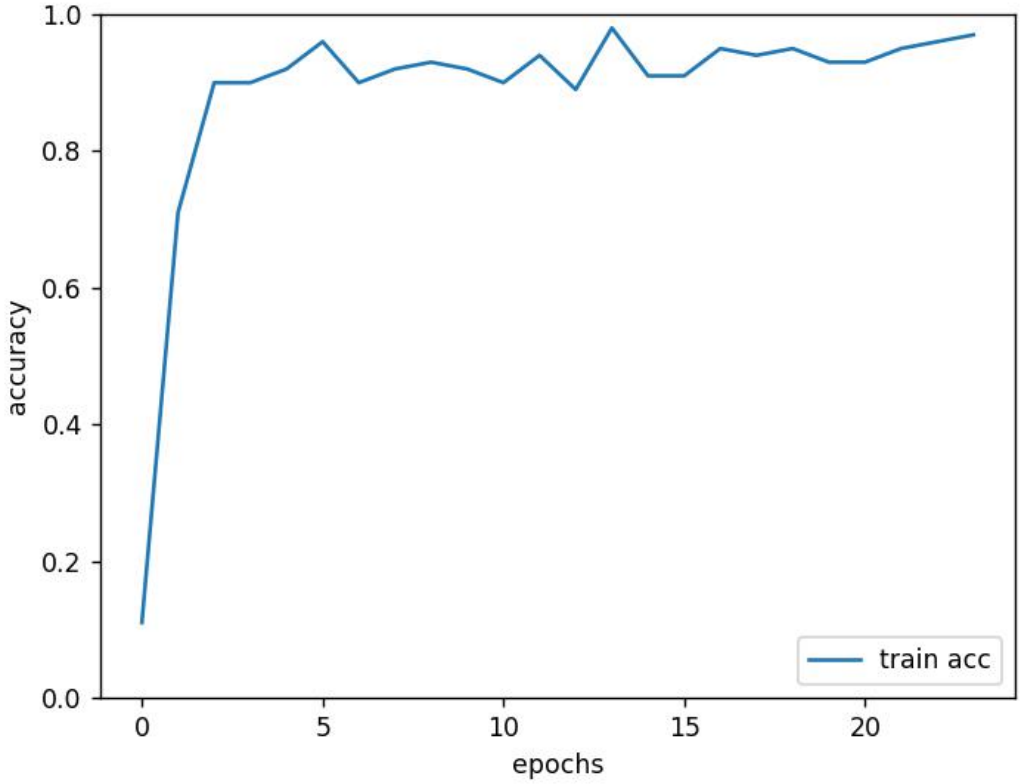

图 3-2 原始准确率图像

经查阅资料可知，影响准确率稳定的因素可能有如下几个：

①激活函数和损失函数：目前激活函数较优的选择是 ReLU 函数，sigmoid 激活函数激活范围太小，容易造成梯度弥散、消失；而多分类问题中，误差函数更优选择是交叉熵误差。

②batch size 是否合适：一般是较大会有比较好的效果，一是更快收敛，二是可以躲过一些局部最优点。但是也不是一味地增加 batch size 就好，太大的 batch size 泛化性不好。较小的 batch size 可能会使得网络有明显的震荡。

③学习率：学习率太大，一步前进的路程太长，会出现来回震荡的情况，但是学习率太小，收敛速度会比较慢。

④训练次数：个人认为，训练次数越多，参数调整的次数越多，越容易接近

最优解。但训练次数越多，所需要的时间越长。

⑤神经网络层数：在一定数值范围内，增加层数可以提高准确率，但随之带来的是训练时间的增长，但可能层数达到一定数值之后，准确率增加效果不明显，因此需要调整别的参数。

⑥算法：算法的编写可能让计算不够简练，也有可能会影响准确率。

下面是调整后的超参数与准确率图像，可以看出效果好了一点。

```
iters_num = 40000
batch_size = 400
learning_rate = 0.2
iter_per_epoch = max(train_size / batch_size, 1)
```

图 3-3 调整后超参数

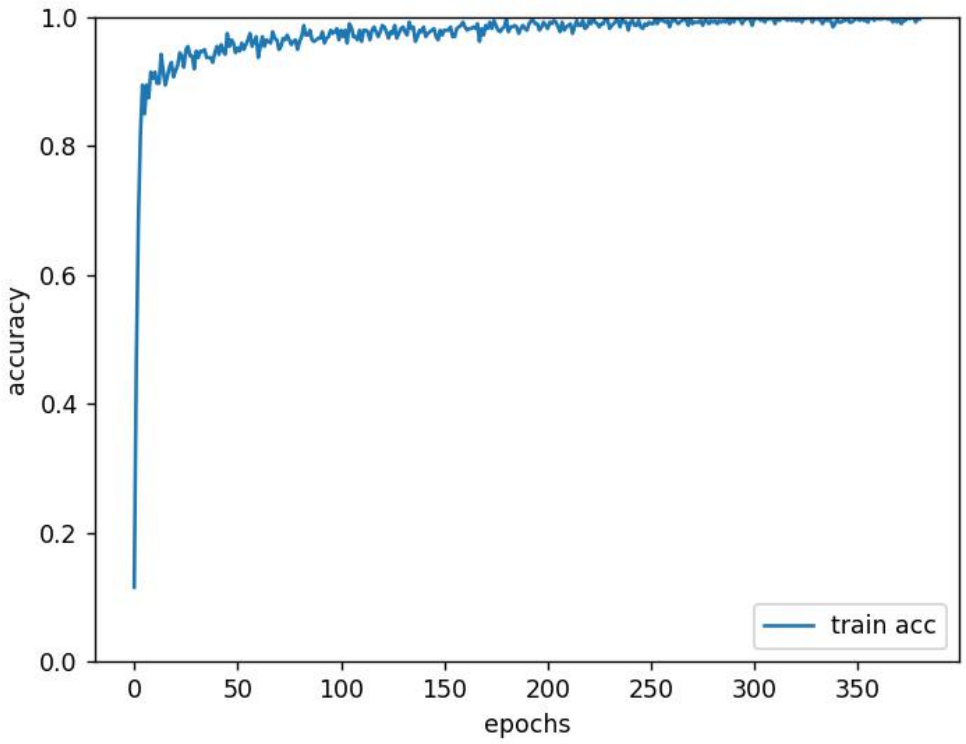

图 3-4 调整后准确率图像

将测试集的测试结果写入文件并提交至 kaggle，可知本模型具有一定的效果，但还需要进一步优化。

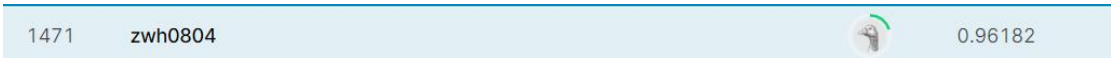

图 3-5 测试结果

实验小结

4.1 模型优缺点分析

BP 神经网络具有以下优缺点：

优点：

①选取非线性函数作为激活函数，可以解决不能用线性标准进行分类的问题。

②采用误差反向传播，从输出层到隐藏层逐层进行参数修正，进行自我学习并记忆学习结果，随着学习的不断进行，误差会越来越小。

③网络结构较为简单，采用很少的神经网络层就可以获得较好的训练结果。同时隐藏层的加入，使得数据在传播过程中可以被挖掘出更多的信息，训练效果较好。

缺点：

①需要设定的超参数较多，batch size，learning rate 等参数的选取会在很大程度上影响训练效果（在结果分析中有提到），不同的问题需要不同的选取策略，没有普适性的选取算法。

②容易陷入局部最优，使训练在局部最优左右徘徊，无法到达全局最优。

③权重和偏置等参数是随机初始化的，前后往往会得到不同的训练结果。

④训练结果与训练集有很大关系，如果训练集不够典型，不能很完整包含所有情况，训练结果就很难达到预期。

⑤训练速度，准确率收敛速度较慢。

⑥可能会出现过拟合现象。

4.2 遇到的问题

之前学习过 python，但并没有用 python 来解决一些实际问题，对 python 算不上太熟悉，对神经网络相关知识也是十分陌生，在网上查阅了大量书籍和资料之后才着手编写代码。编写过程中，由于对误差反向传播不太熟悉，花了大量的时间去研究写法，最终结果差强人意。

如何处理读入的文件也是一个困扰了我很长时间的问题，在查阅资料的过程中，重新学习了之前不太熟悉的文件操作，对 list 的操作，切片操作等，最终参考网上资料，成功解决问题。

4.3 心得体会

重新拾起一个已经十分陌生的语言对我来说是一个不小的挑战，但因为自己本身对人工智能有兴趣，所以自己在接触一个新的领域——神经网络的时候，虽然有很多困难，很多 python 的语法不是很熟悉，刚开始也有很多看不懂的地方，但深入钻研其中的数学原理的时候，发现神经网络模型还是非常有意思的，自学习的核心思想与之前简单编写算法有很大的不同。

在编写程序的过程中，让我印象最深刻的，就是要深入了解其中的数学原理，多问几个为什么，否则还是会感觉很迷茫，就像误差反向传播算法，本身并没有很复杂，需要多下点功夫的地方就是误差项的推导，在遇到困难的时候，查阅资料和求助老师同学都是不错的选择。

在遇到没有接触过的语法和库函数的时候，要多去查阅资料弄清其使用方法，并横向比较与其他库函数的区别，选择最适合的。

这次由于时间较为紧迫，并没有对模型进行很大的优化，而是选取了一种较为简单的方案，搭建的神经网络比较简单，最终得到的结果差强人意，希望以后可以继续优化自己的模型和算法，得到更好的结果。

总之，有困难的时候最先想到的不应该是放弃，而是应该耐心思考，把原理多看几遍，有些问题可能就会迎刃而解，这次经历对我来说既有挑战又有意义，让我渐渐摸索出了自己探索不太熟悉的知识领域的方法，希望以后可以继续努力，在人工智能领域进行更深刻更广袤的探索。

Be the best AI trainer！

附录 源代码

network.py:

```
import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def d_sigmoid(x):
    t = sigmoid(x)
    return t * (1 - t)

def softmax(x):
    if x.ndim == 2:
        x = x.T
        x = x - np.max(x, axis=0)
        y = np.exp(x) / np.sum(np.exp(x), axis=0)
        return y.T
    x = x - np.max(x) #防止溢出的修正
    return np.exp(x) / np.sum(np.exp(x))

class Network:

    def __init__(self, input_size, hidden_size, output_size, weight_init_std=0.01):
        # 初始化权重
        self.params = {'W1': weight_init_std * np.random.randn(input_size,
hidden_size), 'b1': np.zeros(hidden_size),
                       'W2': weight_init_std * np.random.randn(hidden_size,
output_size), 'b2': np.zeros(output_size)}

    def predict(self, x):
        #前向传播
        W1 = self.params['W1']
        W2 = self.params['W2']
        b1 = self.params['b1']
        b2 = self.params['b2']
        z1 = sigmoid(np.dot(x, W1) + b1)
        z2 = softmax(np.dot(z1, W2) + b2)
        return z2

    def gradient(self, x, t):
        #求梯度
        W1 = self.params['W1']
        W2 = self.params['W2']
```

```
        b1 = self.params['b1']
        b2 = self.params['b2']
        grads = {}
        batch_num = x.shape[0]
        a1 = np.dot(x, W1) + b1
        z1 = sigmoid(a1)
        a2 = np.dot(z1, W2) + b2
        y = softmax(a2)
        dy = (y - t) / batch_num
        grads['W2'] = np.dot(z1.T, dy)
        grads['b2'] = np.sum(dy, axis=0)
        da1 = np.dot(dy, W2.T)
        dz1 = d_sigmoid(a1) * da1
        grads['W1'] = np.dot(x.T, dz1)
        grads['b1'] = np.sum(dz1, axis=0)
        return grads

    def train(self, x, t, lr):
        #训练
        y = self.predict(x)
        grad = self.gradient(x, t)
        y = np.argmax(y, axis=1)
        t = np.argmax(t, axis=1)
        accuracy = np.sum(y == t) / float(x.shape[0])
        #误差反向传播
        for key in ('W1', 'b1', 'W2', 'b2'):
            self.params[key] -= lr * grad[key]
        return accuracy
```

main.py:

```
import numpy as np
import csv
import load
import matplotlib.pyplot as plt
import network

#读入数据
x_train, t_train0 = load.loadTrainData()
t_train = load.one_hot_label(t_train0)
x_test = load.loadTestData()
#初始化神经网络
network = network.Network(input_size=784, hidden_size=50, output_size=10)
train_size = x_train.shape[0]
```

```
#设置超参数
iters_num = 40000
batch_size = 400
learning_rate = 0.2
iter_per_epoch = max(train_size / batch_size, 1)

train_acc_list = []
i = 0
train_acc = 0

#mini-batch 方法选取数据进行训练
for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]
    train_acc = network.train(x_batch,t_batch,learning_rate)
    #达到一个 epoch，储存训练结果
    if i % iter_per_epoch == 0:
        train_acc_list.append(train_acc)
        print("train acc:" + str(train_acc))

#画出准确率图像
markers = {'train': 'o'}
x = np.arange(len(train_acc_list))
plt.plot(x, train_acc_list, label='train acc')
plt.xlabel("epochs")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.legend(loc='lower right')
plt.show()

#处理测试结果，将测试结果写入数组
num = []
result = []
for i in range(28000):
    num.append(i+1)
    y = network.predict(x_test[i])
    result.append(np.argmax(y))

#测试结果写入文件
with open("submission.csv","w",newline="") as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(["ImageId", "Label"])
    for i in range(28000):
```

```
            writer.writerow([num[i], result[i]])
```

load.py:

```
import csv
import numpy as np

#直接读取 csv 文件的结果是字符串，将字符串转换成整型变量
def toInt(array):
    array=np.mat(array)
    m,n=np.shape(array)
    newArray=np.zeros((m,n))
    for i in range(m):
        for j in range(n):
                newArray[i,j]=int(array[i,j])
    return newArray

#数据归一化，方便处理数据
def nomalizing(array):
    m,n=np.shape(array)
    for i in range(m):
        for j in range(n):
            if array[i,j]!=0:
                array[i,j]=1
    return array

#读入训练数据
def loadTrainData():
    l=[]
    with open('train.csv') as file:
        lines=csv.reader(file)
        for line in lines:
            l.append(line)
    l.remove(l[0]) #去掉第一行标签
    l=np.array(l)
    label=l[:,0] #取第一列作为标签
    data=l[:,1:] #取其余的列作为图像
    return nomalizing(toInt(data)),toInt(label)

#读入测试数据
def loadTestData():
    l=[]
    with open('test.csv') as file:
        lines=csv.reader(file)
        for line in lines:
```

```
                l.append(line)
    l.remove(l[0]) #去掉第一行标签
    data=np.array(l)
    return nomalizing(toInt(data))

#转换成独热标签
def one_hot_label(X):
    T = np.zeros((X.size, 10))
    for idx, row in enumerate(T):
        row[int(X[0][idx])] = 1 #每一行对应下标处置 1
    return T
```