Lab2 的 part2 文字分类部分我使用 java，一开始使用 python 写的，但是训练的时间过长，因此使用 java 又写了一遍，以提升速度。具体代码说明和实验过程见如下说明：

一、函数方法说明

一）BPNetwork 类：该类是分类的 bp 网络的核心部分，其方法基本和 part1 的方法差不多。

1. private double rand(double a,double b)：该方法返回在区间（a，b）的任意随机数，且保证每次调用的时候返回的都是与之前不同的随机数
2. private double[][] generate_w(int m,int n)：该方法返回生成的 m*n 的 weight 矩阵，且通过在该方法中调用 rand(a,b)来初始化每个 weight 的值
3. private double[] generate_b(int m)：该方法返回生成的 m 长度的 bias 值，且通过在该方法中调用 rand(a,b)来初始化每个 bias 的值
4. private void init_array(double num,double[] array)
   private void init_array(int num,int[] array)：这两个函数是将数组初始化为 num
5. private double[] get_shift(double[] score)：得到数组 score 的偏置，使 score 的每个元素 s 减去 score 中最大的值作为 s 的新值，这个方法调用的目的是为了防止在进行数字指数化时溢出。
6. private int get_max_index(double[] array)：得到 array 中最大的元素的 index
7. private double[] softmax(double[] x)：进行 softmax 计算，公式为
   $\exp(x[i])/\Sigma_{j=0}^{x.len}\exp(x[j])$
8. private double tanh(double x):返回 tanh(x)计算的结果
9. private double tanh_deriv(double x)：返回 tanh 的导数，1-tanh(x)*tanh(x);
10. public BPNetwork(int input_n, int output_n, int[] hidden_set)：进行 bp 网络的初始化，与 part1-1 的 setup 方法的效果相同，不再赘述。只有如下修改：
    this.predict 为进行 softmax 之后的网络的输出值，this.output_cells 为未进行 softmax 之前的输出值。this.output_cells 需要计算 loss 的时候使用，因此分离开来。
11. private double[] forward_proagate(int[] input)：输入层→隐藏层和隐藏层之间的传播方式与 bp 神经网络相同，依旧使用公式
    $Output[h]=fit\_function((\Sigma_{i=0}^{输入层神经元个数}weight[i][h]*input[i])+bias[h])$
    不同的只有最后一层隐藏层到输出层的计算改为：
    $Output\_cells[h]=\Sigma_{i=0}^{输入层神经元个数}weight[i][h]*input[i]$
    Predict[h]=softmax(output_cells)[h]
12. private void get_deltas(int label)：输出层到最后一层隐藏层的计算改为 output_deltas[i] = true_label[i]-preict[i]，这里的输入值 label 与 true-label 有所不同，输入的 label 是指应该分到的 k 类，输入的是 k，但是 true_label[i],如果 i=k，则其为 1，如果 i≠k，则为 0
13. private void renew_w(double learn)
    private void renew_b(double learn):
    这两个方法的思路同 bp 网络 part1-1 设计的思路，因此不再赘述。所不同的只有一点，因为最后一层隐藏层到输出层不需要 bias，因此 output_b 的更新就可以不需要了。
14. private double back_propagate(int[] input, int label, double learn)：需要说明的是 label 指的是第 label 类。可以直接调用以上方法就可以了。
15. private double get_loss(int label)：得到此时的 loss 值。
    Score_shift = get_shift(this.output_cells);
    $Loss=\log(\Sigma_{i=0}^{score\_shift.len}\exp(score\_shift[i]))-score\_shift[label]$
16. public void train(int[][] input_datas,int[] labels,double learn,int limit):给一组输入样本 input_datas 和该样本对应的正确输出类别 labels,学习率 learn，以及训练次数 limit，进行循环训练，需要注意的是，为了保证训练结果尽量不出现局部最小值，需要在每次训练之后

对训练样本进行打乱。

17. public void upset_train_data()：这个方法就是打乱训练样本的方法。
18. public double test_result():这个方法测试测试集的正确率，返回的就是测试集的正确率。
19. public double train_test()：这个方法测试训练集的正确率，返回的就是正确率。

二）BmpReader 类：读取图片的类

1. public static int[] read(String filepath)：读取指定路径的图片，返回一个一维数组。
2. public static int[] transform(int[][] img)：将一个图片像素二维数组转化为一维数组。服务于上一个函数。

三）ReadImage 类，读取助教给的图片信息，得到 train_data，train_label，test_data，test_label.

1. public int[] getTest_label()
   public int[][] getTest_data()
   public int[] getTrain_label()
   public int[][] getTrain_data()

返回 train_data，train_label，test_data，test_label.

2. public String[] get_image_paths(String path)：返回图片的路径数组
3. public int[][][] get_total_images()：返回所有的图片，image[i][j][]为类别为 i 的第 j 个图片。

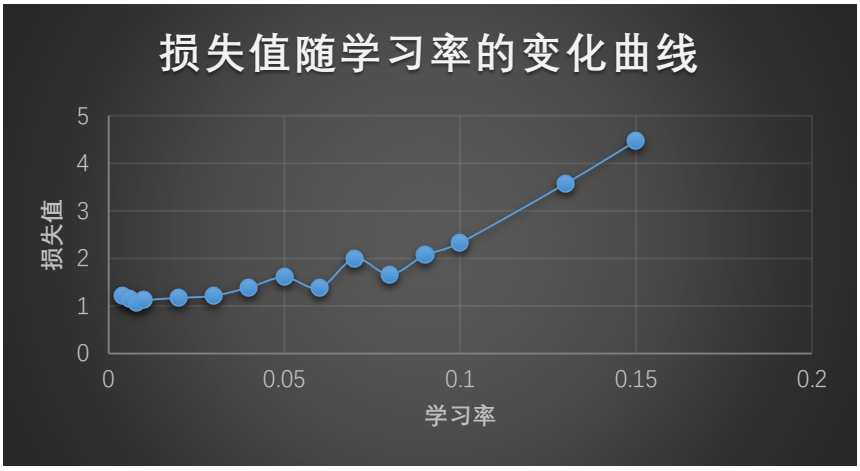
四）BPNetworkClassify 类，这个类是用来测试实验的类。

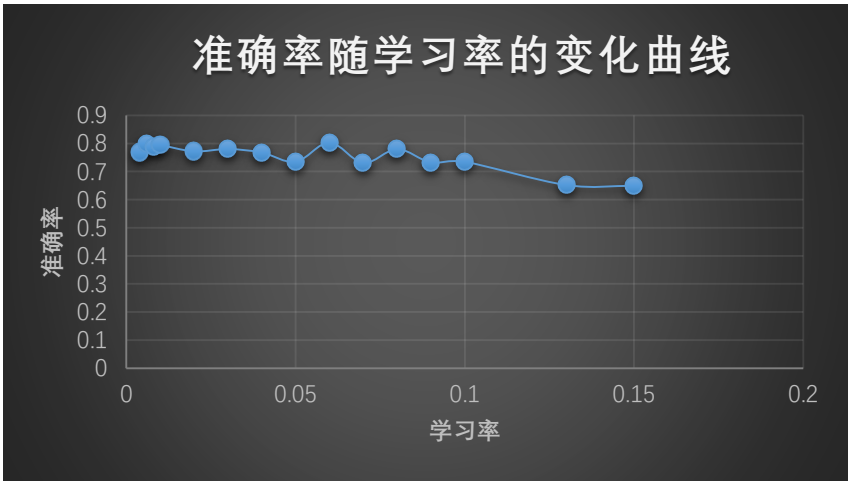
二、实验内容——各种因素对损失值及正确率的影响

前提是 weight 取（-0.9，0.9）的随机值，bias 取（-1.0，0）的随机值。

1. 学习率对损失值和正确率的影响（训练 150 次）：

   前提是训练 150 次，bp 神经网络结构为 784，50，50，14.

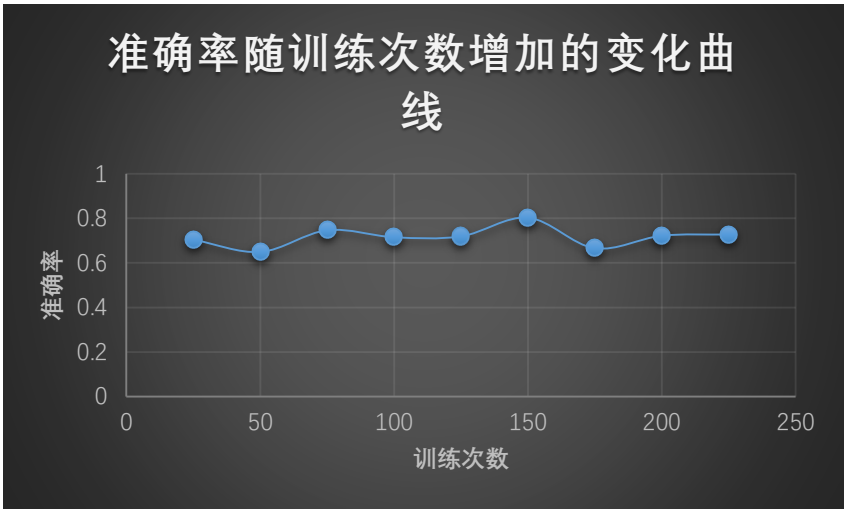


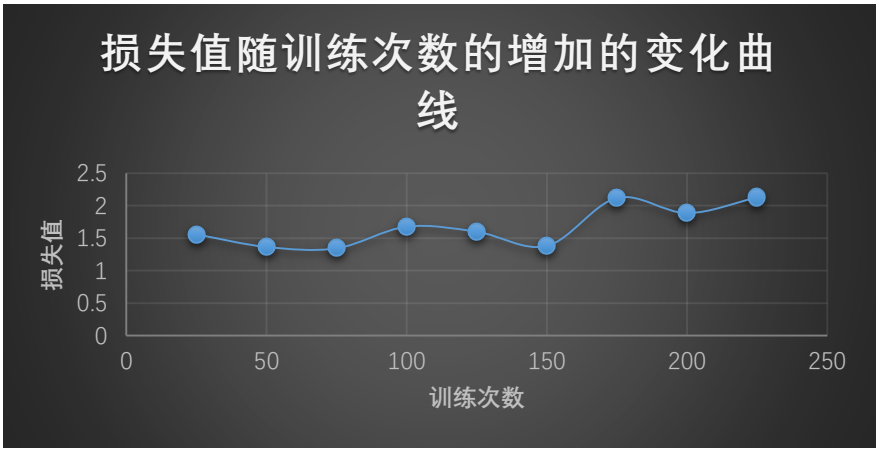


从图中可以看出准确率在学习率为 0.06 左右比较高，最高可以达到 80.35%左右。且随着学习率升高，会看到 loss 值呈上升趋势，准确率呈下降趋势。

2. 训练次数对损失值和正确率的影响：

前提是神经网络结构为 784，50，50，14，学习率为 0.06



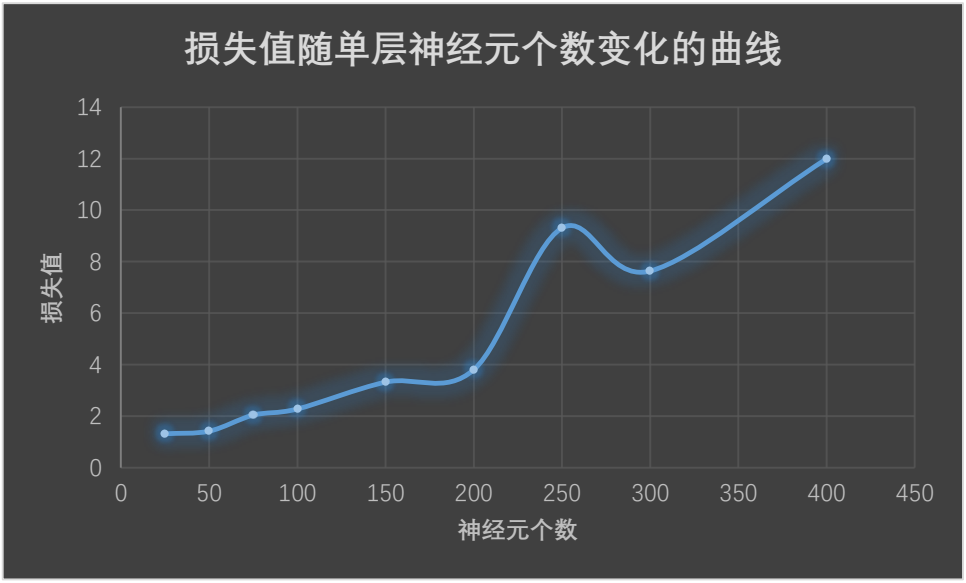


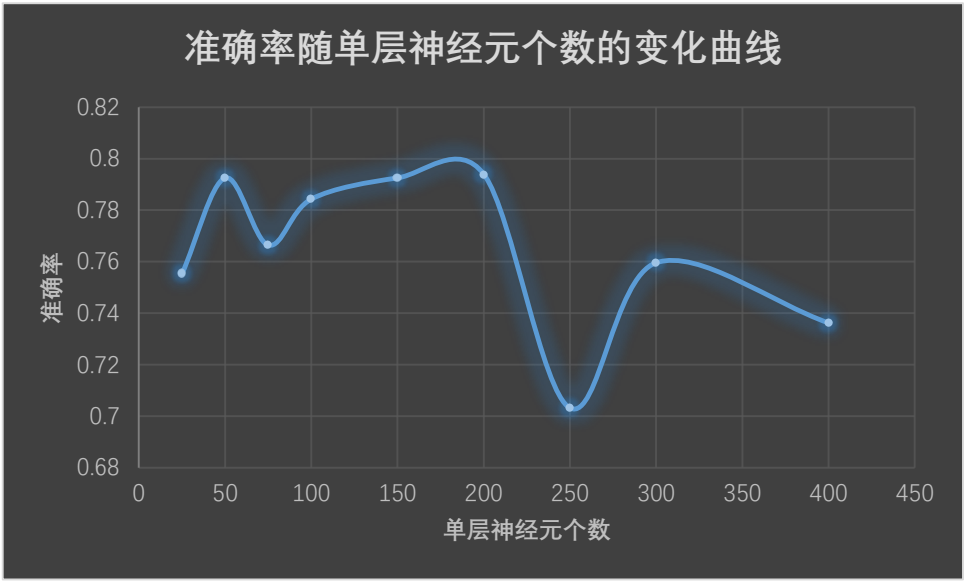
在图中可以看出，此种前提下，训练次数在 150 次左右的准确率较大，训练次数增大，损失值的变化范围不是很大，整体来讲变化比较平稳，我猜想是因为隐藏层数量较多的原因导致训练次数对最后结果的影响不是很大。

3. 隐藏层神经元个数对损失值和正确率的影响：

①隐藏层数为 1 时：

前提为训练 50 次，学习率为 0.06。（训练 50 次是为了节约时间，如果训练 150 次效果会更好）





在图中可以看出，随着神经元个数的增加，损失值整体上是呈上升趋势的，准确率也有所波动，但大概稳定在 70%-80%之间，由图中还可以看出，在单层有 200 个神经元左右的时候，准确率比较大，损失值也比较小，这只是训练了 50 次，在训练 150 次（比较优的训练次数）时，准确率可以突破 80%

经过实验，在 200 个神经元，训练 150 次时可以达到 82.56%左右的正确率。损失值为 4.407，关于为什么准确率大，而损失值也大的原因，我猜想与测试集文字的差别有关。

②隐藏层数大于等于 2 时：

同样训练 50 次，则隐藏层数为 2 时：

隐藏层设置	准确率	损失值
[30,30]	0.5961	1.9263
[40,40]	0.7308	1.1836
[50,50]	0.7184	1.4992
[80,80]	0.71016	1.64944

隐藏层设置	准确率	损失值
[30,30,30]	0.651	1.3122
[40,40,40]	0.5467	1.7064
[80,80,80]	0.53159	1.4573
[100,100,100]	0.5315	1.4573

从这些准确率和损失值的变化中我们可以看到，隐藏层设置层数多并不能给来准确率的上升，相反，一层神经元情况下准确率会更高一些。

4. Weight 和 bias 的初始值对 loss 和准确率的影响

调 weight：

w（-0.9，0.9） b（-1.8,0） 0.06 76% 50 次

w(-1,1) b(-2,0) 0.06 67% 50ci

w(-1,1) b(-2,0) 0.1 100ci 65%

w(-1,1) b(-2,0) 0.06 100ci 73.2%

w(-1,1) b(-2,0) 0.07 100ci

w（-0.9，0.9） b（-1.0,0） 0.06 76% 50 次 (50,50)

循环训练打乱样本 w(-0.9,0.9) b(-1.0,0) 0.06 78% 100 次

循环训练打乱样本 w(-0.9,0.9) b(-1.0,0) 0.06 80.35% 150c (50,50)

循环训练打乱样本　w(-0.9,0.9) b(-1.0,0) 0.059　150c　(50,50)　80.35%

这是进行准确率调整的数据，这里就不再一一展示了。

5. 不同线性函数对网络准确率和损失值的影响。Sigmoid 由于没有负数，它的拟合没有 tanh 的效果好，但是在做分类器时没有像拟合 sinx 那样差别很大，具体数据就不一一展示了。

三、实验过程遇到的问题及解决

实验中遇到了很多问题，第一个问题就是 softmax 求导的问题，搜了很多资料，在几篇博客里找到了答案，这个过程是真的很繁杂。

第二个问题就是 python 运行慢，而且训练的准确率不高，两层神经元网络训练 50 次就要 1 个多小时，之后询问助教，这个太慢助教也不能接受，建议我用 java 写一遍，然后我又用 java 写了一遍，真的是很心累。但是写完之后准确率也上升了，而且运行速度非常感人。

第三个问题就是图片读取的问题，大部分是在网上找的代码，然后拼凑出来的，但是感觉拼出来的效果还不错。