LAB1 文档

闫世艺 16302010076

第一部分：灵活反向传播算法

1.1 代码结构

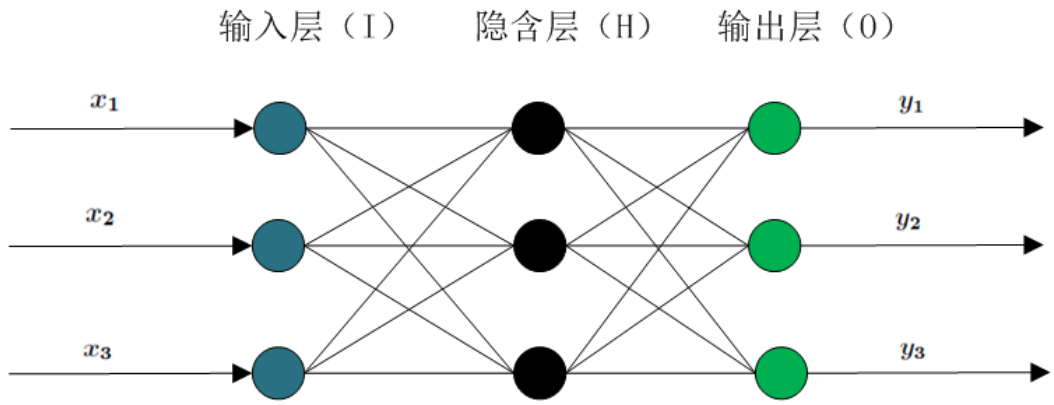
函数或属性	作用
工具函数	
rand(a,b)	生成 a，b 之间的随机数
init_weight	初始化权值，初始化为(-1，1)之间的随机数
init_bias	初始化阈值，初始化为(-1，1)之间的随机数
tanh	双曲正切激活函数
tanh_derivative	双曲正切激活函数的梯度函数
类：BPNetwork	**BP 神经网络的基础结构和方法，可在拟合 sinx 以及做文字分类时候重复使用**
属性：	
self.input_n = 0 self.input_cells = [] self.output_n = 0 self.output_cells = [] self.input_w = [] self.output_w = [] self.hidden_set = [] self.hidden_ws = [] self.hidden_bs = [] self.output_b = [] self.hidden_results = [] self.output_deltas = [] self.hidden_deltases = []	输入神经元数 输入神经元数组 输出神经元数 输出神经元数组 输入层到隐藏层的权值数组 隐藏层到输出层的权值数组 隐藏层的每层网络节点数 隐藏层之间的权值数组的数组 隐藏层之间的 Bias 数组的数组 输出层的 Bias 数组 隐藏层的输出 输出层的δ数组 隐藏层的δ数组

函数：	
set_up(input,output,hidden_set)	初始化神经网络的基本结构，输入层，输出层，隐藏层，及其中间的网络结构。可自定义网络层数和每层节点数目
forward_propagate	向前传播，第 k 层第 j 个节点的输出值 Z[k][j] = tanh((Σj (Σi=前一层输入神经元个数 weight[k-1][i][j] * input[k-1][i])) + bias[k-1][j])
calculate_delta	计算δ，根据梯度下降算法寻求局部最优解
update_weight	更新权值，从输出层向前更新
update_bias	更新阈值，从输出层向前更新

1.2 对反向传播算法的理解

1.2.1 神经网络的结构

如下图 (图源网络):



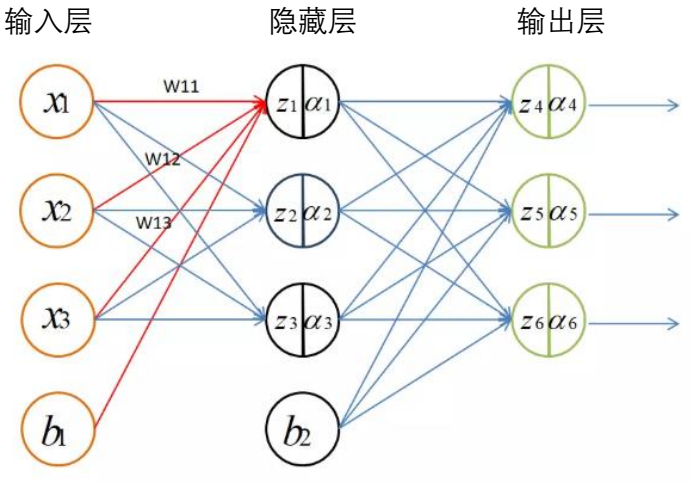
- 输入层（I）：输入数据，单个数据的 n 维的输入
- 隐含层（H）：隐藏层可以又多层，每一层可以有不同的节点数目，而这些因素都会影响神经网络的学习效果
- 输出层（O）：输出数据，单个样本的 m 维的输出

每一层的节点与上一层都有连接，即全连接。除输入层外，其他每一层的每个节点的值，都是通过前一层节点的值，和对应的权值，阈值，以及激活函数计算之后得到的。除输出层外，每一层的输出又作为下一层的输入。

1.2.2 反向传播算法的过程

第一步、向前传播

如下图(图源网络):



向前传播的公式如下:

$Z_1 = X_1 * W_{11} + X_1 * W_{11} + X_1 * W_{11} + b_1$

$\alpha_1 = \varphi(Z_1)$

计算公式如下:

$$Z[k][j] = \varphi((\Sigma j\ (\Sigma i\ weight[k-1][i][j] * input[k-1][i])) + bias[k-1][j])$$

其中:

$X_1, X_1, X_3 \cdots\cdots X_n$ 是前一层的输入

$W_{11}, W_{12}, W_{13} \cdots\cdots W_{1n}$ 是对应的权重值

b_1 是对应的阈值

φ 是激活函数

第二步、计算误差，求得δ

由于实际输出（d_j）与期望输出值（y_j）是存在误差的，所以反向传播算法的核心就是通过不断“训练”，计算前向传播产生的误差，不断调整输入层，隐藏层之间的权值和阈值，来不断减少误差。通过多次向前传播，计算误差，调整权阈值，这几个步骤的重复，最终达到学习的效果。

方法：**梯度下降法**

定义损失函数为均方差函数

$$E(w,b) = \frac{1}{2}\sum_{j=0}^{n-1}(d_j - y_j)^2$$

根据梯度下降法，通过沿着最小平方和的最快下降方向，调整网络的权值和阈值

- **对于 W(i,j) $\delta E(w,b)/\delta W(i,j) = \delta_{ij} * y_i$，**$y_j$是输出值

  神经元 j 是输出层节点时，局部梯度 δ_j等于倒数$\varphi'(z_j)$和误差 $d_j - y_j$的乘积。

  $$\delta_{ij} = \varphi'(Z_j) * (d_j - y_j)$$

  神经元 j 是隐藏层节点时，局部梯度 δ_j等于倒数$\varphi'(z_j)$和下一层的 δ_k与权值加权和的乘积。

  $$\delta_{ij} = \varphi'(Z_j) * \Sigma_k(\delta_k W_{kj})$$

- **对于，**$B(i,j)\delta E(w,b)/\delta B(i,j) = \delta_{ij}$，$y_j$是输出值

此处省略求解**δ**的化简过程，因为输出层有直接的期望输出，所以此处的局部梯度可以直接由误差得到。但是隐藏层以及之前的每一层必须通过后面与之直接相连的网络层递归得到。

第三步、更新权值和阈值

- $\Delta W = \mu * \delta E(w,b)/\delta W(i,j) = \mu * \delta_{ij} * y_i$，μ是学习率

  从最后一层输出层向前更新权值

- $\Delta B = \mu * \delta E(w,b)/\delta B(i,j) = \mu * \delta_{ij}$，μ是学习率

  从最后一层输出层向前更新阈值

第四步、重复以上三步操作，重复次数即为学习次数

1.2.3 影响算法学习效果的因素

- **激活函数**

  激活函数一般是非线性的，因为如果是一个线性的激活函数，那么无论神经网络有多少层，都可以线性转化为一层，这样多层神经网络的意义就不存在了。其次不同的激活函数有不同的激活特性，比如 Sigmoid 激活函数的输出范围都是大于零的，tanh 激活函数的输出范围是（-1，1），这样就会产生不同得先熬过，这与神经网络的需求也是有关的，还有激活函数一般要求处处可导，因为如果我们用到梯度下降法，就需要激活函数是可导的。反向传播时多个导数相乘，会出现梯度消失的情况。

- **梯度下降法**

  梯度下降法本身无法保证得到全局最优解，其思想是得到局部最优解，所以训练的结果很有可能掉入一个小的局部最优解

- **损失函数**

  损失函数用来计算期望值与实际输出之间的误差，损失函数的结果越小，则说明输出结果越接近期望值，学习效果越好。如果损失函数能够更加准确的估计模型的误差，那么通过训练就更能逼近正确的结果。

- **神经网络层数和每层节点数**

  神经网络层数和每层节点数会直接影响到权值加权之后的结果，层数以及每层的节点数都会直接影响神经网络学习的效果。

- **学习率**

  学习率一般取（0，1）之间的数值，当学习率较小的时候，每次的反向传播的权值，阈值调整较小，学习速度比较慢，学习率变大之后则很容易使变化量不稳定。

第二部分：拟合 sinx

2.1 代码结构

方法	**作用**
get_train	生成训练集（-pi，pi），精度为 0.1
get_test	生成测试集（-pi，pi），精度为 0.01
back_propagate（input,output,learn,times）	反向传播，调用 BPNetWork 类中的方法。主要是向前传播，计算δ，更新 W，B，重复以上三步。
__main__	1. 初始化神经网络层数结构 2. 反向传播算法对 sinx 进行拟合 3. 绘图

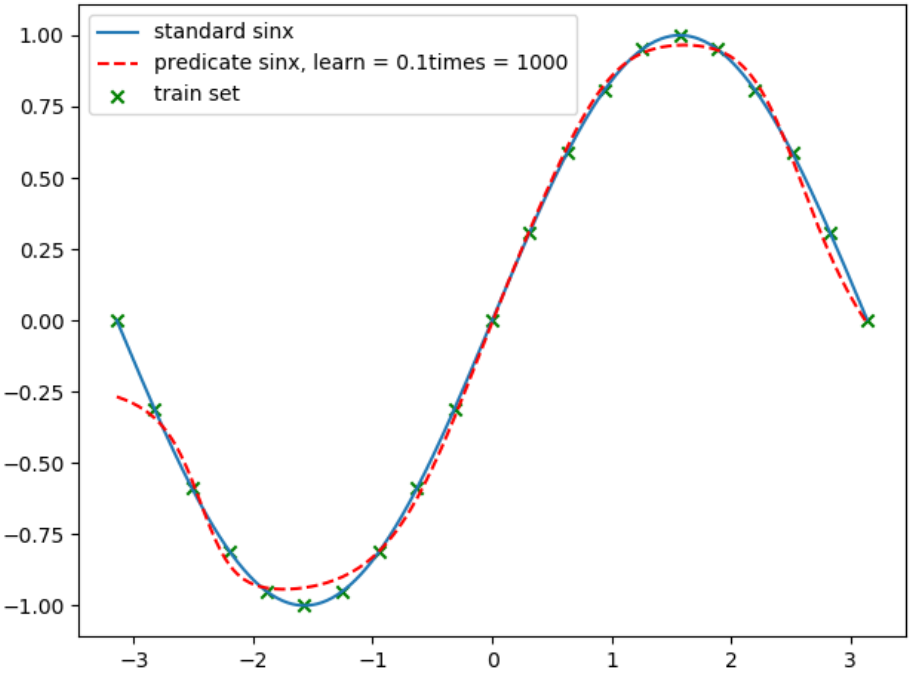
2.2 不同实验参数的拟合效果

自定义参数：

- 网络层数
- 每层节点数
- 学习率
- 训练次数

* 根据经验公式：每层节点数 h = $(n+m)^{1/2}+a$　(0<a<10)

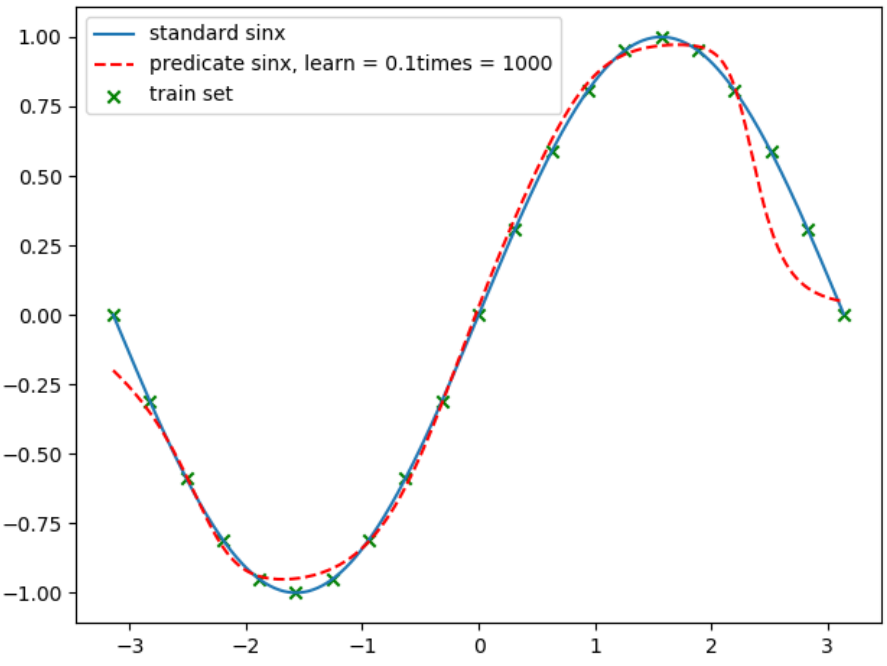
先设定一个训练的基础标准：网络层数 2 层，每层节点数 5 个，学习率 0.1，训练次数 1000，看一下训练效果，如下图：



平均误差为 0.0012614827478683035

1. 不同网络层数和不同节点数

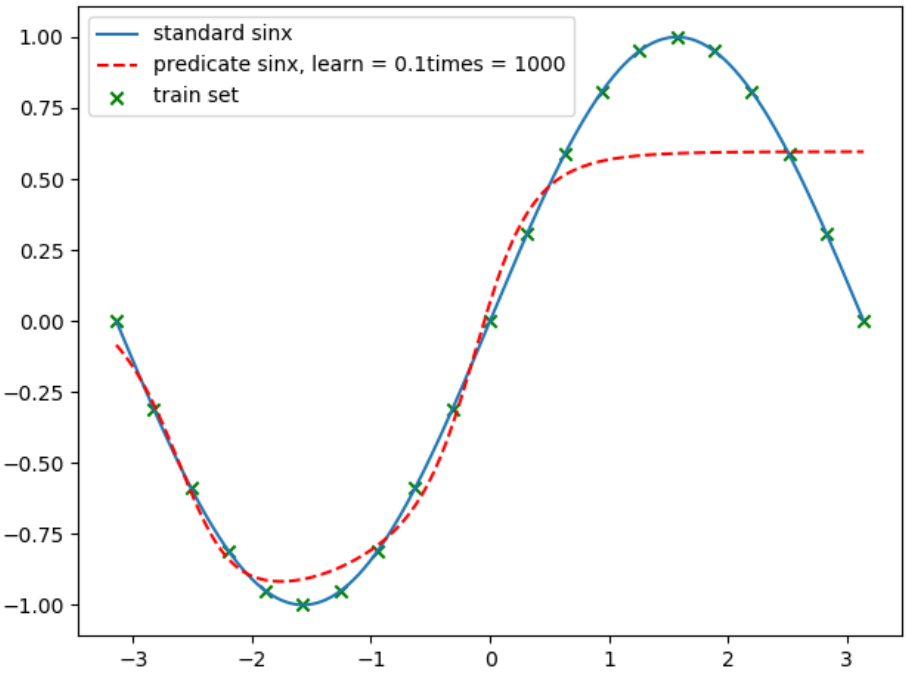
1.1 隐藏层节点设置为[5, 5, 5]的时候，如下图：



可以看到网络层数增加，拟合效果反而下降了，所以决定不再增加网络层数（继续增加之后，loss 继续增大，此处不再贴图片）

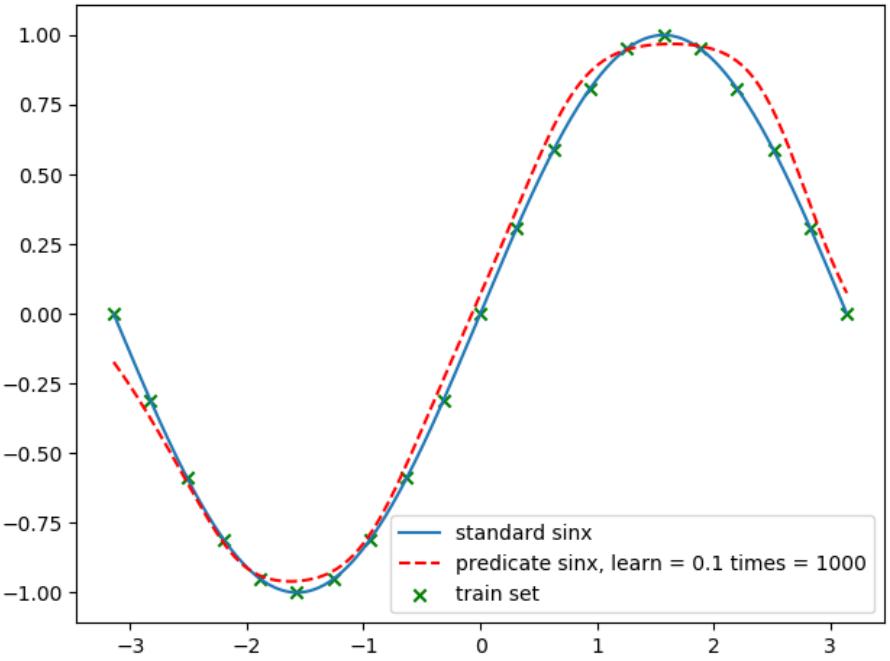
2. 不同节点数

2.1 隐藏层节点设置为[2,2]的时候，如下图:



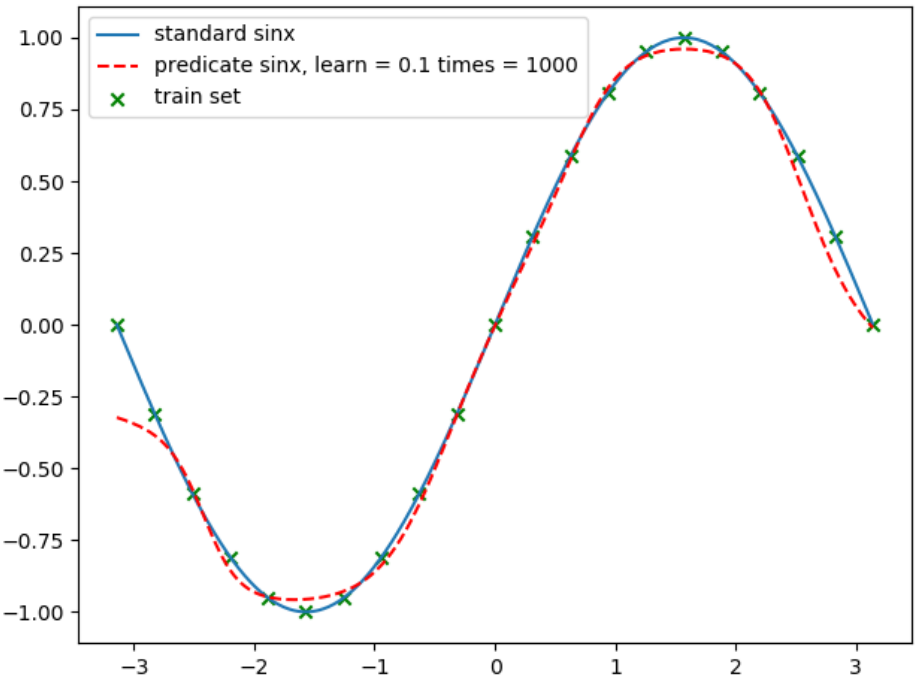
此时拟合效果出现较大偏差，平均误差为 0.020550235227835684

2.2 隐藏层节点设置为[10,10]的时候，如下图：



拟合效果相比没有得到改善，平均误差为 0.0021691601667659433

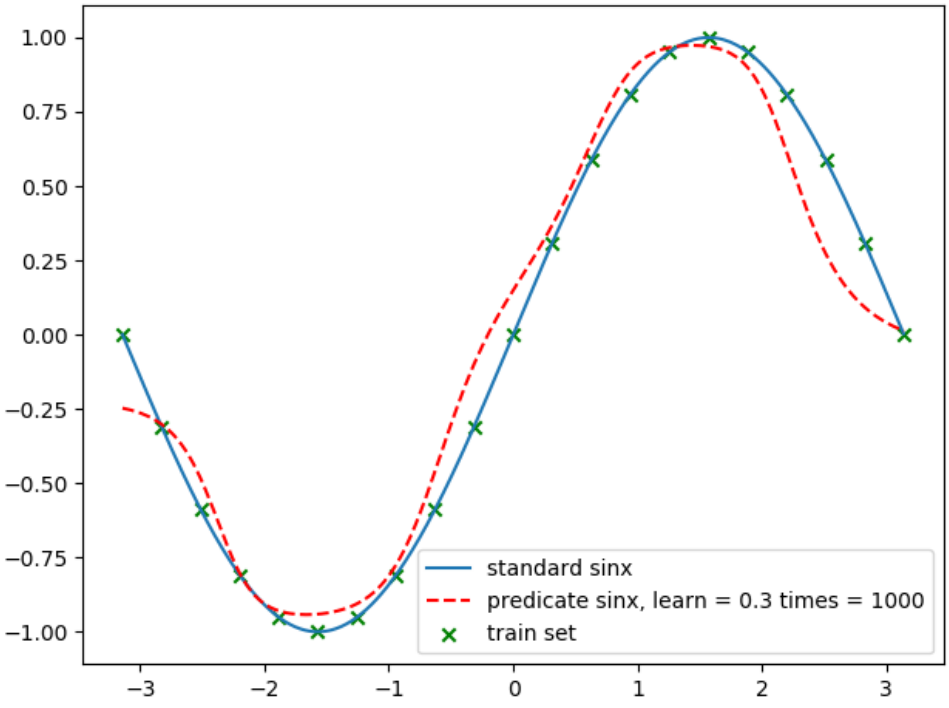
2.3 隐藏层节点设置为[9,5]的时候，如下图



平均误差为 0.0019329660307345088

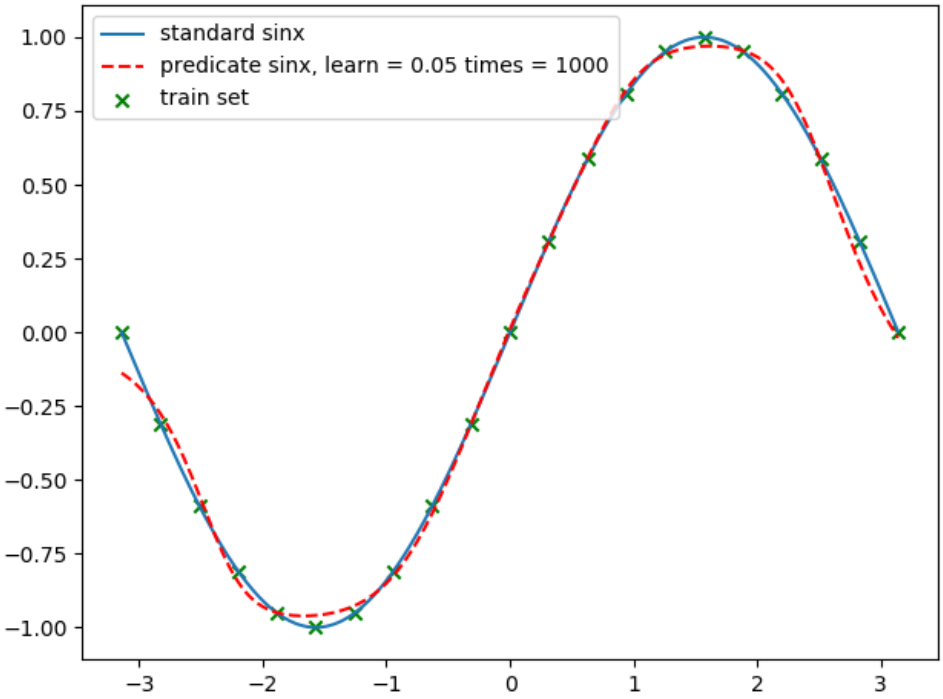
3. 不同学习率

3.1 学习率为 0.3 的时候，如下图：



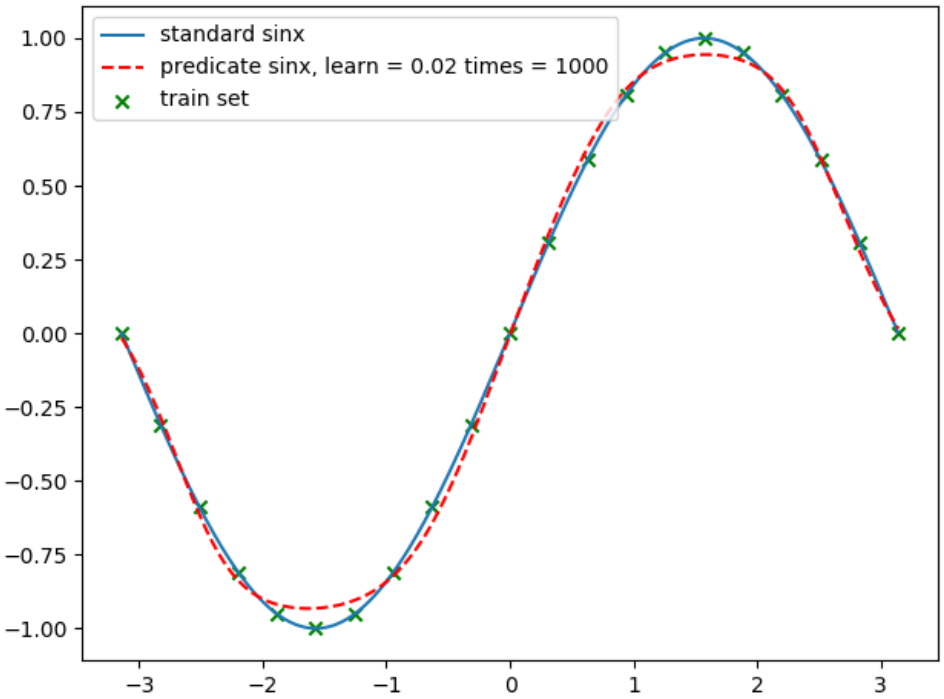
可以看到学习率增加，拟合效果反而下降了，所以决定不再增加学习率（继续增加之后，loss 继续增大，此处不再贴图片）平均误差为 0.008076051286641548

3.2 学习率为 0.05 的时候，如下图:



拟合效果变好，误差有效减小，平均误差为 0.0005469358051587906

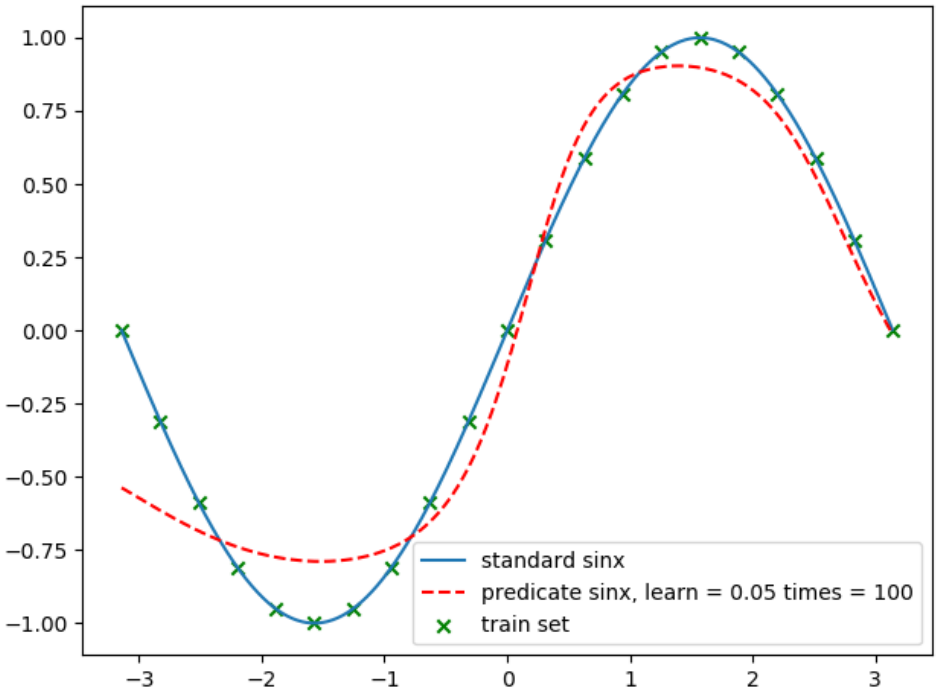
3.3 学习率为 0.02 的时候，如下图：



拟合效果没有明显变好，平均误差为 0.0005999300463614698

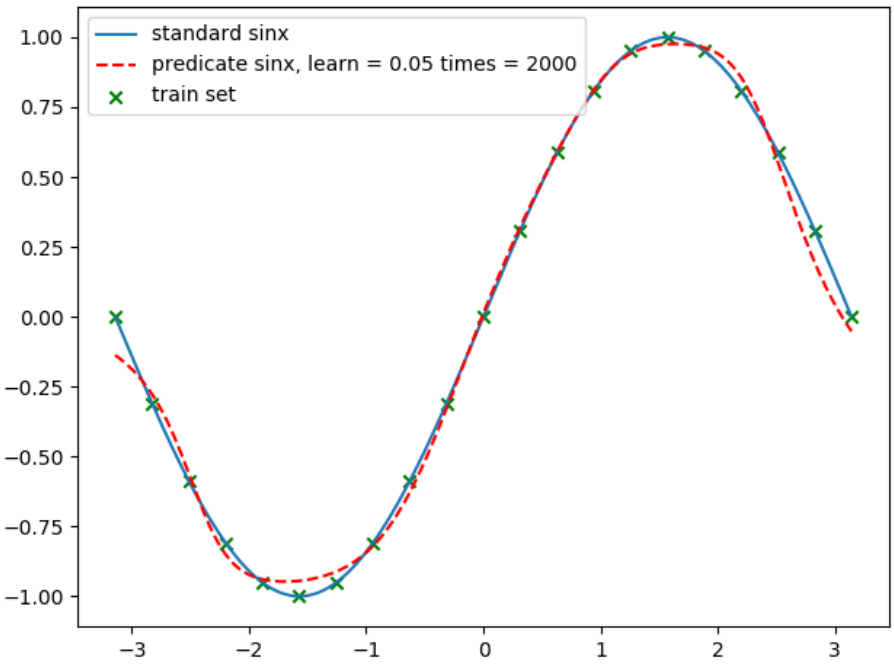
4. 不同学习次数（此时将学习率调整为 0.05）

4.1 学习次数为 100 的时候。如下图：



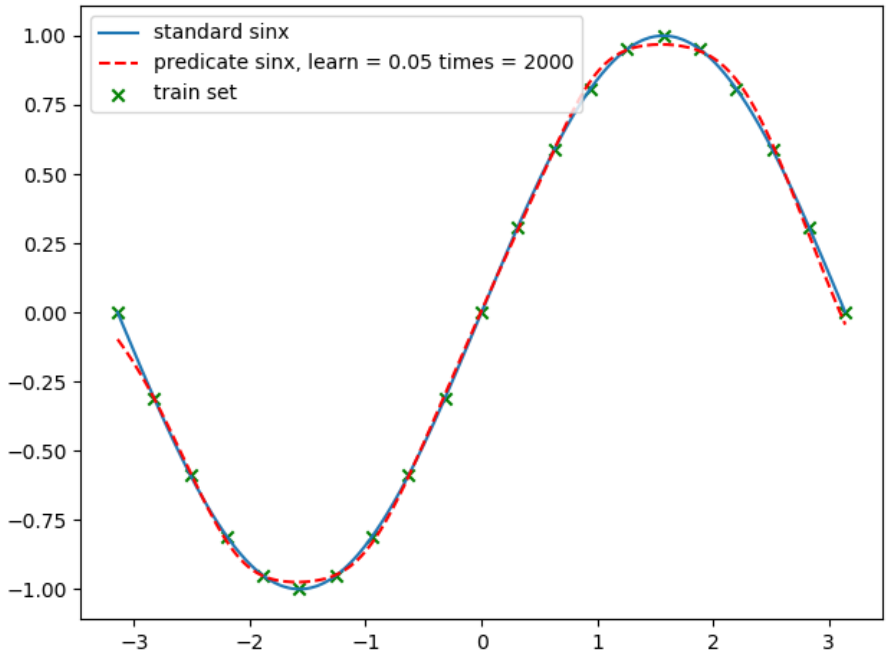
学习次数减少，拟合效果明显变差，平均误差为 0.011209506145333975

4.2 学习次数为 2000 的时候，如下图：



此时的训练数据与 3.2 只是训练次数不同，训练次数增加，误差反而增大了，可以看出**不一定训练次数越多，效果就越好**，平均误差为 0.0009090724792659862

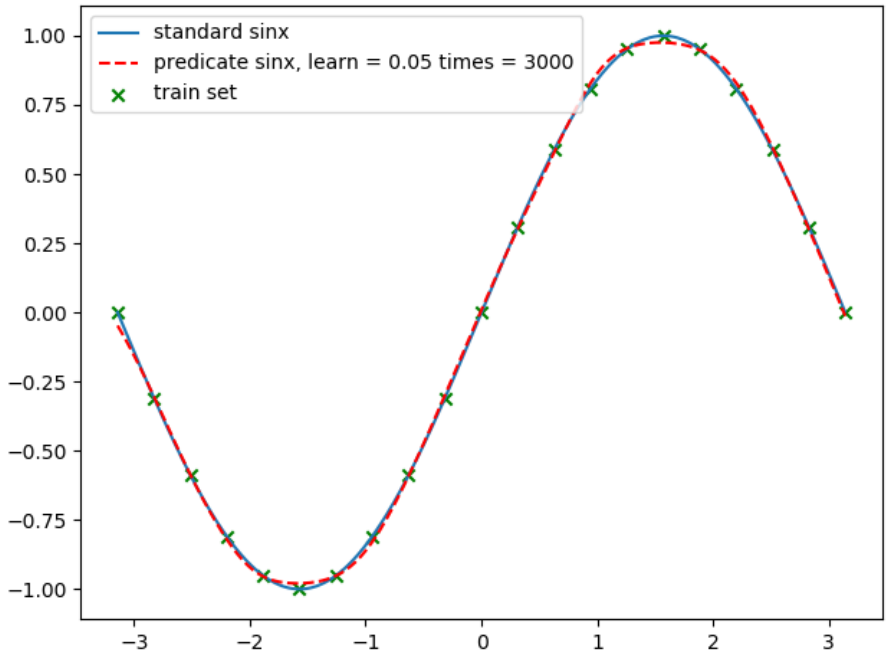
4.4 隐藏层设置为[10,10]，学习次数为 2000 的时候，如下图：



相比[10,10] 训练 1000 次的时候，此时拟合效果更好，

平均误差为 0.0002528205347302115

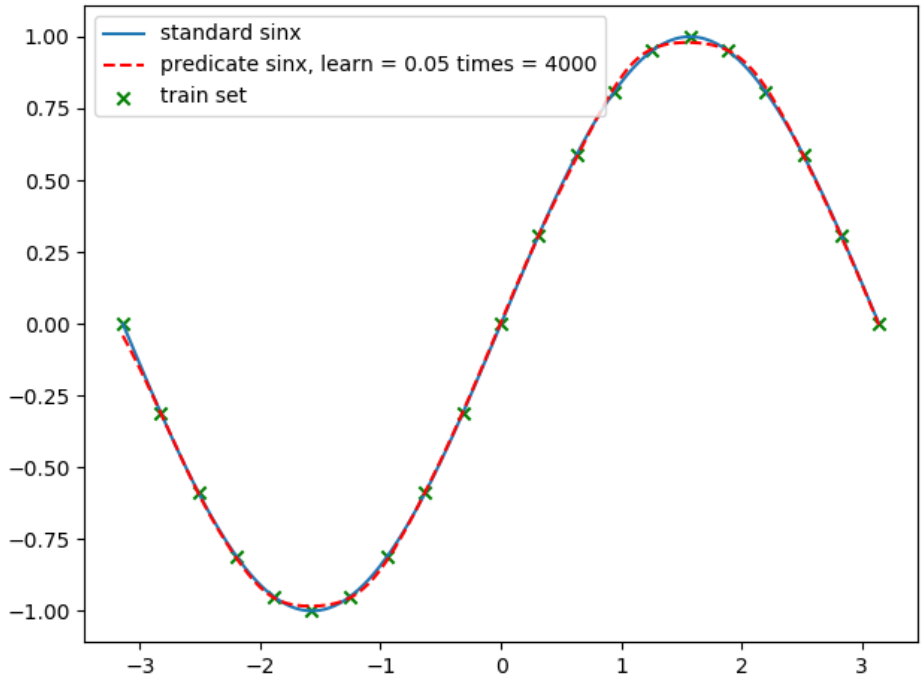
4.5 隐藏层设置为[10,10]，学习次数为 3000 的时候，如下图：



此时拟合效果出现了非常有效的提高，平均误差减小到

```
8.953129929950359e-05
```

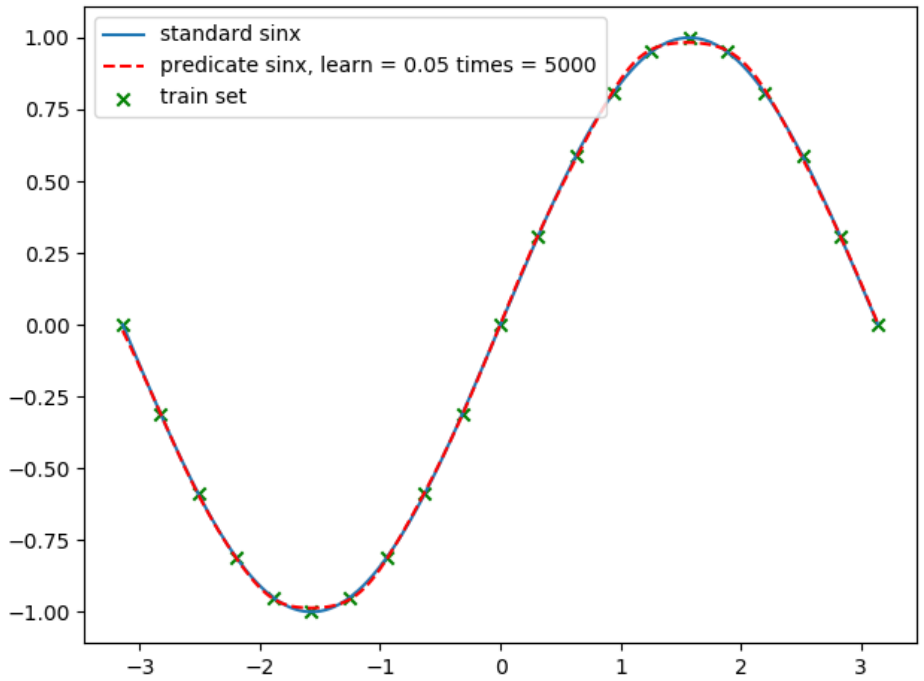
4.6 隐藏层设置为[10,10]，学习次数为 4000 的时候，如下图：



平均误差继续减小

```
5.0322279081281126e-05
```

4.7 隐藏层设置为[10,10]，学习次数为 5000 的时候，如下图：



平均误差继续减小 2.869827595917338e-05

一下训练数据不在贴图，只表格说明参数和训练结果：

编号	隐藏层设置	学习率	训练次数	平均误差
01	[5,5]	0.05	5000	0.002860167284416198
02	[9,5]	0.05	3000	0.0012922150360915113
03	[5,9]	0.05	5000	0.0006257829255978239
04	[9,5]	0.05	5000	6.520933857104006e-05
05	[10,10]	0.03	5000	0.00011302094589733894
06	[10,10]	0.03	3000	0.00021456745185871896
07	[10,10]	0.08	3000	0.00016916353482451443
08	[10,10]	0.08	5000	2.9177292093819762e-05

总结:

在选取了合适的隐藏层设置的时候, 训练次数增加, 可以看到误差在不断减小, 然而如果选取了不是很合适的其他参数的时候，训练次数的增加并不能有效的减少误差，隐藏层设置为[10,10]的时候得到了最好的训练效果，学习率 0.05，0.08 均是比较好的，训练次数在 3000 及以上误差不断减小。

第三部分：文字分类

3.1 代码结构

函数或属性	作用
工具类 Utils	
get_file_list(path)	读取 path 路径下的所有 bmp 图片
get_image_matrix()	将图片信息存储为 bit 文件

split(x, y, scale = 0.25)	分割训练集和测试集用于自测
get_data(scale = 0.25)	返回训练集，训练集 Label，测试集，测试集 Label
draw(train_corrects, test_corrects)	画图，训练次数-正确率
write_data()	将 get_data 的数据写入文件
类：BPNetwork	**BP 神经网络的基础结构和方法，用于文字分类**
属性：	
self.input_n = 0 self.input_cells = [] self.output_m = 0 self.output_cells = [] self.hidden_set = 0 self.hidden_result = [] self.input_w = [] self.output_w = [] self.input_b = [] self.output_b = [] self.output_delta = [] self.input_delta = []	输入神经元数 输入神经元数组 输出神经元数 输出神经元数组 隐藏层神经元数 隐藏层的输出 输出层到隐藏层的权值数组 输入层到隐藏层的权值数组 输入层到隐藏层的阈值数组 隐藏层到输出层的阈值数组 输出层的δ数组 隐藏层的δ数组
函数：	
set_up(input_n,output,hidden_size)	初始化神经网络的基本结构，输入层，输出层，隐藏层。
softmax(x)	Softmax 函数，对输出层的结果进行 softmax 运算
activation_function(x)	激活函数，对隐藏层的结果进行激活函数运算
forward_propagate(train)	一次向前传播，第 k 层第 j 个节点的输出值 Z[k][j] = tanh((Σj (Σi=前一层输入神经元个数 weight[k-1][i][j] * input[k-1][i])) + bias[k-1][j])
back_propagate(target, learn)	反向传播，包括计算δ，并且更新权值，阈值
calculate_correct(train_data, train_label)	计算一次前向传播的平均误差

train(times, train_data, train_label, learn)	对训练集进行训练

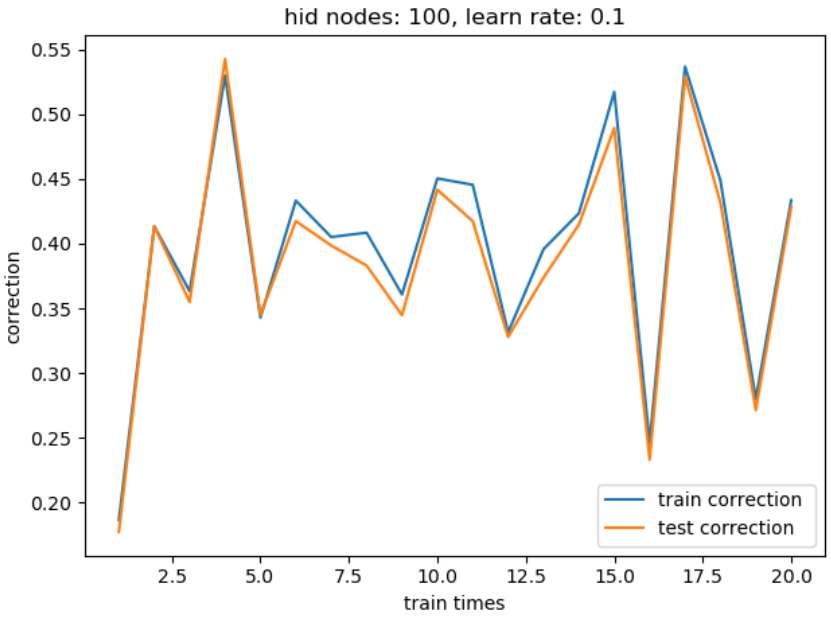
3.2 不同实验参数的分类效果

自定义参数：

- 隐藏层节点数
- 学习率
- 训练次数

1. 不同学习率(隐藏层节点数：100)

1.1 learn = 0.1



峰值的训练效果：

```
第 4 次反向传播训练
训练集正确率为: 0.5297379032258065
测试集正确率为: 0.5423387096774194
```

观察趋势图可以看到，learn 过大导致训练效果，没有达到梯度下降的作用，而是左右摇摆

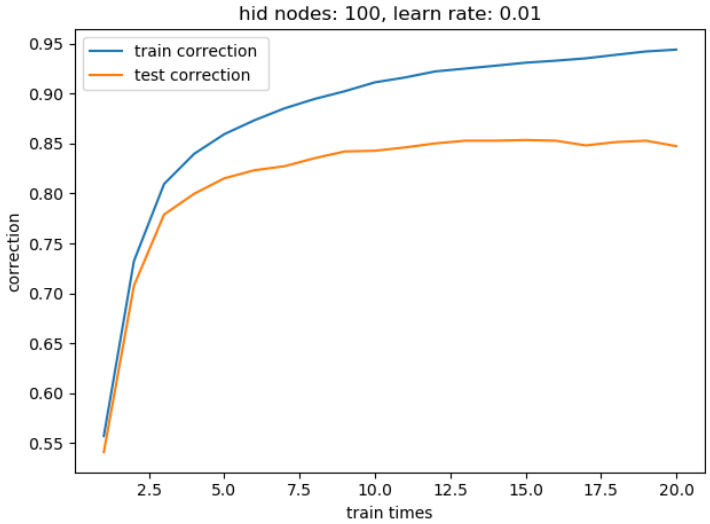
1.2 learn = 0.05

峰值的训练效果：

```
第 14 次反向传播训练
训练集正确率为: 0.7854502688172043
测试集正确率为: 0.7600806451612904
```

最高的正确率有所提高，但是依旧不稳定

1.3 learn = 0.01



峰值的训练效果：

```
第 15 次反向传播训练
训练集正确率为: 0.9309475806451613
测试集正确率为: 0.853494623655914
```

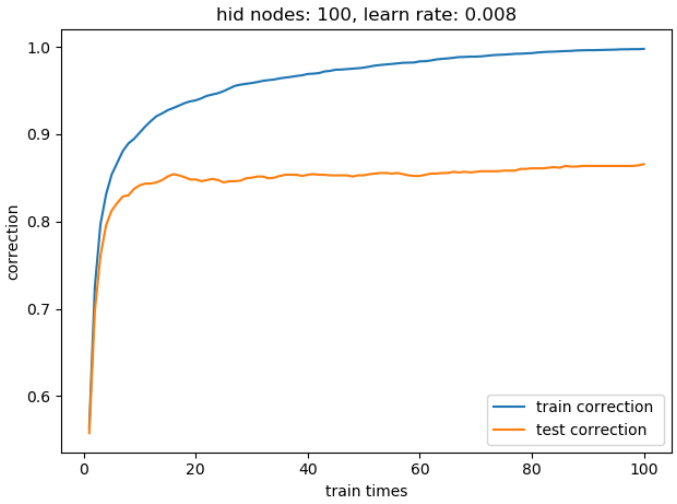
在 130 次达到瓶颈：

```
第 130 次反向传播训练
训练集正确率为: 0.9994959677419355
测试集正确率为: 0.8689516129032258
```

减小 learn 之后，训练效果的趋势逐渐稳定，最高正确率也提高了。

1.4 learn = 0.008

并且增加了训练次数

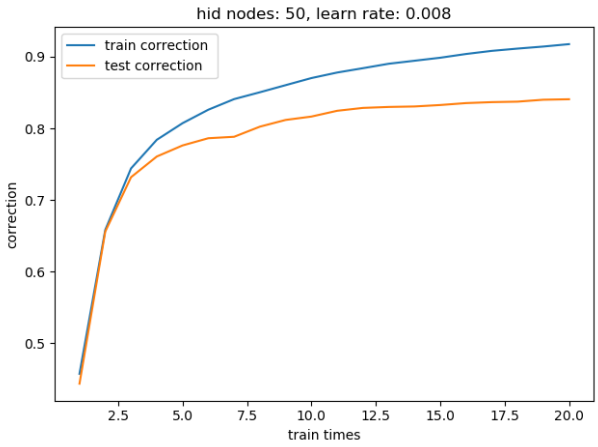


峰值训练效果:

```
第 100 次反向传播训练
训练集正确率为: 0.9976478494623656
测试集正确率为: 0.8655913978494624
```

2. 不同隐藏层节点数

2.1 hid = 50

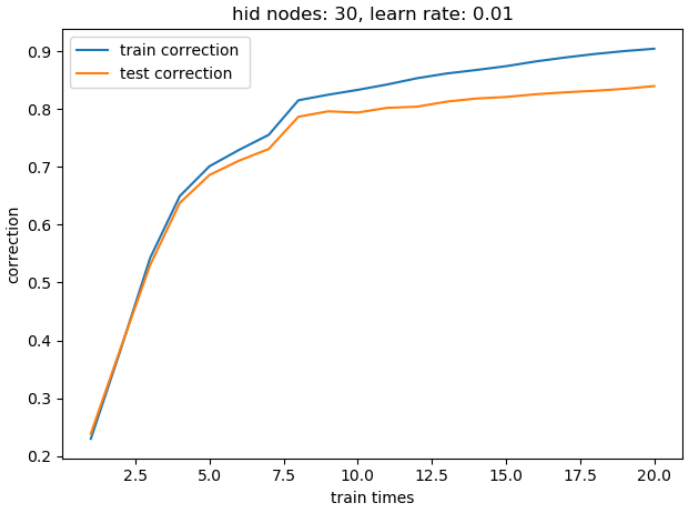


峰值训练效果:

```
第 20 次反向传播训练
训练集正确率为: 0.9176747311827957
测试集正确率为: 0.8407258064516129
```

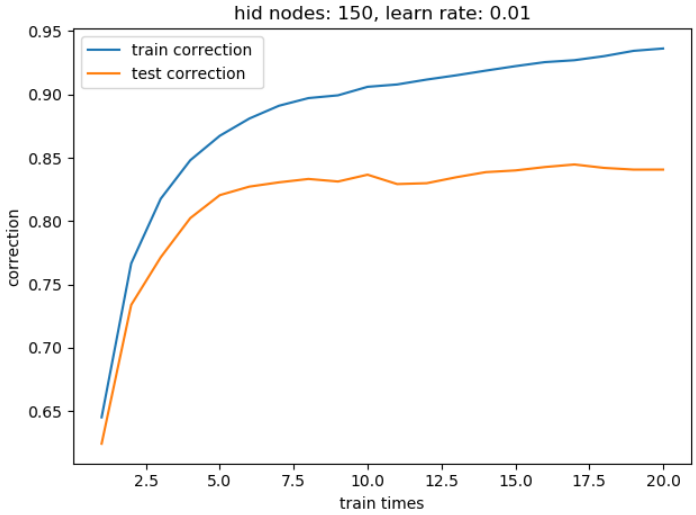
与 hid = 100 相比，上升的速度更慢一些，不如 100 效果好

2.2 hid = 30



上升速度更慢一些，可能需要更多的训练次数。

2.2 hid = 150

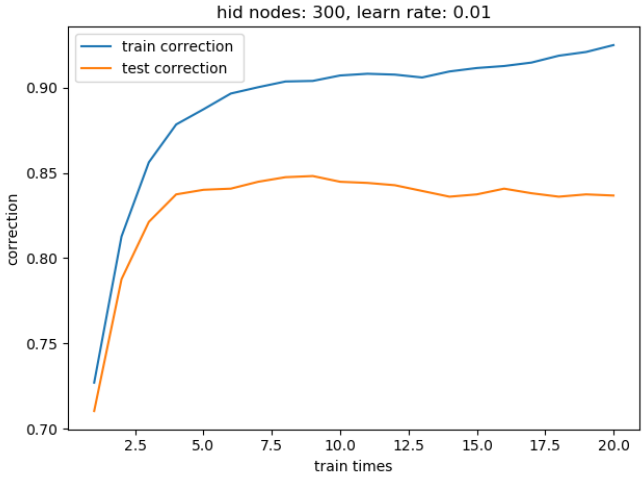


训练集正确率不断升高到 0.9 以上

测试集的最好结果：

```
第 17 次反向传播训练
训练集正确率为: 0.9270833333333334
测试集正确率为: 0.844758064516129
```

2.5 hid = 300



训练集的最好结果:

```
第 9 次反向传播训练
训练集正确率为: 0.9038978494623656
测试集正确率为: 0.8481182795698925
```

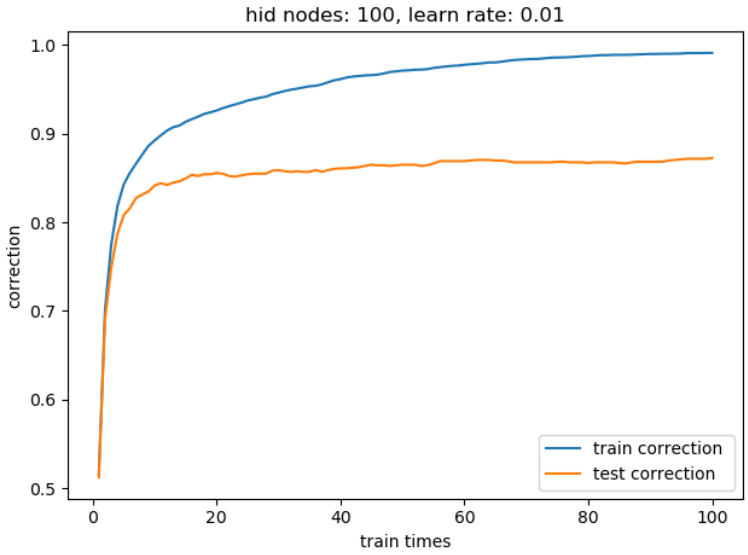
总结:

通过调节不同参数的实验，发现，当隐藏层节点取 100 的时候，学习率选择 0.01，训练 20 次差不多能达到较好的正确率，大概在 0.85 左右。
并且，隐藏层节点越多，经过越少次数的训练可以达到当前最好的训练效果，但是节点相对少一点，则更有可能通过更多次的训练达到更高的训练正确率。
选取隐藏节点数 100， 学习率 0.01， 训练 130 次作为测试的神经网络的参数。
训练时间大概 15 分钟。

第四部分、改进分类神经网络

改进: 在第三部分的基础上，不使用 softmax 函数，对最后一层输出也使用 sigmoid 函数。

hid = 100， learn = 0.01



训练 20 次的效果：

```
第 20 次反向传播训练
训练集正确率为: 0.9260752688172043
测试集正确率为: 0.855510752688172
```

增大训练次数，训练结果变更好：

```
第 100 次反向传播训练
训练集正确率为: 0.9910954301075269
测试集正确率为: 0.8723118279569892
```

```
第 165 次反向传播训练
训练集正确率为: 0.9971438172043011
测试集正确率为: 0.8770161290322581
```

此后再继续增大训练次数，正确率不在提高，并且略微下降：

```
第 200 次反向传播训练
训练集正确率为: 0.9983198924731183
测试集正确率为: 0.8723118279569892
```

总结:

选取隐藏节点数 100， 学习率 0.01，训练 160 次为测试的神经网络的参数。训练时间大概 20 分钟。通过与第三部分的比较可以看到，使用 softmax 函数，训练集的正确率提高的更快，不使用 softmax，只使用 sigmoid 函数，训练集的正确率提高的速度稍差一点，但是训练集的正确率却可以达到更好的效果。