Lab1 的 part1 拟合 sinx 部分我使用 python3.6 语言，利用 numpy 库进行部分数学操作，具体见如下说明：

一、函数方法说明：

A. 全局方法：

1. rand(a,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. generate_w(m,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. generate_b(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. fit_function(x,deriv=False)：该方法返回拟合函数，如果该方法传入的 deriv 是 True,则返回拟合函数的导数。我试验了两种拟合函数，一种是 sigmoid。另一种是 tanh 函数，在实验数据来看，tanh 拟合效果比较好，而 sigmoid 由于只能输出 0-1 的正数，只能拟合 0-π 的一部分，且拟合效果不是很好

B. 定义了 BPNetwork 类，其内部函数方法说明如下：

1. __init__(self)：定义类中所需要的数据，类中所需要维持的参数名称在此函数中事先定义。这些数据包括：
   ⑴ self.input_n = 0：表示输入层的神经元个数，拟合 sinx 需要 self.input_n = 2，一个是输入的数据，另一个用来调节 bias
   ⑵ self.input_cells = []：表示输入层的神经元输入的数据，有两个元素，一个是输入的数据；另一个用来调节 bias，默认为 1
   ⑶ self.output_n = 0：输出层的神经元个数
   ⑷ self.output_cells = []：输出层的神经元的输出值
   ⑸ self.input_w = []：输入层到第一层隐藏层的 weight
   ⑹ self.output_w = []：最后一层隐藏层到输出层的 weight
   ⑺ self.hidden_ns = []：隐藏层的设置，它的长度是指隐藏层个数，它的每个元素指该层隐藏层的神经元个数
   ⑻ self.hidden_ws = []：隐藏层的 weights，其中的每个元素为上一层隐藏层到下一层隐藏层的 weight，一共有 n-1 个 weight
   ⑼ self.hidden_bs = []：其中的每个元素 self.hidden_bs[i]都是第 i 层隐藏层的 bias 设置，其长度为隐藏层的个数
   ⑽ self.output_b = []：输出层的 bias
   ⑾ self.hidden_results = []：其中的每个元素是每一层隐藏层的输出值
   ⑿ self.output_deltas = []：指输出层的∂Error/∂(output_w)
   ⒀ self.hidden_deltases = []：其中的每个元素 self.hidden_deltases[i] 指从第 i 层出发的∂Error/∂(weight),该 weight 指从第 i 层到第 i+1 层的 weight
2. setup(self, input_n, output_n, hidden_set)：初始化类中所需参数，输入的参数中，
   input_n 是定义输入的参数个数，但并不等于输入层神经元个数，self.input_n 等于输入的参数 input_n+1,原因是需要新增一个神经元用来调节 bias，且这个神经元的输入值记为 1，这样就可以通过用 output_deltas 来调节 input_b 了
   output_n 是输出层个数
   hidden_set 是一个 list，里面的每个元素是隐藏层的神经元个数，如[10,10]表示有两层隐藏层，两层隐藏层都有 11 个神经元，+1 的原因同 input_cells 中增加一个神经元的作用是一样的，是为了调节该隐藏层的 bias

通过这些信息就可以初始化类中所需要的数据了。即在__init__方法中的那些数据.
需要注意的是在初始化 weight 的时候使用 generate_w 方法形成一个由随机数的矩阵，初始化 bias 的时候使用 generate_b 形成一个由随机数组成的一维数组

3. forward_propagate(self, input)：向前传播。
(1) 将输入的 input 作为前 n-1 个输入层神经元的输入值，第 n 个输入层神经元设为 1，然后进行向前传播
(2) 传播分为三个部分，输入层到隐藏层第一层，在隐藏层之中，然后从最后一层隐藏层到输出层。 传播公式为：
Output[h]=fit_function((Σ$_{i=0}^{输入层神经元个数}$weight[i][h]*input[i])+bias[h])
传播的思路就是：前一层的输入的值乘以前一层到后一层的 weight，然后得到积的加和，再用这个加和加上下一层的对应的 bias，得到后一层的输出。
4. get_deltas(self, label):该函数是得到 self.output_deltas 和 self.hidden_deltases 的函数。

self.output_deltas = ∂Error/∂(output_w):指输出层的 deltas 公式为

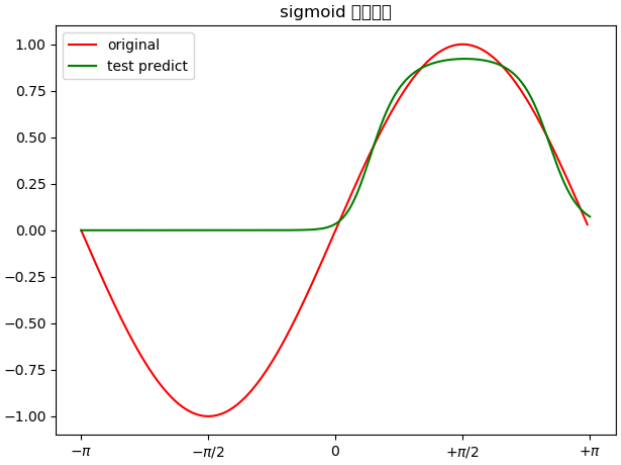
output_deltas[o]=fit_function(output_cell[o])*(label[o]-output_cells[o])

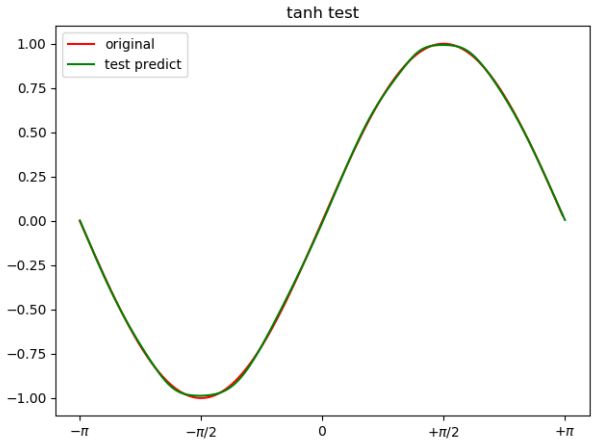
self.hidden_deltases = ∂Error/∂(weight) weight 是指 hidden_ws 中的元素。

hidden_deltases[k][o]=fit_function_deriv(hidden_result[k][o])*(deltas[i]*weight[o][i]
这里的 deltas 是指的上一层神经元的 deltas
5. renew_w(self,learn)：更新 weight 的函数，仍然分成三个部分，先更新最后一层隐藏层到输出层的权重，再是隐藏层之间的权重，然后再更新第一层隐藏层到输入层的权重。更新 weight 的公式为：
Weight[i][o] += deltas[o]*input[i]*learn，其中的 o 是输出层神经元的 index，i 是输入层的神经元的 index，input 是指输入层的输入，learn 是学习率。
6. renew_b(self,learn)：更新 bias 的函数，分成两个部分，第一部分是隐藏层的 bias，第二部分是输出层的 bias。更新 bias 的公式为：
bias[i]+= deltas[i]*learn
7. back_propagate(self, input, label, learn):先进行前向传播，再进行反向传播，然后获取 deltas，更新 weight 和 bias，之后再计算此次正向传播得到的损失值，并返回。这个函数的每一行都在调用其他的函数，是一次完整的反向传播。
8. get_loss(self,label,output_cell)：用损失函数获取损失值，公式为
Σ$_{o}^{输出值的个数}$$_{=0}$((label[o]-output_cell[o])**2) *0.5
9. test(self):设置 bp 网的结构，然后得到 20 组数据进行训练，然后再用 200 组数据进行测试。直接调用 test()方法就可以测试。

二、遇到的困难及解决

1. 遇到的一个比较烦的问题就是 bias 的处理问题，在网上搜了一些 bp 网 bias 处理的文章，发现是通过增加一个输入层神经元，并且在每一层隐藏层上增加一个神经元来解决的。也就是说，如果输入值的个数是 1，那么就要两个输入层神经元，其中①号的神经元的值为这个输入值，②号神经元的值为 1，②号神经元与下一层的每个神经元之间的 weight 就相当于下一层每个神经元的 bias，bias = weight$_{2,i}$*1,这样就可以通过 bias[i]+= deltas[i]*learn 来调整 bias 了，同样的，对每个隐藏层都新增一个神经元也是这样的目的。
2. 遇到的第二个困难就是 sigmoid 函数和 tanh 函数选取问题，一开始我使用 sigmoid 函数，发现拟合出来的是一条直线，通过调参，可以大致拟合出[0,Π]区间的形状，但效果并不是很理想。如图。后来通过使用 tanh 函数，可以很好的拟合出 sinx 的形状。





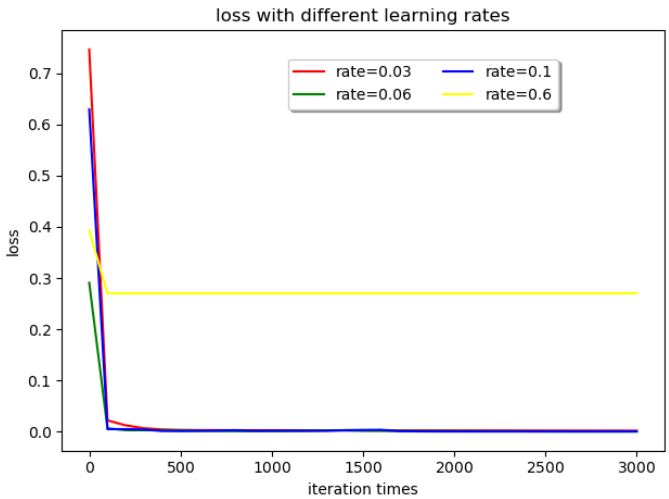
上两张图中左图是用 sigmoid 函数，右图是用 tanh 函数，参数都是 weight 取(-0.69,1)之间的随机数，bias 取(-2.409,0.02)之间的随机数，学习率为 0.05，训练样本为（-Π，Π）之间的 20 个，测试样本为（-Π，Π）之间 200 个，训练次数为 10000.bp 神经网络设置为[1,10,10,1]

可以看出来 sigmoid 拟合在为负值的部分，趋向于直线，这是因为 sigmoid 只能取（0，1）之间的值，所以拟合效果比较差。而 tanh 在这样的参数设置下，已经拟合的非常好了，它的平均误差只有 1.92524882921e-05，可以说非常完美了。

三、实验内容——各种因素对损失值（loss）的影响

一）前提：使用 tanh 非线性函数进行拟合。Weight 初始值为(-0.69,1)的随机值，bias 初始值为(-2.409,0.02)的随机值。在相同的训练集上训练 3000 次，每 100 次选取一个点，对测试集进行测试得到 loss 的平均值作为纵坐标值。

1. 不同的学习率（rate）下，loss 的变化情况。前提：1 个输入值，1 个输出值，6 个中间神经元。这里使用不同的学习率。

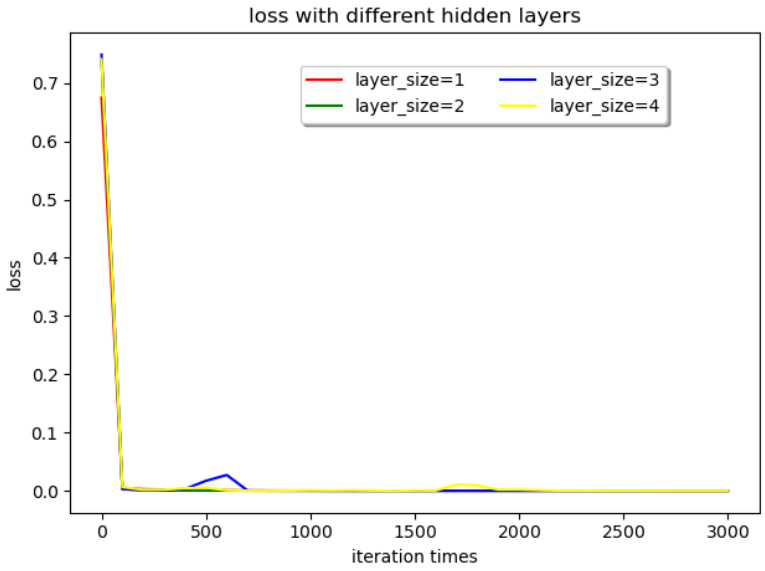


当learning rate = 0.01 时，平均loss为：0.00177692217424
当learning rate = 0.06 时，平均loss为：0.000119781347307
当learning rate = 0.1 时，平均loss为：0.00030590788673
当learning rate = 0.6 时，平均loss为：0.270187966936

，可以看出来，在学习率为 0.03、0.06，0.1 的时候最后的拟合都非常好，但是一开始的 loss 的下降速率却不一样，学习率为 0.1 的时候 loss 的下降速率较小，0.06 次之，最快的下降斜率是 rate=0.03 的时候，说明在 0.03 的时候拟合的速率最快。

训练 3000 次后打印出来的平均 loss 见上图，可以看出来，在学习率为 0.6 的时候平均 loss 最大，拟合效果最差，在学习率为 0.06 的时候平均 loss 最小，说明它的拟合效果最好。

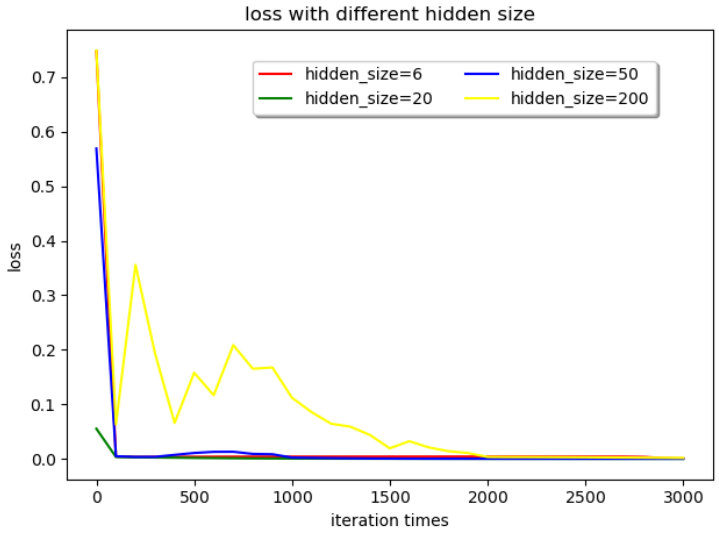
2. 不同隐藏层个数下，loss 的变化。这里使用学习率为 0.05。




```
当隐藏层设置为： [10] 时，平均loss为：0.000151280260482
当隐藏层设置为： [10, 10] 时，平均loss为：0.000246862602714
当隐藏层设置为： [10, 10, 10] 时，平均loss为：7.55183091262e-05
当隐藏层设置为： [10, 10, 10, 10] 时，平均loss为：0.000144285188015
```

在第一张图中可以看出，隐藏层设置 1-4 层走势差不多，只有在设置三层和第四层的时候在不同地方略有波动，可能是因为出现了局部最小值的原因，在循环次数增大后，局部最小值就被跳出来了。在训练 3000 次结束之后，输出的平均 loss 值可以看出，在隐藏层设置三层时，拟合的效果最好，其次是四层的时候。而在隐藏层设置两层时，是拟合效果最差的，原因可能是出现了局部最小值。

3. 隐藏层只有一层的时候，设置不同的神经元个数。这里使用学习率为 0.05。

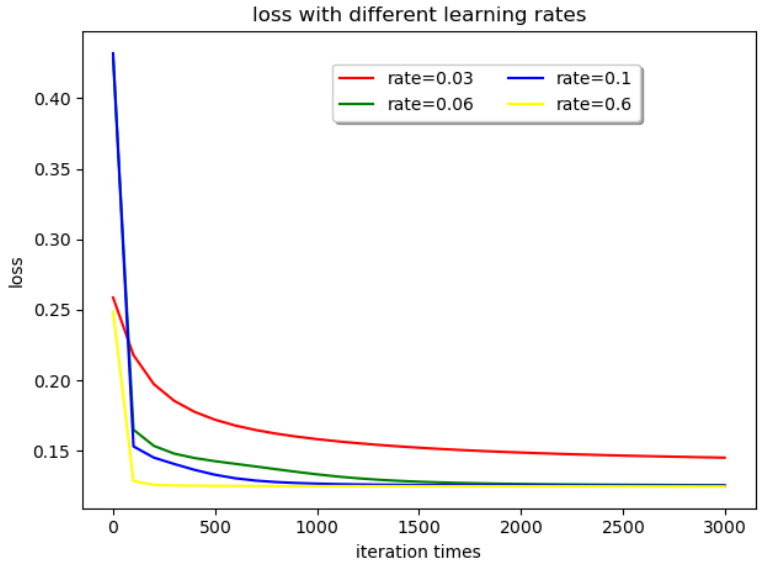


```
当只有一层隐藏层，该隐藏层大小为： [6] 时，平均loss为：0.000824453103088
当只有一层隐藏层，该隐藏层大小为： [20] 时，平均loss为：0.000224852986956
当只有一层隐藏层，该隐藏层大小为： [50] 时，平均loss为：0.000349377716662
当只有一层隐藏层，该隐藏层大小为： [200] 时，平均loss为：0.00183935877302
```

在第一张图中我们可以看到，隐藏层大小为 6，20，50 的时候 loss 值都下降的比较稳定，而隐藏层大小为 200 的时候在 0-2000 次时波动较大，原因可能是因为出现过拟合和出现局部最小值的现象，而随着训练次数的增大，这种状况有所好转。
在第二张图中，我们可以看到，隐藏层为 200 的时候，拟合效果最差，可能是有些过拟合，所以结果不是很好。隐藏层大小为 6 的时候次最差，可能是失拟合，因为神经元个数过少。而隐藏层为 20 的时候效果最好。

二）前提：使用 sigmoid 非线性函数进行拟合。Weight 初始值为(-0.69,1)的随机值，bias 初始值为(-2.409,0.02)的随机值。在相同的训练集上训练 3000 次，每 100 次选取一个点，对测试集进行测试得到 loss 的平均值作为纵坐标值。

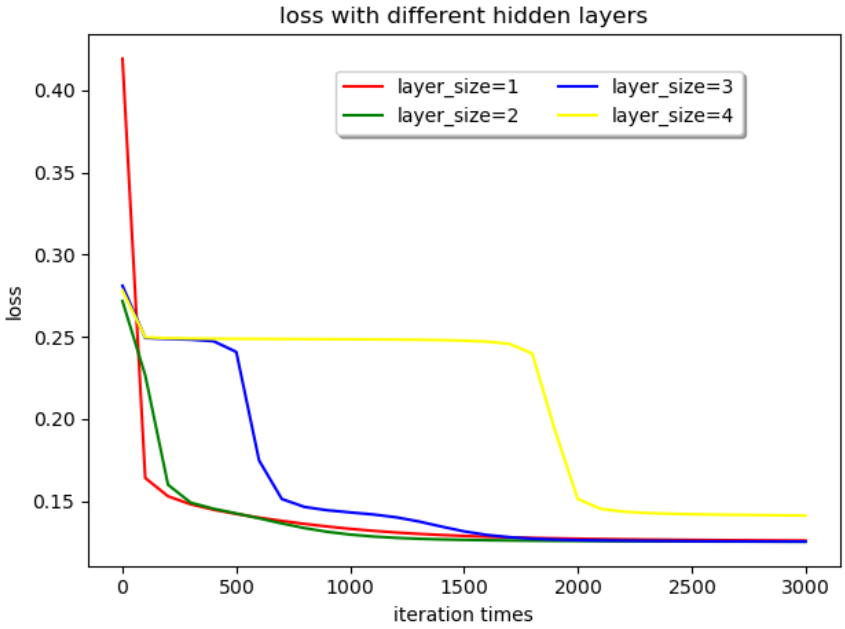
1. 不同的学习率（rate）下，loss 的变化情况。前提：1 个输入值，1 个输出值，6 个中间神经元。这里使用不同的学习率。




```
当learning rate = 0.01 时，平均loss为：0.145179886045
当learning rate = 0.06 时，平均loss为：0.125691943437
当learning rate = 0.1 时，平均loss为：0.125282922722
当learning rate = 0.6 时，平均loss为：0.124763196227
```

可以看出 rate=0.1 的时候，loss 的下降速率最大，rate=0.03 时下降速率最小，且最后的 loss 值最大，拟合效果最差。而在第二张图中可以看出，在 rate=0.6 的时候，训练 3000 次之后的效果最好。0.06 和 0.1 的效果差不多。

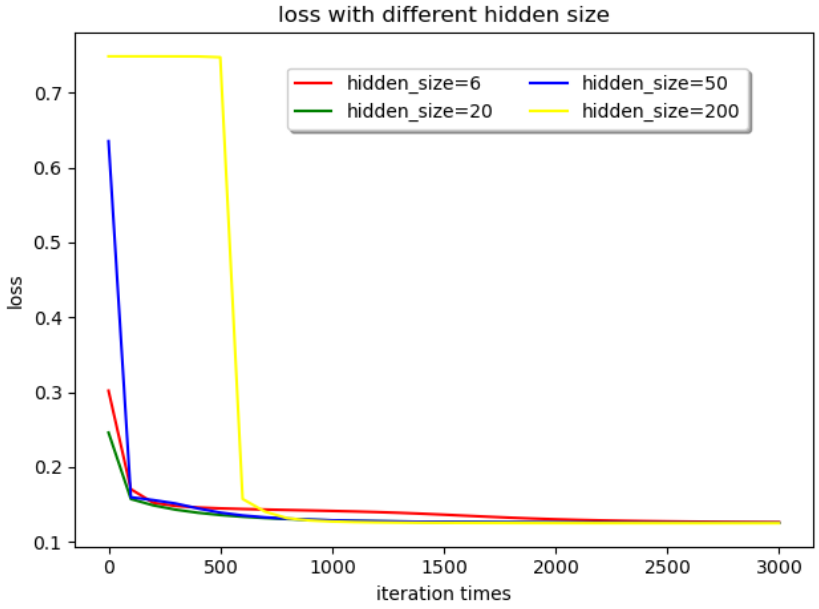
2. 不同隐藏层个数下，loss 的变化。这里使用学习率为 0.05。



当隐藏层设置为： [10] 时，平均loss为：0.126079109639
当隐藏层设置为： [10, 10] 时，平均loss为：0.12538880761
当隐藏层设置为： [10, 10, 10] 时，平均loss为：0.125472471441
当隐藏层设置为： [10, 10, 10, 10] 时，平均loss为：0.141319709322

可以看出在隐藏层为 3 时，在训练次数大概为 200-700 处有一段时间的平缓，loss=0.25 左右，这个时候时候拟合出来的是一条平缓的直线，说明出现了局部最小值的情况。同样的 ，在隐藏层为 4 的时候在训练次数大概为 200-1800 处出现了一样的情况，说明也是出现了局部最小值的情况。

3. 隐藏层只有一层的时候，设置不同的神经元个数。这里使用学习率为 0.05。



当只有一层隐藏层，该隐藏层大小为： [6] 时，平均loss为：0.126358217499
当只有一层隐藏层，该隐藏层大小为： [20] 时，平均loss为：0.125465216005
当只有一层隐藏层，该隐藏层大小为： [50] 时，平均loss为：0.125458451668
当只有一层隐藏层，该隐藏层大小为： [200] 时，平均loss为：0.125059438076

在图中可以看出在隐藏层大小为 200 时，在训练次数为 0-500 时，出现了局部最小值的情况，导致 loss 值没有什么变化。600 次训练之后迅速拟合，loss 也迅速减小。隐藏层大小为 50，6，20 的

曲线下降速率依次降低，隐藏层大小为 50 的拟合效果最好。

四、总结

与 tanh 非线性函数拟合的效果对比可以看出 sigmoid 明显效果差，tanh 在学习率为 0.06 左右，效果比较好，隐藏层个数和层数都需要多实验几次才可以得到较好的拟合。Sigmoid 在学习率为 0.6 左右的时候效果比较好。