GAMES102 第二次作业报告

00002 周玉祺

1、问题描述及目的

任务：

1、使用RBF神经网络函数来拟合数据（仍限制在函数情形）

2、与作业1的方法比较

目的：

1、理解神经网络优化

2、学习使用TensorFlow优化

2、RBF神经网络拟合

背景：

在第一次作业中，曾使用Gauss基函数来进行插值。但是，在操作过程中，应对不同的输入点集，需要不断的更新 σ 的值来保证拟合曲线的光滑性，因此，我们希望在计算权重 ω 的值时，程序也能自动地对 σ 进行优化。

原理：

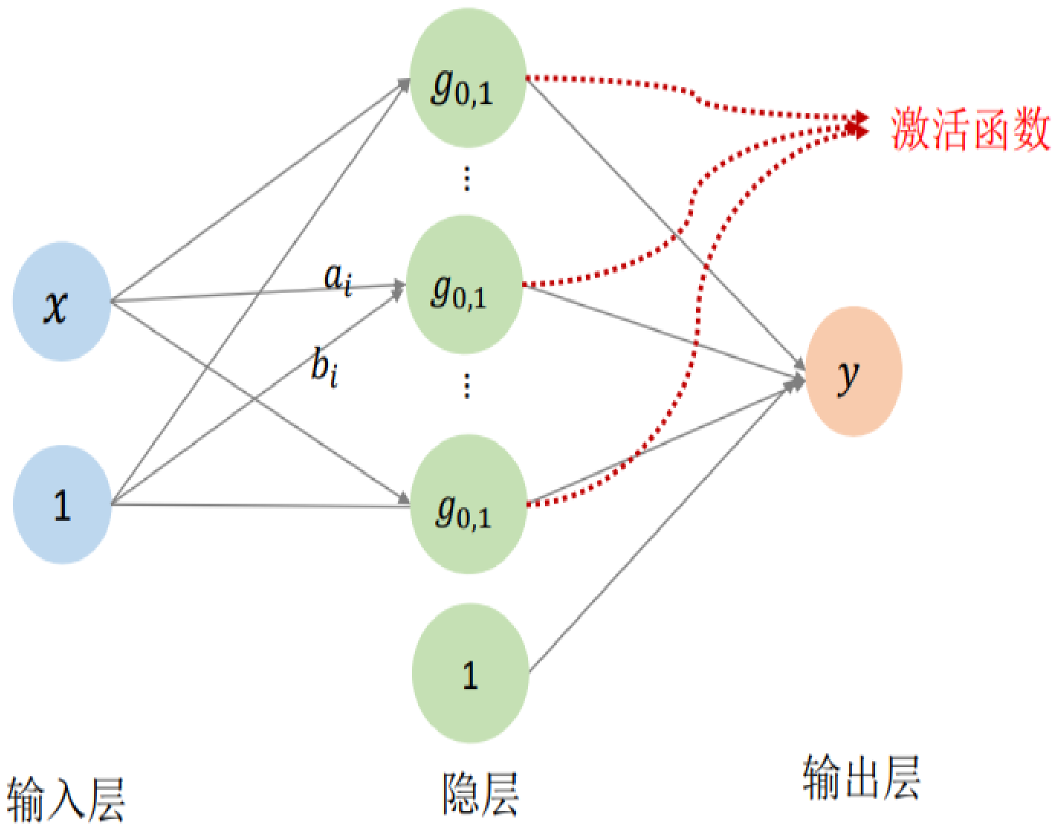
一般的Gauss函数可以表示为：

$$g_{\mu,\sigma} = \frac{1}{\sqrt(2\pi\sigma^2)} e^{-\frac{(x-\mu)^2}{2\sigma^2}} = g_{0,1}(ax+b)$$

由于权重 ω 的存在，可以忽略指数项前面的系数，并假设 $g_{0,1}$的表达式中只含指数项。因此，最终的函数 $f(x)$ 的表达式可以写成：

$$f(x) = \omega_0 + \sum_{i=1}^{n} \omega_i g_{0,1}(a_i x + b_i)$$

若把基函数看成是一个基本函数通过平移和伸缩变换而来的，则可以把整个拟合过程看成神经网络：



假如隐藏层的节点数最够多，便可以使输出的值足够逼近精确解。但是，实际情况中，由于种种原因，无法选择大量的节点，并且，节点数非常多时，对小规模的点进行拟合时，容易造成过拟合的情况。因此，怎样选择隐藏层节点数是个问题。

算法及程序结构：

在操作过程中，我选择了TensorFlow作为主要框架来完成，参考了代码https://github.com/shiluqiang/RBF_NN_tensorflow/blob/master/RBF_tensorflow.py的结构，使用单隐层的RBF神经网络。

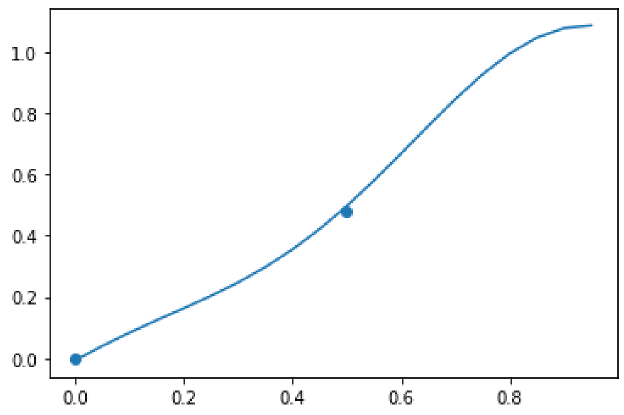
整个程序以正弦函数作为代码编写中的测试函数，用以检验算法的正确性及优劣性。假设平面上点的个数为Size， 则输入层输入 $(x_1, x_2, ..., x_{Size})$ ，输出层输入 $(y_1, y_2, ..., y_{Size})$ ，隐藏层节点数设置为 8*Size 。以训练出的输出层的输出与输入的平方平均作为误差函数，在优化算法上选择了Adam算法（一些梯度下降算法很容易收敛到局部最优解，但并不是全局最优，并且，根据输入点数的不同，学习率不好设置，相比之下，Adam算法好很多）。最后，再选择随机生成平面上Size个点，其中所有点（包括之前正弦函数的点）的横坐标均在[0,1)之中，且均匀分布，以此来检验算法的有效性。

3、输出结果

1）正弦函数的拟合图像举例（更多的图像在文件夹中）：

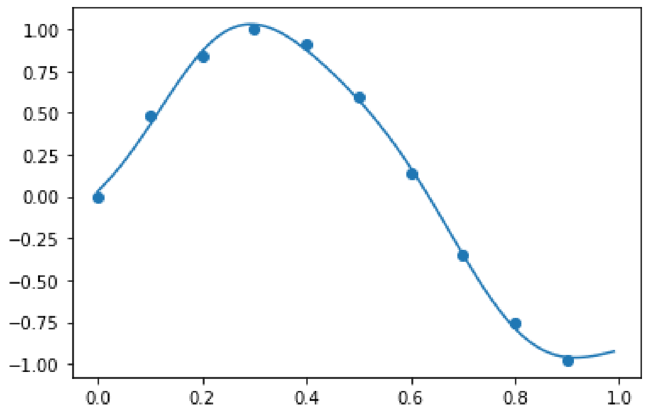
Size=2:

```
------------------------1.Set Data---------------------
------------------------2.Parameter setting-------------
------------------------3.Start Training------------------------
Loss function at step 0 is 24.544014
Loss function at step 30 is 0.07578677
Loss function at step 60 is 0.004592937
Loss function at step 90 is 0.00018232966
Training complete!
------------------------4.Draw FittingCurve------------------------
```



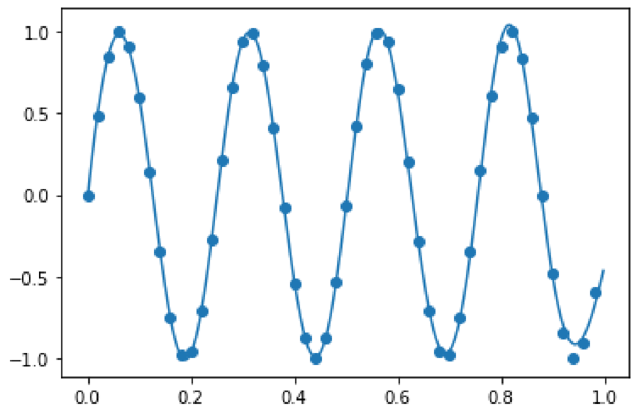
Size=10:

```
------------------------1.Set Data---------------------
------------------------2.Parameter setting-------------
------------------------3.Start Training------------------------
Loss function at step 0 is 1166.2429
Loss function at step 150 is 0.00135391
Loss function at step 300 is 0.0009209259
Training complete!
------------------------4.Draw FittingCurve------------------------
```



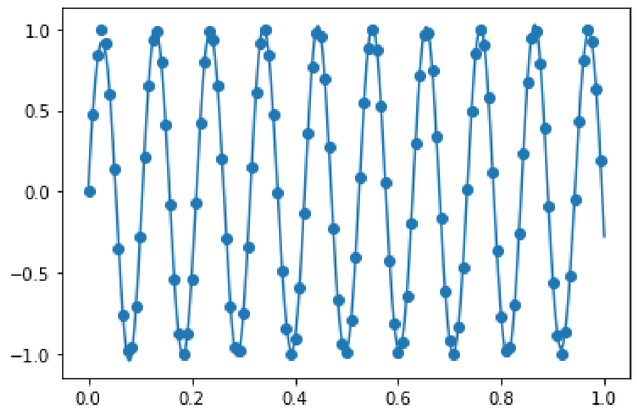
Size=50:

```
------------------------1.Set Data--------------------
------------------------2.Parameter setting-------------
------------------------3.Start Training-----------------------
Loss function at step 0 is 29569.465
Loss function at step 750 is 0.21504995
Loss function at step 1500 is 0.06717448
Loss function at step 2250 is 0.011345858
Loss function at step 3000 is 0.0027883262
Loss function at step 3750 is 0.0017362323
Loss function at step 4500 is 0.0012426659
Loss function at step 5250 is 0.0010130263
Loss function at step 6000 is 0.0007783805
Training complete!
------------------------4.Draw FittingCurve---------------------
```



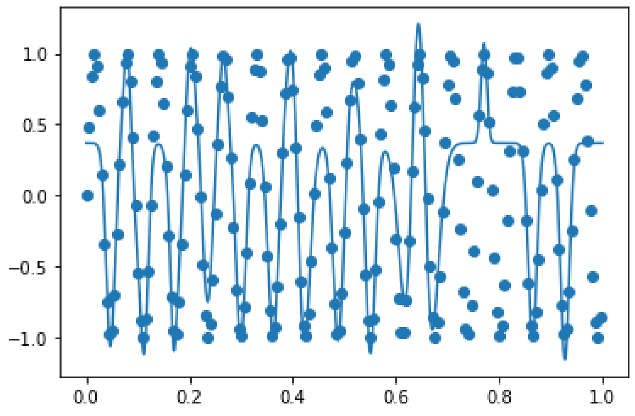
Size=120:

```
------------------------1.Set Data--------------------
------------------------2.Parameter setting-------------
------------------------3.Start Training-----------------------
Loss function at step 0 is 166401.64
Loss function at step 1800 is 0.4765128
Loss function at step 3600 is 0.47490293
Loss function at step 5400 is 0.47303227
Loss function at step 7200 is 0.46513623
Loss function at step 9000 is 0.14455134
Loss function at step 10800 is 0.0010615385
Loss function at step 12600 is 0.00054769847
Training complete!
------------------------4.Draw FittingCurve-----------------------
```



Size=200:

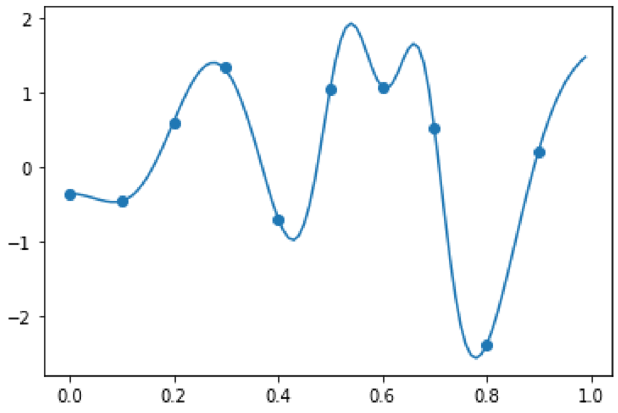
```
------------------------1.Set Data--------------------
------------------------2.Parameter setting-------------
------------------------3.Start Training-----------------------
Loss function at step 0 is 457999.56
Loss function at step 3000 is 0.49338195
Loss function at step 6000 is 0.49142855
Loss function at step 9000 is 0.48788893
Loss function at step 12000 is 0.46455598
Loss function at step 15000 is 0.13997304
Loss function at step 18000 is 0.080926225
Loss function at step 21000 is 0.06333509
Loss function at step 24000 is 0.1608425
Loss function at step 27000 is 0.14371435
Loss function at step 30000 is 0.14004572
Loss function at step 33000 is 0.15869823
Loss function at step 36000 is 0.16970812
Loss function at step 39000 is 0.11691696
Loss function at step 42000 is 0.11993095
Loss function at step 45000 is 0.52001715
Loss function at step 48000 is 0.14637613
Loss function at step 51000 is 0.18864292
Loss function at step 54000 is 0.18702273
Loss function at step 57000 is 0.15498415
Training complete!
------------------------4.Draw FittingCurve-----------------------
```



2）随机生成散点拟合图像举例：

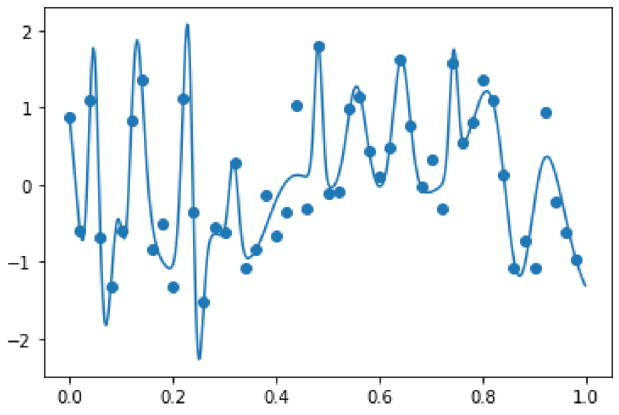
Size=10:

```
------------------------1.Set Data--------------------
------------------------2.Parameter setting-------------
------------------------3.Start Training-----------------------
Loss function at step 0 is 766.4529
Loss function at step 150 is 0.46334442
Loss function at step 300 is 0.27992004
Loss function at step 450 is 0.07902841
Loss function at step 600 is 0.009481664
Loss function at step 750 is 0.0002859346
Training complete!
------------------------4.Draw FittingCurve-----------------------
```



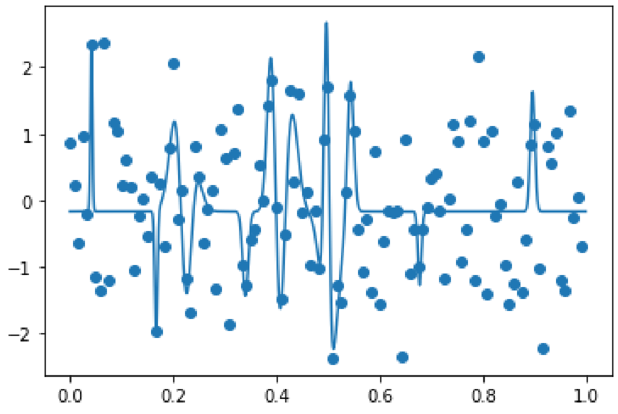
Size=50:

```
-----------------------1.Set Data--------------------
-----------------------2.Parameter setting------------
-----------------------3.Start Training-----------------------
Loss function at step 0 is 25982.195
Loss function at step 750 is 0.574142
Loss function at step 1500 is 0.5661243
Loss function at step 2250 is 0.55303735
Loss function at step 3000 is 0.5425798
Loss function at step 3750 is 0.53727007
Loss function at step 4500 is 0.5254903
Loss function at step 5250 is 0.478073
Loss function at step 6000 is 0.45511577
Loss function at step 6750 is 0.43508613
Loss function at step 7500 is 0.41739863
Loss function at step 8250 is 0.27828968
Loss function at step 9000 is 0.24463211
Loss function at step 9750 is 0.2298366
Loss function at step 10500 is 0.20691839
Loss function at step 11250 is 0.18970035
Loss function at step 12000 is 0.16725478
Loss function at step 12750 is 0.15689194
Loss function at step 13500 is 0.1456646
Loss function at step 14250 is 0.08428392
Training complete!
-----------------------4.Draw FittingCurve-----------------------
```



Size=120:

```
------------------------1.Set Data---------------------
------------------------2.Parameter setting-------------
------------------------3.Start Training-----------------------
Loss function at step 0 is 155131.1
Loss function at step 1800 is 1.0772233
Loss function at step 3600 is 1.0658598
Loss function at step 5400 is 1.0556515
Loss function at step 7200 is 1.0445191
Loss function at step 9000 is 1.0347751
Loss function at step 10800 is 0.98427373
Loss function at step 12600 is 0.91507685
Loss function at step 14400 is 0.81333286
Loss function at step 16200 is 0.7768009
Loss function at step 18000 is 0.76475114
Loss function at step 19800 is 0.7227032
Loss function at step 21600 is 0.69809186
Loss function at step 23400 is 0.76925766
Loss function at step 25200 is 0.7853064
Loss function at step 27000 is 0.75229806
Loss function at step 28800 is 0.727949
Loss function at step 30600 is 0.6815116
Loss function at step 32400 is 0.7292234
Loss function at step 34200 is 0.7171553
Training complete!
------------------------4.Draw FittingCurve-----------------------
```



4、评价与分析

可以看出，在数据量适中的情况下，RBF神经网络拟合是一个非常高效地算法，但是，当数据量过多的时候，由于程序中设置的隐藏层节点数或者训练次数不够，无法有效的进行拟合。并且，在数据量过小的时候，有时会出现过拟合的状况。还有一个比较严重的问题，就是采用的Adam算法会出现不收敛，或者是跳过最优解的可能，但暂时没有好的解决方法。

与作业1的比较：

在数据量较大的情况下（或者点分布较密集），RBF神经网络方法不失为一个更好的选择，并且作业1中许多方法都需要求解线性方程组，在精度方面也会对结果产生影响。但是，在数据量较小的情况下，选择作业1中的Gauss基函数拟合图像会更加光滑（在设定恰当的 σ 的情况下），且不会产生过拟合的情况。总得来说，具体选择什么样的算法需要根据输入数据的规模来判断。