**PJ文档 简易分布式神经网络**

本次pj要求实现一个简易的分布式神经网络根据学生数据对学生进行分类。

**一、整体架构**

1. 代码包目录结构

pj1

|--client.py client进程

|--master.py master进程

|--ws.py worker server进程

|--ps.py parameters server 进程

|--network\_ps.py 包含Network类，和一些ps进程需要调用的函数

|--generate\_data.py 生成训练和测试数据的脚本

|--student\_data.csv pj中预先给出的数据集

|--train\_1.csv student\_data.csv的前300行数据，可以用作训练数据

|--test\_1.csv student\_data.csv的后100行数据，可以用作测试数据

|--test.csv 脚本生成的测试集数据

|--train.csv 脚本生成的训练集数据

|--requirements.txt 项目所需的依赖库

2、项目简介

本次项目使用四个进程模拟分布式系统，分别是client, master, ps, ws，为了增加分布式系统的仿真性，各个进程之间使用socket进行通信。运行时，系统首先开始进行分布式的训练，系统使用过反向传播算法的bp神经网络结构，神经网络的结构可在client中进行灵活调整，支持多层隐含层，支持每层神经元数量调节，默认神经网络结构为输入层、输出层加单层隐含层，隐含层的神经元数量为10。经过100次的训练后，用户可以进行输入，用户分别输入gre, gpa, rank三项数据后，数据交至master, master对数据进行预处理normalization后，将数据交由parameters server, 由worker server进行计算，计算时数据值来自parameters server, 得到计算值后，将返回至client, client接收到结果后进行输出，然后等待下一次用户输入。

在前期的训练过程中，神经网络的结构参数，训练集，测试集在client中，由client经socket发送至master，master接收到相关数据初始化数据集数据，并进行预处理normalization，然后数据传至ps中，ps根据结构参数初始化网络结构，接下来ps负责存储网络参数并在需要时将参数提供给ws，同时负责训练过程当中的反向传播计算，ws进行正向传播计算，二者协作完成训练任务。训练结束后，将发送信号通知master，client，等待用户的下一步输入。

**二、函数及类的功能介绍**

1、类的介绍

本次项目中为了进一步仿真分布式系统，为了能够尽量做到将系统中的不同部署到不同服务器上就可以进行真正的分布式计算，特地使用了四个独立的进程，进程间使用socket通信，所以系统主体是四个独立进程，没有过多定义类，在项目中只使用了一个类，即Network类，用于在parameters server中存储神经网络的参数。对于Network的具体介绍如下：

|  |
| --- |
| class Network |
| attributes:  para 存储神经网络的所有参数，为了方便相应worker server的请求，故使用dict数据类型进行键值对存值  # 神经网络基本参数  para['learning\_rate'] 存储神经网络训练时的学习率  para['size'] 存储神经网络的结构参数，为list类型，默认为[3,10,2]  para['hl\_num'] 神经网络隐含层层数  # 正向传播计算所用参数  para['inputs'] 输入层的输入值  para['outputs'] 输出层的最终实际输出值  para['outputs\_bias'] 输出层偏置  para['outputs\_weight'] 输出层权重  para['outputs\_in'] 输出层的输入值  para['hl\_inputs'] 隐含层的输入值  para['hl\_outputs'] 隐含层的输出值  para['weights'] 隐含层的权重  para['biases'] 隐含层的偏置  # 反向传播计算所用参数  para['thf\_deris'] 隐含层激活函数的导数  para['hl\_out\_deris'] 隐含层输出值的导数  para['weight\_deris'] 隐含层权重的导数  para['thf\_out\_products'] 隐含层激活函数导数值和输出值导数的乘积  para['bias\_deris'] 隐含层偏置的导数  para['outputs\_deris'] 输出层输出值的导数  para['outputs\_weight\_deris'] 输出层权重的导数  para['outputs\_bias\_deris'] 输出层偏置的导数  para['outputs\_in\_deris'] 输出层输入值的导数  para['outputs\_products'] 输出层输入值导数和输出值导数的乘积  para['expec\_outputs'] 期待的正确输出 |
| method:  \_\_init\_\_(sizes, learning\_rate)  Network的构造函数，根据输入的网络结构参数sizes，以及学习率learning\_rate进行构造。对神经网络的参数进行初始化，各权重值范围为[-1,1] 的随机数，所有偏置的初始值为-0.5 |

1. 函数的功能介绍

分不同py文件对函数进行介绍

|  |
| --- |
| master.py |
| csv\_to\_data(file\_name)  将file\_name文件名对应的csv文件转化成数据集数据，最终返回两个list，分别是神经网络训练过程当中的输出值x和期待输出值y。函数接收到文件名file\_name后从文件中读取数据，将数据转化为浮点数后经预处理归一化。对于期待输出值再进行特殊处理，因为该问题实际上为二分类问题，所以将期待输出结果转化为一个长度为2的list，根据此对神经网络进行训练，完成二分类任务。 |

|  |
| --- |
| ws.py |
| request\_para(para\_name, index=None)  向parameters server请求para\_name名称对应的神经网络参数值，神经网络参数值。因为有些参数是存储在二位list当中，index表示所请求的参数的对应位置，若index=None则表示请求整个参数值。 |
| push\_para(para\_name, value, index=None)  该函数用于将worker server本地计算出的数值push到parameters server端。会将值value, push更新到para\_name所对应的网络参数位置上，index同上，表示参数的位置。 |
| sigmiodal(inputs)  神经网络的激活函数，使用了sigmiodal函数，根据输入值inputs计算出激活函数后的值并返回。 |
| forward()  进行神经网络的正向传播计算 |

|  |
| --- |
| network\_ps.py |
| sigmiodal\_deri(inputs)  sigmiodal激活函数的导数，根据inputs计算导数值 |
| back(network)  进行神经网络的反向传播计算并更新权重 |
| tackel\_ws\_request(network, ws\_ps\_sock)  该函数用于处理来自于worker server的参数请求，将存储在parameters server中的神经网络参数值发送给ws |
| loss\_value(network)  计算神经网络的输出值与期待输出值的误差值，误差使用方差公式计算 |
| error\_eval(network, ws\_ps\_sock, test\_x, test\_y)  根据测试集的数据计算神经网络的误差值 |
| max\_index(inputs)  返回list类型的inputs中最大数值的坐标 |
| accuracy(network, ws\_ps\_sock, test\_x, text\_y)  根据测试集数据计算神经网络的分类准确率 |

|  |
| --- |
| generate\_data.py |
| generate\_data(num)  生成格式类似于student\_data.csv中的数据 |

**三、项目解析**

1. 项目运行方式
2. 在项目目录下安装虚拟环境，使用指令 virtualenv venv (项目使用python3版本)
3. 激活虚拟环境，mac/linux指令 source ./venv/bin/activate, windows指令 .\venv\Scripts\activate
4. 安装所需依赖包 pip install –r requirements.txt
5. 依照顺序启动进程(因项目中使用了socket连接，请务必遵循以下顺序启动各进程)

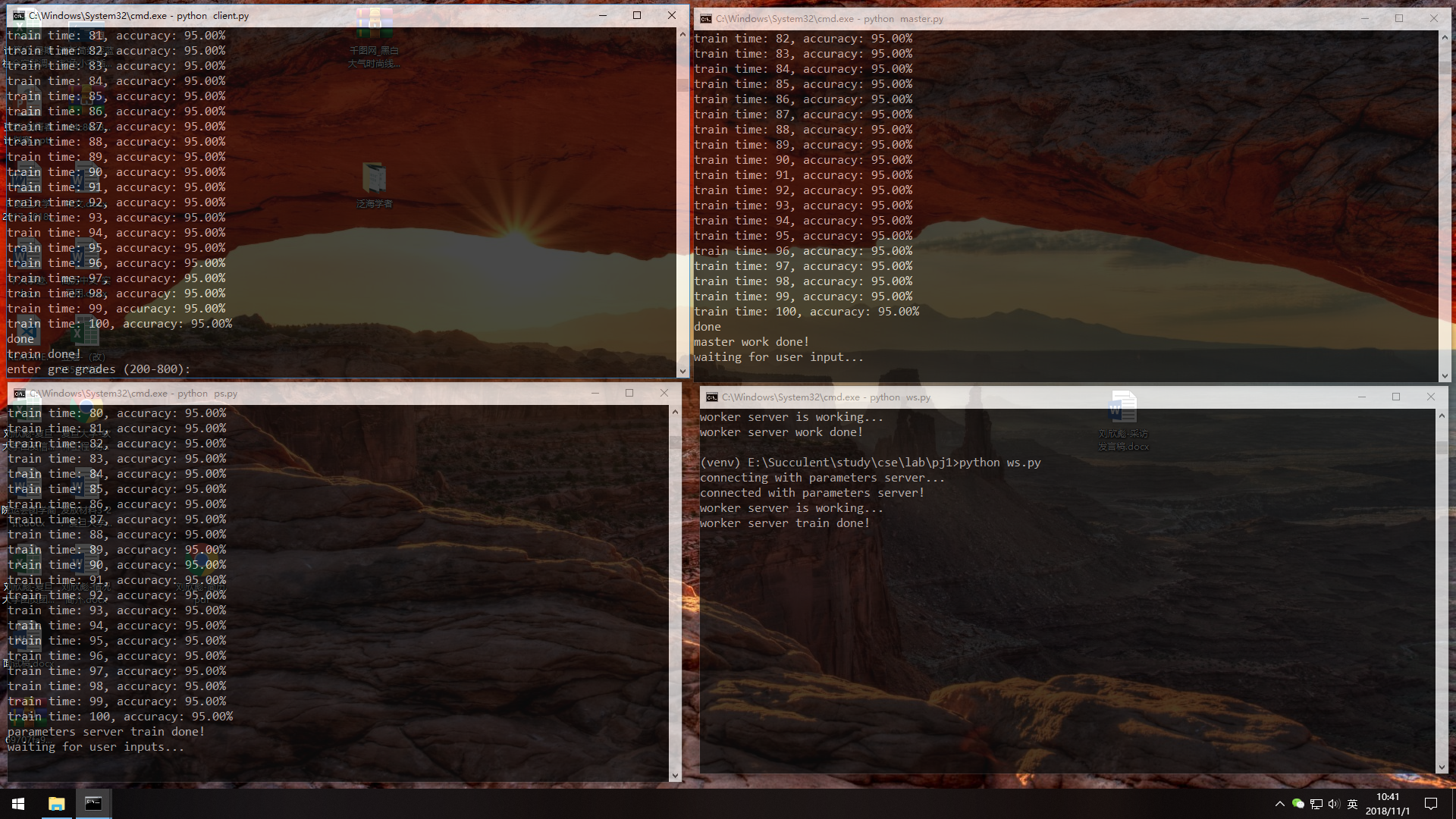
python ws.py

python ps.py

python master.py

python client.py

1. 系统成功启动，请根据提示等待网络训练完成



1. 根据客户端提示，在客户端输入想要分类的学生数据，等待其他进程计算后返回结果显示在client进程中
2. 系统解析

系统整体使用仿真的分布式系统设计。在训练过程中，由client进行配置神经网络结构，定义训练集和测试集数据位置，然后将相关数据传至master。 master在接收到client的数据之后，从指定位置读取训练集和测试集数据，然后分发任务至parameters server 和 worker server中。ps接收到来自master的信息后，对神经网络进行初始化，通知ws神经网络参数已准备就绪。ws开始正向传播计算工作，ps存储神经网络参数并进行神经网络反向传播计算，分布式地完成训练任务。然后等待用户输入计算数据，计算过程与训练过程类似，去掉了反向传播计算步骤。

之所以是按照这样的框架来进行设计是因为这样能够比较平均分配计算任务，将计算任务分配到不同的进程当中，更加符合分布式系统的计算原则，也能够突破集中式系统的性能限制。另外分布式系统还有比较好的容错性，如果在设备数量多的时候，个别设备出现故障仍能够保证整体的任务正常进行，和“不要把鸡蛋放在同一个篮子里”是一样的道理。

1. 系统优缺点解析

优点：

1. 将整体的计算任务分布到多个进程当中，如果多个进程真的运行在多台设备上，可以突破单台机器的计算性能限制
2. 在计算任务较为繁重的情况下，能够较好的提升计算效率。

缺点：

1. 在数据量较小，计算任务不大时，相比于计算开销来说，进程之间的通信开销不划算
2. 故障排除难度高，因为是多进程的仿真分布式系统，所以在系统构建过程当中的debug有一定的难度
3. 对于在单台设备上使用本机的socket进行通信比较稳定，如果放在复杂网络环境下，设备之间通信的稳定性不好保证

**四、项目过程遇到的问题**

1、在本次项目当中首先遇到的问题就是对于分布式系统的设计问题，这次的pj是我第一次自己动手去实现一个分布式系统，虽然只是一个仿真的系统，但对于来说也是一个极大的挑战。所以我充分研读了pj的要求文档，然后在网上找到了一些分布式系统的实例，经过学习后，我对分布式系统的理解也更上了一个台阶，也由此完成了系统的设计。

2、对于本次项目的设计，为了更进一步仿真一个分布式系统，我选择了多进程设计。但是这样就出现了一个问题，就是关于进程之间如何通信，经过比较之后我选择了通讯效率比较高的socket套接字通信。对于socket通信，因为稍微有些偏向底层，所以对于socket的理解和使用也花费了我相当的时间。而且在socket使用当中，对于通信的时序解决也是一个很头痛的问题，经过仔细的分析，我也终于解决了这个问题。

3、另外就是对于训练准确率的努力，经过最开始的尝试之后我发现如果不使用隐含层的神经网络的话，识别的准确率是比较低的，很不理想，所以为了解决这个问题，我在神经网络中加入了隐含层的部分，最终实现的系统是可以对神经网络进行调节，可以选择单层甚至多层的隐含层，并且每层的神经元数量也是由client来决定。通过这样的调整之后也得到了不错的准确率。

**五、对于分布式系统的理解**

根据我学习到的知识，觉得分布式系统可以大致分为三种。分布式存储系统，分布式计算系统，分布式管理系统。我们这次实现的就是一个分布式计算系统。其实最开始知道了分布式系统是懵逼的，感觉这个名字是很难理解的，当逐渐理解了分布式的含义之后我还是拒绝的，因为我觉得，明明一台机器维护起来就已经很麻烦了，为什么要用很多台机器仔细了解到我才知道，分布式系统还就是专门为了突破单个设备性能限制而存在的，分布式系统有很多很多的优点，首先就是他的容错性，假设我们的服务器有多台主机设备，那如果有一台设备发生宕机，我们其他的主机还是可以支撑起服务的正常运行。另外，设备多了，总体的计算性能也自然而然的就会增加，在没有超算的情况下也能用很多设备加起来逼近超算，完成大型的计算任务。但是，我发现分布式系统也还是有一些局限的，首先就是我最开始的想法，维护起来麻烦，相比于单独的一台机器，多台机器维护起来要消耗更多的精力。另外，多台设备机会涉及一个信息传输和同步问题，在传输的过程当中首先就是信息传输的开销，另外还有信息传输的安全性。所以，分布式系统还有相当的发展空间，希望在不久的将来能够对这些问题有一些解决方案。