# PJ1 设计文档

16302010029 谢东方

目录

[PJ1 设计文档 1](#_Toc528832473)

[16302010029 谢东方 1](#_Toc528832474)

[设计架构 2](#_Toc528832475)

[Client 3](#_Toc528832476)

[Master 3](#_Toc528832477)

[Parameter Server 3](#_Toc528832478)

[Worker 3](#_Toc528832479)

[所实现的class和function 4](#_Toc528832480)

[Package client 4](#_Toc528832481)

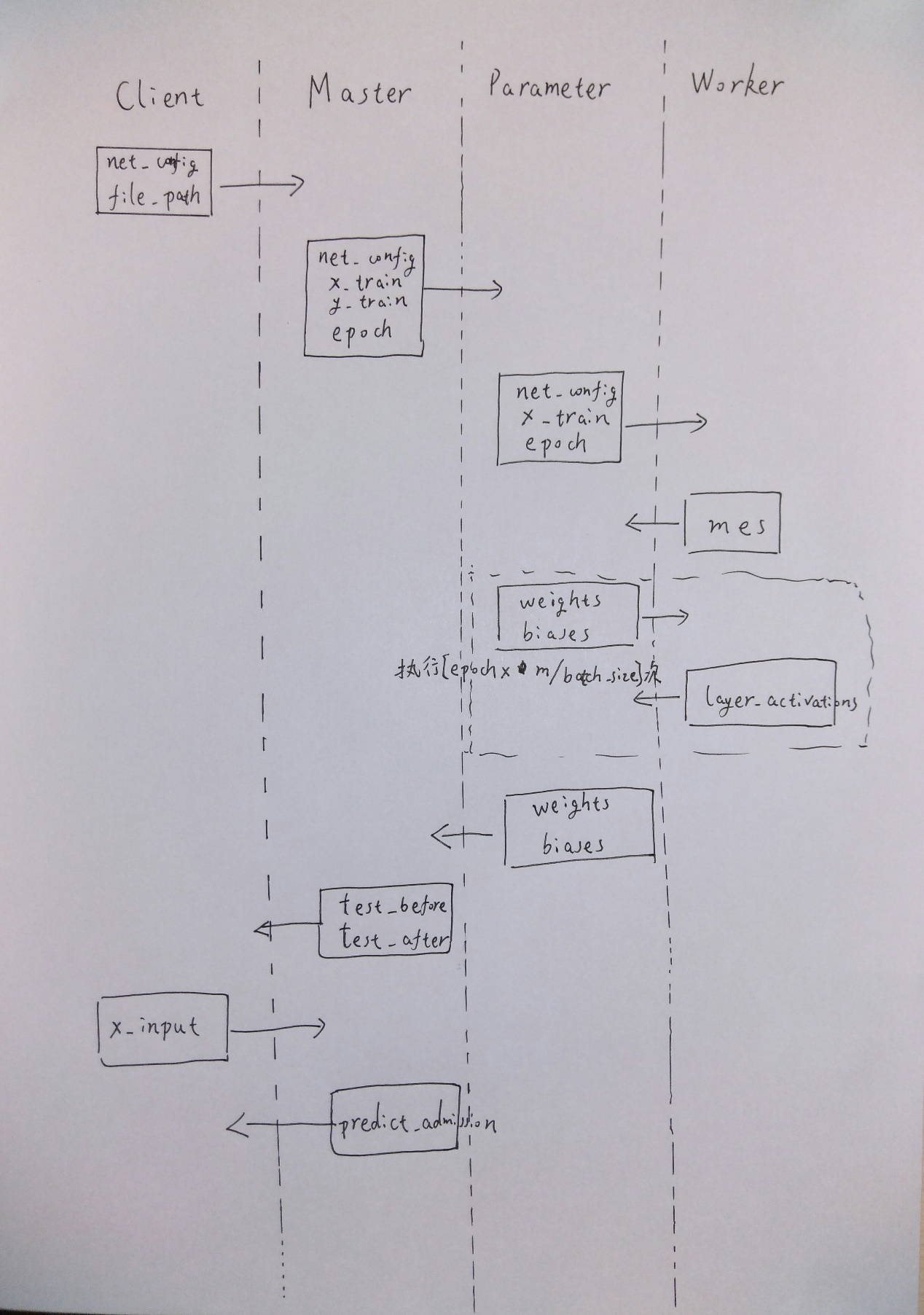
[Package server 4](#_Toc528832482)

[Package util 5](#_Toc528832483)

[Explanation of My Design 6](#_Toc528832484)

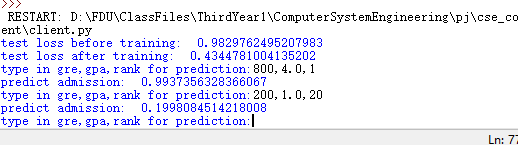
[遇到的困难 6](#_Toc528832485)

## 设计架构



### Client

1. 负责发送初始化的net config 参数，用于初始化神经网络，相当于计算图，并发送file path，需要训练的文件路径，由后端读取。
2. 等待后端训练完成，返回test before(训练前的误差) 和test after（训练后的误差）
3. 客户端可以输入数据进行测试，得到admit的结果，如下图。输入close关闭当前连接。



### Master

1. 接收来自客户端的net config和file path数据，读取相应的数据，并划分为训练集和测试集，将net config，训练集和训练次数发送给parameter server。
2. 接收parameter server的weights和biases数据，并进行测试，返回测试结果。
3. 接收client的x\_input（gre, gpa, rank），进行预测，并返回结果。
4. 当客户端发送close为真时，关闭当前连接。
5. 等待下一个连接。

### Parameter Server

1. 接收来自Master的数据，net config，训练集和训练次数，并向worker发送，net config，x\_train和epoch（训练次数），初始化神经网络。
2. 等待worker返回mes, “Worker is ready!!!”
3. 开始训练循环，具体过程：发送weights和biases到Worker，由Worker返回计算值，由parameter server进行神经网络反向传导。
4. 将weights和biases返回给Master。
5. 等待下一个连接。

### Worker

1. 接收Parameter Server发来的net\_config，训练集和训练次数，返回message，“Worker is ready!!!”
2. 开始训练循环，具体过程：接收weights和biases到Worker，返回计算值。
3. 关闭当前连接，等待下一连接。

## 所实现的class和function

### Package client

**client.py**

|  |  |
| --- | --- |
| **client()** | **一个启动的方法，建立socket连接，并发送数据。等待训练完成后，输入数据进行预测。** |

### Package server

**master.py**

|  |  |
| --- | --- |
| Class: Master | |
| \_\_init\_\_(self, HOST, PORT, listen\_num) | 进行初始化 |
| start(self) | 开启监听端口，执行一个个事务流程。 |
| read\_file(file\_path) | 获取训练集和测试集合，以及数据的取值范围大小，对原始数据进行了归一化处理。 |

|  |  |
| --- | --- |
| Class: Container： 这个类主要用于包装训练流程和最后的预测流程。 | |
| \_\_init\_\_(self, network) | 进行初始化 |
| train(self, x\_train, y\_train, epoch) | 输入训练数据和训练次数，负责和Parameter Server的交互。 |
| test(self, x\_test, y\_test) | 对神经网络进行测试，返回loss的平均值。 |
| predict(self, x\_input) | 对输入进行预测，返回相应的神经网络输出。 |

**parameter.py**

|  |  |
| --- | --- |
| Class ParameterServer | |
| \_\_init\_\_(self, HOST, PORT, listen\_num) | 进行初始化 |
| start(self) | 建立socket连接，开始监听，接收请求，并返回训练过后的weights和biases。 |
| communicate\_with\_worker(self, network, x\_train, y\_train, epoch) | 负责和worker的交互，发送训练数据，接收计算结果，并调整参数。 |

**worker.py**

|  |  |
| --- | --- |
| class Worker | |
| \_\_init\_\_(self, HOST, PORT, listen\_num) | 进行初始化 |
| start(self) | 创建连接，等待数据，返回计算结果。 |

### Package util

**jsonsocket.py**

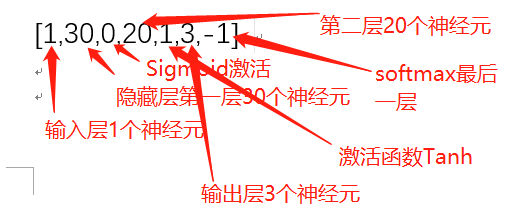
|  |  |
| --- | --- |
| class jsonsocket 继承了socket，不同之处在于，接收和发送的是dictionary。发送前接收后，用json处理。 | |
| recv(self) | 返回dictionary 数据。 |
| send(self, sendDict) | 发送dictionary数据。 |
| accept(self) | 返回对应的jsonsocket 和address。 |

**network\_elements.py**

内部有SoftmaxLayer，DenseLayer，TanhLayer等基础类，代表相应的层，含有相应的前向传播和反向传播方法。这里就不多说了。

|  |  |
| --- | --- |
| class Network | |
| \_\_init\_\_(self,net\_config,learning\_rate=0.01,weight=-0.5,bias=-0.5) | 进行网络的初始化，net\_config是对应的参数。 |
| forward(self,weights, biases, input) | 前向传播，需要提供相应的参数。 |
| train(self,X,y, layer\_activations) | 实际上是个调参数的过程，layer\_activations是计算返回的结果。 |
| predict(self,X) | 通过输入预测结果。 |
| get\_weights(self) | 获取权重 |
| set\_weights(self, weights) | 设置权重 |
| get\_biases(self) | 获取偏置 |
| set\_biases(self, biases) | 设置偏置 |

**net\_config的诠释，激活层可以用的激活函数有0（sigmoid），1(tanh)，2(ReLU)**



**另外，其实，master，parameter server和worker都放了一个network，由于是四个进程，他们显然都是不同的，他们使用了network的不同功能，比如worker的network只使用了forward功能计算每层的激活值，parameter server的network只使用了train进行调参，而master的network只使用了predict进行预测。**

## Explanation of My Design

1. 对于客户端，相当于，发送计算图和数据，然后输入预测数据，返回结果。相当于做了一层layering，屏蔽掉了后端神经网络训练的种种细节，可用性增强。
2. master端，如果改成多进程，就可以同时监听和服务多个客户端，功能更加强大。并且，将参数调整外包给parameter server和worker，减轻了自身的负担，从而可以将它架设在计算能力不是这么强的服务器上面。
3. parameter server和worker，parameter负责数据的存储，内存和磁盘空间足够即可，worker负责计算，只需计算功能强大即可。这样的分工，使得各个模块的运行更加高效。
4. 当前设计最大的好处在于，master, parameter server 和worker结束完一个服务之后，还可以为其他请求提供服务，如果有两个client，第一个client服务结束之后，服务器不会关闭，下一个client的服务就可以开始了。
5. 另外一个好处是，如果中间有一个环节崩溃了，只要设置好异常处理机制，就不会导致propagation effect，把错误控制在局部。比如这个worker不能用了，我换一个就好了。
6. 分离与独立性：各个服务器与客户端（master和parameter server既作服务器端，又作客户端），只要制定好了通信协议——传输的参数和信息，相当于接口，其自身作为一个模块，是可以进行随意改动的（比如升级服务或者增加服务器），并且不会对整个流程的运转产生影响。模块内部聚合度高，相互之间耦合程度小。客户端只需要知道可以访问的服务器和它们提供的服务就好了，而服务器只负责提供服务。

## 遇到的困难

1. 其实，第一个最重要的问题是，我不能确定是网络编程，我犹豫了很久，因为，我觉得应该不会出这么难的问题（我对网络编程一无所知）。然后，后面肯定了是网络编程之后，进度会快一些。首先，需要知道要做什么，才有开始的可能。
2. 然后最大的问题就是python的网络编程了，我对它一无所知orz。然后就从很简单的模型开始，先实现了一个简单的对答。之后，实现了高级一点的版本，但所有数据都还是手动输入的，所以每次出了bug，我就得把所有数据从头再输过一遍，非常的累。因为我构想的模式是大部分数据都应该由客户端手动输入。后来想了想，现在大部分应用并不是这样实现的，于是乎，我就改为半配置，半手动。这样一来，调试的过程大大加快。然而，网络编程中还是出现了一些麻烦的问题，比如每次我接收的都是字符串，异常麻烦，意味着我每次都要亲自处理字符串、浮点数和整型。这时，凭着我仅存的一点点关于网络编程的知识，我想到用json，然后就方便了很多。但这时候，发送的还是字符串。后来又做了改进，就好了很多。
3. 搭建4个进程，要管理他们的交互是非常麻烦的。我进行了尝试之后，构想出了他们的交互图，其实交互图应该最先构建的，相当于定义接口一样，非常重要。根据交互蓝图，将网络搭建好之后，我遇到了非常棘手的问题，因为交互实在过于，复杂，我基本没法找到好的方法进行debug，所以非常头痛。之后，我采取一步步攻克的方法，设置一些桩，交互一部分一部分地做，确认没有问题再进行下一步操作，这样我终于找到了出现的问题。
4. 接下来的问题就是我发送的数据解析出了问题，原因是发送的数据太大了，却只接收的1024个byte，于是解析出错。然后，用了新的socket，即jsonsocket去解决这个问题，它自带了报头，解决了接收长度的问题。最后，再处理一些边边角角的问题，我的project就完工了。