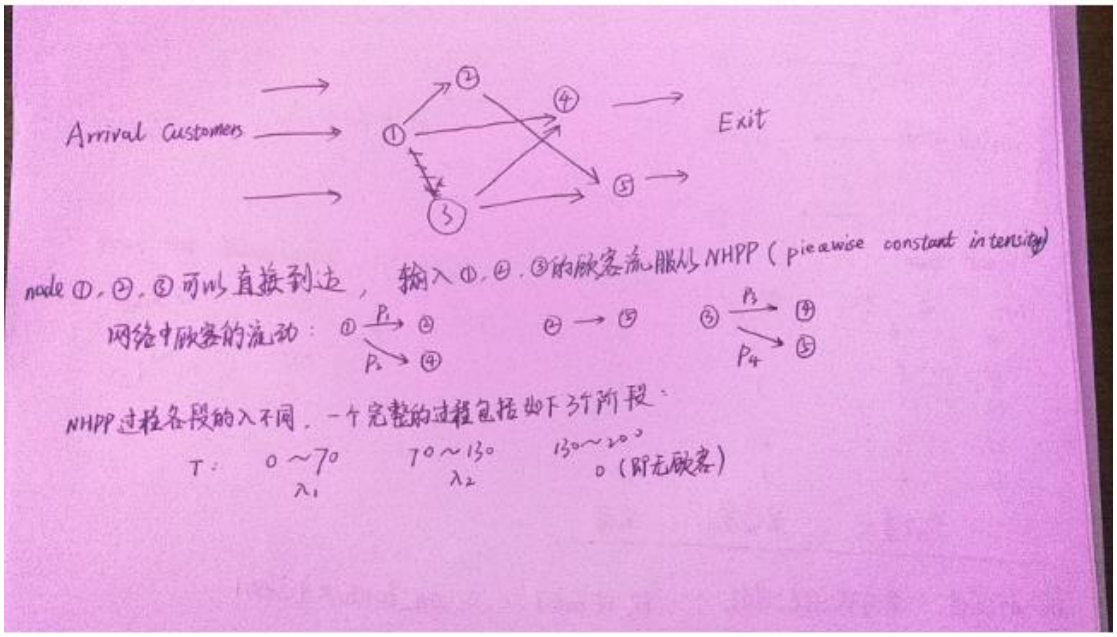
GAN 测试结果报告

用于实验的网络结构如下:



网络的参数调整中使用的一个例子如下:

```
N = ciw.create_network(
    arrival_distributions=[ciw.dists.Nhpp(1.0,0.5),
                           ciw.dists.Nhpp(0.5,0.1),
                           ciw.dists.Nhpp(0.8,0.2),
                           ciw.dists.NoArrivals(),
                           ciw.dists.NoArrivals()],
    service_distributions=[ciw.dists.Normal(2.0, 0.5),
                           ciw.dists.Exponential(0.4),
                           ciw.dists.Exponential(0.25),
                           ciw.dists.Normal(0.5, 0.5),
                           ciw.dists.Normal(1.0, 0.5)],
    routing=[[0.0, 0.4, 0.0, 0.6, 0.0],
             [0.0, 0.0, 0.0, 0.0, 1.0],
             [0.0, 0.0, 0.0, 0.3, 0.7],
             [0.0, 0.0, 0.0, 0.0, 0.0],
             [0.0, 0.0, 0.0, 0.0, 0.0]],
    number_of_servers=[[[3, 70], [2, 130], [1, 200]],
                       [[2, 70], [1, 130], [1, 200]],
                       [[1, 70], [1, 130], [1, 200]],
                       [[2, 70], [2, 130], [1, 200]],
                       [[2, 70], [1, 130], [1, 200]]]
)
```

测试主要分为网络参数的调整和 GAN 的 discriminator 结构的调整。
网络参数方面，输入是 NHPP 过程，一个 cycle 分为三个阶段：顾客到达率高，顾客到达率低，无新的顾客到达，模拟的是一天中不同时间段顾客到达的分布。
服务时间对于不同的节点我选择的是不同类型的分布，有正态，指数，均匀。

对 ciw 模拟输入的这些数据如何处理：
分别测试了只使用 arrival time，service time 和在此基础上加上 waiting time，exit time，block time 等其他数据，但是发现对结果并没有改善，反而使参数更难以收敛。因此最后选择仅使用每个节点的不同阶段的 arrival time 和 service time。

GAN 网络的结构的调整：
先使用简单的多层线性网络（多个 nn.Linear + nn.ReLU 层）但是没有任何效果。考虑认为各个 node 的数据之间的联系区别较大，而且关系较复杂，nn.Linear 则将它们一概而论，将所有的 node 的数据都杂糅在一起，这对于训练的要求很高，基本不可能训练的了。

在处理 ciw 的数据时我测试了两种方法：一是将其整理成 node_num*dim 形式的二维数据；二是将所有 node 的数据连接在一起形成 1*（node_num*dim）形式的一维数据。dim 代表一个 node 的 feature 数据的大小。换言之，我希望是网络能够从各个 node 的局部数据段中找到分布规律，这样很自然想到使用一维卷积网络来做 Discriminator，网络的结构以及一个测试中使用如下：

```
class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()

        self.main_module = nn.Sequential(
            nn.Conv1d(5, 5, 10, 5),
            nn.ReLU(True),
            nn.Conv1d(5, 5, 5, 2),
            nn.ReLU(True),
            nn.Conv1d(5, 5, 5, 2),
            nn.ReLU(True),
            nn.Conv1d(5,5,7),
            nn.ReLU(True),

        )

    def forward(self,x):
        return self.main_module(x)
```

数据的 diversity：
根据具体的 feature 还需要进行进一步调整。之前和汪教授讨论时提到过可能是训练数据的 diversity 不够导致网络无法得到充分的训练，因此我在设置随机数保证任何两次的 simulation 的结构都不同的前提下还对每个 epoch 的训练做了如下处理：
每次模拟 Simulate 多个 cycle（相当于现实中很多天），每个 cycle 的数据都是一个一个 sample，训练 N 个 cycle 相当于获得 N 个 sample。这样每个 epoch 训练时输入给 Discriminator 的 batchsize=N，我选择了 N=5，10，20 进行测试。这样训练数据的 diversity 基本可以满足，对于 real data 和 simu data，输入训练的 sample 的数量均达到了 2000~10000。

训练的结果

但是无论怎么增加训练的量，调整训练的参数（learning rate，discriminator，feature），GAN 的训练结果都得不到明显的改善。具体表现为 GAN 在训练过程中对于输入参数的初始分布异常敏感，每次训练过程中 loss 的变化和 parameter 的收敛都随初始参数的不同而不同。具体的规律为若初始参数对应网络的顾客的 arrival rate 很大，service rate 很快，则训练时它们变得越来越大，反之则越来越小。而 d_loss 和 g_loss 的变化趋势则是 d_loss 越来越小，g_loss 越来越大。随后我将各个 node 的 arrival rate 和 average service time 在训练过程中的变化输出，发现 simu 的 rate 和真实的 rate 之差较大，但是 GAN 网络的训练却并不能减小这个差距。

原因分析：

之后我主要在思考 GAN 训练不好的原因，查找了一些 wgan，mmd gan 训练的例子，特别是对于序列类型数据的例子。可以看到这些 project 中的 gan 的 Discriminator 使用的也均为一维卷积层，针对不同的具体任务 conv1d 的 kernel，stride 有所不同，网络总的层数不同。因此对于我们的这个任务使用一维卷积层来构建 discrimintor 网络理论上是可行的，因为分布体现在序列的局部变化中。

在跟踪 GAN 训练过程中 simulator 的输出数据的时发现对于如下现象：

1.当网络的输入顾客分布很稀疏或者很密集时，GAN 对其根本无法处理。

2.在参数相同的情况下 ciw 训练输出的数据的差别有时会很大，也即输出的数据分布有稀疏的倾向，也即很容易落入 1 的情况。

这两种现象导致 gan 训练很难，总体来说就是 ciw 模拟输出的数据的稀疏性倾向和当数据过于稀疏时 gan 没法处理（对于图像的情况可以看成当 generator 输出的图像几乎全是为像素为 0 的点，gan 很难将其改善）。

替代 loss:

最后我测试了各种 loss，最后发现只有 L1loss 和 Smoothl1loss 有效果，像 MSE，即使它和 L1 相似，但效果和 gan 一样没有训练效果。其中 Smoothl1loss 效果最好，表现为收敛速度很快（即使网络以及参数很复杂，经过 50epoch 左右即可基本收敛）和精度很高（精确地收敛到 groud truth 附近）。取得这些效果只有一个前提：数据量够大（每个 node 的 customer 至少 50），至于 feature，使用之前的 feature 即可。特别的，这种 loss 可以很好的处理稀疏性的情况。