Wgan 训练排队系统

生成器参数：排队系统的参数

判别器：浅层神经网络

训练时需要注意的地方：

- 判别器的层数不能太深，对于很深（看具体情况）的网络，训练过程中基本都会出现梯度消失的问题，即过早出现参数更新停止，而此时网络还远没训练好。
- 判别器最好先用 Conv1d 进行处理，之后再接 Linear 层，当然也可以完全不用 Linear 层，不过从实验效果看最后使用 Linear 更好。
- 判别器层数和第一层的结构不变的条件下，改变网络其他层结构和参数，都可以有较好的训练效果。要注意怎么设置 Conv1d 的步长和核大小来控制网络的深度。（测试发现使用 Conv1d 时，对 channel 没特别要求，输出 channel 直接设为 1 即可，增加这个其实对训练帮助不大）
- ciw 对于具体的一组参数生成的模拟结果的多样性其实并不是很大，所以训练时的 batchsize 不需要特别大（在图像和文本数据训练中常用的 batchsize=64），这里只需要 10 即可，batchsize 过大不但训练速度慢，效果反而更差。
- 之前的想法是定义一个 cycle（可以看作实际生活一天）的时间为 T，然后让 ciw 模拟 NT 时间来得到 N 个样本，虽然可以做到这点，实际模拟时候发现，N 较大时，一次模拟得到的后面的 cycle 的数据相对开始的几个 cycle 的数据畸变越严重。之后想了一下这样得到的 batch 里面 N 个样本也不是独立的，所以即使可以很好的得到 N 个样本，这样训练还是会有问题。解决方法其实也很简单，就是在 forward 时候模拟 N 次，相当于一次 forward 构建了 N 个随机计算图，但是这 N 个计算图有共同的 leaf。（**这个是之前 wgan 一直训不好的主要原因之一**）
- 最后是优化器的 learning_rate 的选择，之前使用 SmoothL1Loss 时候设置步长基本都比较大，训练时参数收敛的速度比较快，loss 也可以快速收敛，但是对于 Wgan，若 learning_rate 设置较大，训练一定会失败。虽然说 wgan 的训练是 simulator 和判别器同时训练，但判别器训练的次数要多于 simulator，而判别器又不能训练得太好，若 learning rate 较大，判别器很快可以训练好，之后排队系统的参数就不太变化了，也就是出现了梯度消失的问题，判别器过早训练好之后没法为 simulation 提供有用的梯度信息。也就是必须要花这么多时间慢慢更新 wgan 的网络，而不能像使用 L1loss 那样通过设置大的 learning rate 缩短训练时间。**（这时之前 wgan 效果不好的第二个主要原因）**
- 在从 ciw 模拟结果中提取的输入给判别器训练的 feature（包括各个节点的顾客到达时间，服务时间，等待时间等）一般维数较大，之前提到过的把这些数据整理成 1*（node_num*node_data）形式的一维 feature 和 node_num*node_data 形式的二维 feature。从结果看这两种形式都可以 work，但一维的情况对应的效果要好一些，表现为收敛更快，更准确。也就是说类似图像有多个 channel 那样把不同 node 看成排队系统的不同 channel 来表示 feature 并没有特别的效果，所以还是用一维 feature 更好。另外对于每个节点，考虑 waiting time 和 service start time 对于训练没有什么帮助，只需要使用 arrival time 和 service time 即可。
- 为满足 Lipschitz 限制，我选择了对所有参数都进行了 weight clipping，将它们限制在一个范围内。虽然 gradient penalty 是个更好的解决方法，但是它带来的好处对于我们这个任务是没有特别意义的，但是却会大大增加训练时间，所以不推荐使用。

网络设置和实验结果:
实验使用的是 3 节点的网络，对更多的节点训练时 loss 的变化和参数的收敛差不多，所以就使用了 3 节点的网络来测试。
实验中设计的判别器网络有两类，Conv+Linear 和 Conv:

```
self.main_module = nn.Sequential(
    #state=10*1*450
    nn.Conv1d(in_channels=1,out_channels=1, kernel_size=5,stride=3,padding=1),
    nn.ReLU(True),
    #state=10*1*150
    nn.Linear(in_features=150,out_features=10),
    nn.ReLU(True),
    #state=10*1*10
    nn.Linear(in_features=10,out_features=1),
    #state=10*1*1
)
```

```
self.main_module = nn.Sequential(
    #state=10*3*150
    nn.Conv1d(in_channels=3,out_channels=6, kernel_size=5,stride=3,padding=1),
    nn.ReLU(True),
    #state=10*6*50
    nn.Conv1d(in_channels=6, out_channels=12, kernel_size=5, stride=3, padding=1),
    nn.ReLU(True),
    #state=10*12*16
    nn.Conv1d(in_channels=12, out_channels=24, kernel_size=16, stride=1, padding=0),
    #state=10*24*1
)
```

Wgan 训练时的参考指标为 Wasserstein distance。（它越小表示模拟数据和真实数据的分布之间差距越小）
实验中它等于 d_loss_real - d_loss_simu
对于不同参数的 3 节点系统，该指标的变化都是先快速增大（可以视为判别器的初步训练过程），之后减小，并且在减小的过程中震荡得厉害，但是尽管 Loss 和 Wasserstein distance 是震荡的，参数却是稳定的朝 ground truth 变化，且参数朝 ground truth 的更新会使 Wasserstein distance 逐渐降下来，并最终收敛。不论参数初始化为何值，wgan 都能成功训练。这解决了使用 SmoothL1loss 时 service time 的参数训练困难和对于较密集（单位时间顾客量较大）网络训练效果不好的问题，代价是训练的时间大大增加。而一个问题是怎样确定停止训练的 epoch，因为可能会出现网络基本训练好了，但是 loss 还没收敛，仍继续在一个较小的值附近震荡，而从实验结果来看这几乎是必然的，所以不能以来 loss 的收敛来作为结束训练的，但以 loss 的范围来判断是否停止训练也是不合适的，选择合适的 epoch 是比较好的做法，这要在 training set 中训练得到。

需要说明的是对于不同的排队系统，Discriminator 网络需要相应的调整，而 learning rate 可能也要变化一些。