

ANN Finale —— DCGAN

ANN Finale —— DCGAN

基础实验

实验设定

`latent_dim` 与 `hidden_dim`

`FID`

深入讨论

超参讨论

纳什均衡

Linear Interpolation

Mode Collapse Problem

MLP GAN

基础实验

实验设定

图例中的线条图例遵从如下的实验命名原则：

```
1  GAN_{backbone}_latent_dim_{latent_dim}_generator_hidden_dim_{genera  
tor_hidden_dim}_discriminator_hidden_dim_{discriminator_hidden_dim}  
_seed_{seed}
```

其中，`backbone` 为 `{"MLP", "CNN"}`；默认 `seed` 为 42（如果 `seed` 为 42 则省略该项）

其余部分采用默认参数，如下所示：

```
1 batch_size = 64
2 num_training_steps = 5000
3 learning_rate = 0.0002
4 beta1 = 0.5
```

为了完成本次作业，我一共进行了 18 组基础实验，其配置参数如下：

```
1 latent_dims = [16, 64, 100]
2 hidden_dims = [16, 64, 100]
3 # 将 generator_hidden_dim 与 discriminator_hidden_dim 设定为相同的大小
4 backbones = ["CNN", "MLP"]
5 experiments = product(latent_dims, hidden_dims, backbones)
```

此外，为了测试 `seed` 参数对于实验的影响，我测试了如下 4 组实验：

```
1 seeds = [43, 107, 213, 996]
2 latent_dim = 16
3 generator_hidden_dim = 16
4 discriminator_hidden_dim = 16
5 backbone = "CNN"
```

由于 GAN 的初始化对于实验结果的影响是非常显著的，我进行了两组实验；所有的实验结果可以参考[第一次实验结果与第二次实验结果](#)。

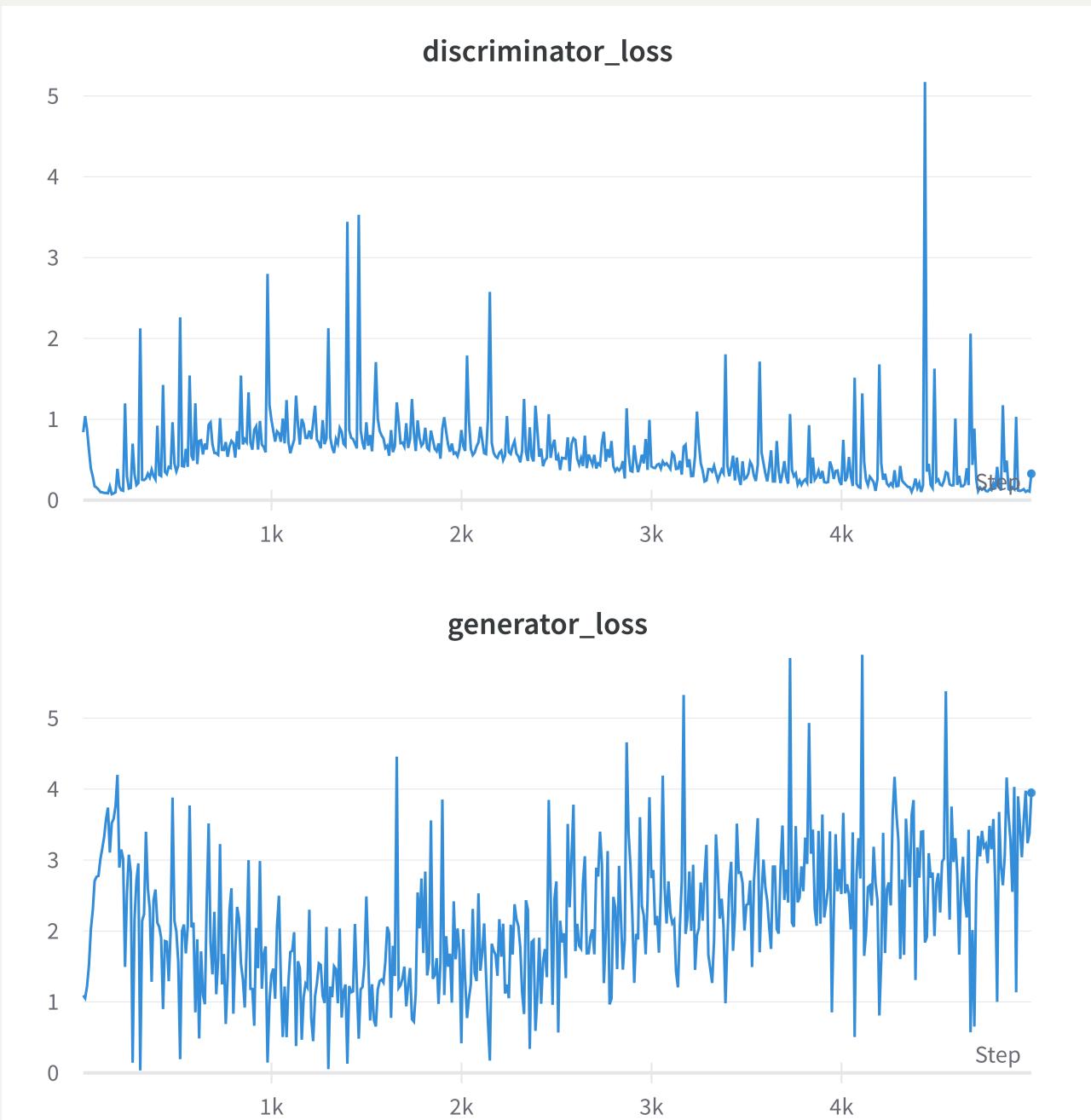
`latent_dim` 与 `hidden_dim`

为了更加显著地显示 `latent_dim` 与 `hidden_dim` 带给模型的差异，我选择展示如下四组实验的训练曲线与最后一次生成结果：

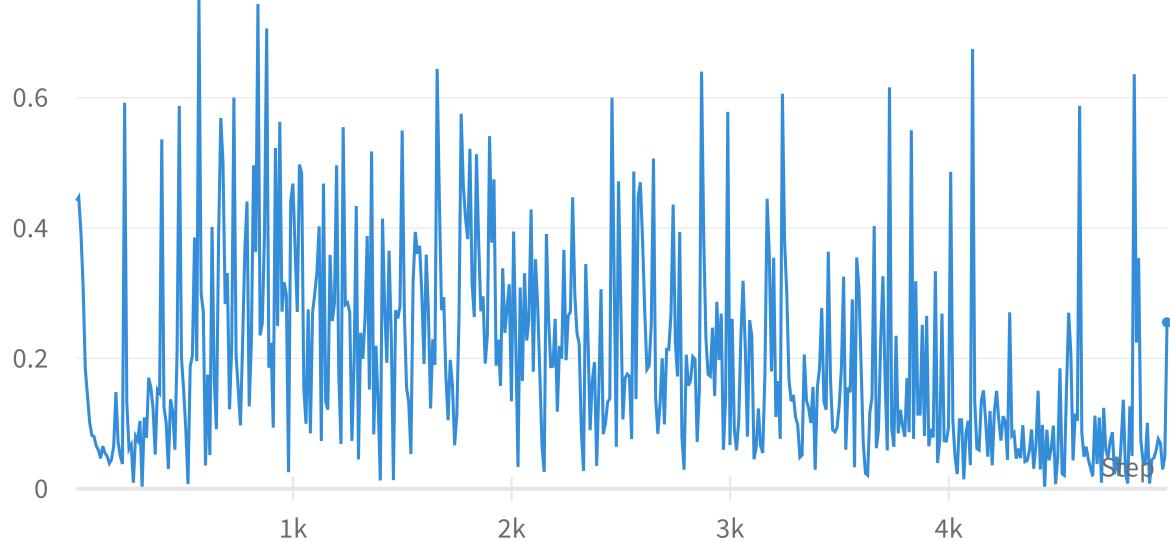
```
1 latent_dims = [16, 100]
2 hidden_dims = [16, 100]
3 backbones = "CNN"
```

以下小节标题为 {backbone} {latent dim} {hidden_dim}

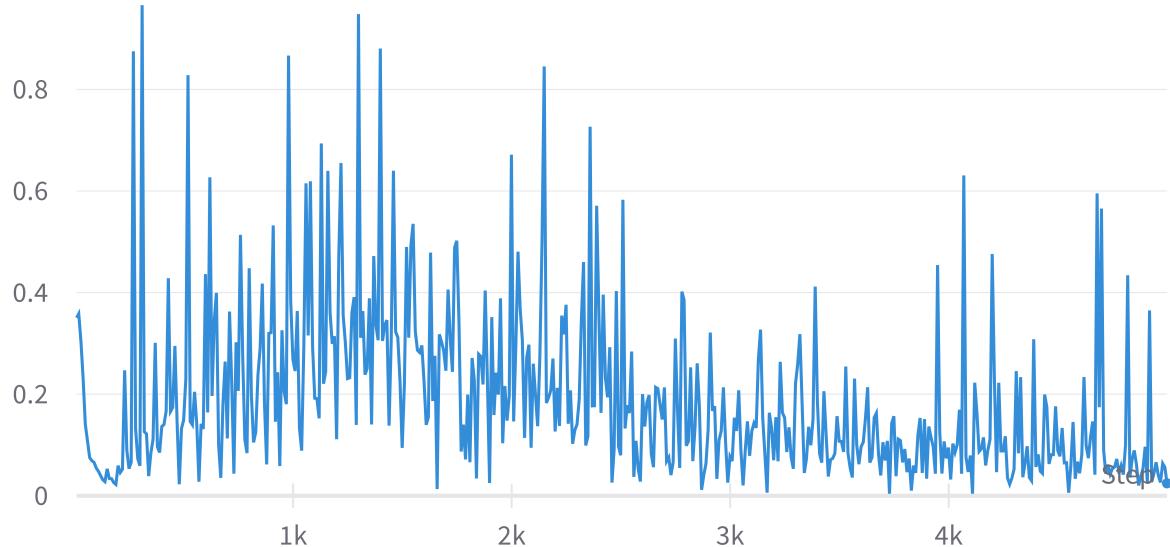
CNN 16 16

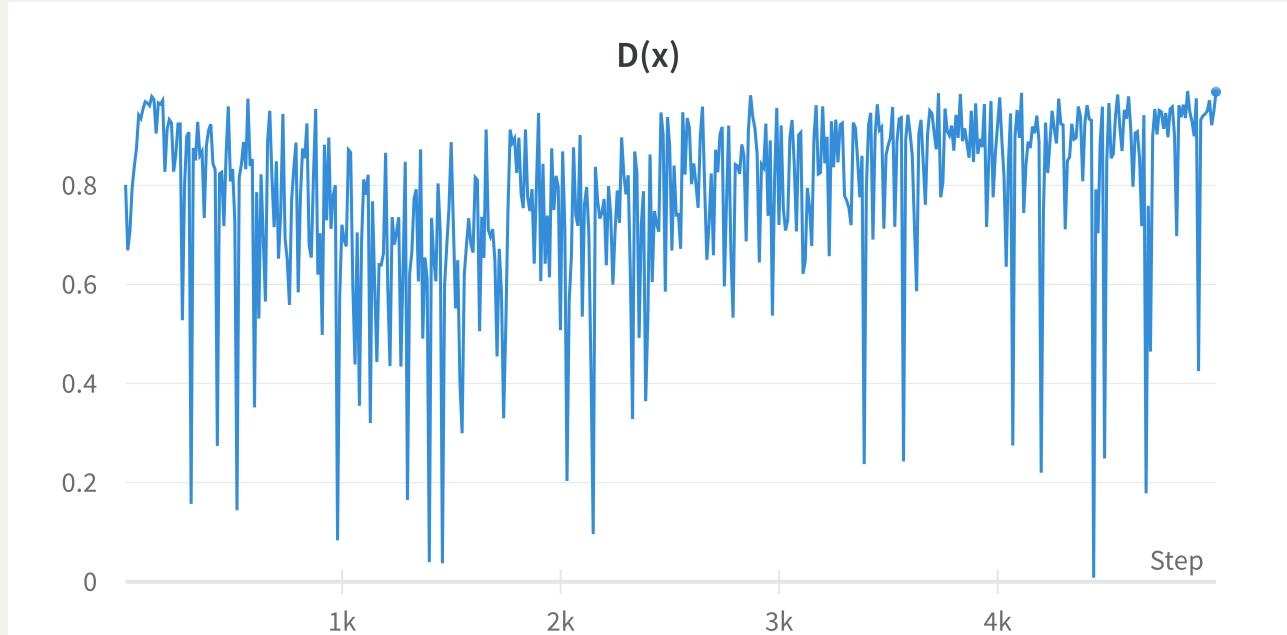


$D(G(z1))$



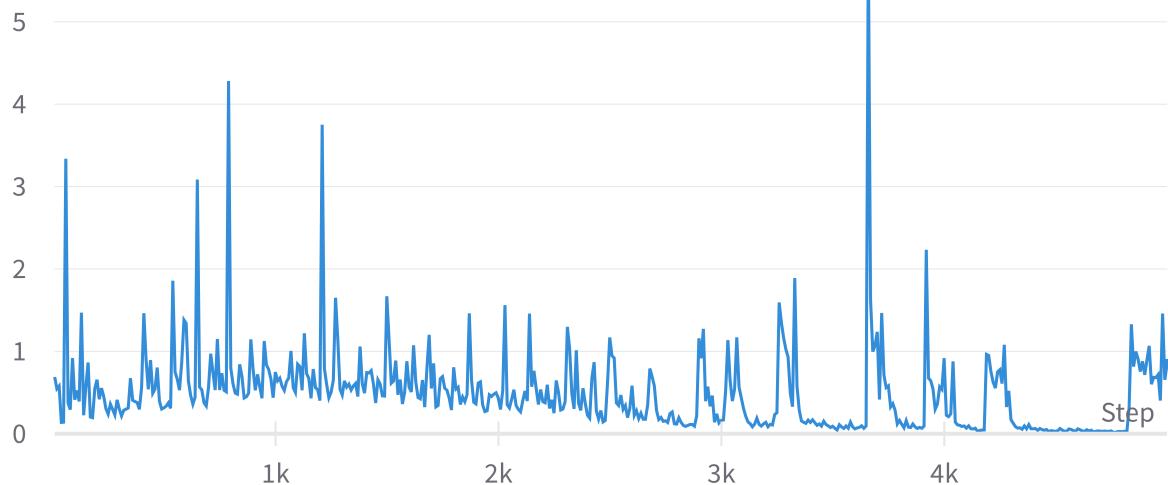
$D(G(z2))$



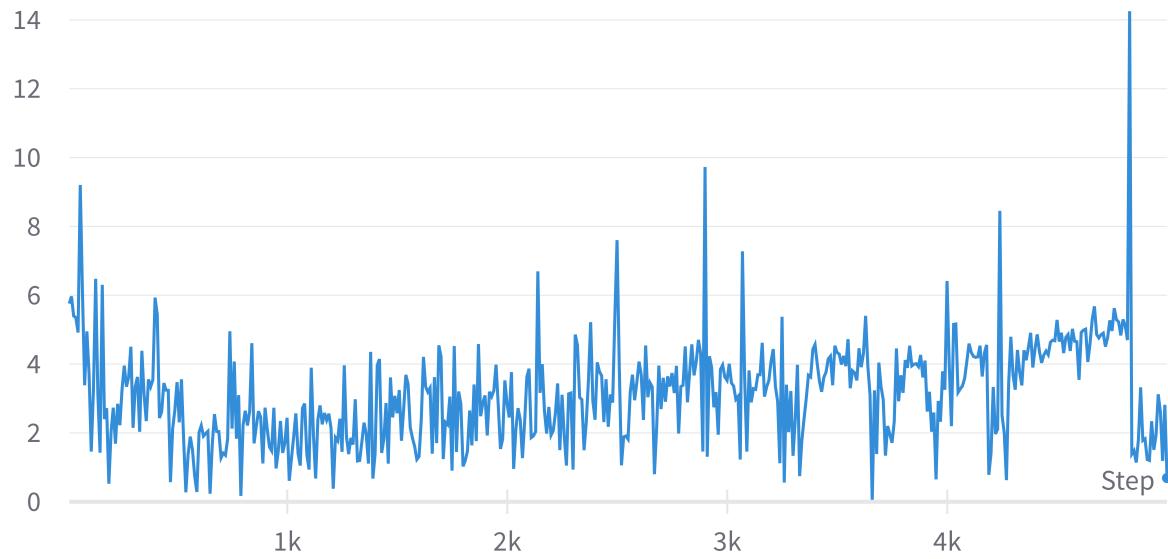


CNN 16 100

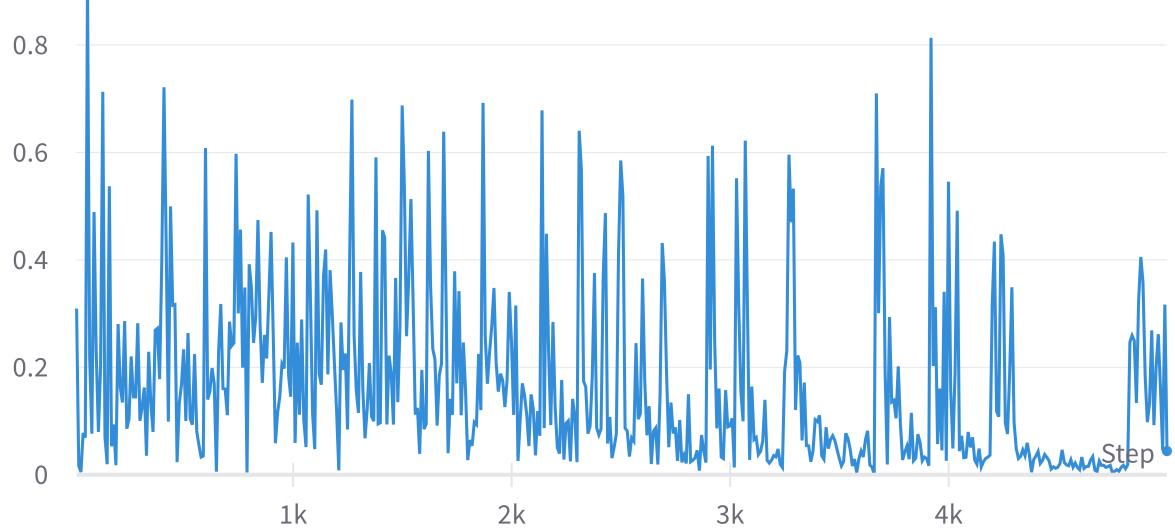
discriminator_loss



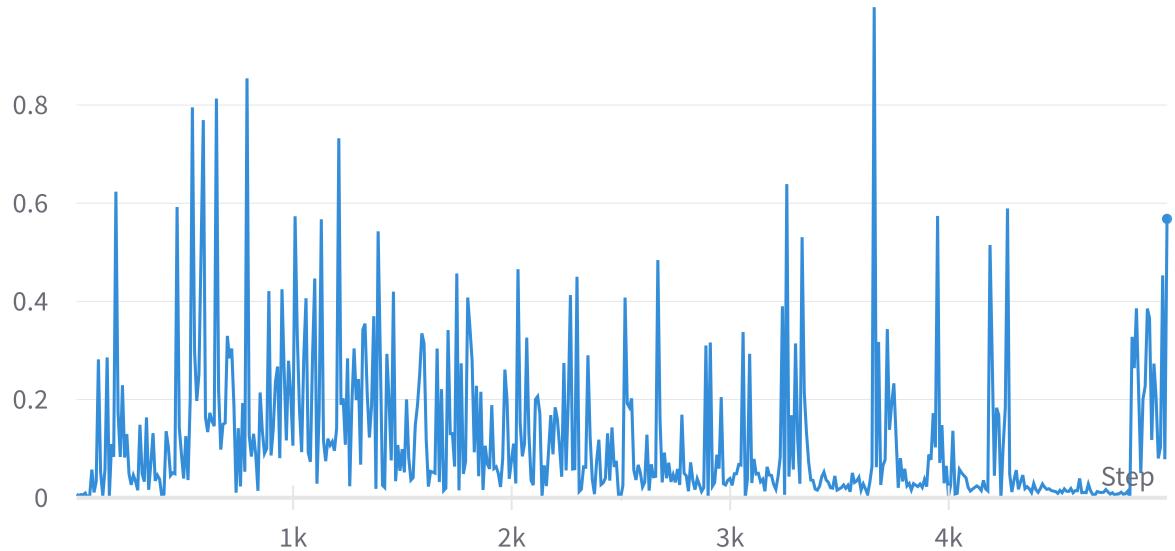
generator_loss

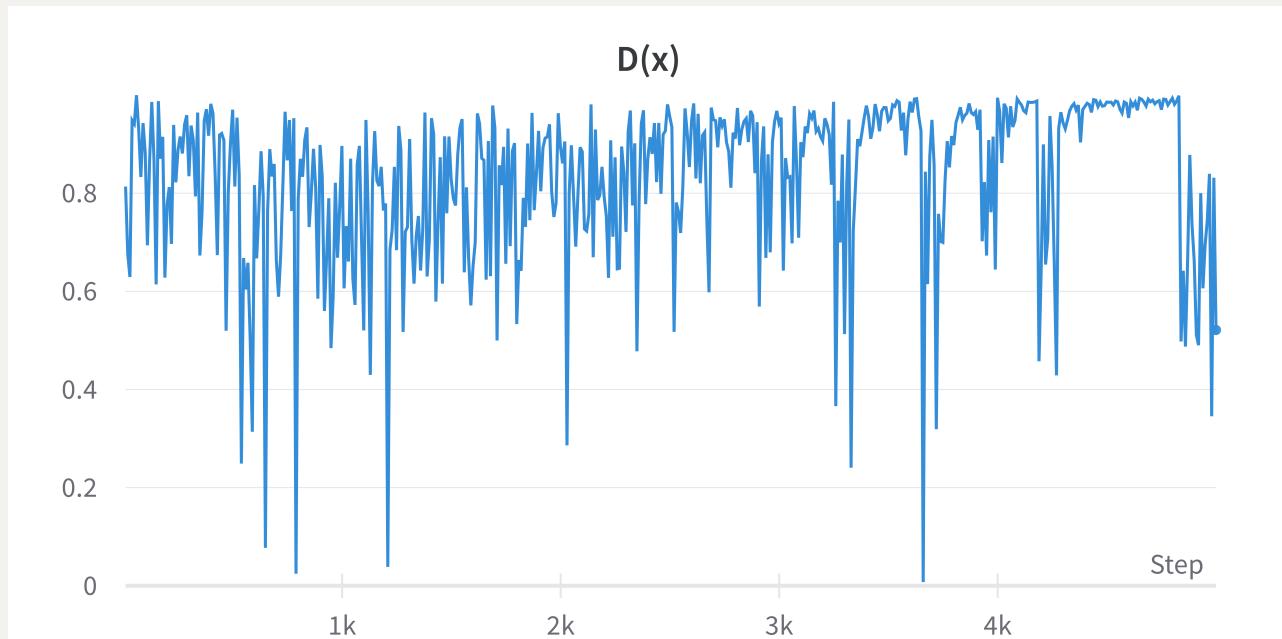


$D(G(z1))$



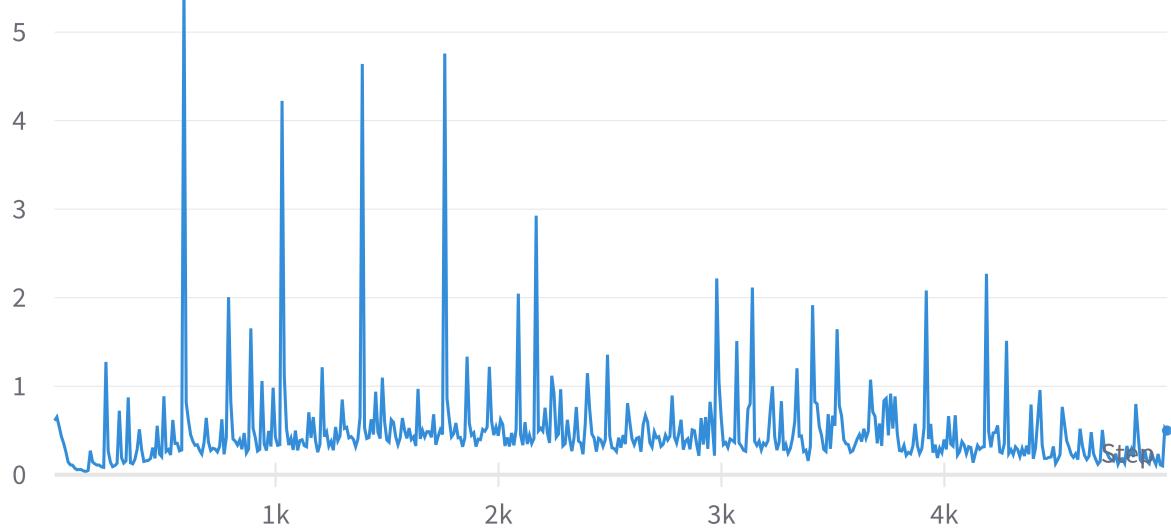
$D(G(z2))$



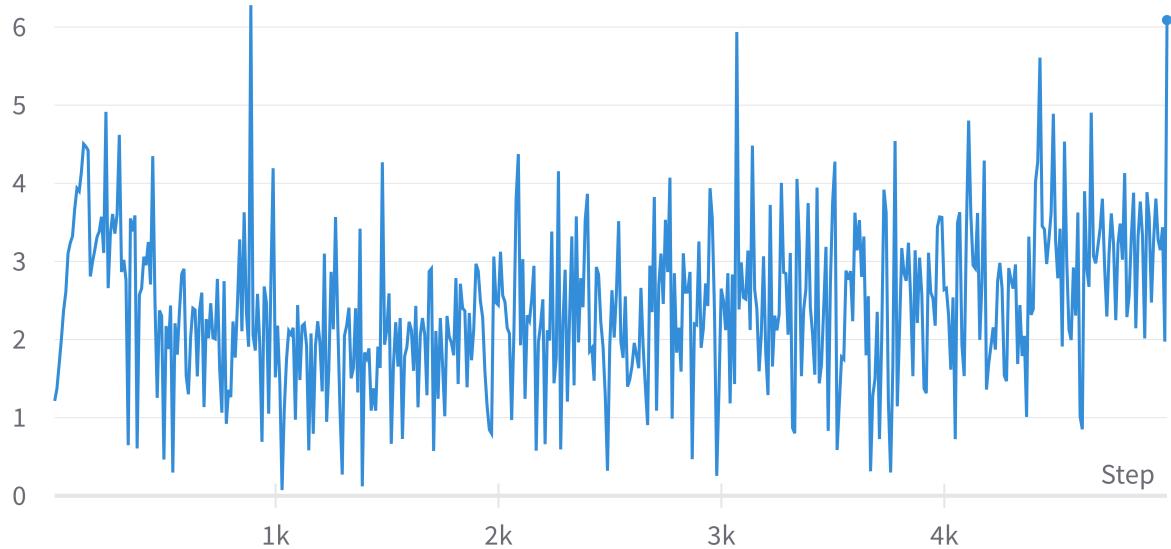


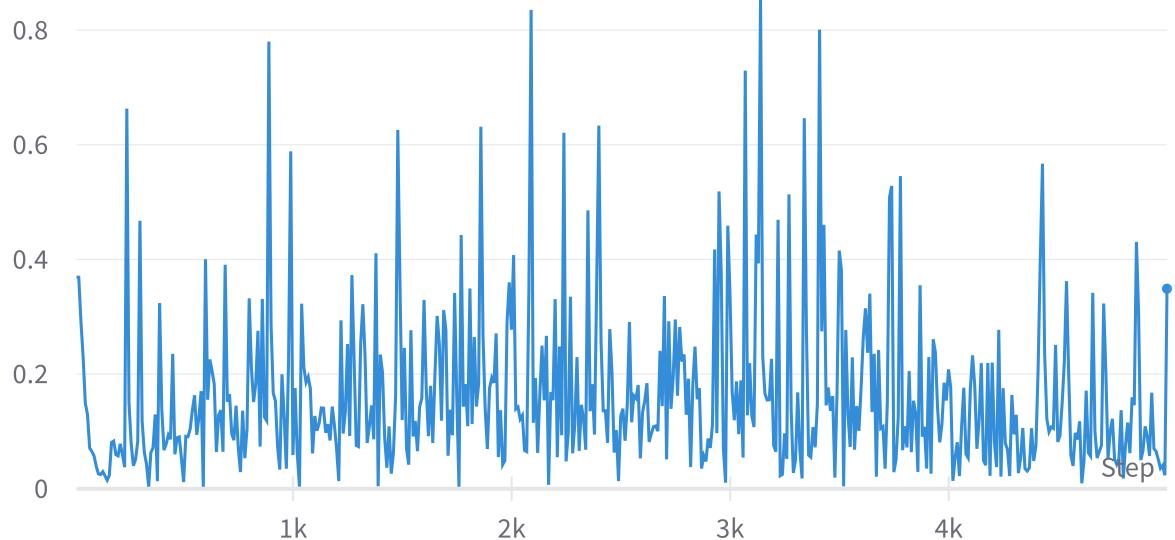
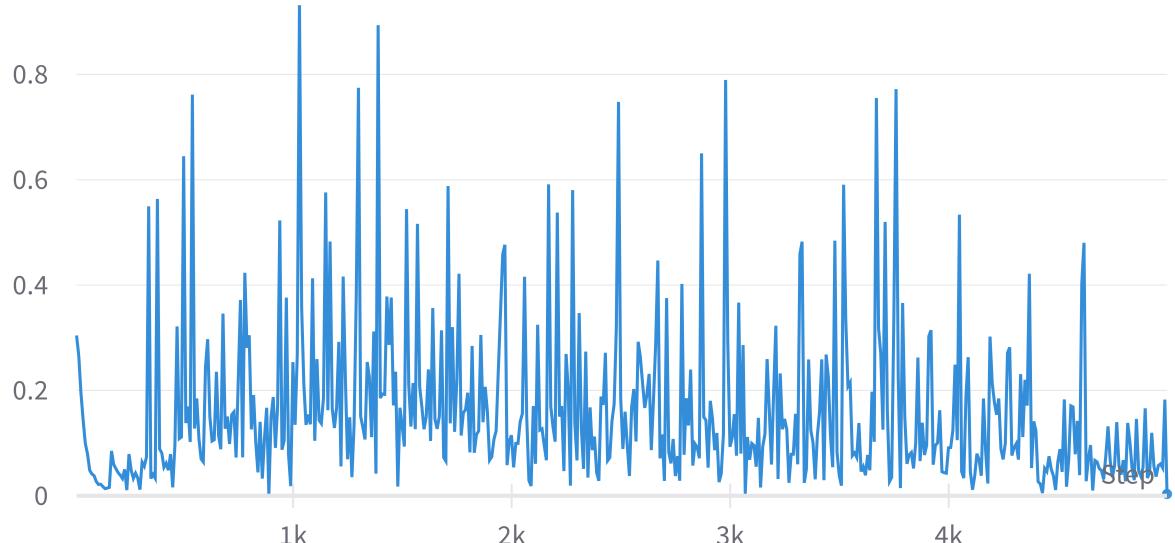
CNN 100 16

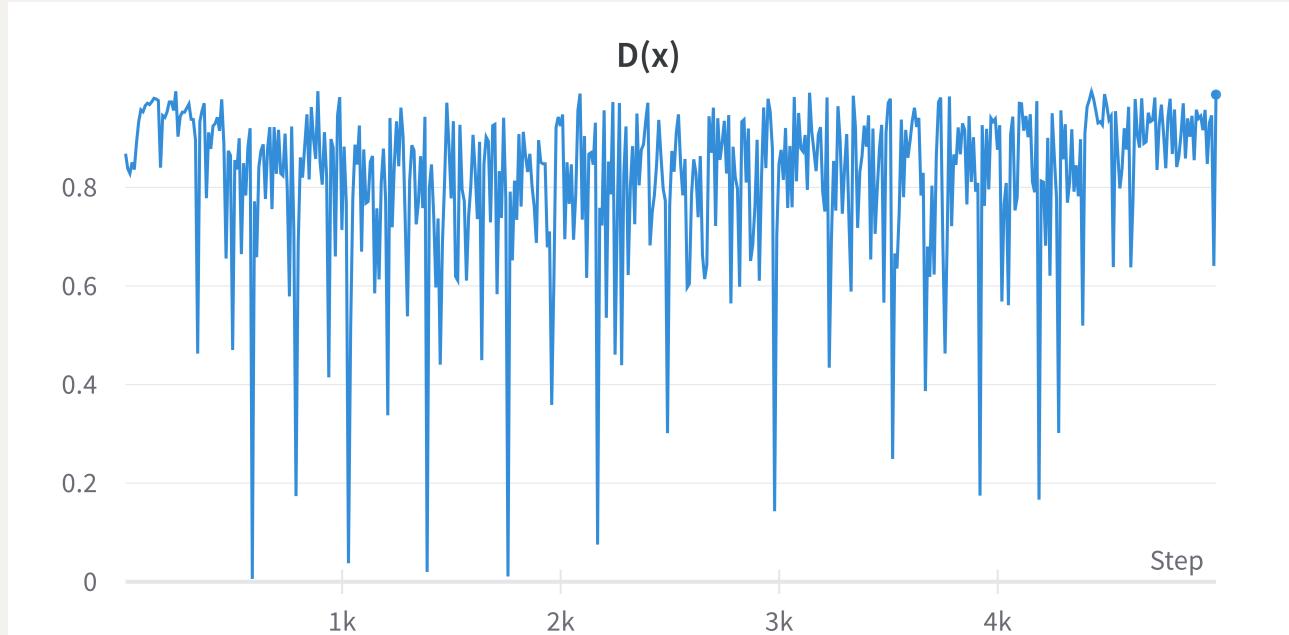
discriminator_loss



generator_loss

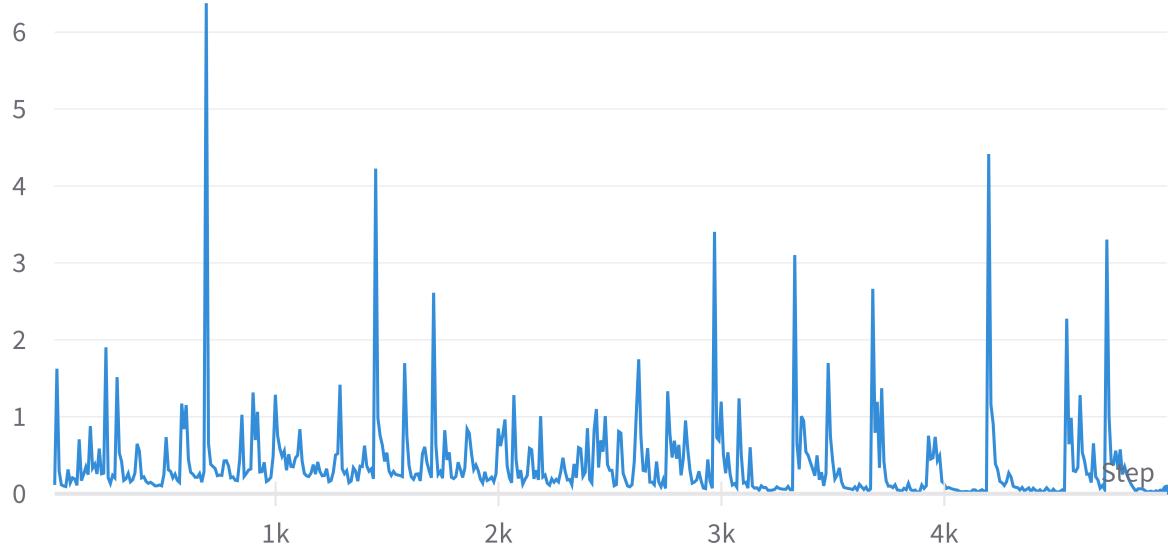


$D(G(z1))$  $D(G(z2))$ 

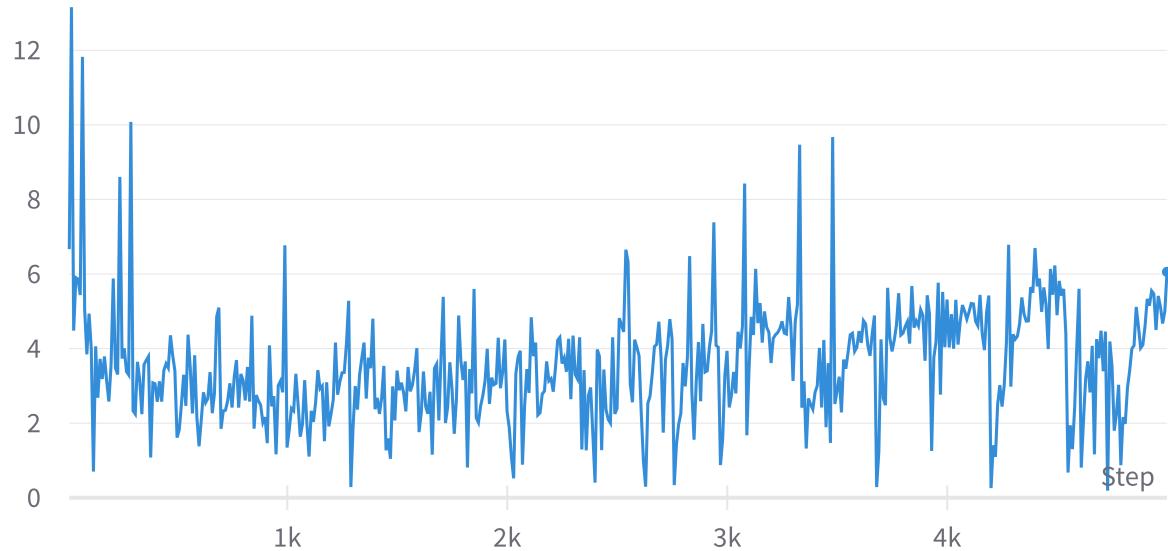


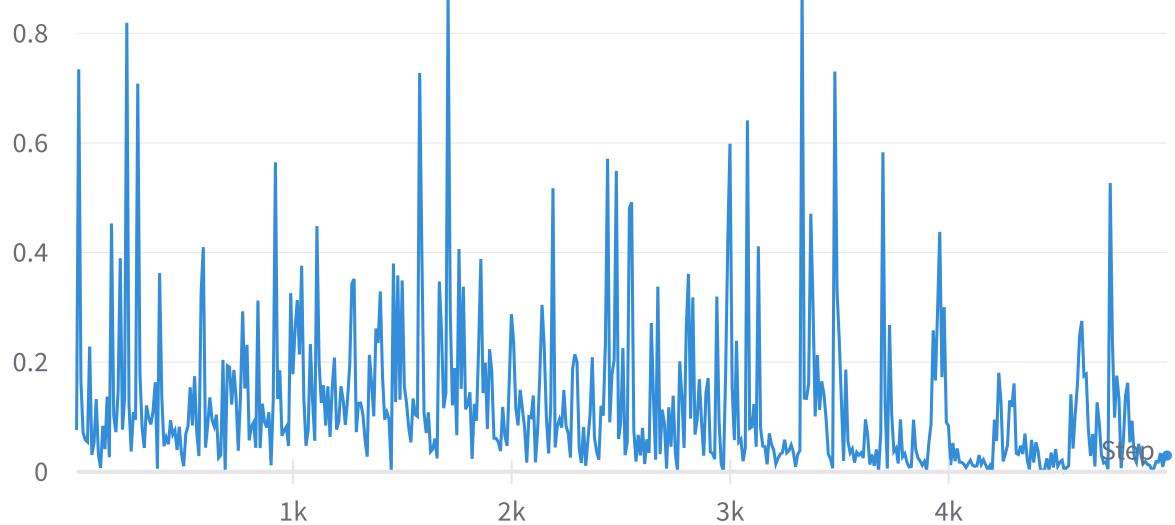
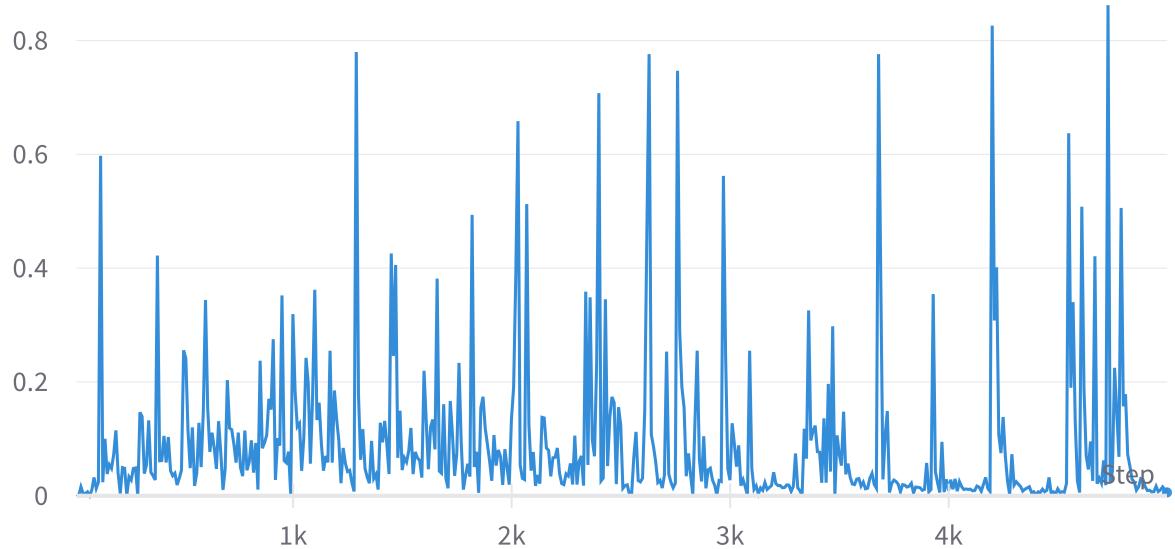
CNN 100 100

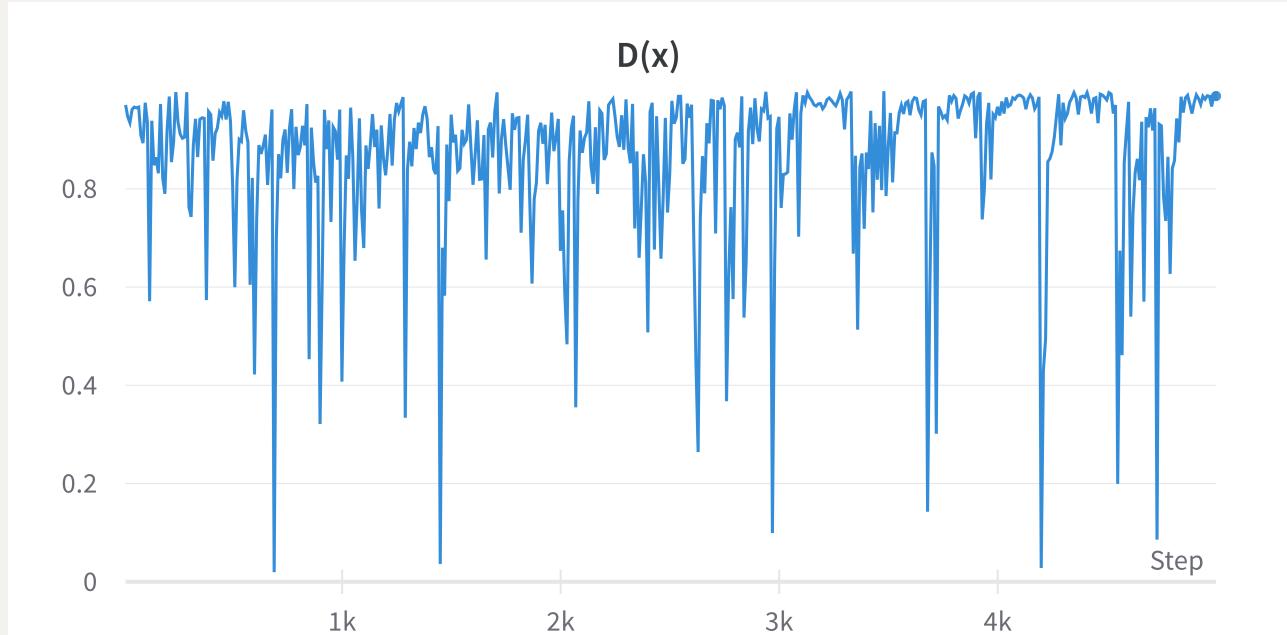
discriminator_loss



generator_loss



$D(G(z1))$  $D(G(z2))$ 



FID

MODEL CONFIG	FID SCORE
CNN_16_16_16	104.45376654460628
CNN_16_100_100	31.057871640232264
CNN_100_16_16	65.38224467437774
CNN_100_100_100	36.13799152833505

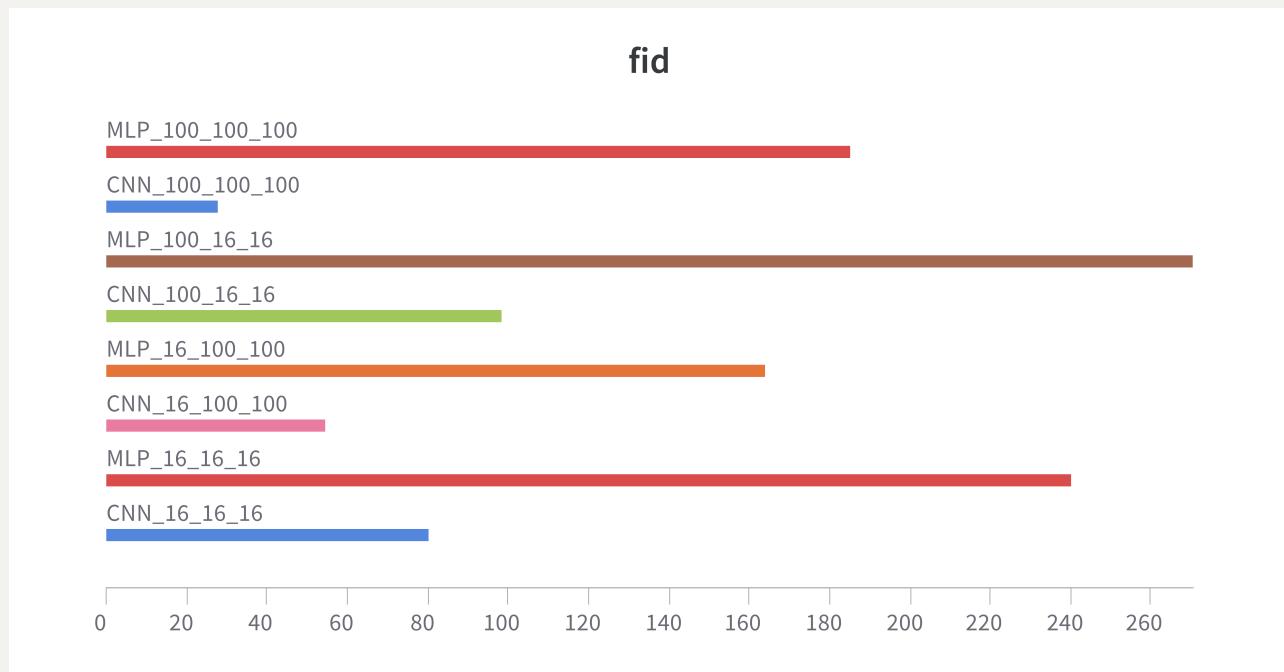
可以见得，四组模型中，CNN_16_100_100 模型的 FID 最佳，取得了 31.06 的优秀表现。

此外，注意到当 FID 超过 100 时实际上意味着训练效果较为糟糕，我反复测试了 CNN_16_16_16 模型的 FID，其值在 70 与 110 之间反复波动。

深入讨论

超参讨论

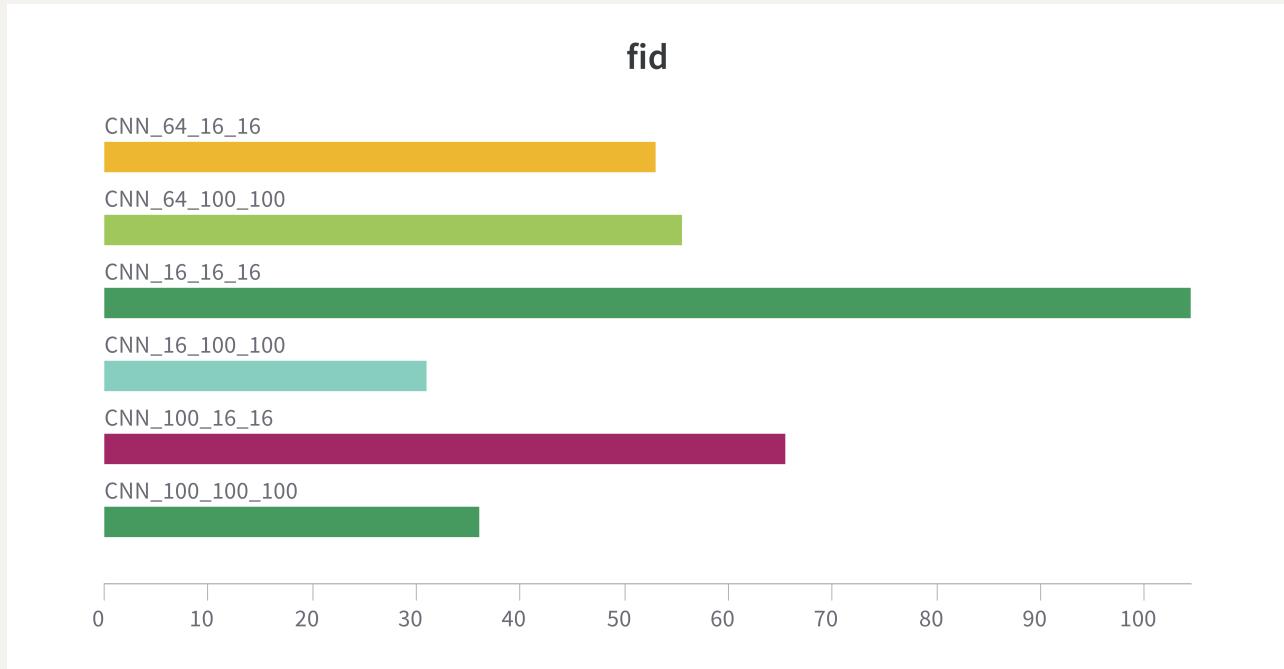
下图中涉及到关于 MLP 的数据均来自[第二次实验结果](#)，而仅含有 CNN 的数据来自[第一次实验结果](#)。



从上图可见，就 FID 指标而言，对于 MLP 和 CNN，`hidden_dim` 增大都能带来训练效果的提升。

`hidden_num` 几乎直接体现了模型的表示力，其增大导致指标的提升是合理的。在不考虑训练成本的前提下，增大 `hidden_dim` 会增强 generator 的生成质量；同时，也有利于增强 discriminator 的分类能力。

训练过程中，generator 的生成潜能得到了强化，而更强劲的 discriminator 能够迫使生成器产生更好的生成效果。无论是从 generator 还是 discriminator 的角度而言，增大 `hidden_dim` 都有助于提升模型的表现能力。



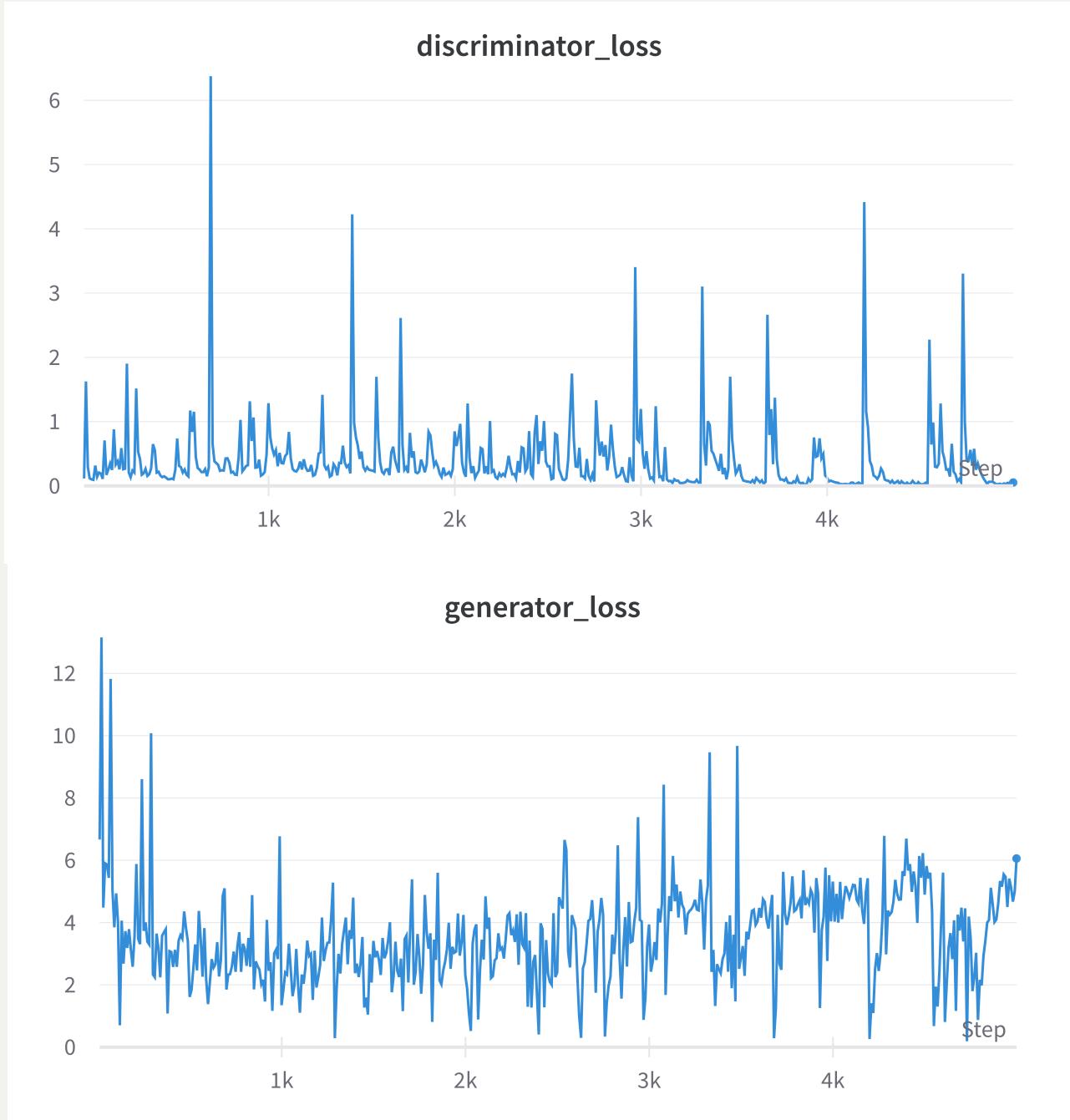
如上图所示，而考虑 `latent_dim` 时，可以见得其参数变动对于模型的影响相对没有规律可言。考虑到 `latent_dim` 主要是用于随机噪声的维度，而过大的 `latent_dim` 相当于更随机的噪声，这对于一个生成器来说并不一定会导致更好的效果，反而可能加大学习难度致使指标下降。

总体来看，目前来看过大或过小的 `latent_dim` 都不合适、取一个适中的值会比较好。

纳什均衡

在 GAN 一文的原文中，作者提到了 GAN 模型的纳什均衡点为 0.5。这是最理想的情况下，生成器生成的图像完全可以以假乱真，辨别器随机的输出正误。

此处参考 CNN_100_100 模型的训练曲线：



训练全程的 `discriminator_loss` 和 `generator_loss` 都在剧烈波动，远远没有达到纳什均衡。实际上，台大李宏毅老师曾经幽默地形容道：“No Pain No GAN!” GAN 的训练难度出奇的高，往往需要用到如下的一些技巧：

1. 梯度惩罚
2. 优先训练 Discriminator
3. 标签平滑
4. 添加噪声
5. 不使用基于动量的优化算法，推荐 RMSProp 等等

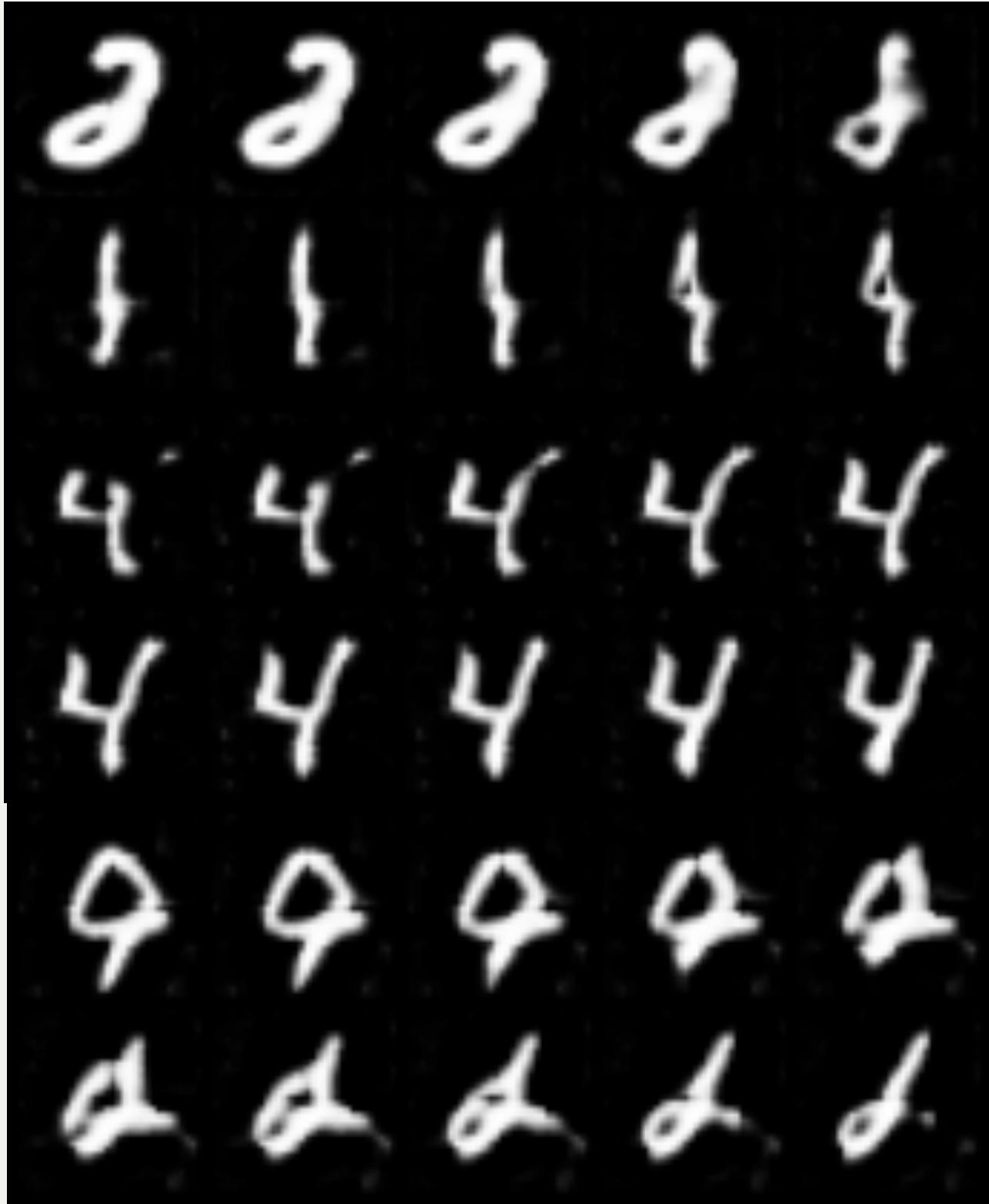
6. 在生成器和判别器中使用 Batch Normalization

本次实验没有实现其中任一技巧，用朴素的同时训练法难以达到优良的训练效果可想而知。

Linear Interpolation

按照题干要求，我为每次实验均选取了五对 (z_1, z_2) 并选取 $K = 10$ 进行插值实验。在此，我选取效果最佳的 CNN_16_100_100 模型展示其效果：





可以见到，随着 i 从 1 过渡到 10， $z_1 + \frac{i}{K}(z_2 - z_1)$ 从 z_1 逐渐过渡到 z_2 ，在图像上也呈现出了渐变的效果。相连的图片间差距不大，然而首尾两端的图片有明显差距；特别是第二张图片，从起初的 4 逐渐过渡为 9。虽然其他几组的生成效果不容乐观，然而在此如此朴素的训练方式下，模型还是有了基本的生成能力；不同的 latent variable 可以生成具有高频信息的可识别数字，甚至能够达到渐变效果。

美中不足，模型的生成能力还是有所欠佳，最后一组数字从 9 开始渐变，然而最后一张图片完全无法辨识，且中间渐变的图片也非常扭曲。

Mode Collapse Problem

同样选取 CNN_16_100_100 模型，检测其随机生成结果：



利用 HW1 训练得到的 CNN 模型进行统计，得出每个数字的出现次数如下：

0	1	2	3	4	5	6	7	8	9
5	2	4	7	2	1	4	9	13	3

如上的数据分布显著偏离了 $[0, 9]$ 之间的离散随机分布，模型的确出现了 Mode Collapse Problem。

此外，查阅 CNN 分类模型给出的结果，出现频率较低的数字的生成质量也非常堪忧。比如模型唯一出现的 5 在第二行倒数第四列，而这一数字 manual detect 都无法认定其为 5。对 5 较低的生成质量让我们有理由猜测，在 GAN 的极大极小游戏中，一旦生成器生成了 5，辨别器可以当即判定出生成图像为 fake；进而，生成器会倾向于尽量少生成 5 来躲避鉴别器的惩罚。如此以来，模型陷入了恶性循环，导致生成器更倾向于生成那些他已经能够具有很好生成效果的图像。进而导致了模型 collapse 到某几个，甚至某几个数字上。实际

上，同样有理由猜测，在 **Linear Interpolation** 部分图示中最后一行生成效果极差的数字即是模型试图生成的 5。

至于为什么是 5 无法得到良好的生成，我认为有两方面原因。一方面，5 这个数字需要两笔画才能写完，相对较为复杂（同样需要两笔画的 4 的生成效果也不尽如人意）；另一方面，GAN 的初始化对于模型的生成效果影响非常显著，这样可能影响了生成器无法良好驾驭的生成数字。

MLP GAN

按照如下结构，我试图直接改写原始的生成器与辨别器以使用 MLP 来实现 GAN 的生成器与辨别器：

```
1 class Generator_MLP(nn.Module):
2     def __init__(self, num_channels, latent_dim, hidden_dim):
3         super(Generator_MLP, self).__init__()
4         self.num_channels = num_channels
5         self.hidden_dim = hidden_dim
6         self.latent_dim = latent_dim
7         self.decoder = nn.Sequential(
8             nn.Linear(latent_dim, 4 * hidden_dim),
9             nn.BatchNorm1d(4 * hidden_dim),
10            nn.ReLU(),
11            nn.Linear(4 * hidden_dim, 2 * hidden_dim),
12            nn.BatchNorm1d(2 * hidden_dim),
13            nn.ReLU(),
14            nn.Linear(2 * hidden_dim, hidden_dim),
15            nn.BatchNorm1d(hidden_dim),
16            nn.ReLU(),
17            nn.Linear(hidden_dim, num_channels * 32 * 32),
18            nn.Tanh(),
19        )
20
21 class Discriminator_MLP(nn.Module):
22     def __init__(self, num_channels, hidden_dim):
```

```
23     super(Discriminator_MLP, self).__init__()
24     self.num_channels = num_channels
25     self.hidden_dim = hidden_dim
26     self.clf = nn.Sequential(
27         nn.Linear(num_channels * 32 * 32, hidden_dim),
28         nn.LeakyReLU(0.2, inplace=True),
29         nn.Linear(hidden_dim, 2 * hidden_dim),
30         nn.Dropout(0.2),
31         nn.LeakyReLU(0.2, inplace=True),
32         nn.Linear(2 * hidden_dim, 4 * hidden_dim),
33         nn.Dropout(0.2),
34         nn.LeakyReLU(0.2, inplace=True),
35         nn.Linear(4 * hidden_dim, 1),
36         nn.Sigmoid(),
37     )
```

如此一来，我最大程度地保留了 CNN 的网络架构，便于直接比较。并且，如我的实验设定所示，所有的实验参数我都将 MLP 与 CNN 设定的相同。如此以来，得到的 MLP 结果相对更加不如人意。

我选择 FID 最低的模型 MLP_16_100_100 作为展示；然而其 FID 仍然高达 163。



为了进行对比，额外展示 CNN_16_100_100 模型的效果：



可以看到同样的设定并且训练相同的 steps，MLP 的训练结果远不如 CNN。MLP 的生成结果看上去有很多边缘噪点，此外，数字覆盖着各种噪点，诸如第四行倒数第三个 6 就已经完全碎裂成了若干个碎片。MLP 的 FID 远超同等条件下的 CNN 可想而知。

究其原因，我认为还是和图像生成任务本身有关系。不考虑视觉的先验经验时，CNN 仅仅是共享了参数的 MLP，计算复杂度更高，而参数更少，相对而言表达力似乎更低。然而，在视觉相关问题上，CNN 有着强大的视觉先验经验。在视觉任务上的效果远超 MLP。

实际上，CNN 模型模拟了人类的视觉直觉；人眼对于图像的理解便是从微观到全局，从高频到低频，且全过程保持着二维（乃至三维）信息的传导。通过从 low-level 到 high-level 的 feature 传递，CNN 高度利用了人眼对于图像的理解机制，可以充分利用图像上相近像素间的关系，而像素间的相互关系在 self attention 当中有着更深入的展现。MLP 将高维的图像展开为一维，导致其显式丢失了图像高维的相互信息。此外，CNN 模型通过参数共享和 pooling 机制，使其具有一定的平移不变性，也提高了 CNN 在视觉任务上的泛化能力。

总归，模型的表现有模型的容量和任务适应性共同决定。在视觉任务上，CNN 惊艳的效果众所周知，在本次实验中训练效果强于 MLP 不足为奇。