深度学习实践实验七

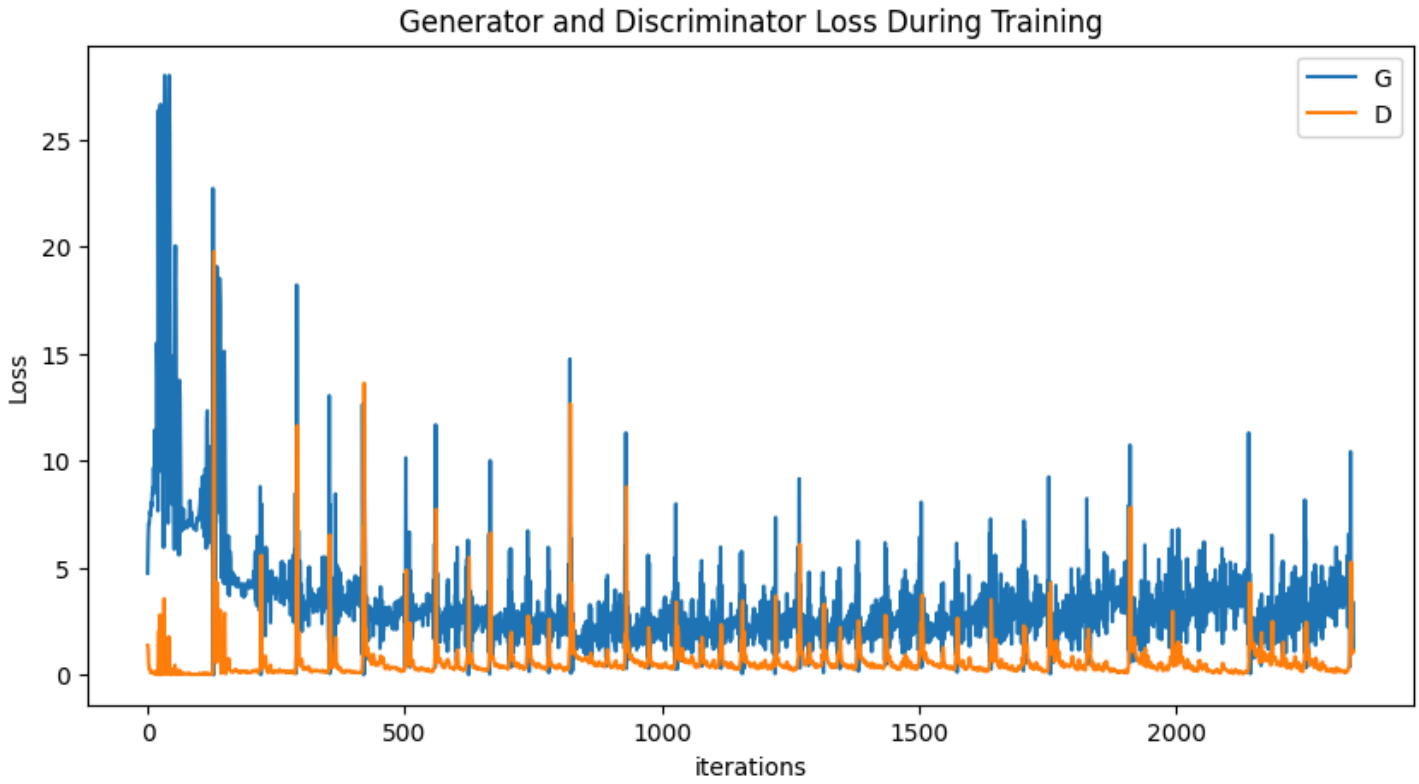
PB22151796-莫环欣

输入噪声的维度是多少?原始图像的维度是多少？生成图像的维度是多少?

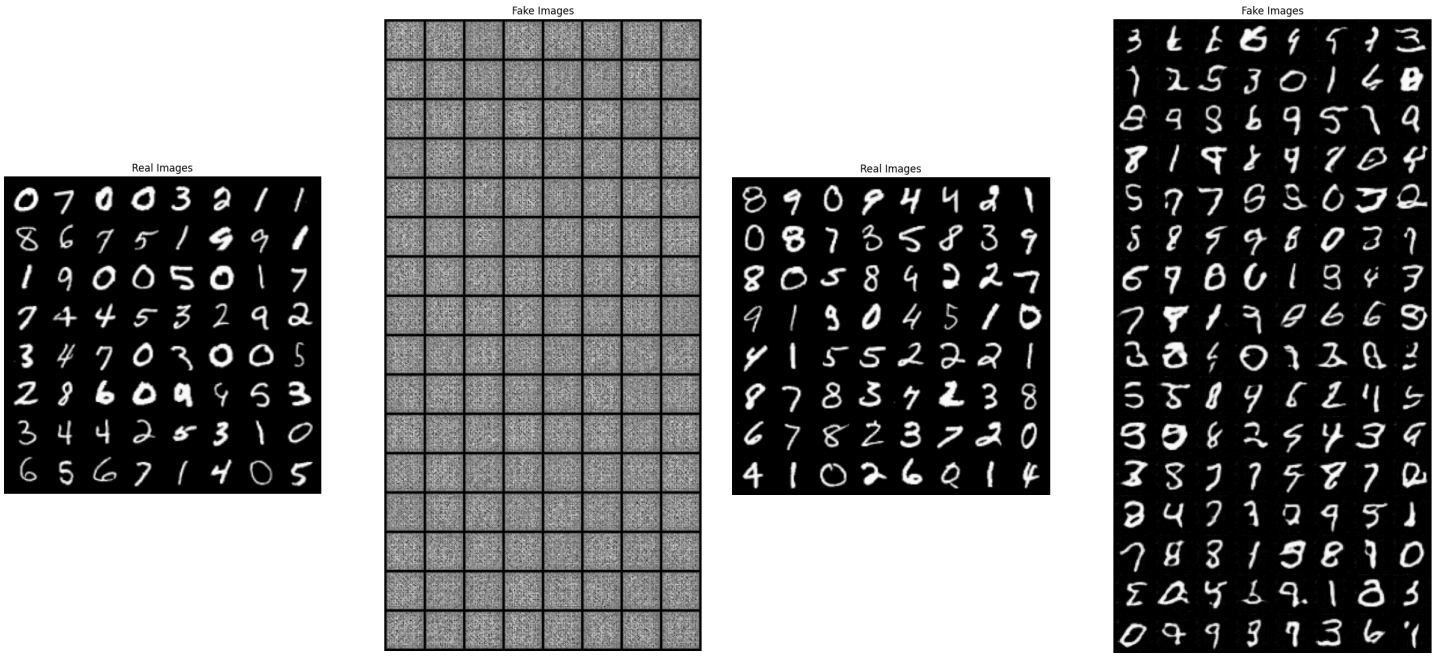
- 输入噪声的维度：
  - `Z_DIM = 100`
  - `(BATCH_SIZE, Z_DIM, 1, 1)`
- 原始图像的维度：
  - 原始图像为`MNIST`数据集，尺寸为`28x28`
- 生成图像的维度：
  - 在加载数据集时重新调整了维度
    - `transforms.Resize(X_DIM),`
    - `X_DIM = 64`
  - `IMAGE_CHANNEL = 1`
    - 单通道灰度图
  - 因此维度为`(BATCH_SIZE, IMAGE_CHANNEL, X_DIM, X_DIM)`

通过查看 img_list 观察生成效果是不是在不断地变好?

是的，并且从生成器与判别器的损失变化情况上也可以验证这一点，大致在`800`轮左右时损失就接近收敛了。



例如下面的两组图，一开始（左侧）生成的图片中全是噪点，最后（右侧）可以生成与原图极为相似的手写数字



如果我们想生成倒立的 MNIST 数字，怎么办？

可以直接在预处理阶段对`MNIST`全部垂直翻转即可，这样生成器就会模仿翻转后的`MNIST`

```
transforms.RandomVerticalFlip(p=1.0),
```

Real Images

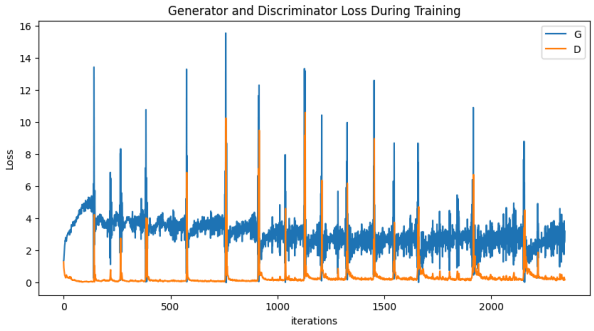
Fake Images

如果我们想直接生成 32*32 的图像，那么代码要如何修改?

首先修改`X_DIM`为`32`，随后在生成器中减少一个上采样层（`ConvTranspose2d`），并对应的在判别器中减少一个卷积层

同时还需要降低学习率，否则损失无法收敛（图片也多是噪点）

```
optimizerD = optim.Adam(netD.parameters(), lr=0.0001, betas=(0.5, 0.999))
optimizerG = optim.Adam(netG.parameters(), lr=0.0001, betas=(0.5, 0.999))
```


Generator and Discriminator Loss During Training
G
D
Loss
iterations
0
500
1000
1500
2000
0
2
4
6
8
10
12
14
16


Real Images


Fake Images

```
dataset = dset.MNIST(
    root=DATA_PATH,
    download=True,
    transform=transforms.Compose(
        [
            transforms.Resize(32),  # 调整为 32x32
            transforms.ToTensor(),
            transforms.Normalize((0.5,), (0.5,)),
        ]
    ),
)
dataloader = torch.utils.data.DataLoader(
    dataset, batch_size=BATCH_SIZE, shuffle=True, num_workers=2
)
class Generator32(nn.Module):
    def __init__(self):
        super(Generator32, self).__init__()
        self.main = nn.Sequential(
            nn.ConvTranspose2d(Z_DIM, G_HIDDEN * 4, 4, 1, 0, bias=False),
            nn.BatchNorm2d(G_HIDDEN * 4),
            nn.ReLU(True),
            nn.ConvTranspose2d(G_HIDDEN * 4, G_HIDDEN * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(G_HIDDEN * 2),
            nn.ReLU(True),
            nn.ConvTranspose2d(G_HIDDEN * 2, G_HIDDEN, 4, 2, 1, bias=False),
            nn.BatchNorm2d(G_HIDDEN),
            nn.ReLU(True),
            nn.ConvTranspose2d(G_HIDDEN, IMAGE_CHANNEL, 4, 2, 1, bias=False),
            nn.Tanh(),
        )

    def forward(self, input):
        return self.main(input)

class Discriminator32(nn.Module):
    def __init__(self):
        super(Discriminator32, self).__init__()
        self.main = nn.Sequential(
            nn.Conv2d(IMAGE_CHANNEL, D_HIDDEN, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(D_HIDDEN, D_HIDDEN * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(D_HIDDEN * 2),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(D_HIDDEN * 2, D_HIDDEN * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(D_HIDDEN * 4),
            nn.LeakyReLU(0.2, inplace=True),
```

```
            nn.Conv2d(D_HIDDEN * 4, 1, 4, 1, 0, bias=False),
            nn.Sigmoid(),
        )

    def forward(self, input):
        return self.main(input).view(-1, 1).squeeze(1)
```