2024-3-25 A brief summary of basic GAN

1. The Theory of GAN

参考文献: Generative Adversarial Nets. Generative Adversarial Nets - new.pdf

总结: GAN_notes.pdf

以及参考了一些网上的实现代码

2. Model Setting

数据集：torchvision.dataset.MINIST，transform 做标准化后，加载到 dataloader

优化器：Adam (lr = 0.0003)

损失函数：BCELoss()

训练：D 和 G 训练次数比为 1:1

batch_size = 64

G 的输入噪声维度为 100

2.1 Generator

一共三个 Generator：

1. 简单的 Linear 层 —— G_1；
2. 稍微复杂点的 Linear 层 —— G_2；
3. Conv 上采样层 —— G_3。

2.2 Discriminator

两个 Discriminator

1. 简单的 Linear 层 —— D_1；
2. 复杂点的 Conv 层 —— D_2。

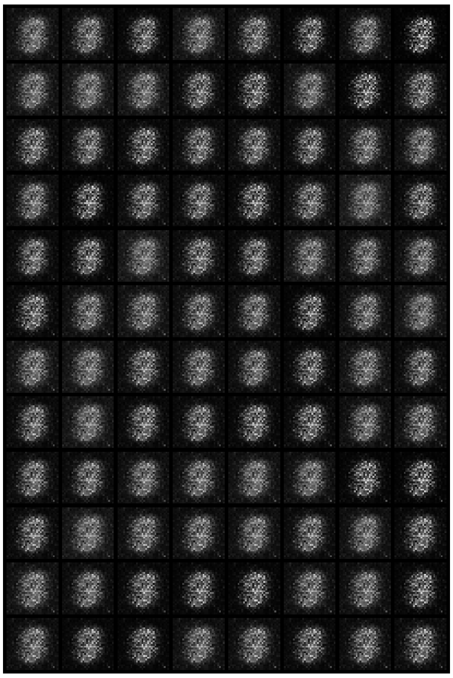
2.3 GAN Structure

1. $G_1 + D_1$：这个模型比较简单，生成器和鉴别器都是 3 层线性层 + LeakyReLU 激活函数，训练很快

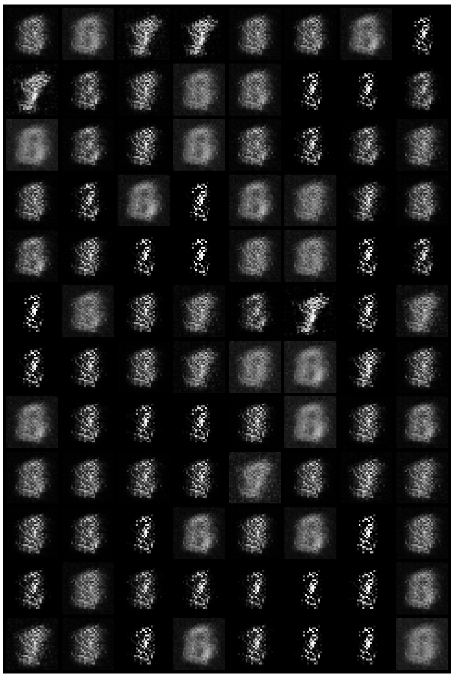
训练 100 epoch，训练结果还可以，大概从 30 个 epoch 后逐渐成型，在后期（80 之后）的效果大概定型，也具有多样性，没有只生成一个数字。

```python
# Discriminator
class discriminator(nn.Module):
    def __init__(self):
        super(discriminator, self).__init__()
        self.dis = nn.Sequential(
            nn.Linear(784, 256),
            nn.LeakyReLU(0.2),
            nn.Linear(256, 256),
            nn.LeakyReLU(0.2),
            nn.Linear(256, 1),
            nn.Sigmoid())

    def forward(self, x):
        x = self.dis(x).squeeze()
        return x
```

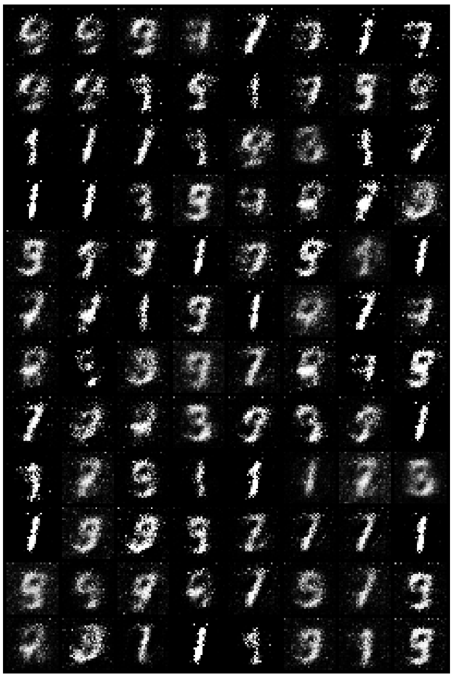
```python
# Generator
class generator(nn.Module):
    def __init__(self):
        super(generator, self).__init__()
        self.gen = nn.Sequential(
            nn.Linear(100, 256),
            nn.ReLU(True),
            nn.Linear(256, 256),
            nn.ReLU(True),
            nn.Linear(256, 784),
            nn.Tanh())

    def forward(self, x):
        x = self.gen(x)
        return x
```



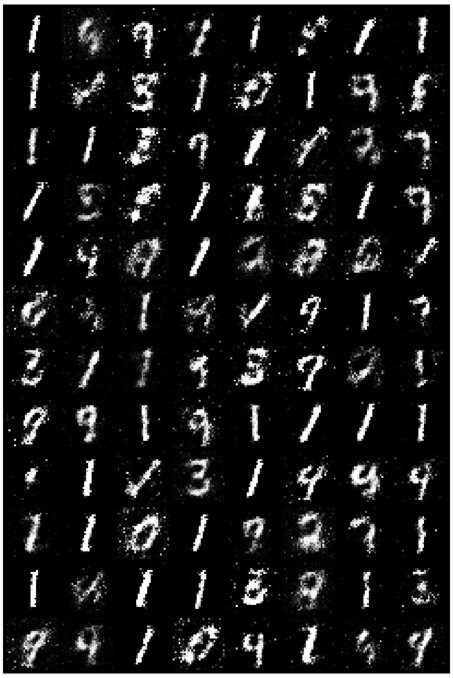
Epoch 5



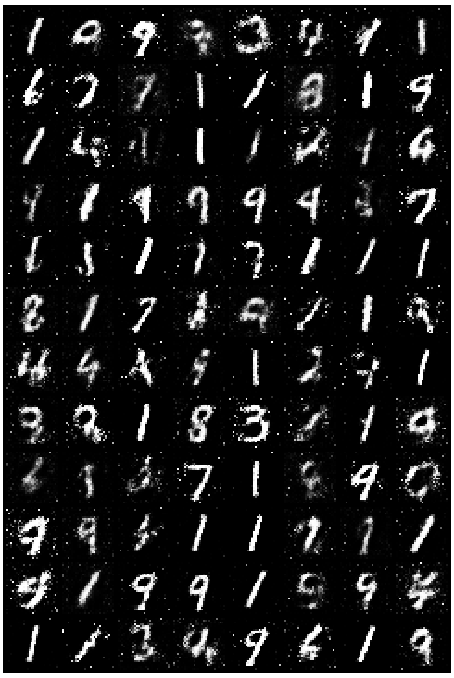
Epoch 15



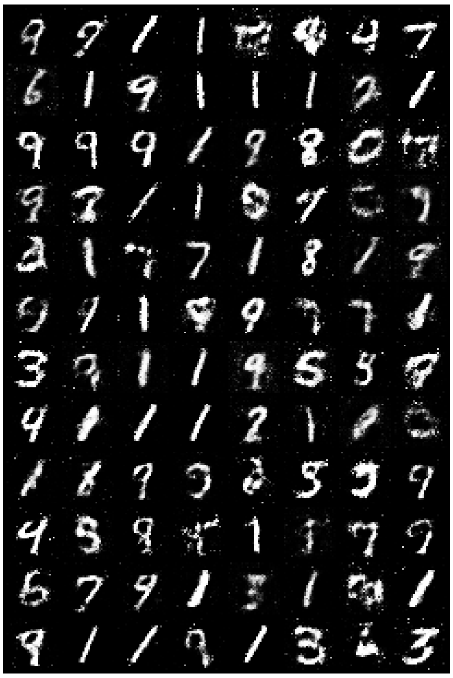
Epoch 30



Epoch 50



Epoch 80



Epoch 100

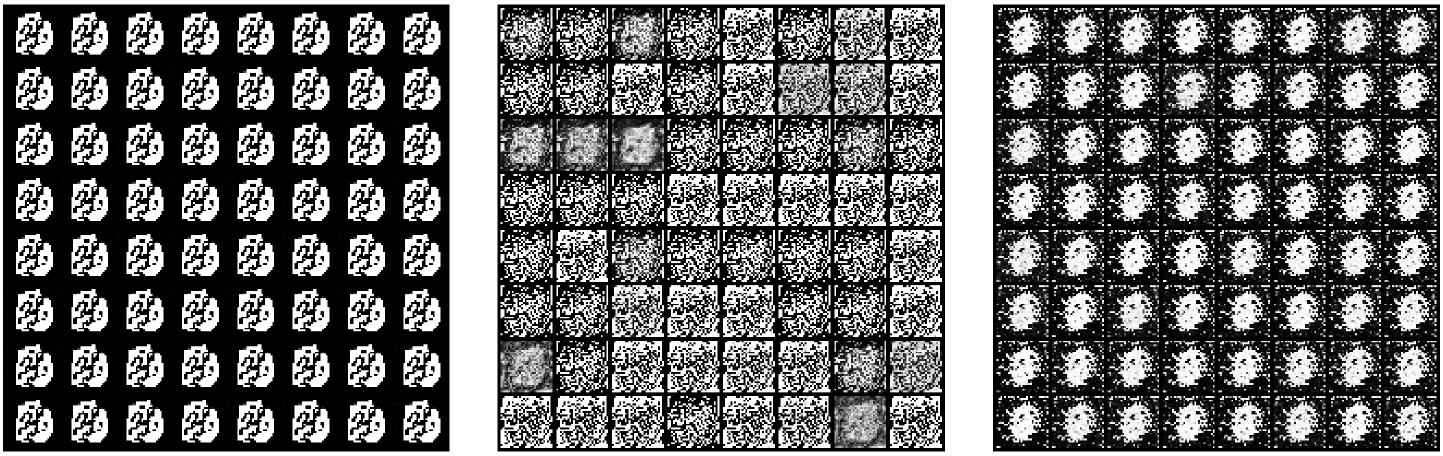
2. $G_2 + D_2$：这个模型的生成器是稍微复杂点的 Linear 层，鉴别器用了 Conv 层，猜测可能是 D 的识别能力经过简单训练就可以非常强大，但生成器太简单，而且因为前期训练改进效果不明显，导致两个 net 不能良好对抗，batchnormal 和 dropout 等都尝试过，会让训练过程表现更差。

训练效果不好，在经过 3~4 个 epoch 后，D 的 loss 会趋近于 0，G 的 loss 会从15 左右突然上升到 40~100 左右，可能是 D 的训练效果太好了？很难产生对抗性，最后生成的图片看起来是纯随机的。

```
class Generator_Linear(nn.Module):
    # based on linear layers
    def __init__(self):
        super(Generator_Linear, self).__init__()
        self.gen = nn.Sequential(
            # 256 is the input size of the generator
            nn.Linear(100, 256),
            # nn.BatchNorm1d(256),
            # dropout layer, 0.3 is the probability of an element to be zeroed
            # nn.Dropout(0.3),
            nn.LeakyReLU(True),

            nn.Linear(256, 512),
            # nn.BatchNorm1d(512),
            # nn.Dropout(0.4),
            nn.LeakyReLU(True),

            nn.Linear(512, 1024),
            # nn.BatchNorm1d(1024),
            # nn.Dropout(0.5),
            nn.LeakyReLU(True),

            # 784 is the output size of the generator, because the image's size is 28 * 28
            nn.Linear(1024, 784),

            # tanh layer, the output of the generator is in the range of [-1, 1]
            nn.Tanh()
        )

    def forward(self, image_noise):
        # image_noise's shape is (batch_size, 256)
        output = self.gen(image_noise)

        # output's shape is (batch_size, 784)
        # reshape the output to (batch_size, 1, 28, 28)
        output = output.view(-1, 1, 28, 28)
        return output
```

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

        self.features = nn.Sequential(
            # 2-d convolutional layer, input channels is 1 (grayscale images), output channels is 32, kernel size is 3
            nn.Conv2d(1, 32, kernel_size=3),

            # batch normalization, 32 is the number of last output channels
            # nn.BatchNorm2d(32),

            # Leaky ReLU, 0.2 is the negative values' slope.
            nn.LeakyReLU(0.2),

            # 2-d convolutional layer, input channels is 32, output channels is 64
            nn.Conv2d(32, 64, kernel_size=3),
            # nn.BatchNorm2d(64),
            nn.LeakyReLU(0.2),
        )

        self.classifier = nn.Sequential(
            nn.Linear(64*24*24, 1024),
            nn.LeakyReLU(0.2),

            nn.Linear(1024, 512),
            nn.LeakyReLU(0.2),

            nn.Linear(512, 1),
            nn.Sigmoid()
        )

    def forward(self, image):
        # image's shape is (batch_size, 1, 28, 28)
        # get images' features
        features = self.features(image)

        # features' shape is (batch_size, 64, 24, 24)
        # reshape features to (batch_size, 64 * 24 * 24)
        features = features.view(features.shape[0], -1)

        # get output
        output = self.classifier(features).squeeze()
        return output
```

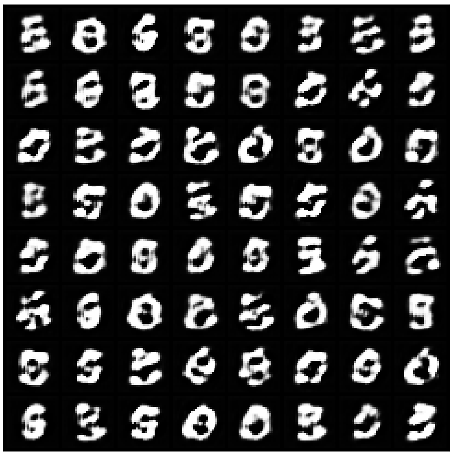
几次不同尝试下前期的训练结果：



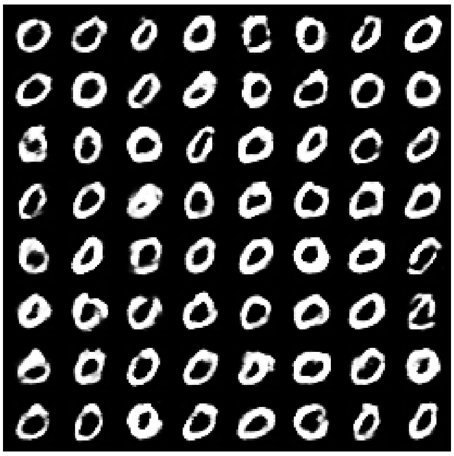
3. $G_3 + D_2$：这里模型使用 Conv 上采样做生成器和使用 Conv 做鉴别器，鉴别器和第二个模型一样。

训练效果在前两个网络之间，比第一个差，比第二个好，因为这个生成器也比较复杂，所以和鉴别器之间能形成比较良好的对抗，loss 的变化过程大概符合预期，而且只需要很少的 epoch 就能生成质量不错的图片，但是最后生成的图片全都是 0，可能是因为卷积网的设置问题？导致模型太关注细节？对这块还不熟悉所以不确定是什么问题。

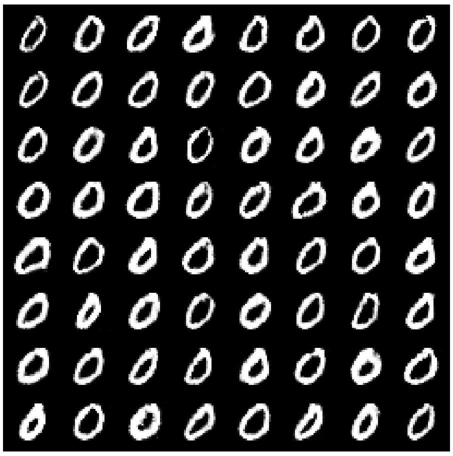
```
class Generator_Conv(nn.Module):
    # based on convolutional layers, up-sampling method is used
    def __init__(self):
        super(Generator_Conv, self).__init__()

        self.expand = nn.Sequential(
            # 256 is the input size of the generator
            nn.Linear(100, 256),
            # nn.BatchNorm1d(256),
            # nn.Dropout(0.3),
            nn.LeakyReLU(True),

            nn.Linear(256, 484),
            # nn.BatchNorm1d(484),
            # nn.Dropout(0.5),
            nn.LeakyReLU(True),
        )
        # the output size of self.expand is 484, because the image's size is 22 * 22
        # this size is set to match the up-sampling process that follows, which will increase the size to 28 * 28

        self.gen = nn.Sequential(
            # 2-d transpose convolutional layer, input channels is 1, output channels is 4, kernel size is 3
            nn.ConvTranspose2d(1, 4, kernel_size=3),
            # the output size of this layer is 22 -1 + 3 = 24 * 24
            nn.BatchNorm2d(4),
            nn.LeakyReLU(True),

            # 2-d transpose convolutional layer, input channels is 4, output channels is 8, kernel size is 3
            nn.ConvTranspose2d(4, 8, kernel_size=3),
            # the output size of this layer is 24 -1 + 3 = 26 * 26
            nn.BatchNorm2d(8),
            nn.LeakyReLU(True),

            # 2-d transpose convolutional layer, input channels is 8, output channels is 4, kernel size is 3
            nn.ConvTranspose2d(8, 4, kernel_size=3),
            # the output size of this layer is 26 -1 + 3 = 28 * 28
            nn.BatchNorm2d(4),
            nn.LeakyReLU(True),

            # 2-d transpose convolutional layer, input channels is 4, output channels is 1, kernel size is 1
            # this layer is used to reduce the number of channels to 1 to match greyscale images
            nn.ConvTranspose2d(4, 1, kernel_size=1),
            # the output size of this layer is 28 -1 + 1 = 28 * 28
            nn.BatchNorm2d(1),
            nn.LeakyReLU(True),

            # tanh layer, the output of the generator is in the range of [-1, 1]
            nn.Tanh()
        )

    def forward(self, image_noise):
        # image_noise's shape is (batch_size, 256)
        output = self.expand(image_noise)

        # output's shape is (batch_size, 484)
        # reshape the output to (batch_size, 1, 22, 22)
        output = output.view(-1, 1, 22, 22)

        # up-sampling the output to (batch_size, 1, 28, 28)
        output = self.gen(output)

        return output
```



Epoch 5



Epoch 10



Epoch 20

3. 改进思路

1. 再继续试试 Linear 层有没有什么改进，第一组最简单的模型组成的网络对抗性还可以？按同样的思路试试把第二个模型改一改看一下有没有提升？
2. 正在看卷积神经网络的部分，看完之后思考第三个模型是什么问题，结合之前看到的网上的介绍，可能是模型太关注细节？
3. 总的来看原始的 GAN 还是很容易崩溃的，有点难协调两个网络的训练过程，后面想再尝试一下 其他的训练方式，比如 D 和 G 用 k:1 的训练比，G 前期用另一个损失函数等