南开大学
Nankai University

南　　开　　大　　学

计　算　机　学　院

深度学习与应用实验报告

GAN 生成对抗网络

艾明旭　2111033

年级：2021 级

专业：信息安全

指导教师：侯淇彬

2024 年 6 月 21 日

目录

一、 基础 GAN 结构

生成式对抗网络是一种深度学习模型，是近年来复杂分布上无监督学习最具前景的方法之一。模型通过框架中 (至少) 两个模块: 生成模型和判别模型的互相博弈学习产生相当好的输出。原始 GAN 理论中，并不要求 G 和 D 都是神经网络，只需要是能拟合相应生成和判别的函数即可。但实用中一般均使用深度神经网络作为 G 和 D。一个优秀的 GAN 应用需要有良好的训练方法，否则可能由于神经网络模型的自由性而导致输出不理想。

这是一个生成器和判别器博弈的过程。生成器生成假数据，然后将生成的假数据和真数据都输入判别器，判别器要判断出哪些是真的哪些是假的。判别器第一次判别出来的肯定有很大的误差，然后我们根据误差来优化判别器。现在判别器水平提高了，生成器生成的数据很难再骗过判别器了，所以我们得反过来优化生成器，之后生成器水平提高了，然后反过来继续训练判别器，判别器水平又提高了，再反过来训练生成器，就这样循环往复，直到达到纳什均衡。

训练时先训练判别器：将训练集数据打上真标签和生成器生成的假图片打上假标签（0）一同组成 batch 送入判别器，对判别器进行训练。计算 loss 时使判别器对真数据输入的判别趋近于真（1），对生成器生成的假图片的判别趋近于假（0）。此过程中只更新判别器的参数，不更新生成器的参数。

然后再训练生成器：将高斯分布的噪声 z 送入生成器，然后将生成器生成的假图片打上真标签（1）送入判别器。计算 loss 时使判别器对生成器生成的假图片的判别趋近于真（1）。此过程中只更新生成器的参数，不更新判别器的参数。

(一) 基础 GAN 的网络结构

```
Discriminator(
  (fc1): Linear(in_features=784, out_features=128, bias=True)
  (nonlin1): LeakyReLU(negative_slope=0.2)
  (fc2): Linear(in_features=128, out_features=1, bias=True)
)
Generator(
  (fc1): Linear(in_features=100, out_features=128, bias=True)
  (nonlin1): LeakyReLU(negative_slope=0.2)
  (fc2): Linear(in_features=128, out_features=784, bias=True)
)
```

构建 GAN 模型的基本逻辑: 现实问题需求 → 建立实现功能的 GAN 框架 (编程)→ 训练 GAN(生成网络、对抗网络)→ 成熟的 GAN 模型 → 应用

生成对抗网络（GANs）由 2 个重要的部分构成：

- 生成器 (Generator)：通过机器生成数据（大部分情况下是图像），目的是“骗过”判别器
- 判别器 (Discriminator)：判断这张图像是真实的还是机器生成的，目的是找出生成器做的“假数据”

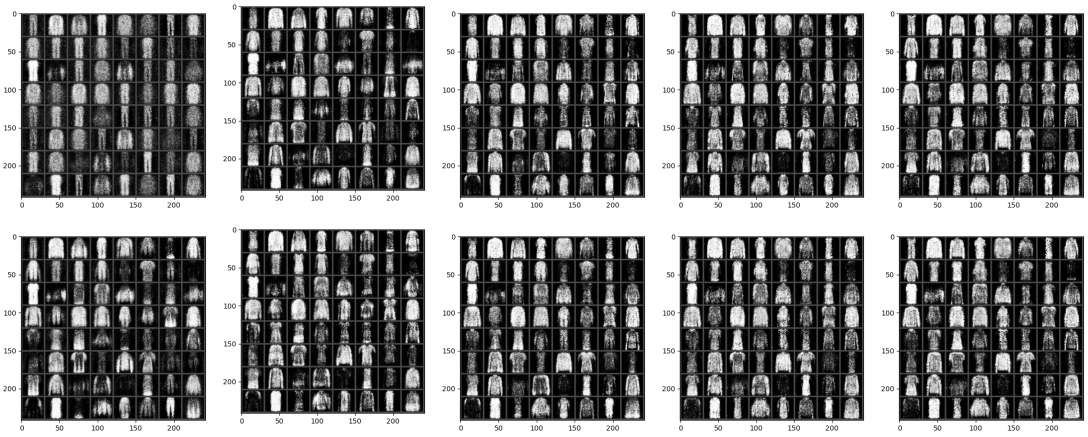

Image Image Image Image Image

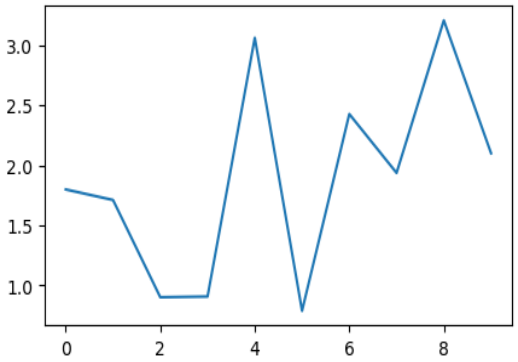

图 1: 基础 LSTM validation loss

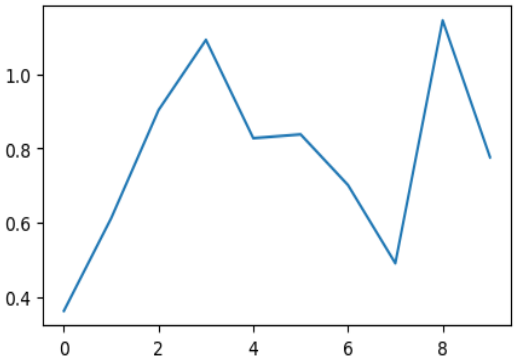

图 2: 基础 LSTM validation accuracy

(二) 图形展示实验结果

二、 Randomlist

(一) 图形结构

这个结构对判别器进行了大量的优化, 其中很多部分需要我们变换网络结构以及层数的加深

```
Discriminator(
(model): Sequential(
  (0): Conv2d(1, 16, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
  (1): LeakyReLU(negative_slope=0.2, inplace=True)
  (2): Dropout2d(p=0.25, inplace=False)
  (3): Conv2d(16, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
  (4): LeakyReLU(negative_slope=0.2, inplace=True)
  (5): Dropout2d(p=0.25, inplace=False)
  (6): BatchNorm2d(32, eps=0.8, momentum=0.1, affine=True,
      track_running_stats=True)
  (7): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
  (8): LeakyReLU(negative_slope=0.2, inplace=True)
  (9): Dropout2d(p=0.25, inplace=False)
```

```
    (10): BatchNorm2d(64, eps=0.8, momentum=0.1, affine=True,
        track_running_stats=True)
    (11): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (12): LeakyReLU(negative_slope=0.2, inplace=True)
    (13): Dropout2d(p=0.25, inplace=False)
    (14): BatchNorm2d(128, eps=0.8, momentum=0.1, affine=True,
        track_running_stats=True)
  )
  (adv_layer): Sequential(
    (0): Linear(in_features=512, out_features=1, bias=True)
    (1): Sigmoid()
  )
)
Generator(
...
    (9): Conv2d(64, 1, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (10): Tanh()
  )
)
```

(二) 结果图像展示

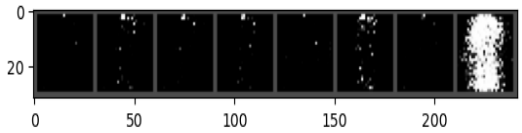

图 3: gan 迭代过程

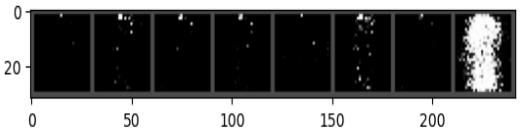

图 4: gan 迭代过程

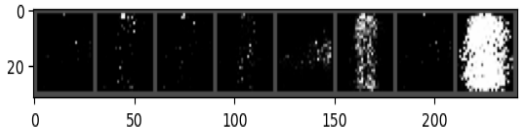

图 5: gan 迭代过程

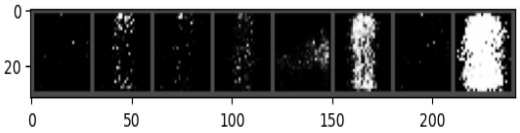

图 6: gan 迭代过程

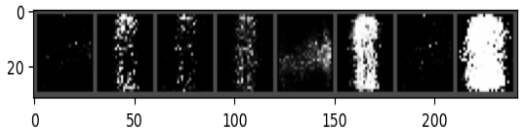

图 7: gan 迭代过程

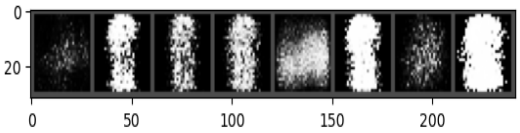

图 8: gan 迭代过程

我们可以看到，过大的噪声值会使得生成的图像中存在更多的噪声和不规则性，从而使图像变得更加模糊。生成的图像可能会失去一些细节和质感。太大的噪声值可能会使整个图像更加“扁平”，失去细节和质感。这是因为过大的噪声值会使得生成的图像过多地关注“特别嘈杂”的区域，从而影响了整体的质感和细节表现。

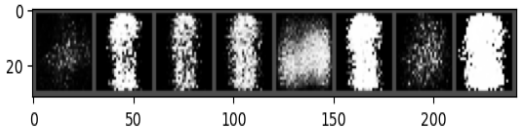

图 9: gan 迭代过程

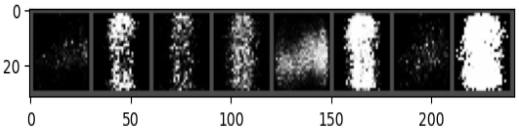

图 10: gan 迭代过程

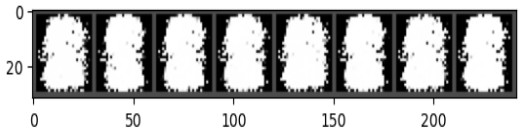

图 11: gan 迭代过程

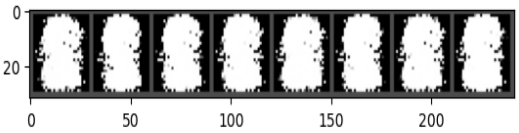

图 12: gan 迭代过程

对于较小的噪声来说，也不一定就完全是对的。较小的噪声会导致生成器干扰减少，能迅速拟合出图像，但此时图像多样性会减小，比如“衣服”的“领子”“袖”等位置有一些细节其实在很小的噪声时会消失。因此需要根据应用场景和需求权衡选择合适的噪声大小。关键是要保证噪声的分布合理，比如与原来的数据保持一致，就是一种合理的分布。

图 13: 生成对抗网络的预测结果

可以看到，在引入时间长短因子之后，整个深度学习的网络层次都发生了一定的变化。

三、 带有 Depthwise conv 的 MobileNets

Depthwise Convolution: 这一步中，每个输入通道分别进行卷积操作，而不是像传统卷积那样对所有通道进行卷积。这大大减少了计算量。

(一) 自己编写的 GAN

这里我自己定义了一个 GAN 方法，网络层结果大致如下。

```
Generator(
  (main): Sequential(
    (0): ConvTranspose2d(100, 256, kernel_size=(4, 4), stride=(1, 1))
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): ConvTranspose2d(256, 128, kernel_size=(3, 3), stride=(2, 2), padding
        =(1, 1))
    (4): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
    (5): ReLU(inplace=True)
    (6): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding
        =(1, 1))
    (7): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
    (8): ReLU(inplace=True)
    (9): ConvTranspose2d(64, 1, kernel_size=(4, 4), stride=(2, 2), padding
        =(1, 1))
    (10): Tanh()
  )
)
Discriminator(
  (main): Sequential(
    (0): Conv2d(1, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): LeakyReLU(negative_slope=0.2, inplace=True)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
    (4): LeakyReLU(negative_slope=0.2, inplace=True)
    (5): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
    (7): LeakyReLU(negative_slope=0.2, inplace=True)
    (8): Conv2d(256, 1, kernel_size=(4, 4), stride=(1, 1))
    (9): Sigmoid()
  )
)
```

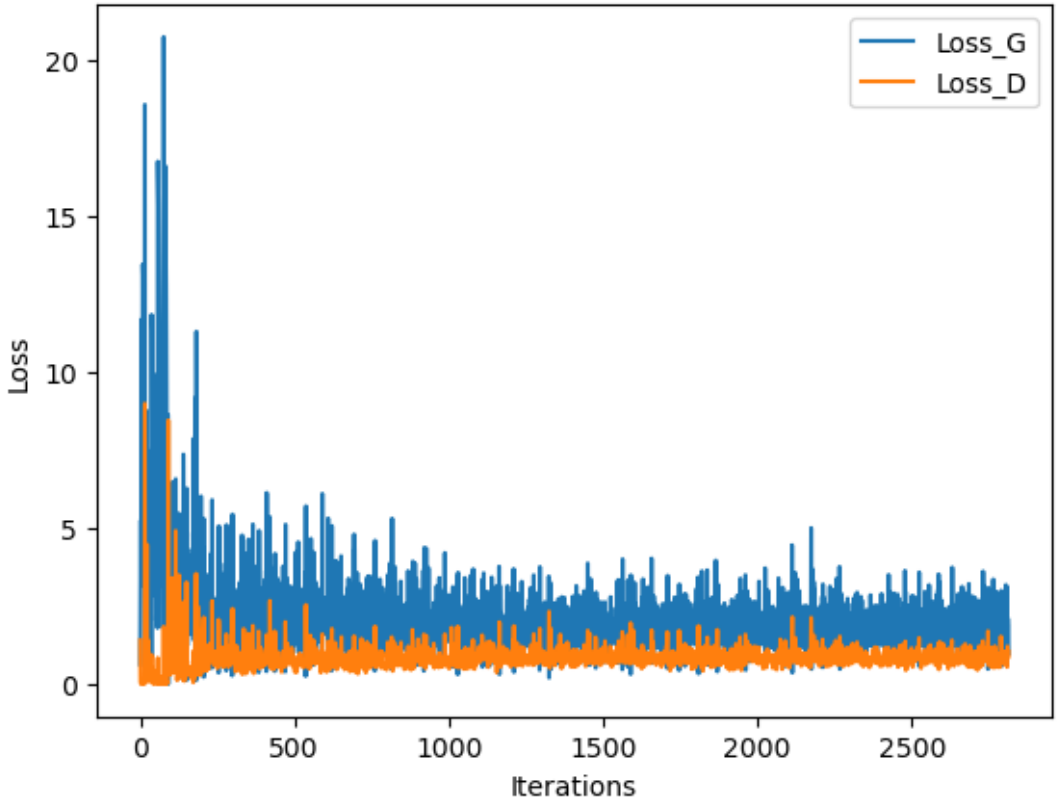

图 14: 生成对抗网络的预测结果

(二) 结果图形展示

可以看到损失整体上是下降并且接近相同的趋势。数据在快速收敛过后逐渐趋向于稳定。

四、 总结

本次实验主要复现了两种生成对抗网络，并进行了自主编写，生成对抗网络是重要的深度学习分支，在编写的过程当中我感到受益匪浅，希望未来学习的道路更加有信心，学习的更多有用的深度学习知识。