Image Generation Using Generative Adversarial Networks

Student ID:230161210

Student name: Qing Zhao

1. Generative Adversarial Networks Review

A generative adversarial network (GAN) is a deep learning architecture. It trains two neural networks to compete against each other to generate more authentic new data from a given training dataset.A GAN is called adversarial because it trains two different networks (Generator and Discriminator) and pits them against each other. Generator Network aims to create sythetic data samples. It takes a random noise vector as input and transforms it into synthetic data .Discriminator Network acts as a critic, trying to distinguish between real data samples from the training dataset and the generated samples produced by the generator.

Training a GAN model involves an iterative competition between the generator and discriminator. At first,The generator initially creates random data samples.Then The discriminator receives both real data and the generated samples. It tries to accurately classify them as either real or fake.in this phase, the discrimination is being trained. Then in the second phase, based on the discriminator's feedback, the generator updates its weights to improve its ability to generate data to fool the discriminator, during this phase, the generator is being trained and the discriminator is not trainable. By integrating these two phases, the generator becomes better at creating synthetic data until it is hard for the discriminator to distinguish real or fake.

GANs are popular in video generation, image generation, and data augementation domain. Although GANs are powerful generative tools, it is a tough task to train a GAN model. Mode collaspe and how to balance the performance of generator and discriminator are two typical challenges.

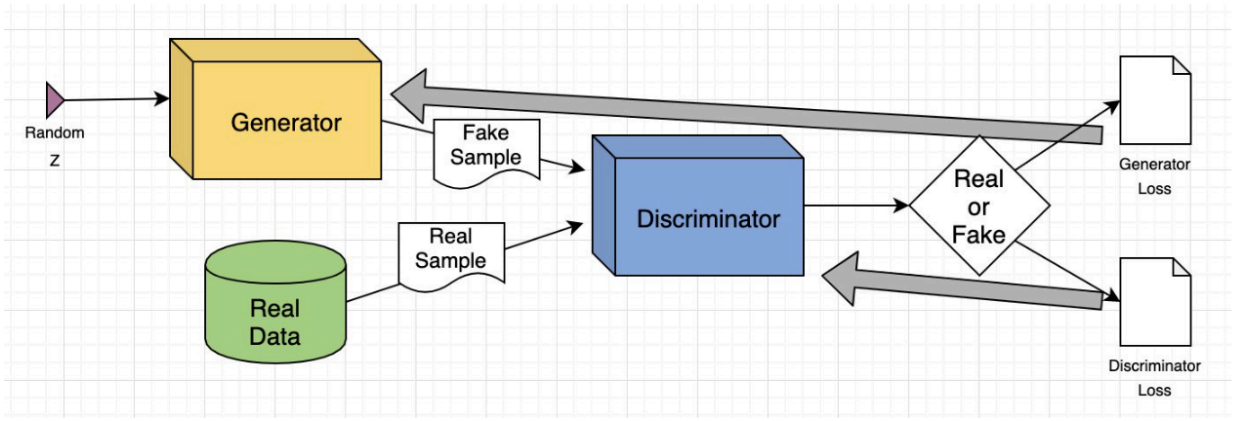

Figure 1: the architecture of GANs

There are several types of GANs tailor to different kinds of problems, such as Vanilla GAN, Conditional GAN, Deep Convolutional GAN, StyleGAN and so on.

2. Problems Formulation

In the experiment, I will use Deep Convolutional GAN(DCGAN) to address the image generation task. Using google colab built-in sample dataset “mnist_train_small” (a part of the famous MNIST dataset) as the training dataset. By comparing the images generated by the initial generator without training with images generated by the generator after 20 epochs of training, to demonstrate the mystery of GAN model and try to avoid Mode collapse.

Deep convolutional GAN (DCGAN) integrates CNN architectures into GANs.With DCGAN, the generator uses transposed convolutions to upscale data distribution, and the discriminator also uses convolutional layers to classify data.

3. Build a DCGAN model

3.1 Generator

The generator takes in random noise and uses it to create fake images.In this experiment, I use the 100-dimension vector random noise as the input for the generator, after flowing through the network, the generator turn this noise vector into an image with the same dimensions of the MNIST dataset.

Guided by the best practice in DCGANs, to design the architecture of the generator, I use batch normalization to contribute to stabilize the training process and choose "`selu`" as the activation for the intermediate deconvolution while "`tanh`" for the output.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 6272)              633472

 reshape (Reshape)           (None, 7, 7, 128)         0

 batch_normalization (Batch  (None, 7, 7, 128)         512
 Normalization)

 conv2d_transpose (Conv2DTr  (None, 14, 14, 64)        204864
 anspose)

 batch_normalization_1 (Bat  (None, 14, 14, 64)        256
 chNormalization)

 conv2d_transpose_1 (Conv2D  (None, 28, 28, 1)         1601
 Transpose)

=================================================================
Total params: 840705 (3.21 MB)
Trainable params: 840321 (3.21 MB)
Non-trainable params: 384 (1.50 KB)
_________________________________________________________________
```

Figure2 : the architecture of the generator

3.2 Discriminator

The discriminator takes in both fake and real images as input and determines if it is fake or not. Thus, the input shape will be that of the training images.In the design of the discriminator, I use strided convolutions to reduce the dimensionality of the input images. Instead of "selu", "LeakyRELU" is chosen as the activation, aiming to avoid mode collapse. Since the output of the discriminator is a binary class, the last layer is a dense layer with "sigmoid" as the activation. Additionally, I add two dropout layers to reduce the risk of overfitting.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 14, 14, 64)	1664
dropout (Dropout)	(None, 14, 14, 64)	0
conv2d_1 (Conv2D)	(None, 7, 7, 128)	204928
dropout_1 (Dropout)	(None, 7, 7, 128)	0
flatten (Flatten)	(None, 6272)	0
dense_1 (Dense)	(None, 1)	6273

Total params: 212865 (831.50 KB)
Trainable params: 212865 (831.50 KB)
Non-trainable params: 0 (0.00 Byte)

Figure 3: the architecture of the discriminator

4. Training DCGAN and Image Generation

Assembling the generator network and discriminator network together, we get the DCGAN model. The following step is to train the DCGAN model.the training loop consists of two phases: the first one to train the discriminator to distinguish between fake or real data, while the second one to train the generator to generate images that could trick the discriminator. In the second phase, the weights of the discriminator can be changed.

Figure 4 shows the initial output from the untrained generator, the generator just turned the random noise vector into an image with the same dimensions of the MNIST dataset, but nothing similar with the image in the MNIST dataset. However, after 20 epochs of training, images (figure 5) generated by the generator are totally different and close to the training images(figure 6).

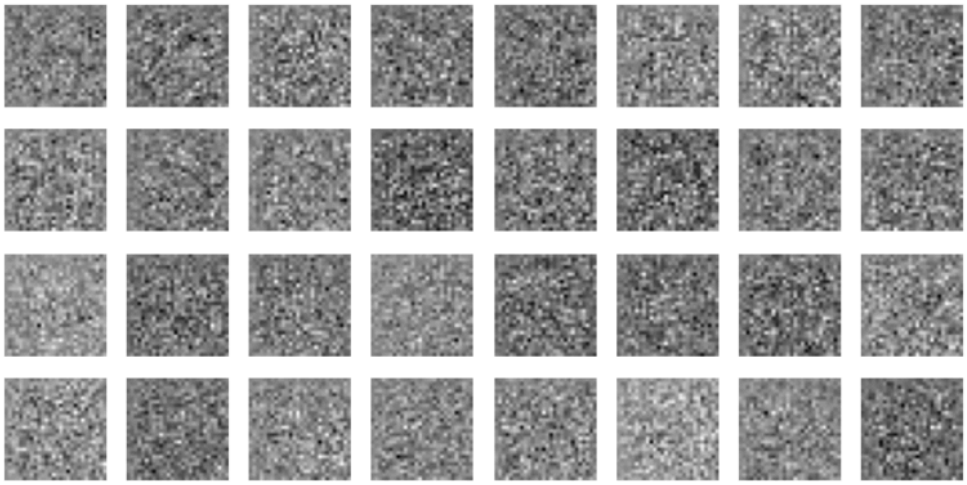

Figure 4: images generated by untrained generator

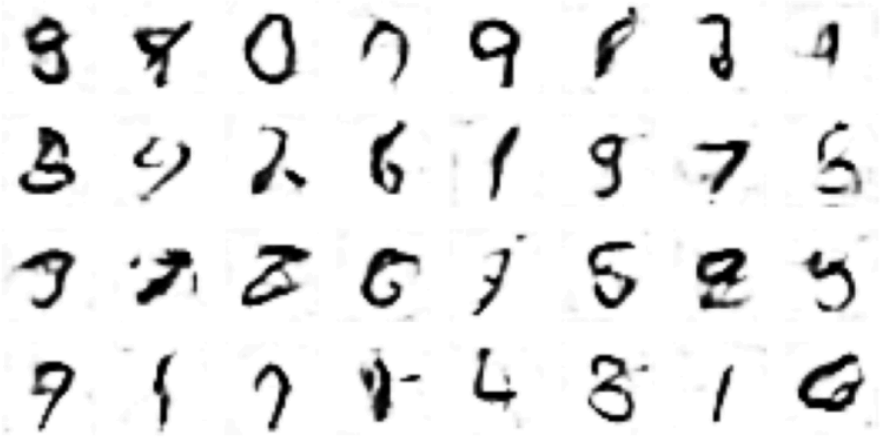

Figure 5: images generated by generator trained after 20 epochs

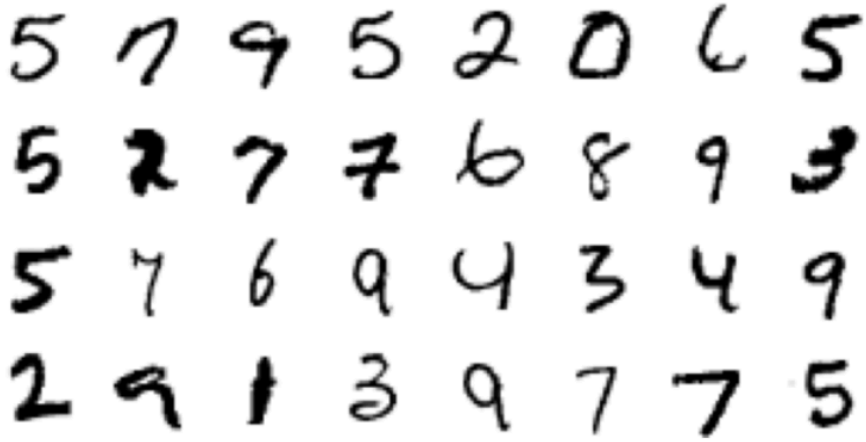

Figure 6: images in training dataset

After 20 epochs of training, the loss of the discriminator is 1.41 and the loss of the generator is 0.53. I tried to increase the training epochs to 100, in that case, although the loss of the generator keeps on decreasing, the loss of the discriminator goes in the opposite direction. Meanwhile, the images generated by the generate occured getting stuck producing a limited set of outputs(1,5,7) instead of exploring the full range of possibilities(0-9), indicating that mode collapse occurred.

References

1. course video:MIT 6.S191: Deep generative modeling
2. course video:Generative Adversarial Networks (GANs) Specialization
3. Course video:Generative Deep Learning with TensorFlow
4. textbook:Deep Learning by Ian Goodfellow, Yoshua Bengio, Aaron Courville
5. paper:Generative Adversarial Nets by Ian J. Goodfellow, Jean Pouget-Abadie , Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair , Aaron Courville, Yoshua Bengio
6. website:https://aws.amazon.com/what-is/gan/