DL HW6: InfoGAN

Ting-Hsuan, Wang (0756045)

May 22, 2019

1 Introduction

A basic GAN comprises a generator and a discriminator. Generator's duty is to generate images that are as real as real images from a random noise, while discriminator's responsibility is to judges the whether the input image is a real image or not. However, GAN does not put constraints on how generator should utilize the noise. In other words, generator may use the noise in a way such that all dimensions in the noise are highly entangled. Therefore, individual dimensions of noise may not correspond to semantic features of the data.

To solve this, a group of engineers proposed InfoGAN in 2016. InfoGAN is aimed to

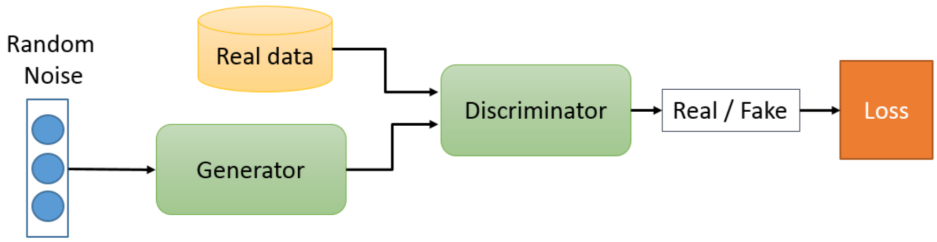

Figure 1: Network architecture of InfoGAN.

decompose the input noise vector into (1) incompressible noise and (2) latent code which can represent semantic features. They add a subneural network Q to distinguish generator's output of semantic features. In this lab, we're going to implement a simplified version of InfoGAN which supports only discrete latent variables.

2 Experiment setups

2.1 How you implement InfoGAN

2.1.1 Adversarial loss

The loss of generator and discriminator of InfoGAN looks like this:

$$\begin{aligned} L_D &= -E_{x\sim p_r}[logD(x)] - E_{x\sim p_g}[log(1-D(x))] \\ L_G &= E_{x\sim p_g}[log(1-D(x))] + L_l(G,Q) \end{aligned} \tag{1}$$

By excluding the lower bound mutual information term, we get the exactly same loss function as the original GAN:

$$\begin{aligned} L_D &= -E_{x\sim p_r}[logD(x)] - E_{x\sim p_g}[log(1-D(x))] \\ L_G &= E_{x\sim p_g}[log(1-D(x))] \end{aligned} \tag{2}$$

Both the loss of discriminator and generator can be updated using binary cross entropy.

2.1.2 Maximizing mutual information

The input noise of generator is made up of an incompressible noise term, z and a latent code c. So the form of the generator becomes $G(z, c)$. Nevertheless, we have to put constraints on how generator use latent code. To achieve this, we add a new term $I(c; G(z, c))$, where I is the mutual information between two variables. That is, upon seeing latent code c/generator output, we can anticipate the output of generator/latent to be close to a certain number of image/latent code. However, it is difficult to directly optimize $I(c; G(z, c))$ since it requires the posterior $P(c|x)$. Fortunately we can obtain a lower bound of it by defining an auxiliary distribution $Q(c|x)$ to approximate $p(c|x)$:

$$\begin{aligned} I(c; G(z,c)) &= H(c) - H(c|G(z,c)) \\ &= E_{x\sim G(z,c)}[E_{c'\sim P(c|x)}[logP(c'|x)]] + H(c) \\ &= E_{x\sim G(z,c)}[D_{KL}(P(c|x)||Q(c|x)) + E_{c'\sim p(c|x)[logQ(c'|x)]}] + H(c) \\ &>= E_{x\sim G(z,c)}[E_{c'\ P(c|x)[logQ(c'|x)]}] + H(c) = L_l(G,Q) \end{aligned} \tag{3}$$

As such, we can get generator's loss:

$$L_G = E_{x\sim p_r}[logD(x)] + E_{x\sim p_g}[log(1-D(x))] + L_l(G,Q) \tag{4}$$

where $L_l(G, Q)$ can be updated using cross entropy.

```
output_q = netQ(fake)
errQ = ce_criterion(output_q, img_labels.to(device))
```

Figure 2: Q update.

2.1.3 How you generate fixed noise and images

To observe how our model learns, we fix noise at the beginning of the program. Fixed noise is composed of 56d random noise generated by torch.randn function and 10d latent code generated by torch.arange(0, 10). We use this fixed noise to generate images through generator.

2.2 Which loss function of generator you used

When training GAN, we have two loss functions for generator to use. One is $L_G = E_{x\sim p_g}[log(1-D(x))]$,and another one is $L_G = E_{x\sim p_g}[-log(D(x))]$. We choose the second one. The first

one suffers from saturation problem at early stage of training. Since at the beginning, the generator performs poorly, the discriminator can reject generator's image in high confidence as stated in the first GAN paper by Goodfellow.
To prove that we use the second function as our generator's loss function, we start the deduction from applying cross entropy to the output of discriminator and real data :

$$\begin{aligned} -(y\log(\hat{y}) + (1-y)\log(1-\hat{y})) &= -(y\log(D(\hat{x})) + (1-y)\log(1-D(\hat{x}))) \\ &= -(y\log(D(\hat{x})) + \log(1-D(\hat{x})) - y\log(1-D(\hat{x}))) \end{aligned} \tag{5}$$

As $y = 1$ for real data, we get:

$$-(\log(D(x)) + \log(1-D(x)) - \log(1-D(x))) = -\log(D(x)) \tag{6}$$

3 Results

We implement InfoGAN in two different architectures. The first architecture is the same as TA's note. As for the second architecture, we modify generator's transpose convolution into convolution layer and upsampling layer as mentioned in "Discussion" part, and modify discriminator's batch normalization into spectrum normalization. We list both results below. We can see that the second architecture performs better.

3.1 Results of your samples

InfoGAN samples are shown at Figure 3 for first architecture and Figure 4 for second architecture.

Figure 3: InfoGAN sample from first architecture.

Figure 4: InfoGAN sample from second architecture.

3.2 Training loss curves

Training loss curves shown at Figure 5 and Figure 6 are first architecture's loss. Loss curves for second architecture are listed at Figure 7 and Figure 8.

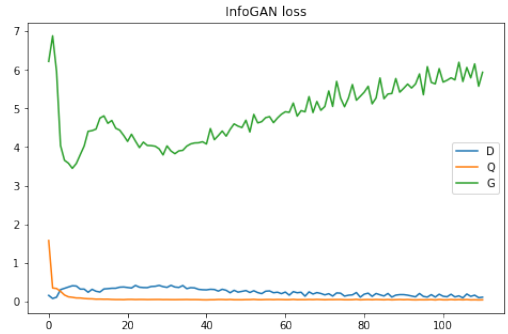

Figure 5: InfoGAN loss for first architecture.

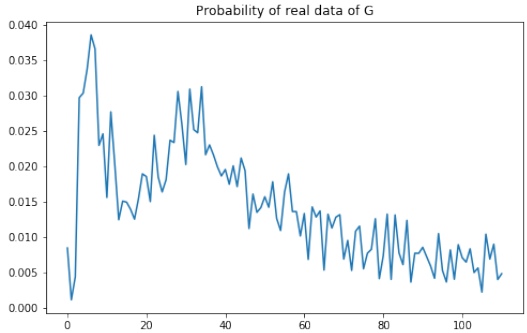

Figure 6: Real probability of generator for first architecture.

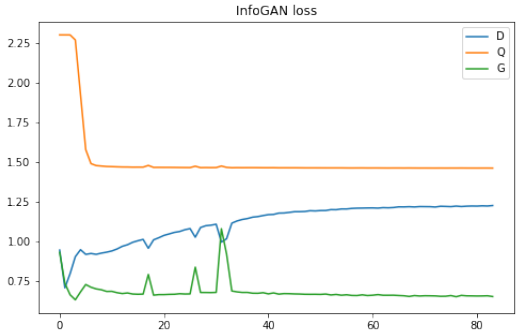

Figure 7: InfoGAN loss for second architecture.

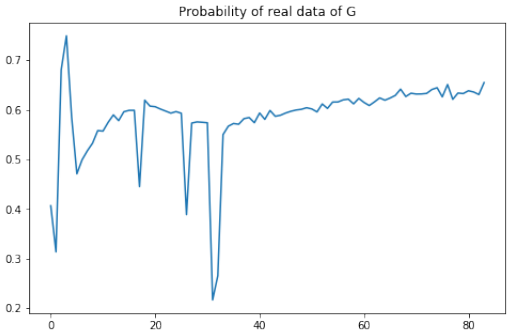

Figure 8: Real probability of generator for second architecture.

4 Discussion

4.1 checkerboard effect caused by Transpose Conv2d

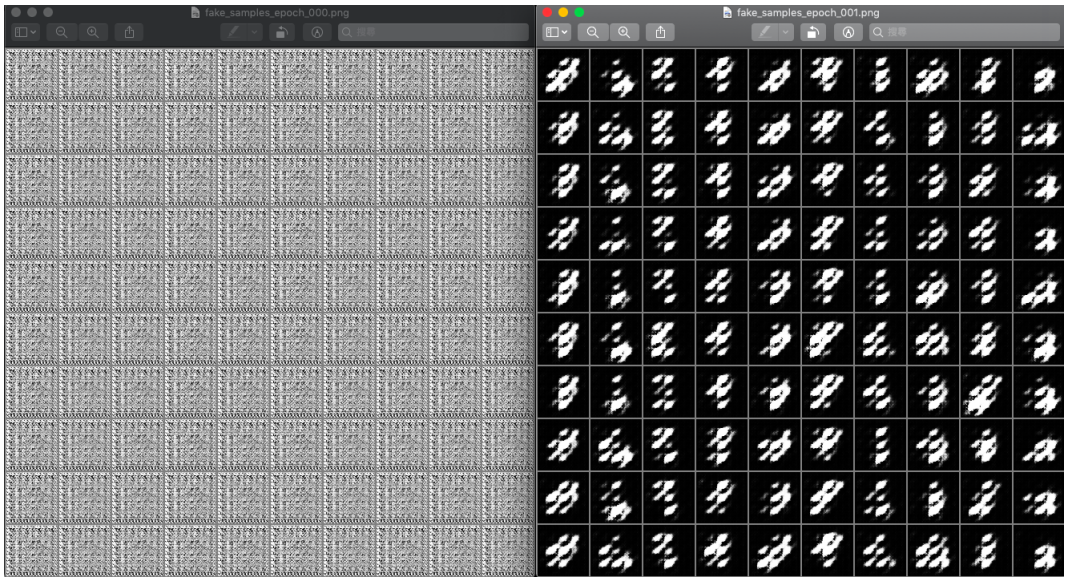

Figure 9: Early stage of training process.

In the early stage of training, we find out that by using transpose convolution layer, we'll get checkerboard effect as shown in Figure 9. From other people's experience, we can also observe same artifacts in Figure 10. This is caused by the uneven overlap of transpose

Figure 10: Transpose 2d artifact.

convolution in Figure 11.

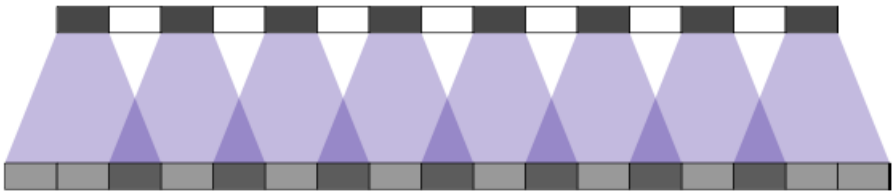

Figure 11: Uneven overlap of transpose convolution.

To solve this, we replace transpose convolution layer with convolution layer and upsampling layer and get Figure 12.

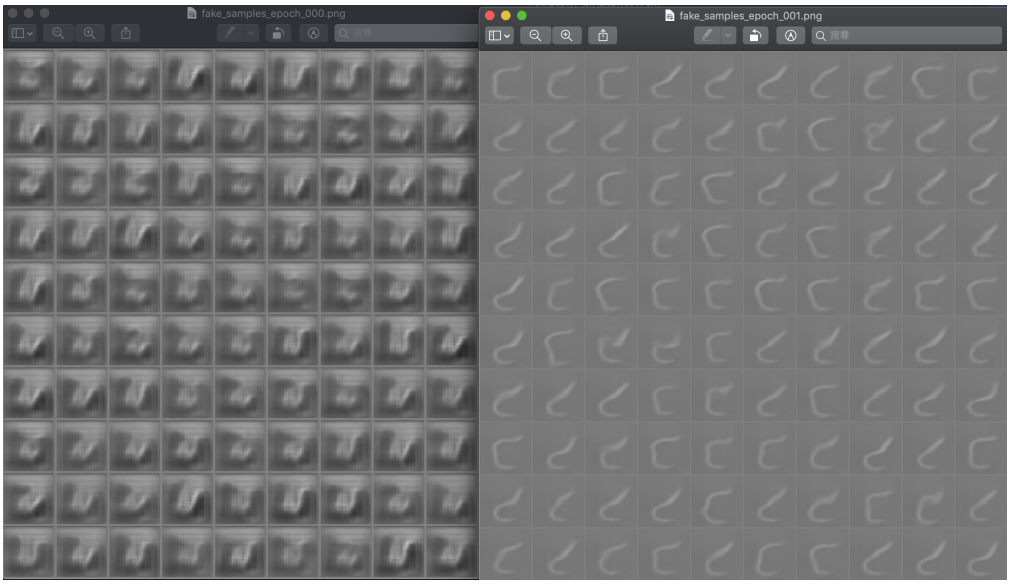

Figure 12: Replacing transpose convolution with convolution layer and upsamping layer.