IMVFX Homework 2: Conditional GAN Practice

Zhi-Yi Chin
joycenerd.cs09@nycu.edu.tw

December 16, 2021

1 Introduction

In this homework, we apply two conditional GANs: cGAN on the MNIST dataset and pix2pix on the Facades dataset. The goal of cGAN is to perform an image generation task, which is given a condition (give a number range from 0 to 9) and a random latent vector; it can generate a digit image specified by the condition. The goal of pix2pix is to perform an image-to-image translation, which is given an input image (building segmentation mask) and the target image (real building photo); based on the input image, it can generate an output image similar to the target image.

2 Network Structure

2.1 cGAN

Conditional generative adversarial network [3], or cGAN for short, is a type of GAN that involves the conditional generation of images by a generator model. In cGANs, a conditional setting is applied, meaning that both the generator and discriminator are conditioned on some auxiliary information (such as class labels or data) from other modalities. As a result, the ideal model can learn a multi-modal mapping from inputs to outputs by being fed with different contextual information.

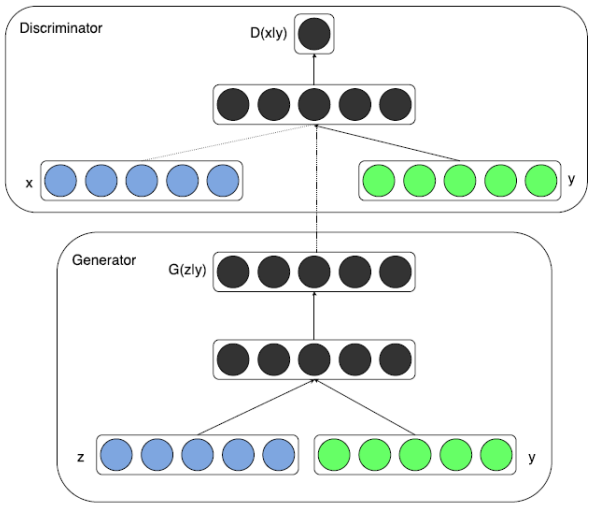


Figure 1: Conditional adversarial net.

Figure 1 shows the structure of a simple conditional adversarial net. Generative adversarial nets can be extended to a conditional model if the generator and discriminator are conditioned on extra information y. y could be any auxiliary information, such as class labels or data from other modalities. The conditioning is performed by feeding y into the discriminator and generator as additional input layers. The prior input noise $p_z(z)$ and y are combined in joint hidden representation in the generator. In the discriminator, x and y are presented as inputs to a discriminative function. The objective function of a two-player minimax game become Eq. 1.

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}\,(\boldsymbol{x})}[\log D(\boldsymbol{x} \mid \boldsymbol{y})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z} \mid \boldsymbol{y})))] \tag{1}$$

2.2 pix2pix

The Pix2Pix Generative Adversarial Network [2], or GAN, is an approach to training a deep convolutional neural network for image-to-image translation tasks. The Pix2Pix model is a type of conditional GAN, or cGAN, where the generation of the output image is conditional on input, in this case, a source image. The discriminator is provided both with a source image and the target image and must determine whether the target is a plausible transformation of the source image. The generator is trained via adversarial loss, enabling the generator to generate plausible images in the target domain. The generator is also updated via L1 loss measured between the generated and expected output images. This additional loss encourages the generator model to create plausible translations of the source image.

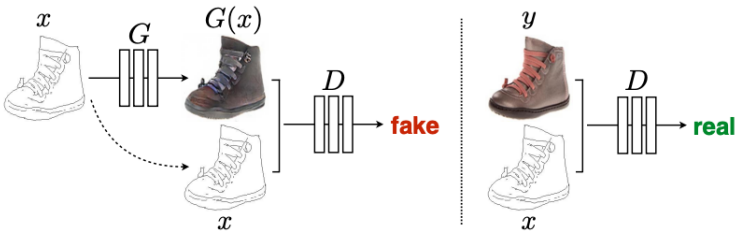


Figure 2: The discriminator, D, learns to classify between fake (synthesized by the generator) and real input, target tuples. The generator, G, learns to fool the discriminator. Unlike an unconditional GAN, both the generator and discriminator observe the input edge map.

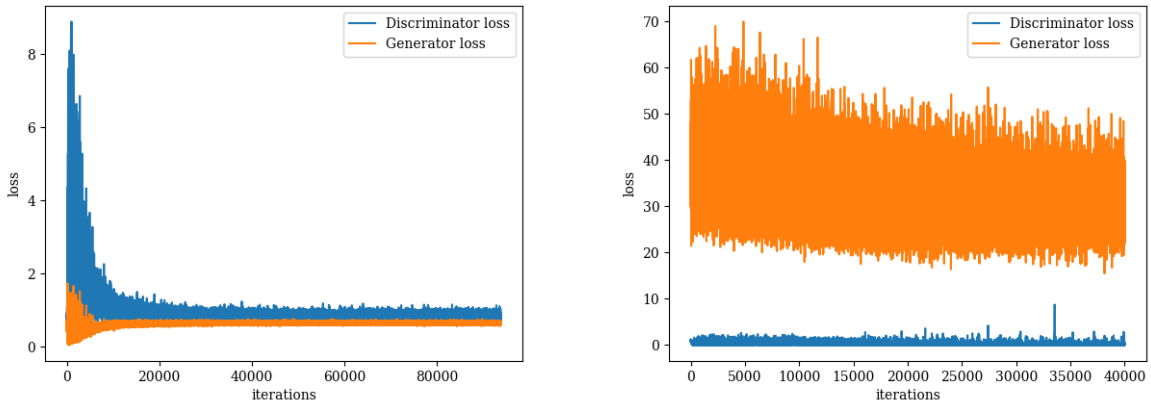
The discriminator design is based on the effective receptive field of the model, which defines the relationship between one output of the model to the number of pixels in the input image. This is called a PatchGAN model and is carefully designed so that each model's output prediction maps to a 70×70 square or patch of the input image. The benefit of this approach is that the same model can be applied to input images of different sizes, e.g., larger or smaller than 256×256 pixels. The generator is an encoder-decoder model using a U-Net architecture. The model takes a source image and generates a target image. It does this by first downsampling or encoding the input image down to a bottleneck layer, then upsampling or decoding the bottleneck representation to the size of the output image. The U-Net architecture means that skip-connections are added between the encoding and corresponding decoding layers, forming a U-shape.

The discriminator model is trained directly on real and generated images, whereas the generator model is not. Instead, the generator model is trained via the discriminator model. It is updated to

minimize the loss predicted by the discriminator for generated images marked as "real." As such, it is encouraged to generate more authentic images. The generator is also updated to minimize the L1 loss or mean absolute error between the generated and target images. The generator is updated via a weighted sum of both the adversarial loss and the L1 loss, where the authors of the model recommend a weighting of 100 to 1 in favor of the L1 loss. This encourages the generator to generate plausible translations of the input image and not just plausible images in the target domain. The discriminator is updated standalone, so the weights are reused in this composite model but are marked as not trainable. The composite model is updated with two targets. First, indicating that the generated images were real (cross-entropy loss), forcing significant weight updates in the generator toward generating more realistic images. Second, the executed real translation of the image is compared against the output of the generator model (L1 loss).

3 Experimental Results

3.1 Plot of loss curve

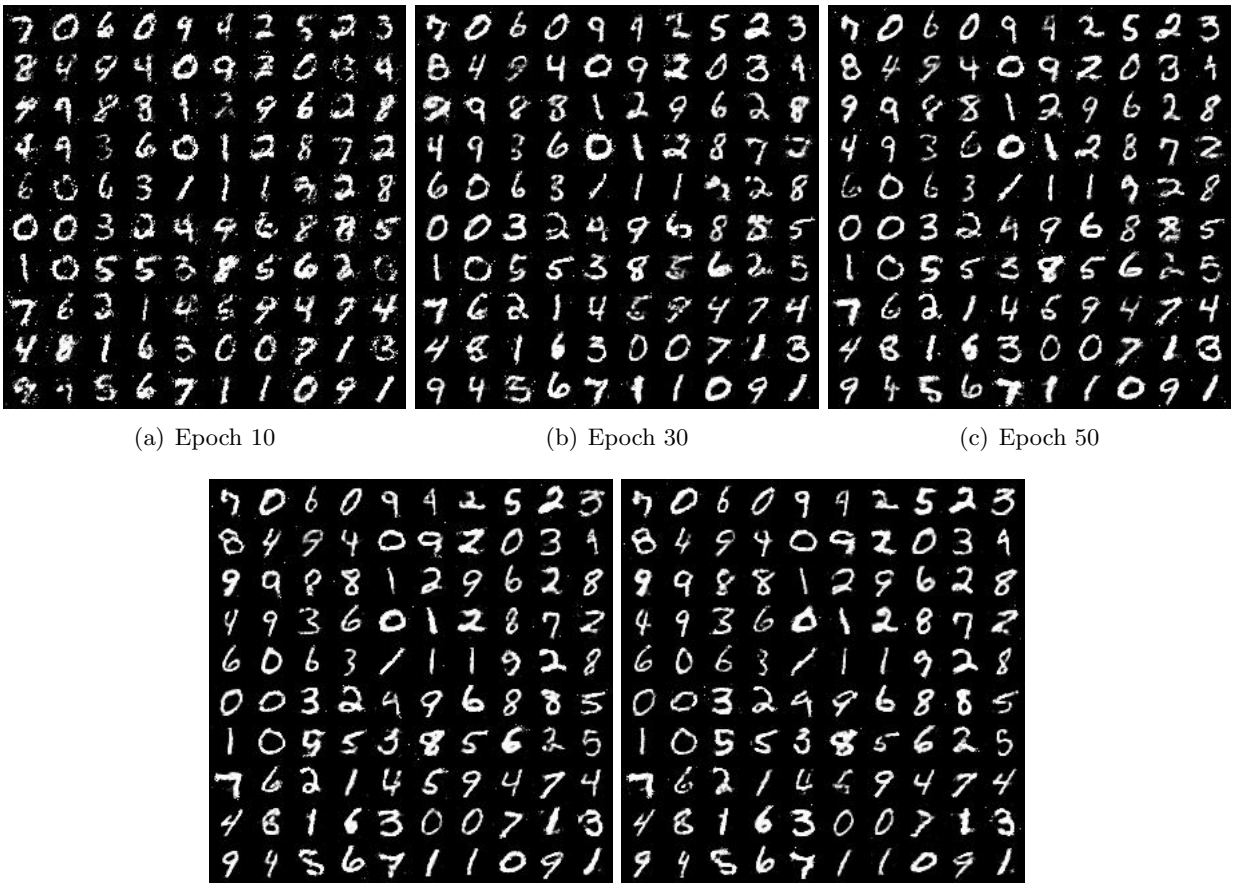


(a) cGAN loss curve

(b) pix2pix loss curve

Figure 3: The generator and discriminator loss curve of the two methods.

3.2 Generated results in different epoch



(a) Epoch 10 (b) Epoch 30 (c) Epoch 50

(d) Epoch 70 (e) Epoch 100

Figure 4: The generated 10*10 Grid images in different epochs.

3.3 Generate images with specified condition labels

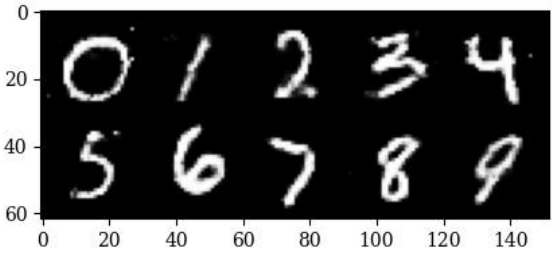


Figure 5: Condition from 1 to 9

3.4 Use trained model to do image-to-image translation

(a)

(b)

(c)

(d)

(e)

Figure 6: For every subfigure, from left to right are input, generated output and target, respectively

4 Discussion

4.1 The difference between cGAN and pix2pix

Both cGAN and pix2pix are conditional GANs. The two methods have some differences. cGAN is the first conditional GAN, and its idea is relatively simple. We can add a one-hot encoded condition (e.g., digit) as a condition, and the generator will try and generate an image with that one-hot encoded condition. The condition of pix2pix is an input image, and the ultimate goal is to preserve the input image's content and transform its style to the target image. pix2pix is more like style transfer. Since pix2pix has a more complex goal, the network architecture is a lot more complex than cGAN. Both the generator and the discriminator of cGAN are just simple CNN. For pix2pix, the generator is an encoder-decoder model using U-Net, and the discriminator utilizes a PathGAN model.

4.2 Why using BCE loss for training?

There are two reasons. First, the BCE objective can more accurately be stated as "the output of the images by the generator should be assigned a high probability by the discriminator." That is $BCE(D(G(Z)), 1)$ where $D(G(Z))$ is the probability assigned to the generated image by the discriminator. Given a "perfect" generator that always has photorealistic outputs, the $D(G(Z))$ values should always be close to 1. There are difficulties getting this kind of convergence (the training is inherently unstable), but that is the goal. Second, in the standard GAN algorithm, the latent vector (the "random noise" which the generator receives as input and has to turn into an image) is sampled independently of training data. If using the MSE between the outputs of the GAN and a single image, we might get some results out. However, we would effectively be saying, "given this (random) Z, produce this specific X," and this would be implicitly forcing the generator to learn a nonsensical embedding of the image. If we think of the Z vector as a high-level description of the image, it would be like showing it a dog three times and asking it to generate the same dog given three different (and uncorrelated) descriptions of the dog. Compare this with something like a VAE, which has an explicit inference mechanism (the encoder network of the VAE infers Z values given an image sample) and then attempts to reconstruct a given image using those inferred Zs. The GAN does not attempt to reconstruct an image, so it does not make sense to compare its outputs to a set of samples using MSE or MAE in its vanilla form.

A more intuitive way to explain is we update our model based on the discriminator's results. Furthermore, the discriminator has two possible outcomes, fake (0) or real (1). In this case, BCE loss is an excellent loss function.

4.3 Result of different settings

Most of the hyperparameters we use are the same as the sample code provided by the TAs, except we change the number of training epochs. In the sample code, the number of training epoch is 5, and after running the code, we find out that the 5 epoch is not enough since the generated quality is not that great. We change the training epoch to 100. From our observation, both methods converge after 50 epochs of training. The result is surprising because the GAN project we have done before always train at least 200 epochs.

4.3.1 How to evaluate convergence?

We can evaluate model convergence based on training loss for many other tasks. If the training loss has not decreased for a long time, it is most likely that the model is converged. However, the loss function does not have this kind of information for GAN. For cGAN, the loss seems to converge after 20 epochs, but the generated quality seems stable after 50 epochs. For pix2pix, we cannot tell the image quality from the loss curve at all. For GAN, we find out that we may need to save the generated images during the training process and look at the generated image quality to decide whether the model converges. If the image quality does not improve anymore, we can stop training. If this is a big project, we can define a metric to evaluate the generated image quality (e.g., FID, PSNR, SSIM); this is an objective way to evaluate model convergence.

5 Bonus

5.1 Train pix2pix on different dataset

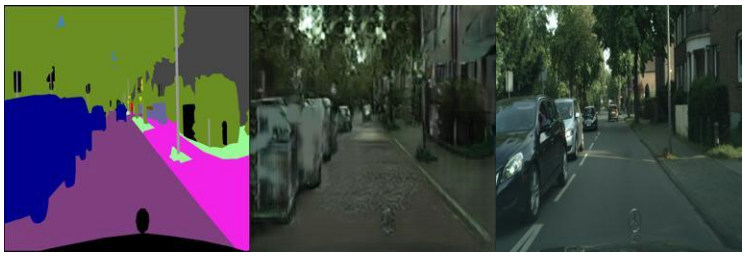
We train pix2pix on cityscapes dataset via this link. The results are shown in Figure 7. The quality of the result is not as good as performing on the facades dataset. We think it is because, for this dataset, the translation may be more complex than the faces dataset. There is only one translated object in the facades dataset, a building. However, there may be roads, trees, buildings, cars, and pedestrians in the cityscapes dataset. The model needs to handle more objects in the cityscapes dataset. If we train this model for a longer time, the quality will become better. However, for the time limit, we train 100 epochs only.

(a)

(b)

(c)

(d)



(e)

Figure 7: For every subfigure, from left to right are input, generated output and target, respectively

5.2 Try with different type of cGANs

We apply conditional Self-attention GAN (SAGAN). A Self-attention GAN (SAGAN) [5] is a DCGAN that utilizes self-attention layers. For convolution, they convolve nearby pixels and extract features out of the local block; they work "locally" in each layer. In contrast, self-attention layers learn from distant blocks. [5] states that DCGAN could fail in capturing geometric or structural patterns that occur consistently with multi-class datasets.

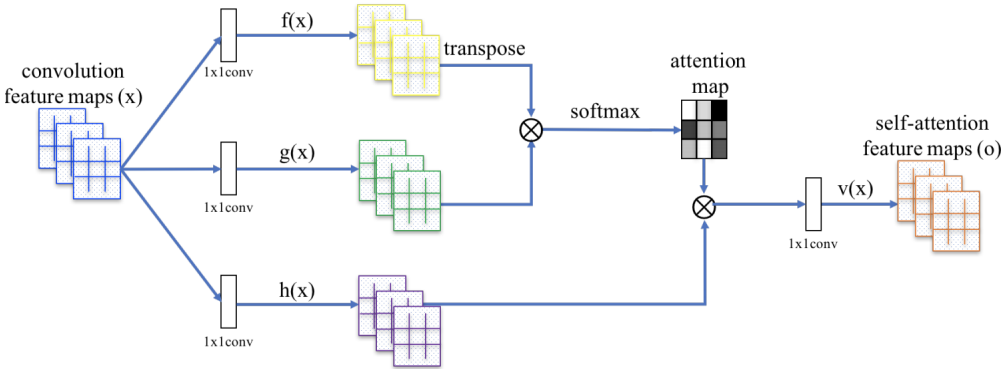


Figure 8: Self-attention mechanism for SAGAN.

The self-attention module is applied to both generator and the discriminator. For the normalization layer, SAGAN uses spectral normalization. It is a novel weight normalization technique proposed in [4] for a stabilized training process instead of the most common batch normalization. The original SAGAN implementation is not a conditional GAN. Also, the accepted image size should be equal to or larger than 64×64. To cope with this problem, we reference the implementation from christiancosgrove/pytorch-sagan [1].

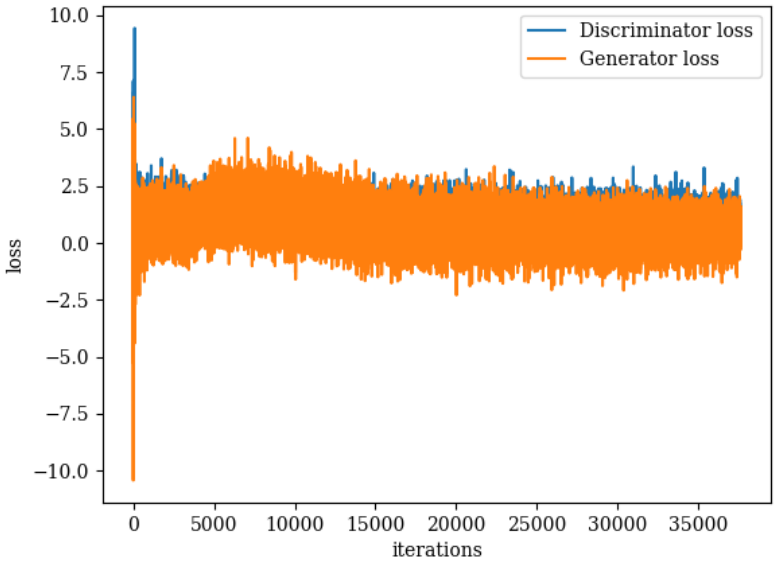
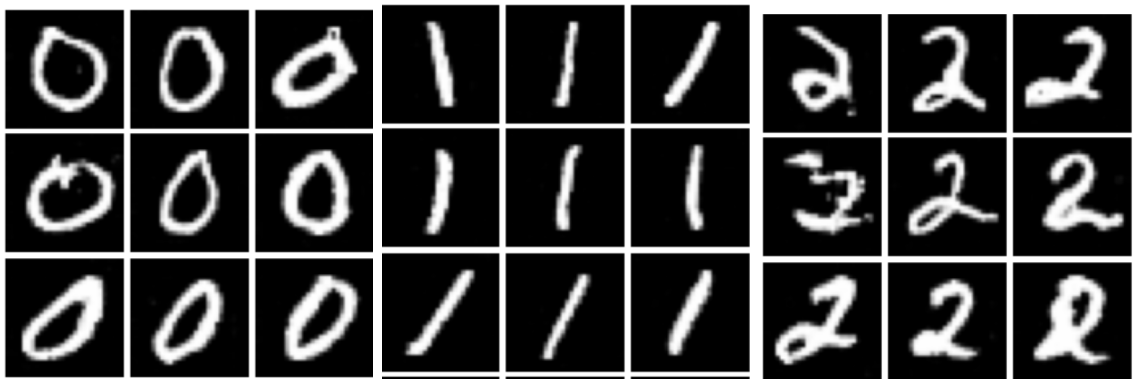
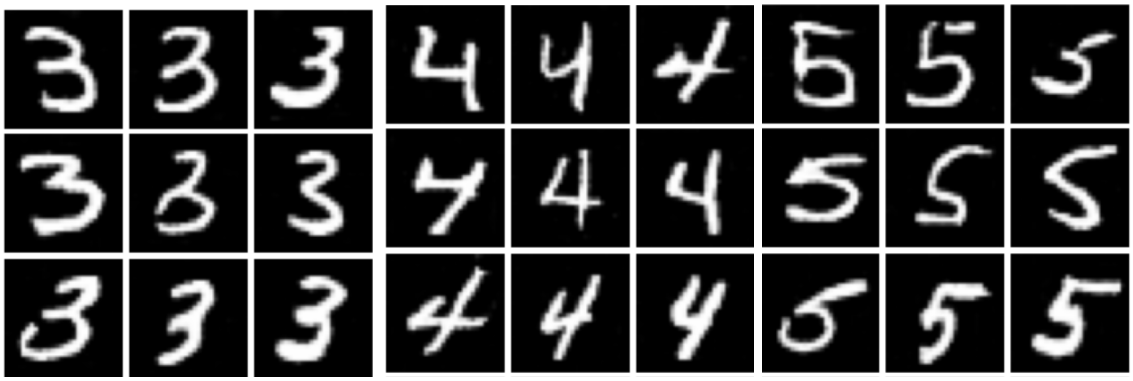


Figure 9: SAGAN loss curve

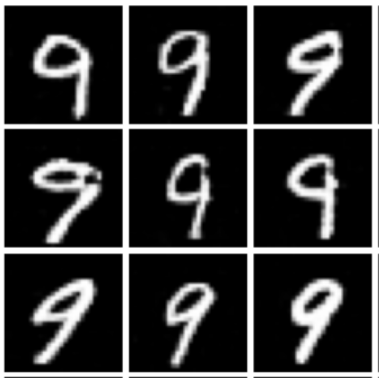


(a) Number 0 (b) Number 1 (c) Number 2



(d) Number 3 (e) Number 4 (f) Number 5

(g) Number 6 (h) Number 7 (i) Number 8



(j) Number 9

Figure 10: The qualitative results of SAGAN.

The result of SAGAN is excellent, but this method converges slower than cGAN. cGAN converge after 50 epochs, but SAGAN converge after 100 epochs. Also, from the results, we found out that digit 2 visualization is not as good as other digits.

References

[1] C. Cosgrove. Self-attention generative adversarial networks (sagan) in pytorch, 2018. URL https://github.com/christiancosgrove/pytorch-sagan.

[2] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 1125–1134, 2017.

[3] M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[4] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

[5] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena. Self-attention generative adversarial networks. In *International conference on machine learning (ICML)*, pages 7354–7363. PMLR, 2019.