Conditional Generative Adversarial Networks

Joseph Kawiecki MS ECE

1. Introduction

Before the landmark paper *Generative Adversarial Nets* (Goodfellow et al., 2020) detailing a new framework for generative models, such models were relatively obscure and difficult to operate. This was due to the nature of generative models being difficult to predict and approximate computations. Furthermore, many of the predominant activation functions at the time, such as piecewise linear units, were challenging to implement in a generative network. But then generative adversarial networks (GAN) were introduced. These networks employ a generative and discriminative model competing against each other in an adversarial training setup. The discriminative network, called the discriminator, will determine if a given input is real or not. The generative network, called the generator, will produce fake data that is input to the discriminator in an attempt to "fool" it. Generative adversarial networks brought an improvement over previous methods in efficiently calculating model gradients without the use of Markov chains or inference. Furthermore, generative adversarial nets have a wide array of uses in being able to generate realistic data. This high-quality data generation is what makes generative adversarial nets special, but it is also what this paper seeks to improve upon. Specifically, without guidance on what to produce, the generator is left to construct a large spectrum of output possibilities. This generative process may be controlled through condition data, primarily through class labels. The issue of incorporating data conditioning into both the generative and discriminative networks is the primary contribution of the paper *Conditional Generative Adversarial Nets* (Mirza & Osindero, 2014) (cGAN).

I was drawn to this problem as it is related to my current research. Specifically, my research involves style transfer for robotic learning. Prior to this project, I had limited experience working with GANs and wished to build my knowledge.

My aim was to first re-implement the original paper and then add a few extensions in an attempt to produce more realistic images. A sample output of the model is in Figure 1. The primary objective was to evaluate if conditioning the data inputs to the generative and discriminator networks would improve results. An improvement would be generating more realistic images and running more computationally efficient. Conditional generative adversarial nets could be beneficial across a wide range of applications where image generation is used. This includes entertainment, health, facial recognition, reconnaissance, photo editing, and much more.

Figure 1. Sample generated output.

2. Related Works

2.1. Generative Adversarial Networks

One key drawback associated with the image generation process of a generative adversarial net is control. While the generator should eventually learn a data distribution capturing the intended style of output, this process can take a long time to learn and may be broad. Conditional generative nets present a solution to provide more control over image generation through the use of condition labels. Such labels can guide a generative net to produce more specific images after training. For example, after training a network with images of dogs, a user may want to view generated images of a specific breed. However, assuming the model was trained over many different breeds, without conditioning it is highly unlikely the model would be able to produce the specific images.

2.2. Deep Convolutional Generative Adversarial Networks

Although published later, one related work attempting to solve a similar problem is titled *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks* (Radford et al., 2015). This paper incorporates a new architecture with convolutional neural networks (CNN) being used for the discriminator and generator layers. The method also forgoes conditioning. Deep convolutional generative adversarial networks (DCGAN) are intended to be used for image classification and generation. For the purpose of simply providing more control during the image generation process, DCGAN are more than capable but require significantly more resources relative to conditional generative nets.

2.3. Evaluation of Generative Models

The original cGAN authors recommend Parzen window log-likelihood for evaluation. In attempting to quantify and compare model performance, the method of evaluation is a critical component. In the paper *A Note on the Evaluation of Generative Models* (Theis et al., 2015), the authors analyze a few evaluation methods of generative models. Three main methods are discussed: average log-likelihood, Parzen window estimation, and visual examination. Parzen window estimation consists of a kernel to estimate density that is built from samples of the evaluation model. From there, log-likelihood is used to evaluate the kernel to compare with the true log-likelihood of the overall model. It is thought that generally, the greater the number of samples, the better the estimator performance. However, it is shown that for multiple datasets and model evaluations with a range of samplings, the Parzen window estimator does not come close to the true log-likelihood estimation, oftentimes barely reaching 50%. The authors therefore recommend avoiding Parzen window estimates except in narrow use cases. For these reasons and poor online documentation, the decision was made to not use the Parzen window estimator for evaluation. Instead, I opted for Frechet Inception Distance (FID) shown in eq. (1).

$$FID = ||u_r - u_r||^2 + T_r(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2}) \quad (1)$$

Frechet Inception Distance evaluates the quality of the generated images relative to the real image by measuring the difference in pixels between the two. Great documentation and being a more modern, standard method of evaluation were the primary factors for why FID was selected.

3. Conditional Adversarial Networks

3.1. Generative Adversarial Networks

At a high level, generative adversarial nets operate by scheduling a discriminator model, D, against a generator model, G, during training. The discriminator is fed both a real and fake data distribution. The real distribution is from the input image source and the fake from the generator output. The generator is fed with random noise in order to replicate the real distribution and attempt to trick the discriminator. The discriminator then decides what is the real distribution. Over time, the generator learns this distribution from the discriminator output to produce better replicas. Consequently, the discriminator continually learns new features to be a better judge. This process is captured at a high level in Figure 2 below.

In mathematical terms, this match between the generator and discriminator can be modeled as a min-max loss function:

$$E_x[log(D(x))] + E_z[log(1 - D(G(z)))] \quad (2)$$

The generator will attempt to minimize eq. (2) while the discriminator will maximize it. The two components on either end of the addition are the discriminator loss and the generator loss respectively. More specifically, the discriminator will attempt to maximize eq. (3):

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [log(D(x)) + log(1 - D(G(z)))] \quad (3)$$

The discriminator is concerned with two components, the $log(D(x))$ term indicating the classification of the real data and the $log(1 - D(G(z)))$ term indicating the classification of the fake data. The x term represents real image data and z indicates random noise.

The generator will attempt to minimize the error in the eq. (4) below. This function produces fake outputs in an attempt to trick the discriminator.

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} [log(1 - D(G(z)))] \quad (4)$$

Generative adversarial nets are known to be challenging to train due to the difficulties keeping both models synchronized. Ideally, both models will eventually stabilize and converge together, yet this rarely occurs. As the generator gets better, usually the discriminator will perform worse. Likewise, as the discriminator improves, the generator usually does worse.

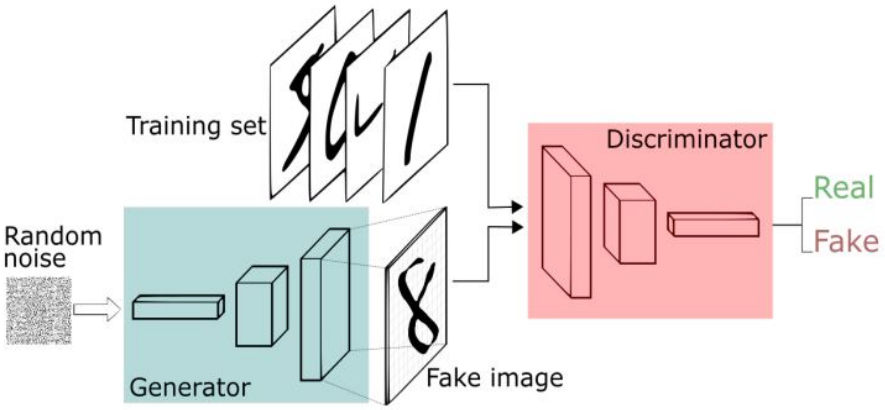

Figure 2. Diagram of a standard generative adversarial network.

3.2. Conditional Generative Networks

One improvement to generative nets is the addition of condition labels. The input data to both the discriminator x and generator z can be conditioned on extra data y to improve control of the generative process. This extra data can be input directly into each model alongside the respective data. This change to the input layer is captured in the min-max function below. Eq. (5) is the engine that comprises cGAN.

$$\min_G \max_D V(D,G)) = \\ E_x[log(D(x|y))] + E_z[log(1 - D(G(z|y)))] \tag{5}$$

4. Experiment

4.1. Model Architecture and Setup

The cGAN model was built following the original paper with a few upgrades to improve GAN performance. As mentioned, the MNIST dataset was used. 10 true class labels paired with 10 randomly generated class labels were used as the real and fake conditioning labels respectively. The discriminator was fed a batch of images along with conditioning labels. The condition labels are fed through an embedding layer before being combined with the image data. The combined data batch is fed through four additional linear layers each with a Leaky ReLU activation function and a dropout layer. Last, the combined batch is fed through a sigmoid layer. Likewise, the generator is fed a batch of random noise with conditioning labels. The noise is constructed with a latent dimension of 100 as recommended. Similar to the discriminator, the condition labels are fed through an embedding layer before being combined with the noise. The combined data is then fed through four more layers each followed by a Leaky ReLU activation function and a dropout layer. Finally, the batch is fed through a hyperbolic tangent function. The general idea of the cGAN model architecture used is depicted in Figure 3. The Leaky ReLU and dropout values used were 0.2 and 0.3 respectively. While the original paper used stochastic gradient descent, I used the Adam optimizer with a learning rate of 0.0001 and 0.0002 for the discriminator and generator respectively. Both the optimizer and learning rates were selected as a result of experimentation. Binary cross entropy loss was computed on the outputs with the appropriate scores.

4.2. Evaluation and Results

Before constructing the cGAN, I built a baseline unconditional network for understanding and comparison. Evaluation metrics for comparison included Frechet Inception Distance (FID), the end loss of the discriminator and generator, and visual inspection. After training for 50 epochs each, the results are captured in Table 1 below. Loss graphs for the unconditional and conditional models over 50 epochs are shown in Figures 4 and 5 respectively. Based on the FID score, the conditional model outperformed the unconditional model by a good margin. However, the conditional loss was more divergent. Only at a few points it seemed to converge while the unconditional loss showed convergence almost the entire training time. As the conditional model was built from the unconditional, both models and training procedures are very similar besides the condition labels. In terms of visual comparison, the results are less clear as both outputs appear very similar to the human eye. However, it does appear the conditional implementation is slightly more clear as seen in Figure 7.

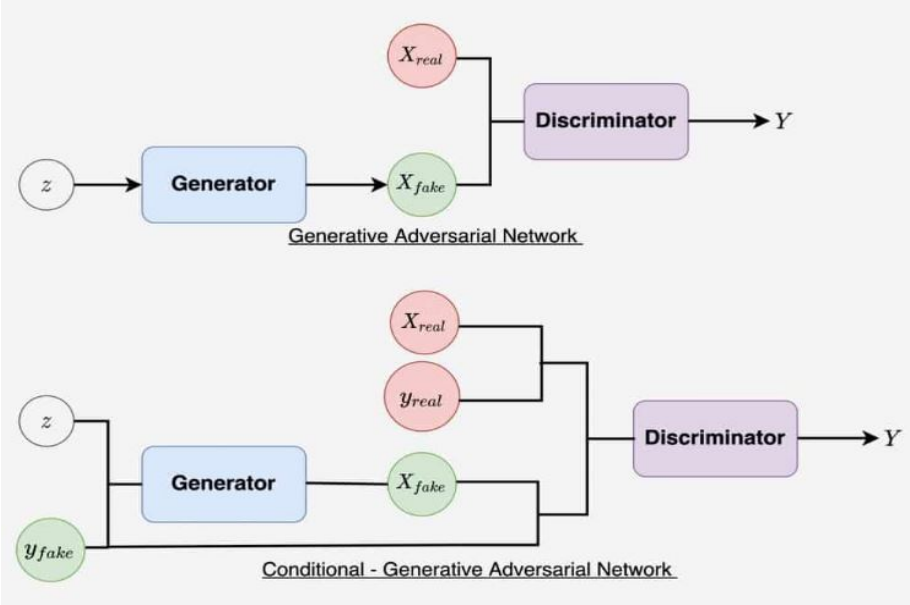

Figure 3. GAN versus cGAN architecture.

	Best FID	D End Loss	G End Loss
Unconditional	76.4712	1.2298	1.2095
Conditional	64.9438	1.2111	0.7946

Table 1. Experimental Results

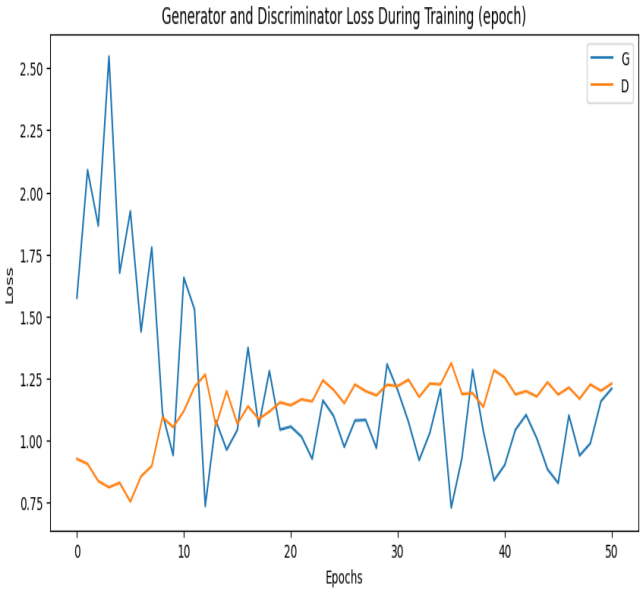

Figure 4. Discriminator and generator loss of the unconditional model over time.

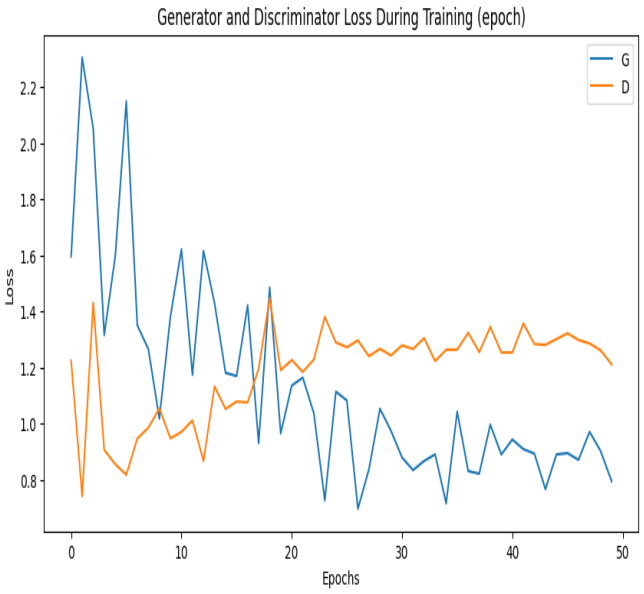

Figure 5. Discriminator and generator loss of the conditional model over time.

4.3. Challenges

There were many unexpected challenges faced in re-implementing the model. Beginning with the unconditional model, it was not very difficult to get running, but producing good results was. The first major issue was mode collapse. The output images would hardly change and the loss val-

ues spiked. To solve this, the discriminator and generator architectures were made to be more similar in terms of the number of parameters. Another major issue with the conditional implementation was the addition of the condition labels. The authors recommend encoding the labels as one-hot vectors. However, this did not yield good results when actually implemented. Further research led to the discovery of embedding layers which serve as a more robust lookup table than one-hot vectors. Adding the embedded layer to handle the conditioned labels provided a sizable boost in performance. Smaller challenges involved configuring the learning rates for the discriminator and generator. Through experimental performance, the discriminator was consistently updating faster than the generator. To account for this, the learning rate for the generator was set to double that of the discriminator. The final output of the unconditional baseline and conditional implementation are shown in Figures 6 and 7.

Figure 6. Final generated output of unconditional baseline model.

Figure 7. Final generated output of the conditional implemented model.

5. Conclusion

This paper demonstrated that conditioning on extra data is an effective means to direct the output generation of a generative adversarial net. Using the MNIST dataset with 10 class labels, the conditional implementation successfully outperformed the unconditional baseline obtaining a better FID score. This indicates the final model was able to generate more realistic images than the baseline model.

This project taught me much about machine learning and generative adversarial networks. In terms of future work, I will first suggest upgrades to the model. I recommend upgrading the model to a DCGAN to use convolutional neural network layers for the generator and discriminator models. I also recommend exploring different activation functions other than ReLU and different optimizer learning rates. Last, I attempted to use the CIFAR-10 dataset, but did not achieve quality results. Future work should explore generalization to different datasets. Regarding general suggestions, applying conditional generative networks to general text-image synthesis could be a useful future application in many fields.

CGAN Code

Acknowledgements

Thanks to soumith: ganhacks

Thanks to Ibtesam Ahmed: GAN in Pytorch with FID

References

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

Marinescu, R. Tutorial on generative adversarial networks - from basics to current state-of-the-art, and towards key applications in medicine, 2020. URL http://www.mit.edu/~razvan/talk/gan-tutorial/pres.pdf.

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

Radford, A., Metz, L., and Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Sharma, A. Conditional gan (cgan) in pytorch and tensorflow, 2021. URL https://learnopencv.com/conditional-gan-cgan-in-pytorch-and-tensorflow/.

Theis, L., Oord, A. v. d., and Bethge, M. A note on

the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015.