Conditional Generative Adversarial Networks

Joseph Kawiecki MS ECE

1. Introduction

A number of years ago, the primary models used for deep learning were discriminative. These models typically involved translating a detailed, high-dimension input to a class. Before the landmark paper *Generative Adversarial Nets* (Goodfellow et al., 2020) detailing GANs, generative models were obscure and difficult to get working. This was due to the nature of generative models being difficult to predict and approximate computations. Utilizing well-understood and commonly used techniques of the day such as piecewise linear units were also challenging to implement in a generative network. But then generative adversarial nets were introduced. These networks employed a generative and discriminative model competing in an adversarial training setup. The generative network would produce fake data in an attempt to "fool" the discriminator that was deciding if a given input was real or not. Generative adversarial nets were 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. But through condition data, primarily through the class labels, the generative process may be controlled. The issue of incorporating data conditioning into both the generative and discriminative networks is the primary contribution of this paper *Conditional Generative Adversarial Nets* (Mirza & Osindero, 2014).

This problem was interesting to me personally as much of my current research is related to generative nets. Specifically, my work involves style transfer for robotic learning. As I had limited experience working with generative nets before, I wanted to build my knowledge and learn their practical implementation.

My aim was to first re-implement the original paper and then add in a few upgrades to hopefully produce more realistic images. A sample output of the baseline model with slight altering is depicted in Figure 1. The primary objective was to evaluate if conditioning the data input to the generative and discriminator networks would improve the output. An improvement of the output would be generating more realistic images and operating 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 of baseline model.

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 some time to learn and will 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 generate specific breed images.

2.2. Deep Convolutional Generative Adversarial Networks

Although published after, 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 model architecture of the traditional generative adversarial net 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 providing more control of the generation process, the model is overly complex and would require significantly more resources to operate than conditional generative nets which are much lighter.

2.3. Evaluation of Generative Models

The suggested method of evaluation for conditional generative adversarial nets was the Parzen window log-likelihood. In attempting to quantify and compare model performance, the method of evaluation was a critical component. In the paper *A Note on the Evaluation of Generative Models* (Theis et al., 2015), the authors analyze a few common methods of generative models and speak to the difficulty of the generative adversarial net evaluation process. This is primarily due to the sheer range of applications. The paper explored three different evaluation techniques: average log-likelihood, Parzen window estimation, and visual examination. Parzen window estimation consists of a kernel to estimate density that is built from samples sourced from the evaluation model. From there, log-likelihood is used to evaluate the kernel for comparison with the overall model's true log-likelihood. 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 advise avoidance of Parzen window estimates except in very narrow 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), the formula is stated below.

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

Frechet Inception Distance evaluates the quality of the generated images relative to the real image instead of the whole generative adversarial net model. It does this by measuring the differences in pixels between the two. The narrow use tailored to the specific application paired with greater online documentation was the primary factor to select FID.

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 mode, G, during training. The discriminator is fed with both a real data distribution along with a fake data distribution from the generator. The generator is fed with random noise to replicate the real distribution in an attempt to trick the discriminator. The discriminator decides what is the real distribution. Over time, the generator learns about this distribution from the discriminator outcome 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:

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

The generator will attempt to minimize the function above while the discriminator will maximize it. The two components of the function separated by the addition are the discriminator loss and the generator loss. The discriminator will attempt to maximize the specific function below:

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

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 function below. This function produces fake outputs in an attempt to trick the discriminator to misclassify a fake image as real.

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

Generative adversarial nets are known to be challenging as it is difficult to keep both models running in sync. 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, the generator will usually perform worse as the discriminator improves.

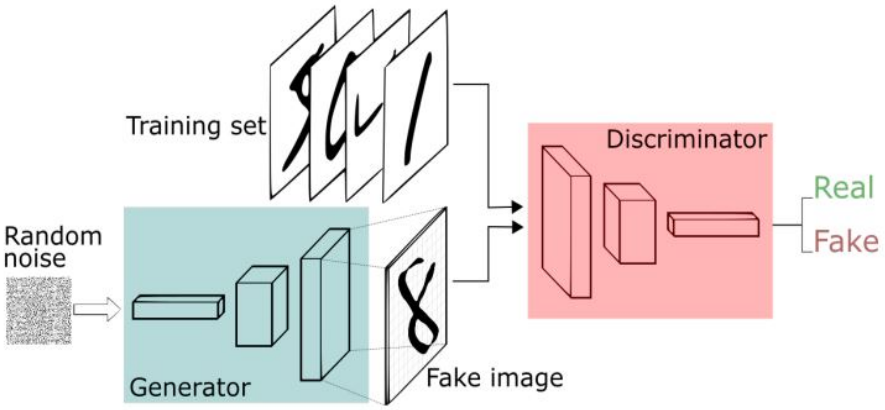

Figure 2. Diagram of a standard generative adversarial network.

3.2. Conditional Generative Networks

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 each's respective data. This change to the input layer is captured in the min-max function below which encapsulates the nature of the CGAN.

$$\min_G \max_D V(D, G)) =$$

$$E_x[log(D(x|y))] + E_z[log(1 - D(G(z|y)))]$$

4. Experiment

4.1. Model Architecture and Setup

The model was built following the original paper as closely as possible with a few minor changes. As mentioned, the MNIST dataset was used. The 0-9 class labels along with a randomly generated 0-9 class label set were both used as the real and fake conditioning labels respectively. The discriminator is fed a batch of images along with conditioning data. Both the images and condition data are fed through separate hidden layers before being combined. The combined data batch is then carried through four additional layers each followed by a RELU activation function. Last, the batch is fed through a final sigmoid layer before being output. Likewise, the generator is fed a batch of noise along with conditioning data. The noise is constructed using a dimension of 100. From input, the noise and condition data are each fed through a separate hidden layer before being combined. The combined data is then fed through two more layers each followed by a RELU activation function. Finally, the batch is put through a sigmoid function. A general idea of the CGAN model architecture used is depicted in Figure 3. Batch normalization and dropout with a probability of 0.5 are used for both the discriminator and generator. While the original paper used stochastic gradient descent, I used the Adam optimizer with a learning rate of 0.002 and 0.001 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 one hot vector labels.

4.2. Evaluation and Results

In terms of evaluation, I utilized a baseline model found online [cite] for comparison. Evaluation metrics for comparison included Frechet Inception Distance (FID), the average loss of the generator and discriminator, and visual inspection of the output images. After training for 50 epochs each, the results are captured in Table 1 below. Graphs of the generator and discriminator losses over time are shown in Figure 4 and Figure 5 respectively. Based on FID score, it is clear my model was outperformed by the baseline model. This may be due to a few factors. One such factor is the use of the Leaky RELU activation versus regular RELU. Another is the different learning rate parameters. The last major difference could be the setup of the training function and specifically the real and fake labels.

	Best FID Score	Avg Discriminator Loss	Avg Generator Loss
Baseline	69.4361	0.5224	2.9591
Implementation	312.476	0.0031	7.5981

Table 1. Baseline and implemented model results.

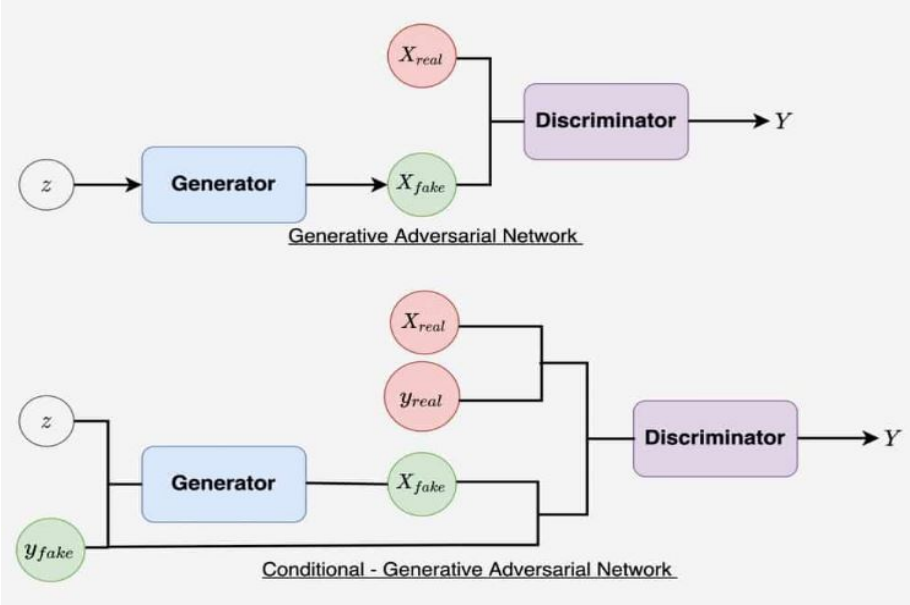

Figure 3. Diagram of the GAN versus CGAN architecture.

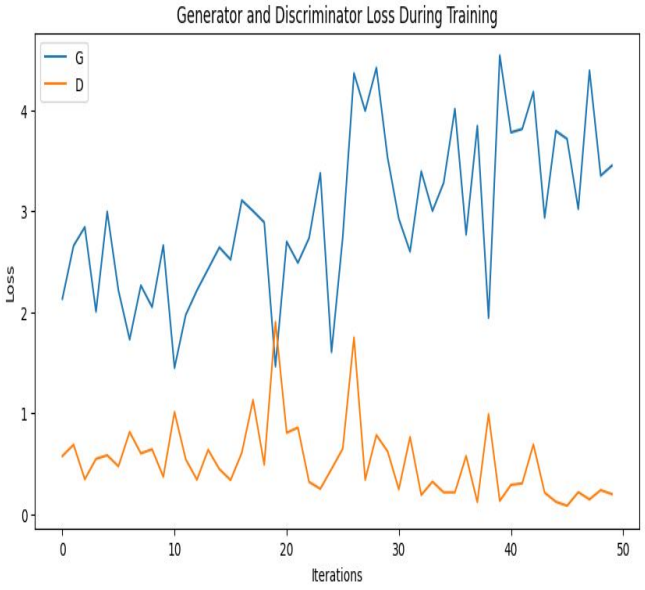

Figure 4. Discriminator and generator loss of the baseline model over time.

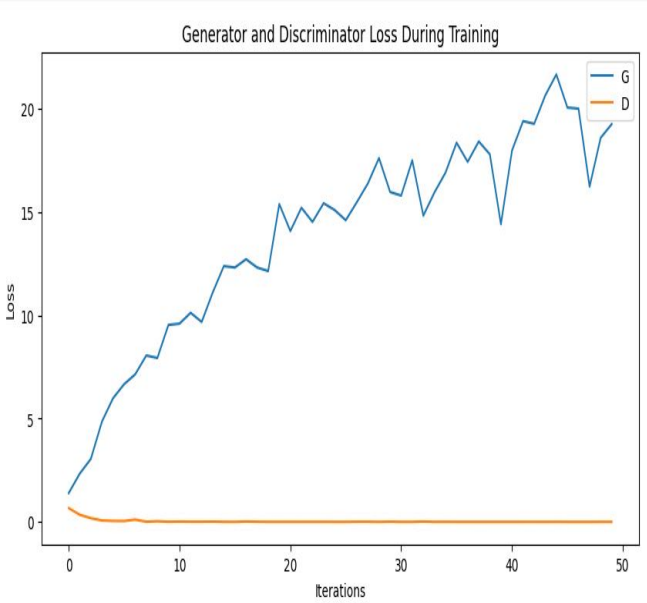

Figure 5. Discriminator and generator loss of the implementation model over time.

4.3. Challenges

There were many unexpected challenges faced in re-implementing the model. Getting the model in a place where it was running and doing something was not all that difficult, but getting relevant data was. The first significant issue was that the loss values were not changing. At first, nei-

ther the discriminator nor the generator was changing at all. Eventually, adding batch normalization helped get the discriminator changing, but it would eventually stabilize at the level the generator loss was stuck at. This issue was finally solved once the appropriate conditioning data was added. Real and random condition data were incorrectly initialized and set input to the wrong models prior to this. Specifically, the random condition data was accidentally generated as a zero vector instead of a one-hot vector which took a while to discover. A sample output of the baseline and implemented model is shown in Figure 6 and Figure 7. Again through visual inspection, it is clear the re-implemented model was outperformed by the baseline. Many fixes were attempted to improve the final implementation with not much success. But, the re-implemented model is functioning and a proper evaluation is able to be conducted through FID score. At the time of submission, the next issue under investigation was the optimizer and learning rate.

Figure 6. Final generated output of baseline model.

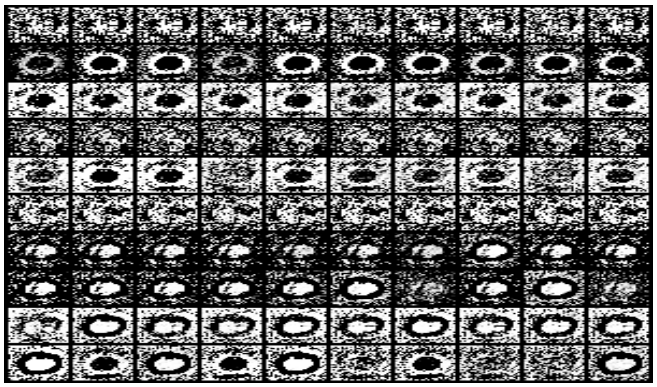

Figure 7. Final generated output of the implemented model.

5. Conclusion

This paper demonstrated that conditioning on extra data is an effective means to direct the output generation of a generative adversarial net. Although the implementation was lacking relative to the baseline, it still portrayed the effectiveness of controlling output generation using class labels. Specifically, the MNIST dataset was used with the ten class labels to format output images as seen in Figure 6. Primarily through the use of Frechet Inception Distance, it was shown that the baseline model settled around a score of 70 while the implemented model settled around a score of 300. This indicates the base model was able to generate much more realistic images than the implemented model.

Despite not seeing great end implementation results, the project taught much about machine learning and generative adversarial networks. In terms of future work, I will first suggest upgrades to the model. I recommend exploring more powerful layers for the generator and discriminator models such as convolutional neural networks. I also recommend exploring different activation functions other than RELU and different optimizer learning rates. 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 TeeyoHuang: conditional-GAN GitHub Repository

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.