Conditional Generative Adversarial Networks

Abhinav Rao [*] [1]

Abstract

Generative Adversarial Networks or GANs (1), introduced in 2014 by Goodfellow et al., present a unique scale-able architecture leveraging neural networks for generative modelling. Conditional Generative Adversarial Networks or C-GANs (2) extend this architecture to incorporate meta-data, usually target labels, in both the generator and discriminator of GANs to be able to generate samples conditioned on specific meta-data and also improve performance. In this paper we reintroduce C-GANs, briefly discuss the incorporation Wasserstein GAN-like (3) method in C-GAN architecture, discuss Fréchet inception distance (FID) (4) method to evaluate generative models and finally tie it all together by implementing C-GAN on a standard data-set to analyse sensitivity of hyper-parameters with respect to model performance and stability.

Introduction

Machine learning is a field which deals with developing methods to leverage data to perform any set of tasks. On a broad level these tasks can be classified as either discriminatory or generative. The former can be viewed as finding the conditional distribution $p(y|x, data)$ with inputs x and targets y, where as the latter can be viewed as finding the joint distribution of the data itself $p(data)$ or $p(x, y)$.

Generative models, although less common, have more general and ubiquitous use cases since they learn the distribution of the data itself rather than a mapping function. First, they make it possible to sample new data from the generated distribution. And secondly, if the data can be separated into inputs (x) and targets (y), as if in a supervised learning scenario, then they can even perform discriminant tasks using statistical inference. $p(y|x) = p(x, y)/p(x)$.

Generative Adversarial Networks (1), or GANs are a subclass of generative models using neural networks. Conditional Generative Adversarial Networks (2), or CGANs extend GANs to incorporate supervised data, that is inputs and targets to improve GAN performance and generate conditionally targeted samples. GAN architecture involves two sub-modules, namely, generator and discriminator, which are pitted against one another to improve their performance. This can at times, lead to unstable training with one model's performance overtaking other while compromising the overall performance. One of the proposed methods to tackle these issues is Wasserstein GAN (3), which allows lopsided training of one of the networks for better convergence.

Unlike discriminant models, there are not many commonly agreed methods to evaluate model performance when it comes to generative models. Fréchet inception distance (4) or FID is a standard method to evaluate generative modeling by approximating the distance between distribution of generated data and the real data. This allows us to benchmark progress and compare between models, thus making it possible to understand sensitivity of hyper-parameters on the training process.

In this paper we select the standard MNIST data-set to train a C-GAN model. We define a control case and change each of the following hyper-parameters - batch size, generator learning rate, discriminator learning rate, number of epochs, dropout rate (5), negative slope of leaky ReLU (6) and finally Wasserstein correction (number of critical steps), to assess their effect on GAN performance using FID.

Project Repo: Link.

Related Work

This work very closely follows the work of the original paper on Conditional Adversarial Networks, in that it aims to analyze the performance of C-GANs as a whole rather than on a specific use-case. Where this work deviates from the seminal original is the focus on hyper-parameter sensitivity and evaluation using data-set specific evaluation metric like FID.

This is also related to conditionally targeted image generation projects such as (7) which focus on using C-GANs for image generation but are more involved in the complications of the conditioning structure and loss function for more involved type of meta-data. Works such as (8) also use MNIST to validate the efficacy of their C-GAN implementation but are more driven by the lack of meta-data or partial target data and how it affects performance. This paper more so relies on the effect of neural network on C-GANs.

Method

To understand C-GANs it is important to understand the mathematical framework of GANs and their generic implementation. The implementation of C-GANs is a simple extension atop the original structure.

1. Generative Adversarial Networks: GANs consist of two networks - the generative network (generator) and the discriminant network (discriminator). The generator takes a random input of pre-specified size and generates outputs the same size as the data. The discriminator takes a sample for either the data or the generated output as input and outputs a single scalar stating the probability the input came from the data distribution. The role of the generator is to learn the data distribution and trick the discriminator and the discriminator trains in a supervised learning paradigm, getting better at detecting generated inputs (1). This adversarial relationship between the networks is mathematically introduced to the structure through the loss function, which can be explained as follows:

Let,
X_p: sample of dimension p from the data distribution $\mathbf{X}$.
Z_d: sample of dimension d of random noise distribution $\mathbf{Z}$.
θ_G: the learn-able parameters of the generator G.
θ_D: the learn-able parameters of the discriminator D.

Thus,
$G(Z_d, \theta_G)$ or more succinctly, $G(Z_d)$ is the generator output for the input Z_d. Dimension of $G(Z_d)$ is p.
$D(X_p, \theta_D)$ or more succinctly, $D(X_p)$ is the discriminator output for the input X_p. $D(X_p)$ is a scalar.
$D(X_p)$, $D(G(Z_d))$ are the discriminator output for real and generated input respectively.

As discussed the discriminator is trying to solve a binary classification problem. This is usually implemented using a binary cross-entropy. In this case we will NOT write the loss but a reward function, which is to be maximized. Thus, the discriminator is rewarded when it predicts X as 1 and $G(Z)$ as 0, and the reward function can be written as:

$$R(D, \theta_D) = \mathbf{E}_{X_p \sim \mathbf{X}}[log(D(X_p))] + \mathbf{E}_{Z_d \sim \mathbf{Z}}[log(1 - D(G(Z_d)))] \tag{1}$$

And it follows that the generator must be optimized to minimize $R(D, \theta_D)$, which is its loss function. We can summarize the entire training process as a mini-max game as a single expression value function V as follows (1):

$$\min_{\theta_G} \max_{\theta_D} V(G, D) = \mathbf{E}_{X_p \sim \mathbf{X}}[log(D(X_p))] + \mathbf{E}_{Z_d \sim \mathbf{Z}}[log(1 - D(G(Z_d)))] \tag{2}$$

The optimization may be done using usual approaches such as stochastic gradient descent, which must converge since the value function is convex (1). Ideally the solution for the $R(D, \theta_D)$ converges to a value of 0.5 which means the discriminator is unable to distinguish between the generated data and the original data better than a random guess (1).

2. Conditional Generative Adversarial Networks: C-GANs as described allow GANs to conditionally generate data when the data-set includes both data X_p and corresponding target y_k. (2):

$$\min_{\theta_G} \max_{\theta_D} V(G, D) = \mathbf{E}_{X_p \sim \mathbf{X}}[log(D(X_p | y_k))] + \mathbf{E}_{Z_d \sim \mathbf{Z}}[log(1 - D(G(Z_d | y_k)))] \tag{3}$$

The exact way to implement the condition on the discriminator and the generator models can be different for different purposes. The expected result is for the model to learn the conditional likelihood distribution $p(X|y)$ which further allows one to sample targeted data from the model. One method that is suggested in the original paper is to add the targets y_k to the inputs X_p and Z_d of discriminator or generator respectively (2). Thus the conditional inputs can be viewed as vectors X_{p+k} and Z_{d+k}. But the details may be more involved especially in more complicated structures such as CNN based C-GANs, also known as Conditional Deep Convolutional Generative Adversarial Networks (C-DCGANs).

3. Wasserstein GANs: Wasserstein GANs, is proposed as a method to improve the stability and convergence of GANs. The details of Wasserstein GANs are out of the scope of this paper. On a high level, Wasserstein GAN replaces the mini-max value function with a similar critic function parametrized by 'nCritic' which defines the number of times the generator is trained for every valuation of the critic function (3). We will test this once with a nCritic of 5 using a "approximate model" for Wasserstein GAN atop the vanilla C-GAN model.

4. Fréchet inception distance: FID is a standard metric used to assess efficacy of generative models. It is based on the Fréchet distance which is defined between any two probability distributions with finite mean and variance (4). A closed form formula can be obtained by assuming both the distributions as Multivariate Gaussian. This is given by:

For two Multivariate Normal written as $\mathcal{N} = \mathcal{N}(\mu, \Sigma)$ and $\mathcal{N}' = \mathcal{N}(\mu', \Sigma')$ the Fréchet Distance is given by:

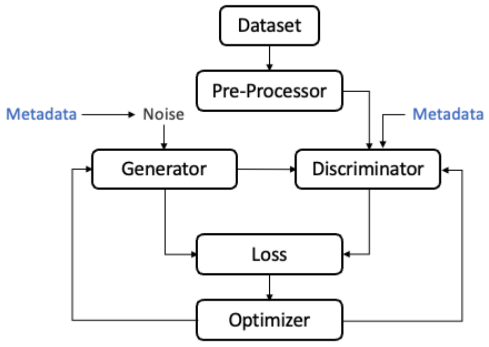

Figure 1. Code block diagram for the full model

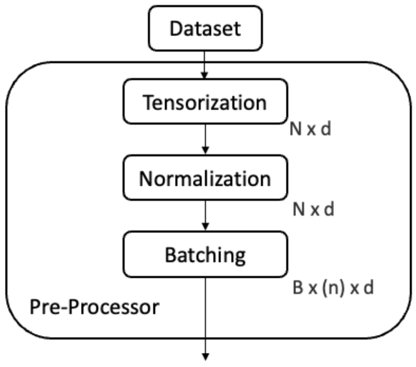

Figure 2. Code block diagram for the Pre-Processing block

$$d_F(\mathcal{N},\mathcal{N}') = \sqrt{\|\mu - \mu'\|^2{}_2 + tr(\Sigma + \Sigma' - 2(\Sigma\Sigma')^{1/2})} \tag{4}$$

This explains the F and D part of FID. Data such as images are not spatially coherent in a way to directly be able to use FID to differentiate between generated and real images. Inception is the practice of using neural encoding of real data and generated data to find FID instead of the data directly (4). Multi-layer perceptrons are very good at taking high dimensional data and compressing them into spatially coherent tensors. We will explain the specifics of our FID implementation in the next section.

Experiment

1. Overall Architecture of C-GAN: On a high level the problem includes six blocks; data-set, pre-processor, generator, discriminator, loss and optimizer. We have already discussed the loss function of C-GANs above, we will discuss the rest here.

The data-set we will be using is MNIST which contains 60000 training samples each being a 28x28 gray-scale image of a handwritten digit between 0 to 9. This data-set was chosen based on the use-case and the overall goal of this experiment which requires multiple runs. MNIST is lightweight data-set with strong community support.

The pre-processing step involves converting the data into appropriate format (tensors), normalizing them, and batching them appropriately for stochastic gradient descent.

2. Neural Network: We define the generator and discriminators as multi-level perceptron networks with linear fully connected hidden layers. Both networks will involve three hidden layers. We will also use leaky ReLU architecture instead of normal ReLU. Leaky ReLU allows learning even when the layer outputs are negative (6). It also allows us to understand the effect of one more parameter on our generator's abilities. The choice to use tanh activation for the generator and sigmoid for the discriminator is based on the output function scope and is also consistent with the original GAN paper (1). We also include dropout in the discriminator as suggested in the original paper. The generator has a net input dimension of 110 which includes 100 for noise and 10 for the target label (one-hot encoded). The output dimension of the generator is same as the data, that is 784 (28x28). The input dimension of the discriminator is 794 which includes the input dimension of 784 and the 10 dimension of the target label, and the discriminator output is a single scalar of dimension 1, between 0 and 1.

3. Loss and optimization: The loss is described at length in the section above. We use a binary cross entropy loss to train each network. We use Adam as our optimizer instead of simple stochastic gradient descent due to more stable and quicker descent. The whole architecture is implemented in pyTorch and is geared towards training in GPU.

4. Custom Inception Model: There are many FID implementations available, but the usual FID libraries are not made for MNIST-like data. They are geared towards 3-channel high resolution images. This motivated a custom trained FID implementation. Thus a simple MNIST classifier with 2 hidden layers was trained, each with 50 neurons. The second layer activation are considered for the FID calculations. The FID score was calculated on 5000 fake and real image samples. To test the custom solution, two different models were chosen, one with visibly bad generation ability. The FID scores reported were 2.69 for the good model and 167.51 for the bad model. This validated the efficacy of the FID evaluation system.

5. Hyper-parameter Analysis: There are seven major hyperparameters of interest; batch-size, learning rate of the gen-

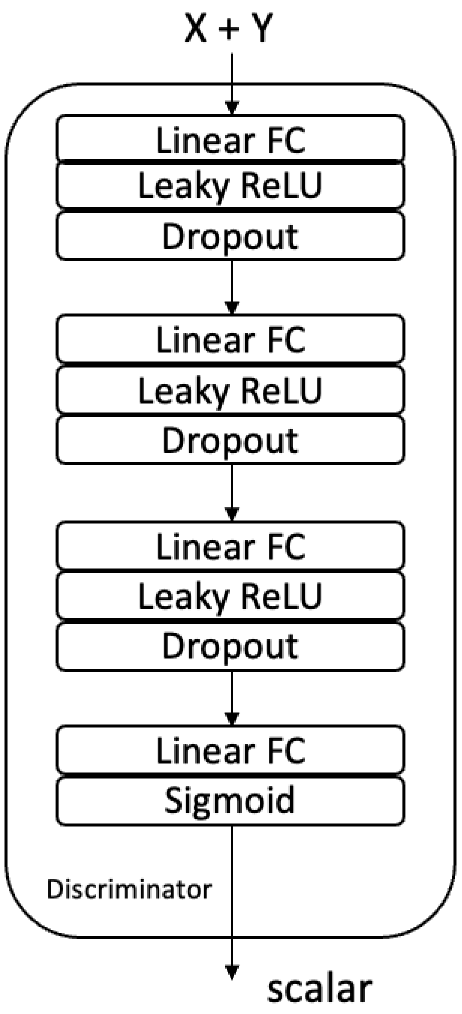

Figure 3. Code block diagram for the Discriminator block

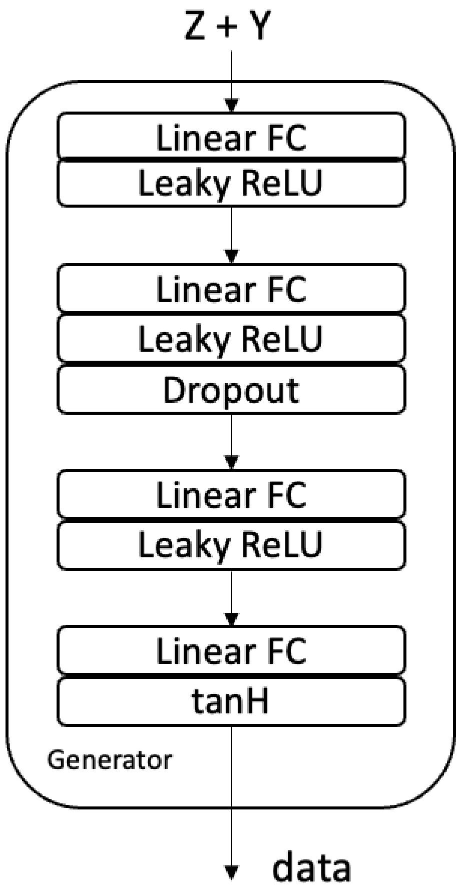

Figure 4. Code block diagram for the Generator block

erator (gLR), discriminator learning rate (dLR), number of epochs (epochs), dropout rate for the discriminator(dropout), the negative slope for the leaky ReLU activation (leaky) and finally Wasserstein critical steps (nCritic). The control case was run with 100 as batch size, both learning rates of 1e-4, for 50 epochs, dropout rate of 0.3, negative slope of 0.2 for leaky ReLU and no Wasserstein critical steps.

After this each hyper parameter was changed independently and FID, final losses and time per epoch were noted. The entire results are as shown in table 1. The models converged in all the tests except one, which is shown in figure 7. It is also to be noted that as the purpose of the entire effort was to run multiple models and do multiple evaluations of FID, a very lean approach towards modelling was considered. The number of weights were reduced as much as possible without affecting the output too much, bring epoch time to 20s (on google colab T4 GPUs).

Conclusion

In the conducted tests C-GANs converged 9/10 times with strong performance within 50 epochs. The targeted generation was tested in creating all the log files and was found to always accurate. The relatively simple execution method of incorporating the targets by just embedding and adding them to the input works and success may also attributed in part to the normalization step in the pre-processing stage. This points to the remarkable ability of neural nets to approximate over multiple modalities without much change in structure.

Table 1. All tested Hyperparameters & evaluations

batch size	gLR	dLR	epochs	dropout	leaky	nCritic	time	FID
100	0.00010	0.00010	50	0.3	0.200	1.0	20.162177	2.6929
100	0.00010	0.00010	50	0.3	0.200	5.0	33.715941	1.2710
100	0.00010	0.00010	50	0.3	0.001	1.0	19.718167	1.7450
100	0.00010	0.00010	50	0.5	0.200	1.0	20.273801	3.6701
100	0.00100	0.00010	50	0.3	0.200	1.0	19.962824	167.5097
100	0.00010	0.00100	50	0.3	0.200	1.0	20.487020	1.7497
50	0.00010	0.00010	50	0.3	0.200	1.0	26.082310	2.0212
50	0.00005	0.00010	50	0.3	0.200	1.0	25.851769	0.9597
32	0.00010	0.00010	100	0.3	0.200	1.0	31.891440	0.6166
100	0.00005	0.00005	100	0.3	0.200	1.0	19.893487	1.4947

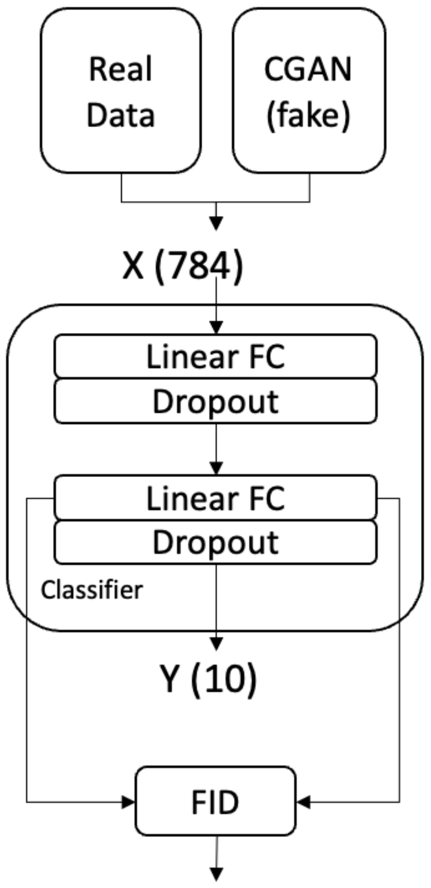

Figure 5. Block diagram for the FID estimation

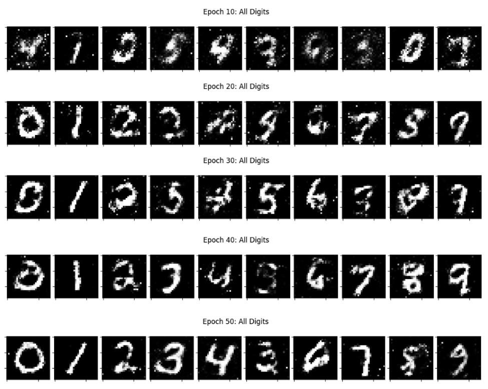

Figure 6. Progress of training every epoch for 50 epochs. Final FID = 2.6929

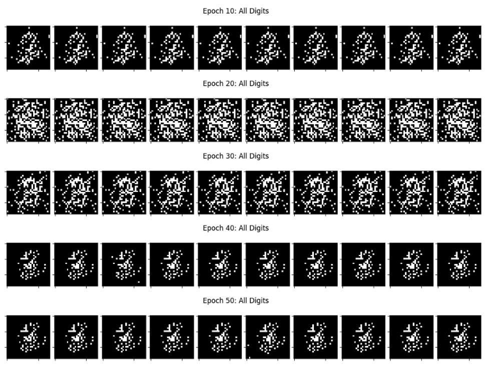

Figure 7. Progress of training every epoch for 50 epochs. Final FID = 167.5087. No convergence case

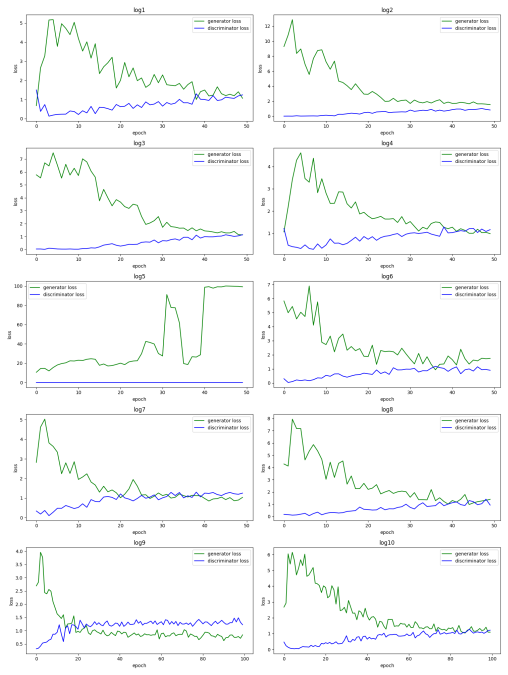

Figure 8. Losses for GAN models for all test cases. Model 5 does not converge.

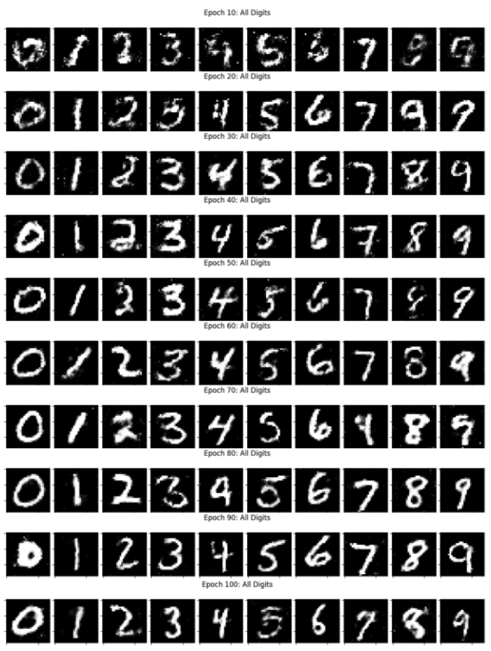

Figure 9. Samples for every 10th epoch when model is run for 100 epochs. Early convergence can be seen and the model only gets marginally better hinting the architecture maybe too lean

The most sensitive hyper-parameter tested was the learning rate of the generator. The test with a high learning rate (0.001) did not converge and the test with the lowest learning rate (0.00005) had the lowest FID, that is the best performance. This was also visually validating, validating the efficacy of the custom FID implementation as mentioned above. The second most effective hyper-parameter tested was the number of epochs. This is to be expected as the lower bound was kept relatively low at just the elbow of convergence. Effects of over-fitting with large number of epochs was not tested. Discriminator learning rate on the other hand showed minimal effect on the output as compared to the generator. A higher learning rate showed marginal improvement in the model. Increasing dropout rate showed negative impact. Leaky ReLU's negative slope rate had minimal impact. The Wasserstein correction showed improvement and had one of the most stable convergence curve.

Another thing to consider is the total training time. Most of the tests had more or less similar training time per epoch, except the one with the Wasserstein GAN implementation and the one with smaller batch-size. This is as expected since the Wasserstein GAN increases the number of backward passes each epoch and a smaller batch size increases the number of passes per epoch proportionally. Of the two, the Wasserstein implementation may be worth a deeper dive even with its 150% increase in training time due to its relatively stable and faster convergence. Batch-size showed minimal effect and smaller batches especially below 50, may not be worth pursuing.

This experiment reinforces the following notions about C-GANs:

- GANs can be extended to produce targeted sample without increasing layers or changing the overall architecture too much. This alludes to the fact that the underlying GAN architecture has enough ability to learn not only the structure of the data but also the relationship between the label and the data.

- The generator stability during training is relatively more sensitive to the convergence of the entire model when compared to the discriminator. This reinforces the motivation for Wasserstein GANs which allow more lopsided training of the generator. The theoretical final loss of discriminator is 0.5 (1) (a random guess) as explained above. But there are cases when the discriminator dominates and trends towards zero loss and drives the overall loss function(3). As visualized in figure 7, this incentives the generator outputs to be very different from the original data-set, which can be identified by the discriminator immediately.

- FID scores matched visual inspection, further reinforcing their efficacy in evaluating generative models. The use of a custom inception model based on layers of a classification model contains enough representational information to effectively quantify success of generative model at the very least. A comparative analysis between evaluation metrics is needed to validate the effectiveness of FID. The FID score in this case was also sensitive to number of samples used to approximate the mean and covariance matrix. FID scores also have numerical shortcomings due to the calculation of square root of the matrix (eq(4)).

- The lean model allows quicker training and hyper-parameter tuning. But this comes at a cost of poor final performance irrespective of number of epochs. This is intuitive as the model does not have enough resolution to learn all artefacts especially in complicated symbols like 5 and 8. (figure 9). This can be improved by changing architecture (CNN based C-GANs) or increasing layers.

Acknowledgement

I would like to acknowledge the following individuals and their respective material which helped in the completion of the above work.

- Diego Gomez Mosquera: Implementation of GAN from scratch. The mathematics and intuition behind

GANs is explained very well and in a concise manner. Link

- Jason Brownlee: For an intuitive explanation of FID (Link) and Wasserstein loss (Link).
- Victor Costa (@vfcosta): For a numerically stable and well-tested implementation of the FID formula for pytorch tensors. Link
- Mats Sjöberg and CSC - IT Center for Science: For a lean implementation of a MNIST classifier used by this paper for making the inception vector Link
- Erik Linder-Norén: For a simple GAN implementation with Wasserstein implementation. Link
- Google Colab: This project was completed on google's free service Colaboratory using free T4 GPUs over 4 different Gmail accounts.

References

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[2] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[4] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[5] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[6] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, Georgia, USA, 2013.

[7] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[8] Francisco J Ibarrola, Nishant Ravikumar, and Alejandro F Frangi. Partially conditioned generative adversarial networks. *arXiv preprint arXiv:2007.02845*, 2020.