A Comparison of Different GANs for Generating Handwritten Digits on MNIST*

Jeffery B. Russell†
Fourth Year Computer Science Student at RIT
CUBRC Research Assistant
RITlug President

Ryan Missel‡
Fifth Year Computer Science Student at RIT
CASCI Research Assistant

Kyle Rivenburgh§
Fifth Year Computer Science Student at RIT

(Dated: April 25, 2020)

Generative Adversarial Networks have emerged as a powerful and customizable class of machine learning algorithms within the past half a decade. They learn the distribution of a dataset to generate realistic synthetic samples. It is an active field of research with massive improvements yearly, addressing fundamental limitations of the class and improving on the quality of generated figures. GANs have been successfully applied to music synthesis, face generation, and text-to-image translation.

Within this work, we will look at a variety of GAN architectures and how they compare qualitatively on the famous MNIST dataset. We will explore how differing architectures affect the time of convergence, quality of the resulting images, and complexity in training. The theoretical justifications and shortcomings of each methodology will be explored in detail, such that intuition can be formed on choosing the right architecture for a problem.

Keywords: Computer Vision, Generative Adversarial Networks, Machine Learning, MNIST

I. BACKGROUND

Neural networks (NN) were first developed by Bernard Widrow and Marcian Hoff of Stanford in 1959 under the name of MADALINE (Multiple Adaptive Linear Element) [11]. Neural networks were designed with inspiration taken from biological neurons in human brains. Artificial neurons aggregate information from other neurons and fire off a signal depending on the strengths of previous inputs, which is analogous with how human neurons operate. Neural networks fall into the categorization of supervised learning in artificial intelligence (AI). Under supervised learning, the algorithm needs to be fed in labeled data to make future classifications/predictions. Supervised juxtaposes unsupervised learning, which needs no training data – an example of unsupervised learning would be clustering.

GANs were first proposed by Ian J. Goodfellow in his Ph.D. dissertation in 2014 at the Université de Montréal [4]. The proposed architecture is a dual neural network system, in which a generative model learns to generate realistic samples from distribution to compete against a discriminator that classifies fake images. These models are trained in tandem with one another, both learning from random initialization how to best one another. A successful result of training is when the Nash Equilibrium between the two models is found. Nash Equilibrium occurs when the generator has learned the distribution of the data well enough to the point that the discriminator is only as good as a random chance.

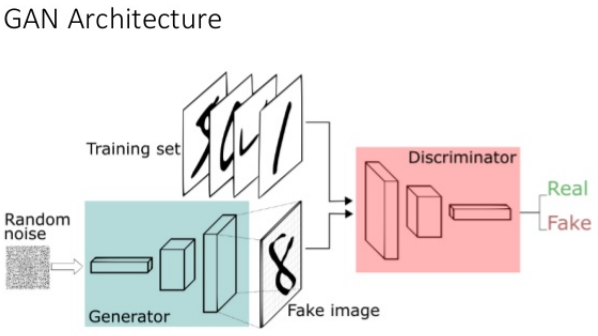


FIG. 1: Architecture of a GAN

Since the advent of GANs in 2014, they have vastly improved and have blown up in the AI research field. State of the art research in GANs is currently focusing on applications in video and voice data.

---

* Submitted as a CSCI-431 assignment at RIT
† jeffery@jrtechs.net, jxr8142@rit.edu
‡ rxm7244@rit.edu
§ ktr5669@rit.edu

A. Applications

GANs have been applied to many problems [8]. A sampling of some of the problems are listed below.

- Image Generation
- Music Generation
- Style Transfer
- Video Prediction
- Super Resolution
- Text to Image

B. Deep Convolutional Generative Adversarial Network

Deep Convolutional Generative Adversarial Networks, DCGAN for short, is an architectural modification on the original GAN, in which the generator and discriminator models are reflections of one another. The makeup of each network is a multi-layer, deep convolutional neural network. The idea behind this architecture is that by reflecting the network structure between the two, the computational capacities of each network to learn their respective tasks are equal [10]. In doing this, it should stabilize competitive learning between the two agents and result in smoother learning, avoiding cases of one network dominance.

C. Wasserstein Generative Adversarial Networks

Wasserstein Generative Adversarial Networks, or WGANs for short, was an improvement on the vanilla GAN proposed by Martin Arjovsky, et al in 2017 [1]. The motivation behind this work is modifying the task of the discriminator in order to stabilize the training between the networks. Instead of having a simple binary classifier that predicts whether an image is real or fake, the discriminator is modified to output the likelihood estimate of the "realness" or "fakeness" of an image. The theoretical idea is that this continuous estimation incentivizes the generator to minimize the distance between the distribution of its generated images and the real images more than the standard discriminator design. Empirically, this design has shown more exceptional results over the standard GAN architecture in terms of training and architecture stability, as well as being more robust to hyper-parameter configurations.

II. GOALS

This project applies three different GAN architectures to generating handwritten images from the MNIST dataset. We are going to compare: vanilla GANs, DCGANs, and WGANs. Using the results of the three different architectures, we wish to judge the performance based on three performance criteria:

- Perceived Quality of Images
- Time required to train
- Training data required

The Modified National Institute of Standards and Technology database (MNIST database) is a dataset comprising of seventy thousand handwritten digits. Sixty thousand of those images are partitioned for training, and the remaining ten thousand are left for testing and validation. We are using the MNIST dataset because it is the de facto standard when it comes to machine learning on images.

A. Research Questions

- Which GAN architecture performs best on the MNIST dataset?
- What are the quantitative differences between these architectures in terms of stability of training, and quality of the results?
- How does required training time and convergence rate differ between GAN architectures?

III. IMPLEMENTATION

We implemented each GAN variety using PyTorch. PyTorch is an open-source machine learning framework. This framework was used due to its popularity in the field and ease of use[9]. Our python implementation can be found on Github in a repository created for this class titled "jrtechs/CSCI-431-final-GANS"[1].

A. Vanilla Generative Adversarial Network

Using boilerplate PyTorch code, we implemented an underlying GAN that uses a generator and discriminator using simple neural networks. We used a Binary Cross-Entropy (BCE) Loss function for the adversarial algorithm [3]. The arching idea of a basic GAN can be observed in figure 1.

[1] `https://github.com/jrtechs/CSCI-431-final-GANs`

B. Deep Generative Adversarial Network

The code to run the DCGAN is identical to the code required to run the regular GAN. The critical difference is that we use different types of neural networks in both implementations. In the GAN, we just used a standard neural network, but in the DCGAN, we used convolutional neural networks where the generator mirrored the discriminator.

C. Wasserstein Generative Adversarial Network

The WGAN implementation was nearly identical to the typical GAN implementation, but, the loss function was changed to be the Wasserstein distance. The key benefit of this is that the loss functions that we are trying to optimize now correlate to image quality.

IV. EXPERIMENTS

This section goes over in-depth the experiments ran in this project and the results produced from them.

A. Data Set

The MNIST database of handwritten digits was used to test the GAN algorithms. The MNIST dataset comprises of seventy thousand handwritten digits already partitioned into a training and test set. The Training set contains sixty thousand images, and ten thousand images are in the test set. This dataset was collected by using approximately 250 writers. Note: the writers in the training and test sets were disjoint from each other.

The MNIST dataset was selected because it is widely used in the field of computer vision and AI. Its popularity makes it an ideal dataset because we can compare our results with the work of other people. Since it is a large set that was used in previous papers, we also know that we could get a really good confidence score if we were solely creating a classifier. However, in this project, we will be generating a discriminator and generator on the most set. Never-less, the dataset has proven by other researchers to be sufficient for use in neural networks. MNIST is ideal to use because images are of a fixed size of 20x20, and images have already been normalized.

The data we used was downloaded from Yann LeCun's website [2].

[2] `http://yann.lecun.com/exdb/mnist/`

TABLE I: This table represents the quality metrics we measured from a random participant.

GAN	WGAN	DCGAN
7	7	10
6	8	10
7	7	10
6	7	10
5	7	10
4	8	10

B. Quality

In this experiment, we aimed to test the quality of the images produced. In this test, we had the GANS generate handwritten digits. After scrambling which GAN produced, which image, we asked a test participant to rank each image on a scale of 1-10 on how it looks. Ten would indicate that it looked like a human drew this digit, and a one would indicate that the image looks terrible. After all the data was collected, we compared which GAN architecture had the best-perceived quality from the participant.

After running all three architectures for 200 epochs, the training metrics indicated that they all were hitting max performance and that more training would be not much more beneficial. We took the last ten sample outputs from each GAN architecture and asked a random participant to rate the handwritten digits on a scale of 1-10; the results are shown in table I. As you can see, the DGGAN outperforms the other two architectures by a large margin.

C. Training

In this experiment, we cut off each GAN after a specif amount of Epochs. We compared the results of the three GAN architectures after a different amount of batches. Note: Each batch contains 64 images. An epoch is when the algorithm has seen all the data in the set. With the MNIST data set, it takes 938 batches to get through all the training data. We sampled after 400 batches, 6000 batches and 187200 batches –200 Epochs. We did 200 epochs because we wanted to see what the algorithm would look like at its best, and we did 400 and 6000 to capture how fast the algorithm learned.

Looking at figure 3, we see that the normal GAN took some time to train and that it looked pretty bad at 400 and 6000 batches but started to look pretty good at 200 epochs.

Looking at figure 4, we can see that the results from 400 and 6000 batches were pretty bad but, the results after 200 epochs look remarkably good.

Looking at figure 5, we notice that training happened remarkably fast. Compared to figure 4 and figure 3, we can observe that the results after 6000 batches

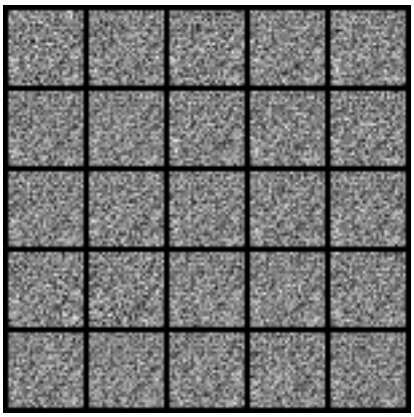


FIG. 2: Output with no training data

looked better than the other two algorithms did after 200 epochs– 187200 batches. The results of the DCGAN after 200 epochs look remarkable and would quickly be passed as human written handwritten digits.

D. Quantity of Training Data

In this experiment, we compare how the GAN algorithms run at different levels of training data from the MNIST set. We compare the GANS using the full training set, and one-sixth of the training data.

The full dataset contained roughly sixty thousand images and took 187200 batches of 64 images to run 200 epochs. The reduced dataset contained ten thousand images and took 31200 batches of 64 images to run 200 epochs.

Figures 3 through 5 show the results of using all the data in the MNIST dataset on 200 epochs. Figure 6 shows the result of the three algorithms at 200 epochs on the data set reduced to one-sixth of the original size. Despite reducing the amount of training data, the DCGAN still performed incredibly well; however, the two other algorithms took a major performance hit.

V. CONCLUSIONS

This project is a useful survey and comparison of three popular GAN architectures. Based on the results, we can conclude that DCGANs make great improvements in terms of what it can do with limited training data and the number of iterations required. DCGANs were also proven to give really crip results.

Future work for this project would entail researching more GAN architectures like Conditional GANS (CGANS), Least Square GANs (LSGAN), Auxiliary Classifier GAN (ACGAN), and Info GANS (infoGAN) [2, 5–7],. Another avenue of research would be to examine how the results of our experiments on the MNIST dataset hold up against different datasets.

Since this is such a new algorithm in the field of Artificial intelligence, people are still actively doing a ton of research in GANs, pushing them at the forefront of cutting edge. As GANs become more widely used in the public and private sectors, we are sure to see a lot more research into the applications of GANs.

VI. ACKNOWLEDGMENT

This was submitted as a RIT CSCI-431 project for professor Sorkunlu's class. Latex files used to generate this report can be found on a Github page created specifically for this project [3].

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein GAN. arXiv:stat.ML/1701.07875

[2] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. 2016. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. *CoRR* abs/1606.03657 (2016). arXiv:1606.03657 http://arxiv.org/abs/1606.03657

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[4] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Networks. arXiv:stat.ML/1406.2661

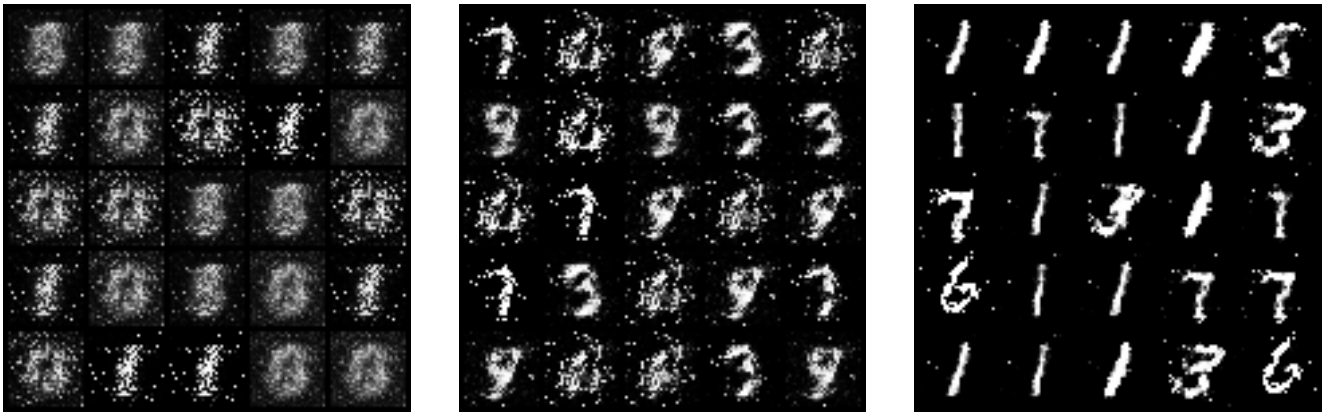
[5] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, and Zhen Wang. 2016. Multi-class Generative Adversarial Networks with the L2 Loss Function. *CoRR* abs/1611.04076 (2016). arXiv:1611.04076 http://arxiv.org/abs/1611.04076

[6] Mehdi Mirza and Simon Osindero. 2014. Conditional Generative Adversarial Nets. *CoRR* abs/1411.1784 (2014). arXiv:1411.1784 http://arxiv.org/abs/1411.1784

[7] Augustus Odena, Christopher Olah, and Jonathon Shlens. 2017. Conditional Image Synthesis with Auxiliary Classifier GANs. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70 (ICML'17)*. JMLR.org, 2642–2651.

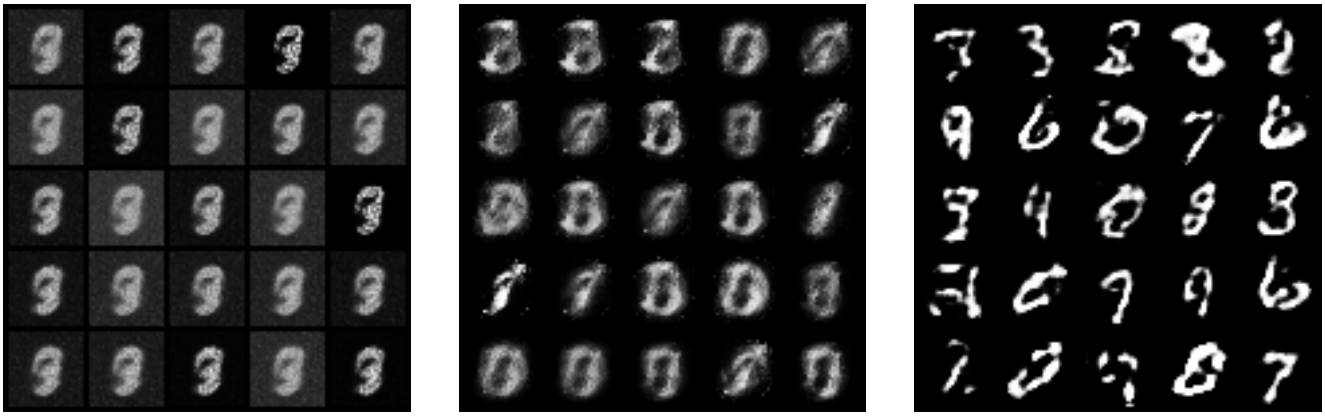
[8] Z. Pan, W. Yu, X. Yi, A. Khan, F. Yuan, and Y. Zheng. 2019. Recent Progress on Generative Adversarial Networks (GANs): A Survey. *IEEE Access* 7 (2019), 36322–36333. https://doi.org/10.1109/ACCESS.2019.2905015

[3] https://github.com/jrtechs/computer-vision-GANs-paper



(a) 400 Batches (b) 6000 Batches (c) 187200 Batches

FIG. 3: GAN Results Sampled at Different Epochs



(a) 400 Batches (b) 6000 Batches (c) 187200 Batches

FIG. 4: WGAN Results Sampled at Different Epochs

(a) 400 Batches (b) 6000 Batches (c) 187200 Batches

FIG. 5: DCGAN Results Sampled at Different Epochs

[9] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).

[10] Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. arXiv:cs.LG/1511.06434

[11] Bernard Widrow. 1962. Generalization and Information Storage in Networks of ADALINE Neurons. Self Organizing Systems. *Yovitz, MC, Jacobi, GT, and Goldstein, GD editors* (1962), 435–461.

(a) GAN (b) WGAN (c) DCGAN

FIG. 6: Results with one Sixth of Training Set and trained for 200 Epochs