Generative Adversarial Networks: A Brief History and Overview

Akhil Gunasekaran[1]

[1]University of California, Santa Cruz

ABSTRACT

Over the past decade, research in the field of Deep Learning has brought about novel improvements in image generation and feature learning; one such example being a Generative Adversarial Network. However, these improvements have been coupled with an increasing demand on mathematical literacy and previous knowledge in the field. Therefore, in this literature review, I seek to introduce Generative Adversarial Networks (GANs) to a broader audience by explaining their background and intuition at a more foundational level. I begin by discussing the mathematical background of this architecture, specifically topics in linear algebra and probability theory. I then proceed to introduce GANs in a more theoretical framework, along with some of the literature on GANs, including their architectural improvements and image-generation capabilities. Finally, I cover state-of-the-art image generation through style-based methods, as well as their implications on society.

Introduction

The purpose of this section is to give a brief introduction to the mathematics and intuition behind Generative Adversarial Networks (GANs). GANs, and Deep Learning Models as a whole, take from a variety of branches of mathematics, including probability theory, multivariable calculus, and linear algebra. Using both mathematics and computation, artificial intelligence (AI) researchers create algorithms which learn meaningful feature representation (patterns) about a dataset; GANs are one of these algorithms.

What makes GANs unique in this regard is their ability to generate novel data samples through a two-player learning and discrimination process. This process facilitates the growth of a generator algorithm and leads to better data sample generation over time. Over the last decade, improvements in GANs have led to better training stability [1], better image generation capability [2], [3], and style-incorporated generation [4]. As a result of this research, computer image generation has seen great strides, so far as piercing mainstream audiences with AI generated artwork and open-source models.

Probability Theory

As its name suggests, the field of probability theory deals with variables that have a random chance of occurring. For example, a coin has a 50% chance of landing on heads and an equal chance of landing on tails. In this scenario, we can define the random variable "coin flip" as x. Moreover, let the instances (possibilities/events) of a random variable be $x_1, x_2, x_3 \ldots\ldots x_n$. The probability of a specific event occurring will be denoted as $P(x = x_1)$, or in other words, the probability of x taking on the value x_1. In Deep Learning, random variables are usually image sample vectors, denoted by a boldfaced $\boldsymbol{x}$.

In cases where the variable x can take on an infinitesimal number of values, i.e., a continuous variable, we model the variable using a probability density function (pdf), denoted by a $p(x)$, with x the variable being distributed. Since $p(\mathrm{x})$ is continuous, we cannot analyze the probability of individual instances of x, rather, we

must integrate $\int_a^b p(\mathrm{x})\ \mathrm{dx}$ over a domain to find an overall probability of x taking on a value in said domain.

When graphing probability distributions in two or more dimensions, we often parameterize the distribution using its mean μ, the average value of the variable, and standard deviation σ, the amount the variable deviates from the center on average. Using these two parameters, we can rewrite a probability distribution as $p(x;\mu,\sigma)$, i.e., the pdf of x parameterized by μ and σ. What does is mean for a distribution to be parameterized? In this scenario, our graph of $p(x)$ depends on the individual values of μ and σ, we cannot properly understand $p(x)$ without the information of μ and σ. Many deep learning models contain hundreds of thousands of individual parameters across an algorithms cycle [5], and optimizing these parameters is often our objective during computation.

Theoretically, one may wish to compare two different PDFs coming from a distribution of data $p(x)$ and the distribution of the model $q(x)$. There are two prominent methods of doing so, the Kullback-Leibler Divergence and the Jensen-Shannon Divergence. The Kullback-Leibler Divergence between two distributions $p(x)$ and $q(x)$ is defined as:

$$D_{KL}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) log(\frac{p(x)}{q(x)}) dx \tag{1}$$

Algorithmically, there are a few issues with KL-divergence. For one, if $p(x)$ and $q(x)$ are dissimilar, the integral evaluates to 0, causing problems in optimization. Moreover, KL Divergence does not preserve symmetry, or in other words:

$$D_{\mathrm{KL}}(P \parallel Q) \neq D_{\mathrm{KL}}(Q \parallel P) \tag{2}$$

To circumvent these issues, researchers may implement the Jensen-Shannon Divergence. One can define the Jensen Shannon Divergence between two probability distributions $p(x)$ and $q(x)$ as:

$$JSD(P \parallel Q) \neq \frac{1}{2} D_{\mathrm{KL}}(P \parallel M) + \frac{1}{2} D_{\mathrm{KL}}(Q \parallel M) \tag{3}$$

Where:

$$M = \frac{1}{2}(P + Q)$$

When $JSD(P \parallel Q) = 0$, it is assumed that $p(x) = q(x)$, since it will zero-out the integral. As we will see later, GANs use an algorithm to minimize the Jensen-Shannon distance between $p(x)$ and $q(x)$, or in a more practical sense, they optimize the model's output to be similar to the distribution of data.

Linear Algebra

The field of linear algebra is crucial for understanding machine learning. Linear algebra allows computer scientists to analyze relationships between variables by represent numbers, images, and other forms of complex data structures as matrices.

Simply, a matrix can be thought of as an array of numbers. The columns of a matrix refer to individual vectors, whereas its rows denote an arbitrary number of individual vectors. (Due to this fact, matrices are often referred to as vectors, specifically image vectors, in Deep Learning literature.) A matrix in m dimensions (rows) with n vectors (columns) is represented as $\mathbf{A}$ belonging to $\mathbb{R}^{\mathrm{m} \times \mathrm{n}}$. Its matrix components a_{ij} belong to the ith row and jth column and are usually scalar values, example:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \dots \\ a_{2,1} & a_{2,2} & \dots \\ \dots & \dots & a_{i,j} \end{bmatrix}. \quad (4)$$

For a more intuitive example, let us look at how to represent an image as a matrix. A square image of $256 \times 256 \times 3$ has exactly 256 rows, 256 columns, and 3 color channels. Each pixel value a_{ij} in the matrix represents the color depth in either red, blue, and green. When we imprint these matrices onto each other, we retrieve our original image.

Neural Networks and Deep Learning

Deep learning and artificial neural networks (ANNs) have been intertwined since the inception of deep learning [6]. ANNs allow for the extraction of feature representations, thereby allowing for the "deep learning" of different data structures, such as images and words. This section aims to cover the architecture of ANNs, specifically how they process, modify, and learn data.

Artificial Neural Networks

Artificial Neural Networks (ANNs) in their most simple form are algorithms designed based on the neurons in our mind [7]. ANNs take input as information and learn patterns (features) by processing the information in “hidden” layers. The "hidden" layers in an ANN are made up of connected neurons, which each process information before sending it to other neurons.

Each neuron connection contains weight parameter. This weight parameter can be interpreted as how much change a neuron incurs on the data; high weight is associated with high change, and low weight value is associated with a low change in the data. This weight is multiplied with the neuron output before being transferred to the next neuron. The ability to find near-optimal weights makes the algorithm very strong, and it is one of the main goals in neural network optimization.

However, before information is sent to the next neuron, both the neuron output and weight must go through an activation function. The activation function determines whether information should be sent to the next neuron. A visualization of this process can be seen in figure 1.

The neurons themselves store probability values between their expected output and actual output from the activation function; worse predictions correlate with more error. The neuron predictions contribute to an “loss function” (or error function), which can be defined for the overall algorithm.

The loss function for an ANN is an overall measure of its ability to learn from the data; generally, higher loss correlates with worse performance and learning. The loss function serves as a guide on how to adjust our weights and/or other parameters, and is typically a non-convex optimization task. There are several methods for weight optimization, but one of the most wideley used is stochastic gradient descent (SGD).

To better understand gradient descent, imagine a mountain range with many different peaks and hills. Different points in this mountain range correlate to different weight values; however, for us to have optimal weight values, we must find the minimum altitude in this mountain range. One possible method would be to simply move down the hill: which is akin to "descending" the "gradient" [8].

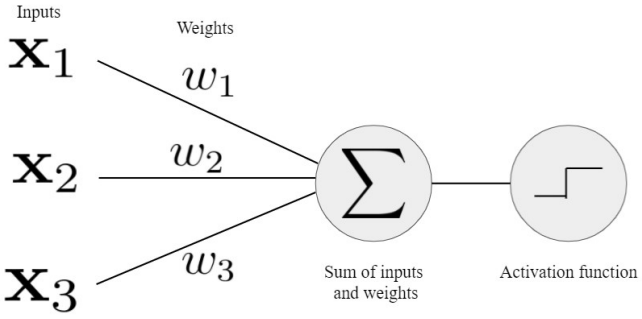

Figure 1. The Rosenblatt Perceptron [7]. Each neuron in a neural network consists of 3 components: the inputs, the weights, and an activation function. The inputs to a neuron consist of multi-dimensional vector information. These vectors are multiplied by their respective weight value which determines their overall impact on the dataset. Finally, the sum of the inputs and weights goes through an activation function which determines if the data should be sent to the next neuron or not. A "0" signals a "do not send," whereas a "1" signals "send."

Convolutional Neural Networks

Early on in their development, ANNs were more abstract in their architecture, and were unable to tackle issues such as image processing. However, since the 21st century, newer architectures have enabled neural networks to excel at image processing tasks, specifically, Convolutional Neural Networks (CNNs) [9].

In the task of image processing, one vital conern is the ability to transform an input image to a vector that is understandable by a neural network. To solve this task, CNNs use convolutional kernels and pooling layers before a network can learn from an image (figure 2). Firstly, the convolutional layer take a specific matrix, known as a filter, and performs a dot product over the span of an entire image. This results in only specific parts of an image correlating with the filter's pattern appearing. The resulting image is then transferred to a pooling layer, which downsizes the image further with different operations. For example, you may take a 2x2 matrix of pixels, and downsize by choosing the highest value of pixel, or average pixel value; this is known as "max pooling" and "average pooling" respectively.

After the convolutional and pooling layers have been performed, the feed-forward neural network recieves a "summarized" version of an image which therby allows it to learn underlying patterns in the image. The network tends to learn more and more patterns the deeper the network layer we are in. For instance, edge detection may come before the detection of objects and small feature attributes, such as strands of hair. In the analysis of CNNs, it is common to denote the feature representation of images as a latent variable $\mathbf{z}$. This variable is referred to as "latent" because us as the designer do not have a priori knowledge of it.

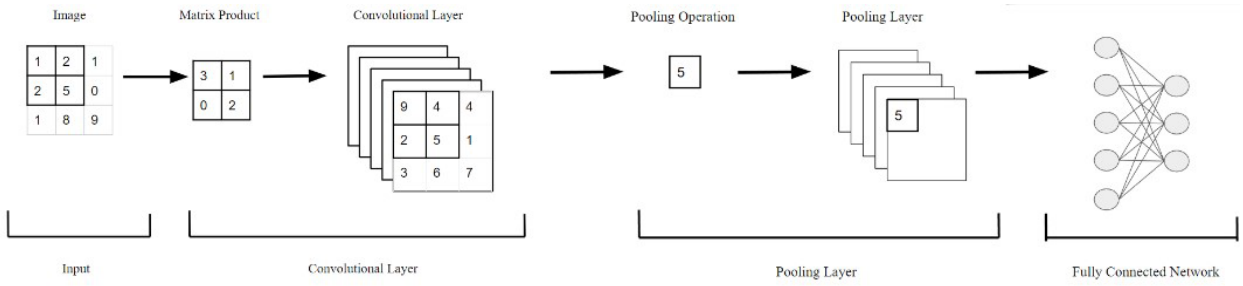

Figure 2. Structure of convolutional neural networks (CNNs). The basic architecture of a CNN includes a convolutional layer, a pooling layer, and a fully-connected neural network (FCN). In this example, we see a portion of a 3x3 image go though this process. The convolutional layer and pooling layer repeat for an arbitrary amount of times before a vector-representation of an image is ready to be processed in the ANN.

Generative Models

Before introducing GANs directly, it is worth going over the broader field of models which GANs belong to: generative models. The goal of generative modeling is to optimize a model $p_{\text{model}}(\boldsymbol{x};\theta)$ trained on a set of data $\boldsymbol{x}$ and parameterized by θ to produce outputs closest to the $p_{\text{data}}(\boldsymbol{x})$ distribution; or in other words, to minimize:

$$\min_{\theta} D_{\text{KL}}(p_{\text{data}}(\boldsymbol{x}) \parallel p_{\text{model}}(\boldsymbol{x};\theta)) \tag{5}$$

When the θ parameter is optimized, the output of our model is ϵ close to $p_{\text{data}}(\boldsymbol{x})$. If our data distribution were a collection of cat images, for instance, we could construct and train a model to produce cat images that look similar to the data. Depending on the model we are analyzing, the θ parameter may take on different roles; in a neural network θ corresponds to the network weights (as a whole) [10]. In reality, data sets can often be incredibly difficult to approximate by simple differentiation and integration, therefore, other methods must be devised to create probability distributions; this is one of the main challenges researchers face in generative modeling [10].

Generative Adversarial Networks

Previously, I have introduced fundamental concepts in probability theory and linear algebra, as well as the architecture behind ANNs and CNNs; an in-depth knowledge of GANs requires a combination of knowledge in these aspects. The following description of GANs is mostly theoretical, as a description of a full implementation is beyond the scope of this paper.

The novel contribution of GANs is a two-player architecture between a generator CNN $\mathbf{G}$ and a discriminator CNN $\mathbf{D}$. One common analogy used to describe these networks is the game between an artwork-counterfeiter and a detective [11]. The goal of the counterfeiter is to produce artwork that is similar to a real piece, whereas the detective must identify the counterfeiter's work as being either real or fake. Much like the detective, the discriminator is trained on a real dataset and learns their patterns prior to analyzing the generator's work. Similarly, the generator begins by producing white noise images from a standard gaussian (images with no information, akin to static noise); this is akin to the counterfeiter attempting to replicate artwork. The images produced by the generator are then fed to the discriminator, which outputs a scalar probability of the image being real (figure 3).

For a more mathematical description, let $\mathbf{X} \in \mathbb{R}^{\text{m}\times\text{n}}$ be the set of all images with dimensionality $\text{m} \times \text{n}$, forming a distribution of images $p_{\text{data}}(\boldsymbol{x})$. This distribution is used to form the discriminator portion of a GAN, defined as being $D(\boldsymbol{x};\theta_{\text{d}})$. Moreover, let the set of all latent variables be $\mathbf{Z} \in \mathbb{R}^{\text{m}\times\text{n}}$ of dimensionality $\text{m} \times \text{n}$ belonging to a gaussian white-noise distribution $p_{\boldsymbol{z}}(\boldsymbol{z})$. This distribution is used to form the generator $G(\boldsymbol{z};\theta_{\text{g}})$, which samples from the gaussian. Altogether, the minimax function can be defined as:

$$\min_{G}\max_{D} V(G,\ D) \ = \ \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}\big[\ln\big(D(\boldsymbol{x})\big)\big] + \ \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\big[\ln\big(1 - D\big(G(\boldsymbol{z})\big)\big)\big]. \tag{6}$$

To better analyze the objective function described above, let us separate the discriminator and the generator. The first portion of the objective function, $\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\ln(D(\boldsymbol{x}))]$, tells us that the discriminator takes an input of real images $\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})$ and seeks to maximize $D(\boldsymbol{x})$, the probability that the images are real. In the second portion of the objective function, $\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\big[\ln\big(1 - D\big(G(\boldsymbol{z})\big)\big)\big]$, we see that minimizing $G(\boldsymbol{z})$ is akin to fooling the discriminator, since this will lead to $D(\boldsymbol{x})$ approaching 0.

This description was quite technical, so it is worth taking a step back and analyzing how a computer achieves this process. The authors of the original GAN paper [12] used a stochastic gradient descent algorithm for both the generator and discriminator. Essentially, the discriminator is given a set of images from the generator and the data pool, and trained to minimize equation 6. After one training cycle, the weights of the discriminator would be updated.

Similarly, the generator would take a sample of latent vectors from white noise images and generate an ouput before having its weights updated. The key to understanding this process is that the decision of the discriminator affects the generator's weights, forcing it to improve, i.e. this process forces the generator to improve its image generation until $p_{\mathrm{g}} = p_{\mathrm{data}}(\boldsymbol{x})$ [10].

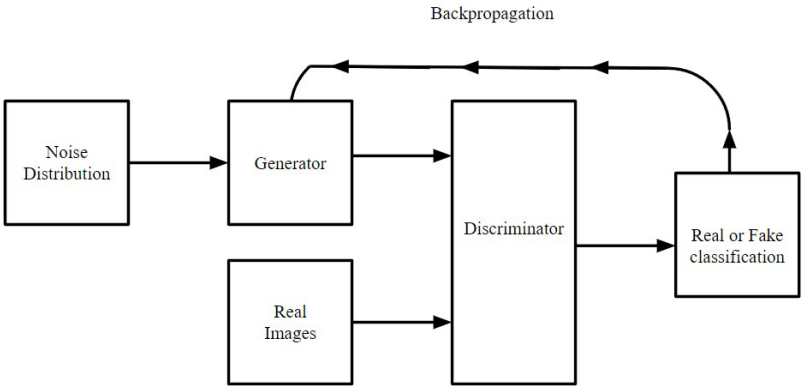

Figure 3. GAN Architecture. A basic GAN architecture consists of a generator and a discriminator which play a minimax game [12]. The generator samples images from a distribution of noise and modifies them, thereby producing a new image. The discriminator is trained on a set of real images, then, it receives the generator's output and must identify the image as being either real or fake with a probability value. Through backpropagation, the updating of neural network weights in a front-to-back order, the output of the discriminator influences how the generator modifies images.

Deep Convolutional GANs

Shortly after the introduction of GANs, their generative capabilities were explored and improved through a variety of research papers. Deep Convolutional GAN (DCGAN) [2], one of the first novel improvements, introduced three key features: replacing the discriminator's pooling layer with convolutional strides, using fractional-convolutions in the generator, and using batch normalization in both networks [13].

Although these architectural improvements may seem overwhelming, they are in fact a simplification of the original GAN framework. More specifically, the strided convolutions used for the discriminator are simply the same stride operation performed in figure 2. Moreover, fractional-strided convolutions add padding to the original input image, allowing for the dot product output to preserve dimensionality.

We previously learned that when data is processed by nodes in a neural network, its weights are shifted by gradient descent. One caveat to this process is that the nodes in a network do not automatically reset after processing data, causing future data to be processed incorrectly. This process is named "internal covariate shift." Batch normalization helps to prevent internal covariate shift by normalizing the inputs at each stage of the network by resetting the variance and mean, leading to faster processing and more stabilized training.

Conditional Image Generation

In the task of image generation, one may wish to generate images belonging to certain categories. For instance, a classical GAN trained on a set of handwritten-digits (MNIST) has no method of controlling the generation of specific digits. To enable conditional image generation, the discriminator and generator of a GAN can be trained on encoded-image-vectors to learn categorical image generation. In other words, we can embed the training data of the generator and discriminator with certain image labels. As a result, the generator learns to generate images of each category it is trained on [14].

There are a multitude of advantages regarding conditional image generation. For one, when obtaining images from a generator, the input of a class label allows for the generation of a specific category of images. This becomes important when GANs are trained on incredibly large datasets with hundreds of categories, such as ImageNet. A classical GAN without image labels has no way to distinguish labels within a dataset, but a conditional GAN can learn each and every category because the generator and discriminator are conditioned on every class label. An example of conditional GAN generation can be seen in figure 4, a collection of categorically generated images using BigGAN [3].

Figure 4. A collection of images generated by BigGAN [3]. BigGAN is trained on a large image dataset, specifically ImageNet [15], and is capable of conditional image generation. The examples here include the categories of wardrobe, radio, beer bottle, Yorkshire terrier, castle, and coffee mug respectively. These examples show the ability of GANs to learn and reconstruct feature representations. Images generated via [16].

GANs and Artwork

Neural Style Transfer

Users of social media sites, such as Instagram and Snapchat, are well familiar with "filters" that can alter one's appearance almost instantaneously. The technology used to make these filters largely originate from style-transfer, a way to mix the style and content of an image using CNNs [17].

We previously learned that a CNN processes images via a convolutional layer, pooling layer, and fully-connected neural-network. To go more in depth, when a processed image is passed through the fully-connected-network (FCN) stage, it is possible to retrieve certain "filter responses" triggered by the image. These filter responses show us how the FCN is learning features in the data. But more importantly, in style-transfer, filter responses are used

to store image information about the "content" of an image; this content stores important information such as position, outlines, placement, etc. Using this logic, the authors of style-transfer proposed sending a white-noise image to adapt the filter-responses of a previously-processed image. This allows the white-noise image to retain the content information of a previous image. As an example, if I were to send an RBG image of a cat through a CNN, I could later pass a white-noise image to adapt the features found in that cat, such as its fur, eyes, etc. Once the content of an image has been extracted, the "style" of a different image can be implanted, an example can be seen in figure 5.

Similarly, to learn the content of an image, an image's style is learned by passing an image and collecting its filter responses in the FCN portion of the CNN. However, instead of directly implanting the filter responses onto a white noise image, a gram matrix of filter responses is first learned. A mathematical description of the gram matrix is beyond the scope of this paper, however, it is important to know that this matrix stores information about the relationship between feature responses between layers in the FCN.

Predictably, once the gram matrix of an image has been extracted, a white noise image is passed. Gradient descent is then performed in order to match the gram matrix of the white-noise image to that of the previously processed image. Notably, a single FCN can store both the content of one image and the style of another image. A single loss function can be computed on a white-noise image, allowing the white-noise image to capture both content and style [17].

One of the problems encountered with style transfer is that it was not possible to arbitrarily apply different styles, meaning that applying a different style would mean retraining the network. In order to solve this issue, Huang and Belongie 2017 introduce Adaptive Instance Normalization, (AdaIN), which normalizes the content and style inputs, and allows for different style inputs to be added during training [18]. The development of AdaIN was key in the development of the style-based generator [4].

Figure 5. The Style Transfer Process. In this example, the content image (Mount Ranier) is combined with the style image (*The Starry Night* by Vincent Van Gogh [19]) to produce an image of Mount Ranier in the style of *The Starry Night* [19]. Images generated via [20].

Style-Based Generation

So far, we have learned about the generative capabilities of GANs, and the artistic capability of Style Transfer. Karras et al. 2019 combine these two algorithms by creating a GAN which is capable of generating images with style. Moreover, they show that StyleGAN produces images of a much higher quality and more diverse images overall.

The architecture behind StyleGAN differs from your standard GAN, in that there are actually two separate neural networks for the generator. One of the neural networks in the generator learns the style of images, whereas the other neural network is a typical generator. The neural network which learns the styles of images is referred to as a "mapping network." Its input, the latent variable $\boldsymbol{z}$ from a set of images, is mapped to the "style-control" variable $\boldsymbol{w}$. The $\boldsymbol{w}$ variable will later be used in the generator network. The generation process begins with a classical generator upsampling a 4x4x512 image via convolutional layers, or in other words, enlarging an image through a

CNN. This image then has noise added to it in order to introduce "stochastic variation." Finally, the style control variable $\boldsymbol{w}$ applies a specific style to the image. This cycle continues twice per image dimension (4x4, 8x8, 16x16, etc), with different styles being added via AdaIN.

Each aspect in StyleGAN's generator contributes to its diversity and quality of images, and I will go over their respective benefits and reasoning here. For one, what is the purpose of the separate mapping network?" The neural network which maps the latent variable $\boldsymbol{z}$ from a set of images to $\boldsymbol{w}$ is a fully connected network whose only purpose is to learn the style of its input images, and then apply these styles, $\boldsymbol{w}$, to the generator. Importantly, $\boldsymbol{w}$ allows for "style mixing" of images created by the other generator. In each convolutional cycle, $\boldsymbol{w}$ is added to the network; this is done before and after convolutional upsampling, allowing for the "mixing" of styles (and more diversity overall). Finally, gaussian noise is added after convolutional upsampling in order to make small changes in images (stochastic variation). This leads to greater image diversities on a micro-level [4].

Related Work

Up until this point, we have seen a mathematical formulation of GANs, improvements to its architecture, and artistic applications. In this section, we will explore other generative models and compare their formulation to GANs, specifically, Boltzmann Machines, Variational Autoencoders, and most recently, Diffusion models.

Restricted Boltzmann Machines

Restricted Boltzmann Machines (RBMs) are one of the earliest examples of generative models. Created well before the invention of convolutional neural networks, RBMs do not have the ability to generate images and pixels, but rather, they are able to estimate a probability distribution. RBMs can be described as a bipartite Markov Random Field of neurons belonging to two classes: hidden units and visible units. In other words, a RBM is a graph-based model that preserves the Markov property, that is, the variable being modeled is independent locally, globally, and conditionally. This model is described as "restricted" because there is no connection between neurons within each of the respective categories [21].

Variational Autoencoders

Unlike GANs, variational autoencoders do not use a generative-discriminative process. Rather, they directly learn patterns in data through learning reconstructions. There are some instances where this can be quite useful. For instance, in the task of inpainting a computer is given an image with a section blacked-out, the computer then attempts to reconstruct the full image.

Firstly, standard Autoencoders work by reducing the dimensions of an input image from training data to latent space, learning a relationship between the data and the latent variable, and then projecting back into image space. These are done in three sections respectively: the encoder, the latent representation, and the decoder. Mathematically, this operation is performed by principal component analysis (PCA). Given an image, the algorithm takes the eigenvalues of the image's covariance matrix to find the best linear subspace of the image, and then analyzes the relationship between variables in that linear subspace. In simpler terms, an autoencoder finds patterns in the data by eliminating variables in an image which do not correlate to a meaningful feature representation. Autoencoders are solely trained on this task, however, the lack of regularization and complexity gives the model too much freedom, causing overfitting.

Variational autoencoders improve upon this framework by replacing the encoder portion of the network with a "latent distribution." Rather than an input image having its latent variable reconstructed, the latent features contribute to a distribution of latent features. Then, a single latent variable is sampled by the network and reconstructed. The

addition of a probability distribution allows for the construction of a KL-divergence based loss function, which is then trained via a neural-network model [22].

Diffusion Models

Recent work in generative modeling has introduced the concept of "diffusion-based models" [23]. Diffusion models work by adding gaussian noise to an image and learning a reversible process. More specifically, noise is added stochastically through a markov chain process to an image at each timestep, and a reverse model must learn a function producing this noise. Therefore, a loss function can be defined as being the mean squared-error between the function prediction noise and the noise applied to the image. Diffusion models derive their name from physics, where they act as a partial-differential-equation for predicting the random movement of particles. In the domain of images, the gaussian noise is analogous the random movement of a particle. Once a neural network has properly learned to reverse this process, it can be trained to produce images. Dhariwal and Nichol 2021 achieved comparable results to state-of-the-art GAN architecture on a variety of metrics, specifically ImageNet [15] and LSUN [24].

Conclusion

In this paper, I introduced the topic of GANs through a mathematical lens by covering foundational concepts in probability theory, linear algebra, and machine learning. A GAN architecture can be described as a model which combines aspects from these fields. More specifically, we have seen that GANs learn image generation through a generator neural network which implicitly minimizes the probability density between the set of real and fake images, which are represented as matrices. In addition, we have seen that GANs are able to learn patterns in large image datasets and create novel images. By covering GAN architecture from a more foundational and mathematical level, we can better understand their novelty, capabilities, and structure.

However, GANs are not without their faults. In their current state, GANs require an incredible amount of computing power and time to achieve results. As a consequence, environmentalists and engineers alike have posed questions regarding the environmental impact of training large machine learning algorithms [25], [26]. In the future, GANs and other generative models could potentially generate images from smaller batches of data, and require much less computational power, helping both the environment and the machine learning practitioner.

Nevertheless, the potential of generative modeling and AI art cannot be understated. In the coming decades, AI generated art trained on many different styles may be competitive with human creativity, potentially displacing artists and other creators. Consequently, this raises some legal questions regarding the ownership of AI art, as the art has no single human creator. Legal and ethical challenges, such as these, must be tackled by our generation in the near future, and understanding the way these algorithms function is critical to solving these dilemmas.

References

[1] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in International conference on machine learning, pp. 214–223, PMLR, 2017. http://proceedings.mlr.press/v70/arjovsky17a/arjovsky17a.pdf

[2] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," arXiv preprint arXiv:1511.06434, 2015. http://doi.org/10.1007/978-3-319-71589-6_9

[3] A. Brock, J. Donahue, and K. Simonyan, "Large scale gan training for high fidelity natural image synthesis," arXiv preprint arXiv:1809.11096, 2018. https://arxiv.org/pdf/1809.11096.pdf

[4] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 4401–4410, 2019. http://doi.org/10.1109/CVPR.2019.00453

[5] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., "Language models are few-shot learners," Advances in neural information processing systems, vol. 33, pp. 1877–1901, 2020. https://arxiv.org/pdf/2005.14165.pdf

[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," nature, vol. 521, no. 7553, pp. 436–444, 2015. http://doi.org/10.1038/nature14539

[7] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," Psychological review, vol. 65, no. 6, p. 386, 1958. http://doi.org/10.1037/h0042519

[8] M. Minsky, "Steps toward artificial intelligence," Proceedings of the IRE, vol. 49, no. 1, pp. 8–30, 1961. http://doi.org/10.1109/JRPROC.1961.287775

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," Communications of the ACM, vol. 60, no. 6, pp. 84–90, 2017. https://doi.org/10.1145/3065386

[10] I. Goodfellow, Y. Bengio, and A. Courville, Deep learning. MIT press, 2016 http://deeplearningbook.org

[11] I. Goodfellow, "Nips 2016 tutorial: Generative adversarial networks," arXiv preprint arXiv:1701.00160, 2016. https://arxiv.org/pdf/1701.00160.pdf

[12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," Communications of the ACM, vol. 63, no. 11, pp. 139–144, 2020. https://doi.org/10.1145/3422622

[13] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in International conference on machine learning, pp. 448–456, PMLR, 2015. https://dl.acm.org/doi/10.5555/3045118.3045167

[14] M. Mirza and S. Osindero, "Conditional generative adversarial nets," arXiv preprint arXiv:1411.1784, 2014. https://arxiv.org/pdf/1411.1784.pdf

[15] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in 2009 IEEE conference on computer vision and pattern recognition, pp. 248–255, Ieee, 2009. https://doi.org/10.1109/CVPR.2009.5206848

[16] DeepMind, https://tfhub.dev/deepmind/biggan-deep-256/1

[17] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," arXiv preprint arXiv:1508.06576, 2015. https://arxiv.org/pdf/1508.06576.pdf, https://doi.org/10.1167/16.12.326

[18] X. Huang and S. Belongie, "Arbitrary style transfer in real-time with adaptive instance normalization," In Proceedings of the IEEE international conference on computer vision, pp. 1501–1510, 2017. http://doi.org/10.1109/ICCV.2017.167

[19] V. van Gogh, "The starry night." 1889, https://www.moma.org/collection/works/79802

[20] R. Nakano, "Arbitrary Style Transfer in the Browser." https://reiinakano.com/arbitrary-image-stylization-tfjs/

[21] G. E. Hinton, "A practical guide to training restricted boltzmann machines," in Neural networks: Tricks of the trade, pp. 599–619, Springer, 2012. https://doi.org/10.1007/978-3-642-35289-8_32

[22] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," arXiv preprint arXiv:1312.6114, 2013. https://arxiv.org/pdf/1312.6114.pdf

[23] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," Advances in Neural Information Processing Systems, vol. 34, pp. 8780–8794, 2021. https://papers.nips.cc/paper/2021/file/49ad23d1ec9fa4bd8d77d02681df5cfa-Paper.pdf

[24] F. Yu, A. Seff, Y. Zhang, S. Song, T. Funkhouser, and J. Xiao, "Lsun: Construction of a large-scale image dataset using deep learning withhumans in the loop," arXiv preprint arXiv:1506.03365, 2015. https://arxiv.org/pdf/1506.03365.pdf

[25] E. Strubell, A. Ganesh, and A. McCallum, “Energy and policy considerations for deep learning in nlp,” arXiv preprint arXiv:1906.02243, 2019. https://arxiv.org/pdf/1906.02243.pdf

[26] A. Lacoste, A. Luccioni, V. Schmidt, and T. Dandres, “Quantifying the carbon emissions of machine learning,” arXiv preprint arXiv:1910.09700, 2019. https://arxiv.org/pdf/1910.09700.pdf