DOI:10.1145/3422622

Generative Adversarial Networks

By Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu,
David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio

Abstract

Generative adversarial networks are a kind of artificial intelligence algorithm designed to solve the *generative modeling* problem. The goal of a generative model is to study a collection of training examples and learn the probability distribution that generated them. Generative Adversarial Networks (GANs) are then able to generate more examples from the estimated probability distribution. Generative models based on deep learning are common, but GANs are among the most successful generative models (especially in terms of their ability to generate realistic high-resolution images). GANs have been successfully applied to a wide variety of tasks (mostly in research settings) but continue to present unique challenges and research opportunities because they are based on game theory while most other approaches to generative modeling are based on optimization.

1. INTRODUCTION

Most current approaches to developing artificial intelligence are based primarily on machine learning. The most widely used and successful form of machine learning to date is supervised learning. Supervised learning algorithms are given a dataset of pairs of example inputs and example outputs. They learn to associate each input with each output and thus learning a mapping from input to output examples. The input examples are typically complicated data objects like images, natural language sentences, or audio waveforms, while the output examples are often relatively simple. The most common kind of supervised learning is classification, where the output is just an integer code identifying a specific category (a photo might be recognized as coming from category 0 containing cats, or category 1 containing dogs, etc.).

Supervised learning is often able to achieve greater than human accuracy after the training process is complete, and thus has been integrated into many products and services. Unfortunately, the learning process itself still falls far short of human abilities. Supervised learning by definition relies on a human supervisor to provide an output example for each input example. Worse, existing approaches to supervised learning often require millions of training examples to exceed human performance, when a human might be able to learn to perform the task acceptably from a very small number of examples.

In order to reduce both the amount of human supervision required for learning and the number of examples required for learning, many researchers today study *unsupervised learning*, often using *generative models*. In this overview paper, we describe one particular approach to unsupervised learning via generative modeling called *generative adversarial networks*. We briefly review applications of GANs and identify core research problems related to convergence in games necessary to make GANs a reliable technology.

2. GENERATIVE MODELING

The goal of supervised learning is relatively straightforward to specify, and all supervised learning algorithms have essentially the same goal: learn to accurately associate new input examples with the correct outputs. For instance, an object recognition algorithm may associate a photo of a dog with some kind of DOG category identifier.

Unsupervised learning is a less clearly defined branch of machine learning, with many different unsupervised learning algorithms pursuing many different goals. Broadly speaking, the goal of unsupervised learning is to learn something useful by examining a dataset containing unlabeled input examples. Clustering and dimensionality reduction are common examples of unsupervised learning.

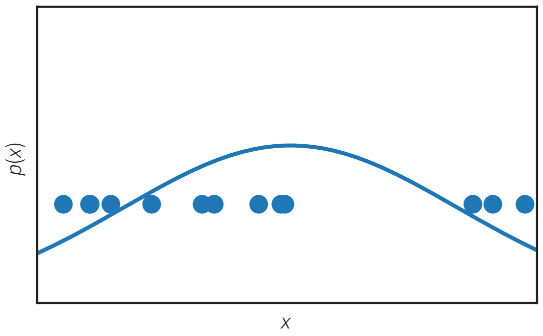
Another approach to unsupervised learning is generative modeling. In generative modeling, training examples $\boldsymbol{x}$ are drawn from an unknown distribution $p_{\text{data}}(\boldsymbol{x})$. The goal of a generative modeling algorithm is to learn a $p_{\text{model}}(\boldsymbol{x})$ that approximates $p_{\text{data}}(\boldsymbol{x})$ as closely as possible.

A straightforward way to learn an approximation of p_{data} is to explicitly write a function $p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta})$ controlled by parameters $\boldsymbol{\theta}$ and search for the value of the parameters that makes p_{data} and p_{model} as similar as possible. In particular, the most popular approach to generative modeling is probably *maximum likelihood estimation*, consisting of minimizing the Kullback-Leibler divergence between p_{data} and p_{model}. The common approach of estimating the mean parameter of a Gaussian distribution by taking the mean of a set of observations is one example of maximum likelihood estimation. This approach based on explicit density functions is illustrated in Figure 1.

Explicit density modeling has worked well for traditional statistics, using simple functional forms of probability distributions, usually applied to small numbers of variables. More recently, with the rise of machine learning in general and deep learning in particular, researchers have become interested in learning models that make use of relatively complicated functional forms. When a deep neural network is used to generate data, the corresponding density function may be computationally intractable. Traditionally, there have been two dominant approaches to confronting this intractability problem: (1) carefully design the model to have a tractable density function (e.g., Frey[11]) and (2) design a learning algorithm based on

The original version of this paper is entitled "Generative Adversarial Networks" and was published in *Advances in Neural Information Processing Systems 27* (NIPS 2014).

Figure 1. Many approaches to generative modeling are based on *density estimation*: observing several training examples of a random variable x and inferring a density function $p(x)$ that generates the training data. This approach is illustrated here, with several data points on a real number line used to fit a Gaussian density function that explains the observed samples. In contrast to this common approach, GANs are *implicit models* that infer the probability distribution $p(x)$ without necessarily representing the density function explicitly.



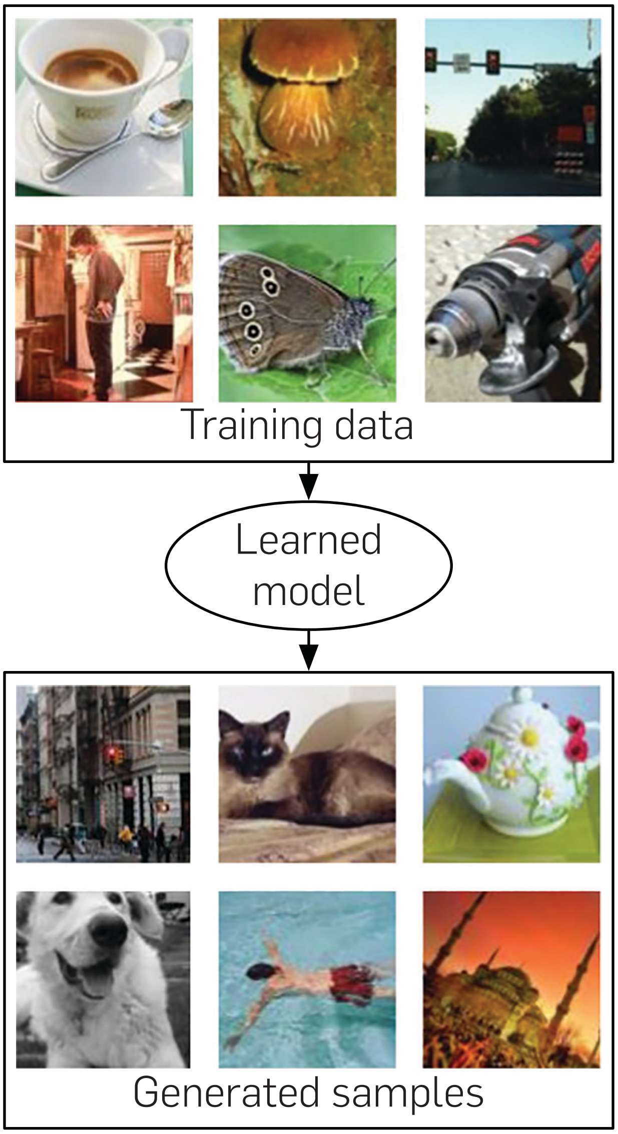
a computationally tractable approximation of an intractable density function (e.g., Kingma and Welling[15]). Both approaches have proved difficult, and for many applications, such as generating realistic high resolution images, researchers remain unsatisfied with the results so far. This motivates further research to improve these two paths, but also suggests that a third path could be useful.

Besides taking a point $\boldsymbol{x}$ as input and returning an estimate of the probability of generating that point, a generative model can be useful if it is able to generate a sample from the distribution p_{model}. This is illustrated in Figure 2. Many models that represent a density function can also generate samples from that density function. In some cases, generating samples is very expensive or only approximate methods of generating samples are tractable.

Some generative models avoid the entire issue of designing a tractable density function and learn only a tractable sample generation process. These are called *implicit generative models*. GANs fall into this category. Prior to the introduction of GANs, the state of the art deep implicit generative model was the *generative stochastic network*[4] which is capable of approximately generating samples via an incremental process based on Markov chains. GANs were introduced in order to create a deep implicit generative model that was able to generate true samples from the model distribution in a single generation step, without need for the incremental generation process or approximate nature of sampling Markov chains.

Today, the most popular approaches to generative modeling are probably GANs, variational autoencoders,[15] and fully-visible belief nets (e.g., Frey[11, 26]). None of these approaches relies on Markov chains, so the reason for the interest in GANs today is not that they succeeded at their original goal of generative modeling without Markov chains, but rather that they have succeded in generating high-quality images and have proven useful for several tasks other than straightforward generation, as described in Section 5.

Figure 2. The goal of many generative models, as illustrated here, is to study a collection of training examples, then learn to generate more examples that come from the same probability distribution. GANs learn to do this without using an explicit representation of the density function. One advantage of the GAN framework is that it may be applied to models for which the density function is computationally intractable. The samples shown here are all samples from the ImageNet dataset,[8] including the ones labeled "model samples." We use actual ImageNet data to illustrate the goal that a hypothetical perfect model would attain.



3. GENERATIVE ADVERSARIAL NETWORKS

Generative adversarial networks are based on a game, in the sense of game theory, between two machine learning models, typically implemented using neural networks.

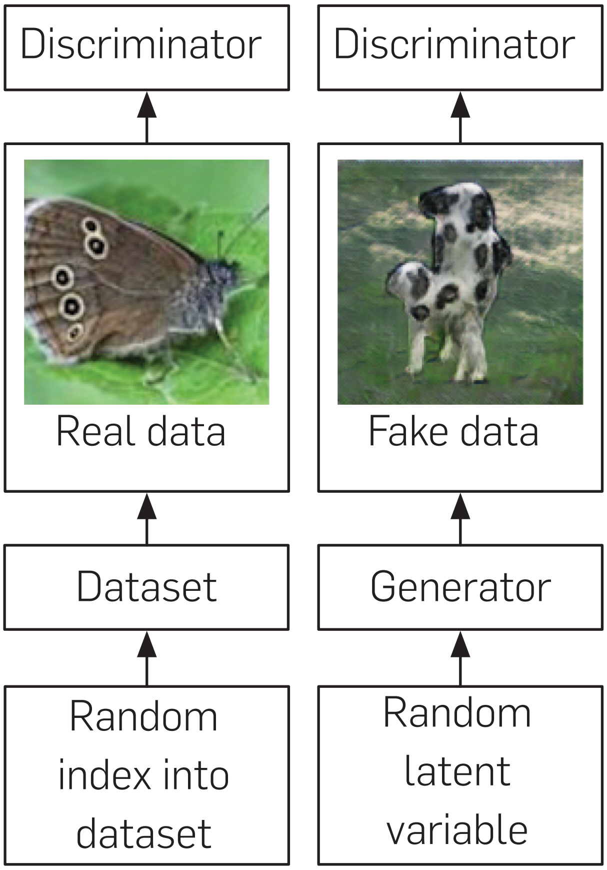
One network called the *generator* defines $p_{\text{model}}(\boldsymbol{x})$ implicitly. The generator is not necessarily able to evaluate the density function p_{model}. For some variants of GANs, evaluation of the density function is possible (any tractable density model for which sampling is tractable and differntiable could be trained as a GAN generator, as done by Danihelka et al.[6]), but this is not required. Instead, the generator is able to draw samples from the distribution p_{model}. The generator is defined by a prior distribution $p(\boldsymbol{z})$ over a vector $\boldsymbol{z}$ that serves as input to the *generator function* $G(\boldsymbol{z}; \boldsymbol{\theta}^{(G)})$ where $\boldsymbol{\theta}^{(G)}$ is a set of learnable parameters defining the generator's strategy in the game. The input vector $\boldsymbol{z}$ can be thought of as a source of randomness in an otherwise deterministic system, analogous to the seed of pseudorandom number generator. The prior distribution $p(\boldsymbol{z})$ is typically a relatively unstructured distribution, such as a high-dimensional Gaussian distribution or a uniform distribution over a hypercube. Samples $\boldsymbol{z}$ from this distribution are then just noise. The main role of the generator is to learn the function $G(\boldsymbol{z})$ that transforms such unstructured noise $\boldsymbol{z}$ into realistic samples.

The other player in this game is the *discriminator*. The discriminator examines samples $\boldsymbol{x}$ and returns some estimate $D(\boldsymbol{x}; \boldsymbol{\theta}^{(D)})$ of whether $\boldsymbol{x}$ is real (drawn from the training distribution) or fake (drawn from p_{model} by running the generator). In the original formulation of GANs, this estimate consists of a probability that the input is real rather than fake assuming that the real distribution and fake distribution are sampled equally often. Other formulations (e.g., Arjovsky et al.[1]) exist but generally speaking, at the level of verbal, intuitive descriptions, the discriminator tries to predict whether the input was real or fake.

Each player incurs a cost: $J^{(G)}(\boldsymbol{\theta}^{(G)}, \boldsymbol{\theta}^{(D)})$ for the generator and $J^{(D)}(\boldsymbol{\theta}^{(G)}, \boldsymbol{\theta}^{(D)})$ for the discriminator. Each player attempts to minimize its own cost. Roughly speaking, the discriminator's cost encourages it to correctly classify data as real or fake, while the generator's cost encourages it to generate samples that the discriminator incorrectly classifies as real. Very many different specific formulations of these costs are possible and so far most popular formulations seem to perform roughly the same.[18] In the original version of GANs, $J^{(D)}$ was defined to be the negative log-likelihood that the discriminator assigns to the real-vs-fake labels given the input to the discriminator. In other words, the discriminator is trained just like a regular binary classifier. The original work on GANs offered two versions of the cost for the generator. One version, today called *minimax GAN* (M-GAN) defined a cost $J^{(G)} = -J^{(D)}$, yielding a minimax game that is straightforward to analyze theoretically. M-GAN defines the cost for the generator by flipping the sign of the discriminator's cost; another approach is the *non-saturating GAN* (NS-GAN), for which the generator's cost is defined by flipping the discriminator's *labels*. In other words, the generator is tried to minimize the negative log-likelihood that the discriminator assigns to the *wrong* labels. The later helps to avoid gradient saturation while training the model.

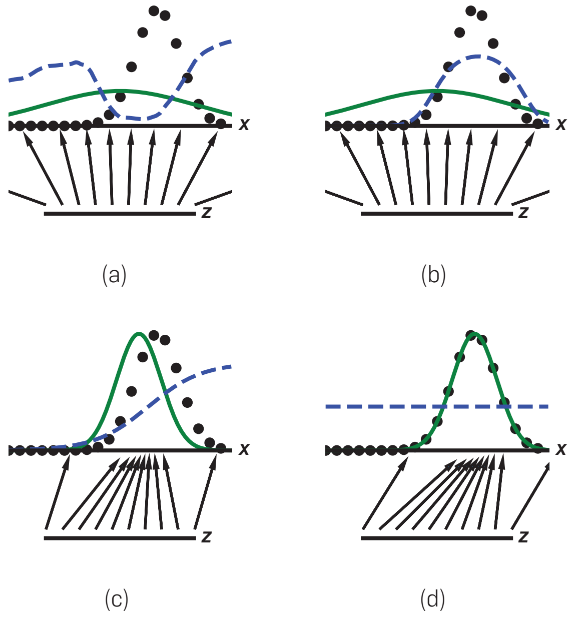
We can think of GANs as a bit like counterfeiters and police: the counterfeiters make fake money while the police try to arrest counterfeiters and continue to allow the spending of legitimate money. Competition between counterfeiters and police leads to more and more realistic counterfeit money until eventually the counterfeiters produce perfect fakes and the police cannot tell the difference between real and fake money. One complication to this analogy is that the generator learns via the discriminator's gradient, as if the counterfeiters have a mole among the police reporting the specific methods that the police use to detect fakes.

Figure 3. Training GANs involves training both a generator network and a discriminator network. The process involves both real data drawn from a dataset and fake data created continuously by the generator throughout the training process. The discriminator is trained much like any other classifier defined by a deep neural network. As shown on the left, the discriminator is shown data from the training set. In this case, the discriminator is trained to assign data to the "real" class. As shown on the right, the training process also involves fake data. The fake data is constructed by first sampling a random vector *z* from a prior distribution over latent variables of the model. The generator is then used to to produce a sample *x* = *G*(*z*). The function *G* is simply a function represented by a neural network that transforms the random, unstructured *z* vector into structured data, intended to be statistically indistinguishable from the training data. The discriminator then classifies this fake data. The discriminator is trained to assign this data to the "fake" class. The backpropagation algorithm makes it possible to use the derivatives of the discriminator's output with respect to the discriminator's input to train the generator. The generator is trained to fool the discriminator, in other words, to make the discriminator assign its input to the "real" class. The training process for the discriminator is thus much the same as for any other binary classifier with the exception that the data for the "fake" class comes from a distribution that changes constantly as the generator learns rather than from a fixed distribution. The learning process for the generator is somewhat unique, because it is not given specific targets for its output, but rather simply given a reward for producing outputs that fool its (constantly changing) opponent.



This process is illustrated in Figure 3. Figure 4 shows a cartoon giving some intution for how the process works.

Figure 4. An illustration of the basic intuition behind the GAN training process, illustrated by fitting a 1-D Gaussian distribution. In this example, we can understand the goal of the generator as learning a simple scaling of the inverse cumulative distribution function of the data generating distribution. GANs are trained by simultaneously updating the discriminator function (*D*, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) p_x from those of the generative distribution p_{model} (green, solid line). The lower horizontal line is the domain from which *z* is sampled, in this case uniformly. The horizontal line above is part of the domain of *x*. The upward arrows show how the mapping $x = G(z)$ imposes the non-uniform distribution p_{model} on transformed samples. *G* contracts in regions of high density and expands in regions of low density of p_{model}. (a) Consider a pair of adversarial networks at initialization: p_{model} is initialized to a unit Gaussian for this example while *D* is defined by a randomly initialized deep neural network. (b) Suppose that *D* were trained to convergence while *G* were held fixed. In practice, both are trained simultaneously, but for the purpose of building intuition, we see that if *G* were fixed, *D* would converge to $D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_{\text{model}}(x)}$. (c) Now suppose that we gradually train both *G* and *D* for a while. The samples *x* generated by *G* flow in the direction of increasing *D* in order to arrive at regions that are more likely to be classified as data. Meanwhile the estimate of *D* is updated in response to this update in *G*. (d) At the Nash equilibrium, neither player can improve its payoff because $p_{\text{model}} = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, that is, $D(x) = \frac{1}{2}$. This constant function shows that all points are equally likely to have come from either distribution. In practice, *G* and *D* are typically optimized with simultaneous gradient steps, and it is not necessary for *D* to be optimal at every step as shown in this intuitive cartoon. See Refs. Fedus et al.[10] and Nagarajan and Kolter[24] for more realistic discussions of the GAN equilibration process.



The situation is not straightforward to model as an optimization problem because each player's cost is a function of the other player's parameters, but each player may control only its own parameters. It is possible to reduce the situation to optimization, where the goal is to minimize

$$J^{(G)}\left(\theta^{(G)}, \underset{\theta^{(D)}}{\arg\min}\, J^{(D)}\left(\theta^{(G)}, \theta^{(D)}\right)\right),$$

as demonstrated by Metz et al.,[22] but the argmin operation is difficult to work with in this way. The most popular approach is to regard this situation as a game between two players. Much of the game theory literature is concerned with games that have discrete and finite action spaces, convex losses, or other properties simplifying them. GANs require use of game theory in settings that are not yet well-explored, where the costs are non-convex and the actions and policies are continuous and high-dimensional (regardless of whether we consider an action to be choosing a specific parameter vector $\theta^{(G)}$ or whether we consider the action to be generating a sample $\boldsymbol{x}$). The goal of a machine learning algorithm in this context is to find a *local Nash equilibrium*[28]: a point that is a local minimum of each player's cost with respect to that player's parameters. With local moves, no player can reduce its cost further, assuming the other player's parameters do not change.

The most common training algorithm is simply to use a gradient-based optimizer to repeatedly take simultaneous steps on both players, incrementally minimizing each player's cost with respect to that player's parameters.

At the end of the training process, GANs are often able to produce realistic samples, even for very complicated datasets containing high-resolution images. An example is shown in Figure 5.

At a high level, one reason that the GAN framework is succesful may be that it involves very little approximation. Many other approaches to generative modeling must approximate an intractable density functions. GANs do not involve any

Figure 5. This image is a sample from a Progressive GAN[14] depicting a person who does not exist but was "imagined" by a GAN after training on photos of celebrities.

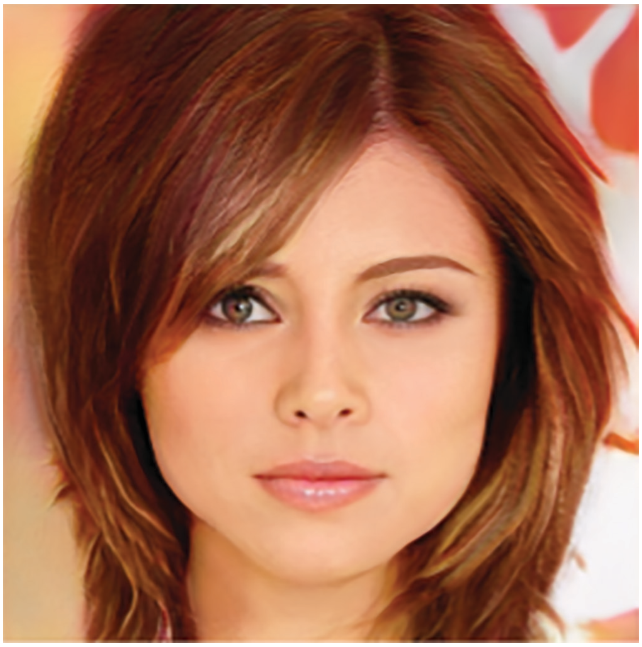
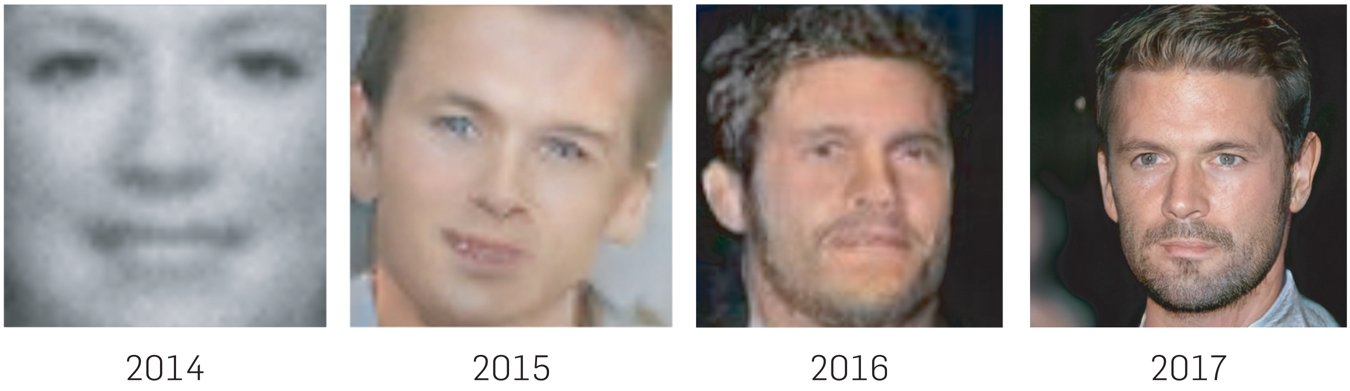


Figure 6. An illustration of progress in GAN capabilities over the course of approximately three years following the introduction of GANs. GANs have rapidly become more capable, due to changes in GAN algorithms, improvements to the underlying deep learning algorithms, and improvements to underlying deep learning software and hardware infrastructure. This rapid progress means that it is infeasible for any single document to summarize the state-of-the-art GAN capabilities or any specific set of best practices; both continue to evolve rapidly enough that any comprehensive survey quickly becomes out of date. Figure reproduced with permission from Brundage et al.[5] The individual results are from Refs. Goodfellow,[13] Karras et al.,[14] Liu and Tuzel,[17] and Radford et al.[27] respectively.



approximation to their true underlying task. The only real error is the statistical error (sampling of a finite amount of training data rather than measuring the true underlying data-generating distribution) and failure of the learning algorithm to converge to exactly the optimal parameters. Many generative modeling strategies would introduce these sources of error and also further sources of approximation error, based on Markov chains, optimization of bounds on the true cost rather than the cost itself, etc.

It is difficult to give much further specific guidance regarding the details of GANs because GANs are such an active research area and most specific advice quickly becomes out of date. Figure 6 shows how quickly the capabilities of GANs have progressed in the years since their introduction.

4. CONVERGENCE OF GANS

The central theoretical results presented in the original GAN paper[13] were that:

1. in the space of density functions p_{model} and discriminator functions D, there is only one local Nash equilibrium, where $p_{\text{model}} = p_{\text{data}}$.
2. if it were possible to optimize directly over such density functions, then the algorithm that consists of optimizing D to convergence in the inner loop, then making a small gradient step on p_{model} in the outer loop, converges to this Nash equilibrium.

However, the theoretical model of local moves directly in density function space may not be very relevant to GANs as they are trained in practice: using local moves in *parameter* space of the *generator* function, among the set of functions representable by neural networks with a finite number of parameters, with each parameter represented with a finite number of bits.

In many different theoretical models, it is interesting to study whether a Nash equilibrium exists,[2] whether any spurious Nash equilibria exist,[32] whether the learning algorithm converges to a Nash equilibrium,[24] and if it does so, how quickly.[21]

In many cases of practical interest, these theoretical questions are open, and the best learning algorithms seem empirically to often fail to converge. Theoretical work to answer these questions is ongoing, as is work to design better costs, models, and training algorithms with better convergence properties.

5. OTHER GAN TOPICS

This article is focused on a summary of the core design considerations and algorithmic properties of GANs.

Many other topics of potential interest cannot be considered here due to space consideration. This article discussed using GANs to approximate a distribution $p(\boldsymbol{x})$ they have also been extended to the conditional setting[23, 25] where they generate samples corresponding to some input by drawing samples from the conditional distribution $p(\boldsymbol{x} \mid \boldsymbol{y})$. GANs are related to moment matching[16] and optimal transport.[1] A quirk of GANs that is made especially clear through their connection to MMD and optimal transport is that they may be used to train generative models for which p_{model} has support only on a thin manifold and may actually assign zero likelihood to the training data. GANs struggle to generate discrete data because the back-propagation algorithm needs to propagate gradients from the discriminator through the output of the generator, but this problem is being gradually resolved.[9] Like most generative models, GANs can be used to fill in gaps in missing data.[34] GANs have proven very effective for learning to classify data using very few labeled training examples.[29] Evaluating the performance of generative models including GANs is a difficult research area in its own right.[29, 31, 32, 33] GANs can be seen as a way for machine learning to learn its own cost function, rather than minimizing a hand-designed cost function. GANs can be seen as a way of supervising machine learning by asking it to produce any output that the machine learning algorithm itself recognizes as acceptable, rather than by asking it to produce a specific example output. GANs are thus great for learning in situations where there are many possible correct answers, such as predicting the many possible futures that can happen in video generation.[19] GANs and GAN-like models can be used to learn to transform data from one domain into data from another domain, even without any labeled pairs of examples from those domains (e.g., Zhu et al.[35]). For example, after studying a collection of photos of zebras and a collection of photos of horses, GANs can turn a photo of a horse into a photo of a zebra.[35] GANs have been used in science to simulate experiments that would be costly to run even in traditional software simulators.[7] GANs can be used to create fake data to train other machine learning models, either when real data would be hard to acquire[30] or when there would be privacy concerns associated with real data.[3] GAN-like models called domain-adversarial networks can be used for domain adaptation.[12] GANs can be used for a variety of interactive digital media effects where the end goal is to produce compelling imagery.[35] GANs can even be used to solve variational inference problems used in other approaches to generative modeling.[20] GANs can learn useful embedding vectors and discover concepts like gender of human faces without supervision.[27]

6. CONCLUSION

GANs are a kind of generative model based on game theory. They have had great practical success in terms of generating realistic data, especially images. It is currently still difficult to train them. For GANs to become a more reliable technology, it will be necessary to design models, costs, or training algorithms for which it is possible to find good Nash equilibria consistently and quickly.

References

1. Arjovsky, M., Chintala, S., Bottou, L. Wasserstein gan. *arXiv preprint arXiv:1701.07875* (2017).
2. Arora, S., Ge, R., Liang, Y., Ma, T., Zhang, Y. Generalization and equilibrium in generative adversarial nets (gans). *arXiv preprint arXiv:1703.00573* (2017).
3. Beaulieu-Jones, B.K., Wu, Z.S., Williams, C., Greene, C.S. Privacy-preserving generative deep neural networks support clinical data sharing. *bioRxiv* (2017), 159756.
4. Bengio, Y., Thibodeau-Laufer, E., Alain, G., Yosinski, J. Deep generative stochastic networks trainable by backprop. In *ICML'2014* (2014).
5. Brundage, M., Avin, S., Clark, J., Toner, H., Eckersley, P., Garfinkel, B., Dafoe, A., Scharre, P., Zeitzoff, T., Filar, B., Anderson, H., Roff, H., Allen, G.C., Steinhardt, J., Flynn, C., hÉigeartaigh, S.Ó., Beard, S., Belfield, H., Farquhar, S., Lyle, C., Crootof, R., Evans, O., Page, M., Bryson, J., Yampolskiy, R., Amodei, D. The Malicious Use of Artificial Intelligence: Forecasting, Prevention, and Mitigation. *ArXiv e-prints* (Feb. 2018).
6. Danihelka, I., Lakshminarayanan, B., Uria, B., Wierstra, D., Dayan, P. Comparison of maximum likelihood and GAN-based training of real nvps. *arXiv preprint arXiv:1705.05263* (2017).
7. de Oliveira, L., Paganini, M., Nachman, B. Learning particle physics by example: location-aware generative adversarial networks for physics synthesis. *Computing and Software for Big Science 1* 1(2017), 4.
8. Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09* (2009).
9. Fedus, W., Goodfellow, I., Dai, A.M. MaskGAN: Better text generation via filling in the ______. In *International Conference on Learning Representations* (2018).
10. Fedus, W., Rosca, M., Lakshminarayanan, B., Dai, A.M., Mohamed, S., Goodfellow, I. Many paths to equilibrium: GANs do not need to decrease a divergence at every step. In *International Conference on Learning Representations* (2018).
11. Frey, B.J. *Graphical Models for Machine Learning and Digital Communication.* MIT Press, Boston, 1998.
12. Ganin, Y., Lempitsky, V. Unsupervised domain adaptation by backpropagation. In *International Conference on Machine Learning* (2015), 1180–1189.
13. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y. Generative adversarial nets. Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, K.Q. Weinberger, eds. *Advances in Neural Information Processing Systems 27,* Curran Associates, Inc., Boston, 2014, 2672–2680.
14. Karras, T., Aila, T., Laine, S., Lehtinen, J. Progressive growing of GANs for improved quality, stability, and variation. *CoRR*, abs/1710.10196 (2017).
15. Kingma, D.P., Welling, M. Auto-encoding variational bayes. In *Proceedings of the International Conference on Learning Representations (ICLR)* (2014).
16. Li, Y., Swersky, K., Zemel, R.S. Generative moment matching networks. *CoRR*, abs/1502.02761 (2015).
17. Liu, M.-Y., Tuzel, O. Coupled generative adversarial networks. D.D. Lee, M. Sugiyama, U.V. Luxburg, I. Guyon, R. Garnett, eds. *Advances in Neural Information Processing Systems 29,* Curran Associates, Inc., Boston, 2016, 469–477.
18. Lucic, M., Kurach, K., Michalski, M., Gelly, S., Bousquet, O. Are GANs created equal? a large-scale study. *arXiv preprint arXiv:1711.10337* (2017).
19. Mathieu, M., Couprie, C., LeCun, Y. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440* (2015).
20. Mescheder, L., Nowozin, S., Geiger, A. Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks. *arXiv preprint arXiv:1701.04722* (2017).
21. Mescheder, L., Nowozin, S., Geiger, A. The numerics of gans. In *Advances in Neural Information Processing Systems* (2017), 1823–1833.
22. Metz, L., Poole, B., Pfau, D., Sohl-Dickstein, J. Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163* (2016).
23. Mirza, M., Osindero, S. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784* (2014).
24. Nagarajan, V., Kolter, J.Z. Gradient descent GAN optimization is locally stable. I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett, eds. *Advances in Neural Information Processing Systems 30*, Curran Associates, Inc., Boston, 2017, 5585–5595.
25. Odena, A., Olah, C., Shlens, J. Conditional image synthesis with auxiliary classifier gans. *arXiv preprint arXiv:1610.09585* (2016).
26. Oord, A. v. d., Li, Y., Babuschkin, I., Simonyan, K., Vinyals, O., Kavukcuoglu, K., Driessche, G. v. d., Lockhart, E., Cobo, L.C., Stimberg, F., et al. Parallel wavenet: Fast high-fidelity speech synthesis. *arXiv preprint arXiv:1711.10433* (2017).
27. Radford, A., Metz, L., Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434* (2015).
28. Ratliff, L.J., Burden, and S.A., Sastry, S.S. Characterization and computation of local nash equilibria in continuous games. In *Communication, Control, and Computing (Allerton), 2013 51st Annual Allerton Conference on.* IEEE, (2013), 917–924.
29. Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X. Improved techniques for training gans. In *Advances in Neural Information Processing Systems* (2016), 2234–2242.
30. Shrivastava, A., Pfister, T., Tuzel, O., Susskind, J., Wang, W., Webb, R. Learning from simulated and unsupervised images through adversarial training.
31. Theis, L., van den Oord, A., Bethge, M. A note on the evaluation of generative models. arXiv:1511.01844 (Nov 2015).
32. Unterthiner, T., Nessler, B., Klambauer, G., Heusel, M., Ramsauer, H., Hochreiter, S. Coulomb GANs: Provably optimal Nash equilibria via potential fields. *arXiv preprint arXiv:1708.08819* (2017).
33. Wu, Y., Burda, Y., Salakhutdinov, R., Grosse, R. On the quantitative analysis of decoder-based generative models. *arXiv preprint arXiv:1611.04273* (2016).
34. Yeh, R., Chen, C., Lim, T.Y., Hasegawa-Johnson, M., Do, M.N. Semantic image inpainting with perceptual and contextual losses. *arXiv preprint arXiv:1607.07539* (2016).
35. Zhu, J.-Y., Park, T., Isola, P., Efros, A.A. Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint arXiv:1703.10593* (2017).

Ian Goodfellow, written while at Google Brain.

Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio, Université de Montréal.

Final submitted 5/9/2018.