Introduction to Generative Adversarial Networks

Nicolas Morizet

Introduction to Generative Adversarial Networks

Nicolas MORIZET, PhD NMORIZET@ADVESTIS.COM
Senior AI Research Scientist
Advestis SAS
69, Boulevard Haussmann
75008 Paris - France

Document: Advestis Technical Report, July 2020.

Abstract

This article aims to introduce the basic ideas and concepts of Generative Adversarial Networks, also known as GANs.

Keywords: Artificial Intelligence, Deep Learning, Generative Adversarial Networks, Machine Learning, Game Theory.

Introduction

In the last few years, researchers have made tremendous progress in the field of artificial intelligence using deep learning. But while deep learning systems can learn to recognize things, they have not been good at creating them. One night in 2014, *Ian Goodfellow*[1] and his colleagues tackled a computer vision challenge: build a machine that could create photos by itself. Neural networks were already used to mimic, in a very simple way, the web of neurons in the human brain. Some of those models were generative by nature as they were intended to create plausible new data. Results were not consistently good and images were often blurry or missing important structures. The first idea of Goodfellow's friends was to perform complex statistical analyses on pictures to help machines come up with images of their own. As the task seemed too tedious, Goodfellow hit on the following original idea "*What if you pitted two neural networks against each other ?*". Despite the scepticism of his friends, Goodfellow went home and developed a functional version of his software in a few hours. What he invented that night is now called *Generative Adversarial Networks* [5] or GANs, and is still considered as a major breakthrough in the field of machine learning. In September 2016, during a seminar entitled "Unsupervised Learning: The Next Frontier in AI", given at Cornell University, *Yann LeCun* (one of the fathers of deep learning) described GANs as "*the coolest idea in machine learning in the last twenty years*".

1. Ian Goodfellow obtained his B.S. and M.S. in computer science from Stanford University under the supervision of Andrew Ng, and his Ph.D. in machine learning from the Université de Montréal, under the supervision of Yoshua Bengio and Aaron Courville. Currently, Goodfellow is employed at Apple Inc. as its director of machine learning in the Special Projects Group.

1. Traditional Machine Learning versus Adversarial Machine Learning

Most machine learning algorithms can be described basically as optimization algorithms where some cost function has to be minimized. A very simple example with two parameters θ_1 and θ_2 and a corresponding cost function $J(\theta_1, \theta_2)$ is shown in Figure 1. In adversarial machine learning, instead, we look to *game theory* involving two *players*. Each player has their own cost which also has to be reduced, but the other players choices also affect their cost. The simplest version of this scenario is a *Minimax zero-sum game* where the two players costs always add up to one. To make this easier to understand and visualize, let us consider only one parameter for each player. The corresponding value function is represented by the 3D surface in Figure 2, where the first player tries to minimize the value and the second player tries to maximize it. If we look at cross-sections through the cost function in terms of the parameters that each player can control, *Player 1* is looking for a local minimum whereas *Player 2* is looking for a local maximum. The *Nash Equilibrium* [9] is reached when both players find such a point simultaneously. The game now has a solution because, in that very specific situation, neither player can improve their cost without controlling the other player [4].

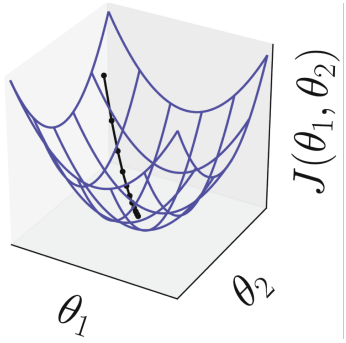

Figure 1: Traditional Machine Learning optimization.

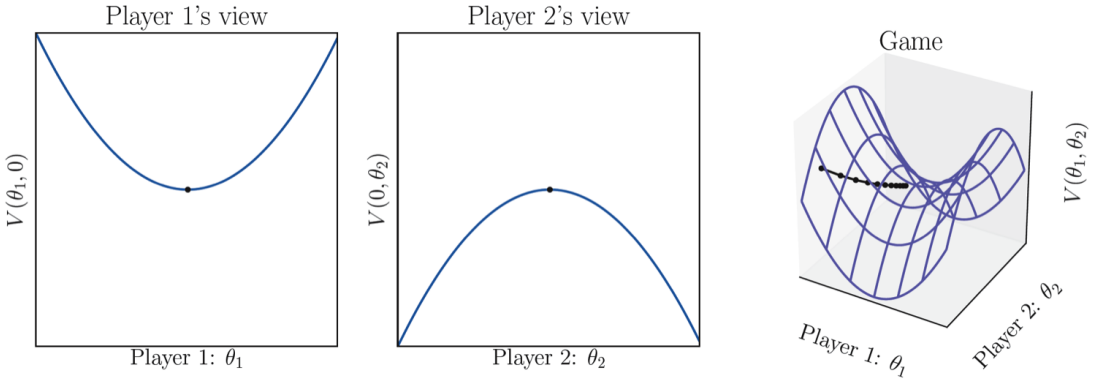

Figure 2: Adversarial Machine Learning and game theory. From left to right : cross-section for $\theta_2 = 0$, cross-section for $\theta_1 = 0$ and the value function.

2. Generative Modeling

Generative modeling is a branch of *unsupervised learning* in machine learning (Figure 3), where no labels are available and the main goal is to learn some underlying structure of the data (e.g. clustering, feature or dimensionality reduction, etc.). Generative models are good at *density estimation* and *sample generation*. Deep generative models represent probability distributions over multiple variables. Some of those models allow an explicit evaluation of the probability distribution function (*explicit density*), others do not (*implicit density*) and require some knowledge of it, such as sampling from the distribution. Many tasks intrinsically require realistic generation of samples from some distribution. Various applications include *single image super-resolution*, *art creation* and *image-to-image translation* [3].

Generative models are worth studying because they allow the manipulation of high-dimensional probability distributions, can be incorporated into *reinforcement learning* [8] and can be trained with missing data, providing predictions on inputs that are missing data. Finally, generative models, and GANs in particular, enable machine learning to work with *multi-modal* outputs.

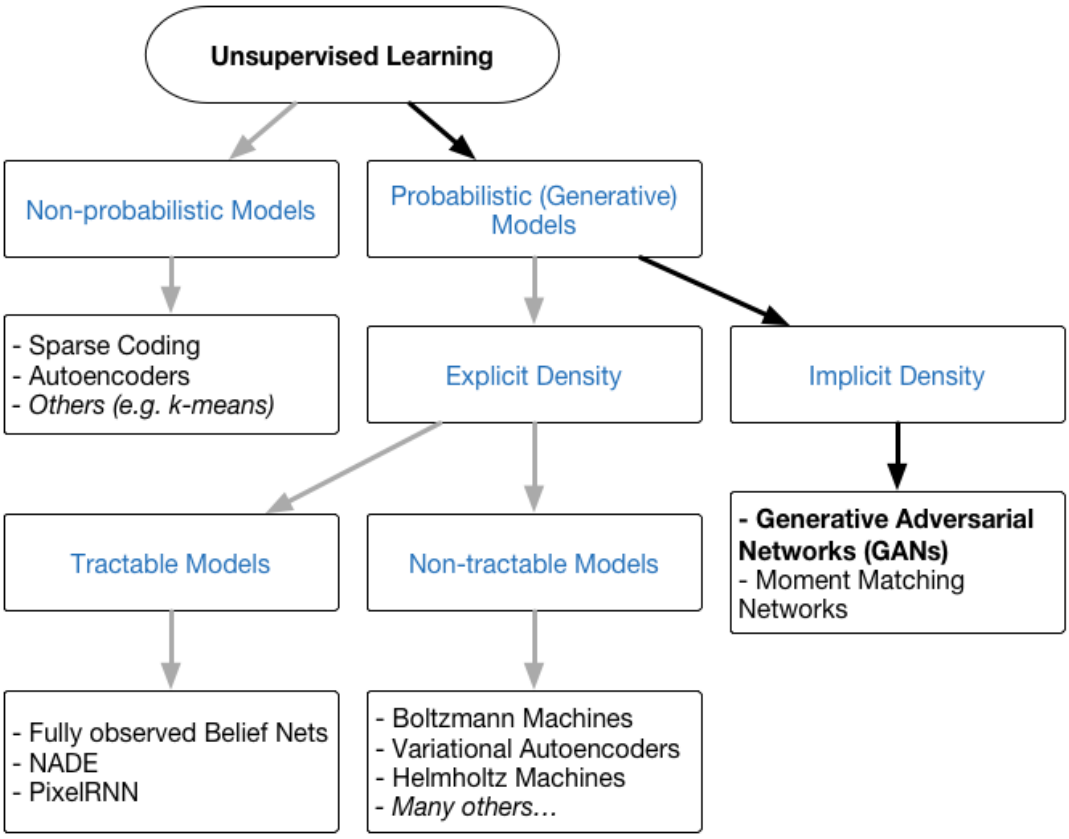

Figure 3: A Taxonomy of Unsupervised Learning.

3. How do GANs work ?

As we have seen in section 1, Adversarial Machine Learning can be represented as a game between two players. Now it is time to understand what GANs really are and how do they work.

3.1 The framework

First, let us give clearer names to the two players involved. One of them is called the **generator**. The goal of the generator is to create samples that are intended to come from the same distribution as the original training data. The other player is the **discriminator**. The discriminator determines whether its input data are real or fake. An intuitive analogy is to think this network as a struggle between a *forger* and the *police*. The generator is the forger and tries to produce fake paintings, while the discriminator is the police, trying to detect the counterfeit paintings. For example, the forger will have to learn how to master the style of the artist or how to use the right tools and materials. The police will have to carry out various analyses like infrared reflectrography, Wood's light, stereoscopic microscope or IR spectrography. Competition in this game drives both entities to improve their methods and skills until the counterfeits are indistinguishable from the genuine paintings.

Formally, GANs are a *structured probabilistic model*[2] containing latent noisy variables z and real variables x. The two players are represented by two differentiable functions, each of which is differentiable with respect to its inputs and its parameters. Both players have cost functions that are defined in terms of both players' parameters. The **generator** is defined by a function G that takes z as input, using $\theta^{(G)}$ as parameters, and wishes to minimize the cost function $J^{(G)}(\theta^{(D)}, \theta^{(G)})$, while controlling only $\theta^{(G)}$. The **discriminator** is defined by a function D that takes x and $G(z)$ as inputs, using $\theta^{(D)}$ as parameters, and wishes to minimize the cost function $J^{(D)}(\theta^{(D)}, \theta^{(G)})$, while controlling only $\theta^{(D)}$ (Figure 4).

The solution to this game is a *Nash equilibrium*, which is a tuple $(\theta^{(D)}, \theta^{(G)})$ that is a local minimum of $J^{(G)}$ with respect to $\theta^{(G)}$ and a local minimum of $J^{(D)}$ with respect to $\theta^{(D)}$ (Table 1).

Name	Input(s)	Parameters	Cost Function
Generator (G)	z	$\theta^{(G)}$	$J^{(G)}(\theta^{(D)}, \theta^{(G)})$
Discriminator (D)	x, $G(z)$	$\theta^{(D)}$	$J^{(D)}(\theta^{(D)}, \theta^{(G)})$

Table 1: Inputs, parameters and cost functions for the GANs framework.

Typically, each differentiable function X is represented by an *universal approximator* such as a **deep neural network** where the parameters $\theta^{(X)}$ denote its *weights* (and *biases*). Depending on the nature of the problem to solve, it can be a Multilayer Perceptron (MLP), a Convolutional Neural Network (CNN), etc.

2. A structured probabilistic model is a way of describing a probability distribution using a graph. This graph holds a set of vertices connected to one another by a set of edges and describes which random variables interact with each other directly. These models are often also referred to as probabilistic graphical models (PGM) [6].

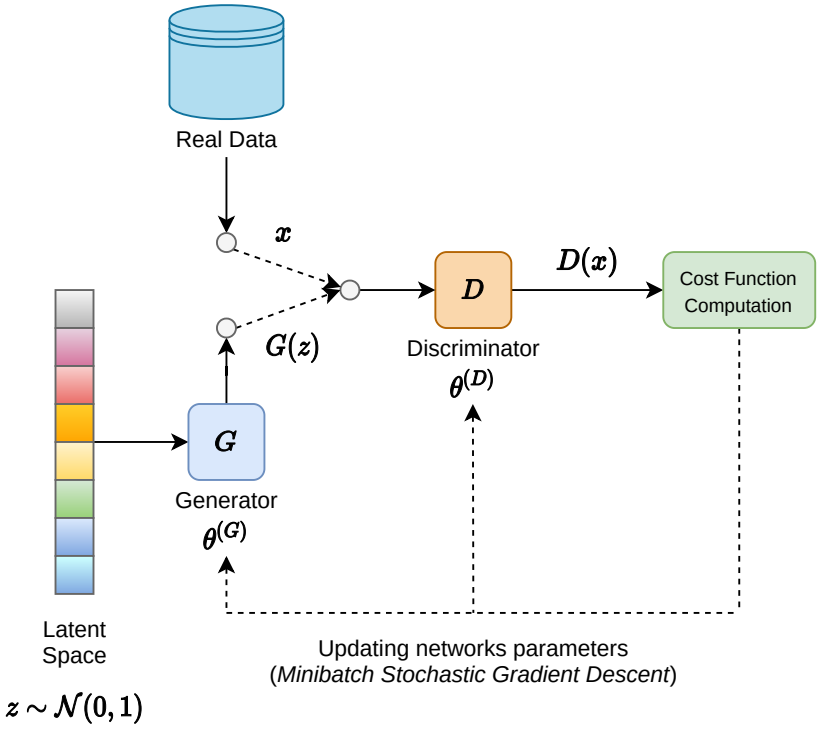

Figure 4: The GANs Framework.

3.2 The training

The training process consists of simultaneous *minibatch stochastic gradient descent*. For each step, two minibatches are sampled : a first minibatch coming from the dataset (labelled as 1) and a second minibatch coming from the generator (where all the labels are 0). Then, two gradient steps are performed simultaneously, one updating $\theta^{(G)}$ to reduce the cost function $J^{(G)}$, the other updating $\theta^{(D)}$ to reduce the cost function $J^{(D)}$.
The cost function used for the discriminator is:

$$J^{(D)} = \mathbb{E}_{\boldsymbol{x}} \log D(\boldsymbol{x}) + \mathbb{E}_{\boldsymbol{z}}[\log(1 - D(G(\boldsymbol{z})))] \tag{1}$$

where $\mathbb{E}_{\boldsymbol{x}}$ is the expectation according to the true distribution, $\mathbb{E}_{\boldsymbol{z}}$ the expectation with respect to the generator's distribution, and $D(\boldsymbol{x})$ the probability that x came from the original data rather that the generator's distribution.

This function may seem intimidating at first, but it derives from the **binary cross-entropy** (also called *log loss*) between two distributions when training a binary classifier. The only difference here is that the classifier is trained on the two aforementioned mini-batches of data. A demonstration can be found in the *Appendix*.

Now that we have specified the cost function for the discriminator, we also must define a cost function for the generator in order to have a complete specification of the game. The simplest version of the game is a zero-sum game (Section 1), in which the sum of all players costs is always zero. This is the **minimax** version:

$$J^{(G)} = -J^{(D)} = -\mathbb{E}_{\boldsymbol{x}} \log D(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{z}}[\log(1 - D(G(\boldsymbol{z})))] \tag{2}$$

The cost used for the generator in the minimax game (equation 2) is useful for theoretical analysis, but does not perform especially well in practice, due to gradient saturation. As Goodfellow notes in [3]:

> "*In the minimax game, the discriminator minimizes a cross-entropy, but the generator maximizes the same cross-entropy. This is unfortunate for the generator, because when the discriminator successfully rejects generator samples with high confidence, the generator's gradient vanishes.*"

To solve this problem, one approach is to also use cross-entropy minimization for the generator, by flipping the target used to construct the cross-entropy cost. Thus, the cost for the generator becomes the **non-saturating heuristic** version (see Algorithm 1):

$$J^{(G)} = -\mathbb{E}_{\boldsymbol{z}} \left[\log D(G(\boldsymbol{z}))\right] \tag{3}$$

Algorithm 1: Minibatch Stochastic Gradient Descent (SGD) Training of GANs

$p_g(\boldsymbol{z})$: noise prior distribution.
$p_{data}(\boldsymbol{x})$: true distribution.

for *n training epochs* **do**

  for *k steps* **do**

    - Sample a minibatch of m noise samples $\left\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\right\}$ from $p_g(\boldsymbol{z})$.
    - Sample a minibatch of m real samples $\left\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\right\}$ from $p_{data}(\boldsymbol{x})$.
    - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right] \tag{4}$$

  end

    - Sample a minibatch of m noise samples $\left\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\right\}$ from $p_g(\boldsymbol{z})$.
    - Update the generator by ascending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \tag{5}$$

end

After several steps of training, if the generator and discriminator have enough *capacity* (i.e. if the networks can approximate the objective functions), they will reach a state where both cannot improve anymore. At this point, the generator generates realistic synthetic data, and the discriminator is unable to differentiate between the two types of input.

Training GANs is a pretty hard challenge because of the imbalance that may exist between the generator and the discriminator, the high sensitivity to the hyperparameters selection, but also for the following major points [7]:

- **Nash equilibrium**: some cost functions will not converge with gradient descent (in particular for non-convex game). Since the opponent is always countermeasuring your actions, the models convergence is harder to be reached,

- **Mode collapse**: the generator keeps fooling the discriminator but is stuck in some sort of "infinite loop", where it delivers the same limited varieties of outputs over and over again,

- **Vanishing gradients**: if the discriminator is too good, the generator learns nothing and its training will fail. One attempt to remedy is to use the *Wasserstein loss* [1], designed to prevent diminishing gradients even when you train the discriminator to optimality.

4. Applications

GANs have a wide range of applications, from computer vision, image generation and text-to-image translation to photo editing, super-resolution imaging and video prediction. It can be observed that there is much less literature concerning the use of GANs for time series. However, this field has been one of the preferred applications of machine learning for several years. This naturally leads to the two following questions: "*Can the spectacular advances of GANs in imaging be transposed to time series processing ?*", and "*Does taking into account the temporal structure of the data pose particular problems for GANs ?*". The use of GANs for the generation of artificial images has been popularized by applications such as "*thispersondoesnotexist*"[3] in the generation of artificial human faces. The transposition of this application to the field of time series could be very useful in the following cases: completion of an insufficient amount of real data with artificial data drawn according to the same law, impossibility to use real data considered too sensitive or confidential (e.g. the cancelation of the 2010 Netflix Prize[4], data requiring strong anonymisation, legal proceedings), etc.

Advestis is already working on generating artificial economic scenarios. The applications seem promising: in a context of limited historical data, it would be possible to empirically approximate a multi-varied probability distribution of unknown law, and to virtually reproduce situations similar to historically observed accidents.

3. The website https://thispersondoesnotexist.com/ uses AI to generate endless fake faces.
4. "Netflix Cancels Contest After Concerns Are Raised About Privacy", The New York Times, March 2010.

Conclusion

GANs are a powerful unsupervised technique for generating artificial data close to real ones. It is considered as a major breakthrough in the field of machine learning with a large range of research and applications to come. Future research opportunities may be found in using GANs for natural language processing, information retrieval or time series generation. Also, GANs might have a great use in the field of security (eg. adversarial attacks on neural networks) [10]. Between 2014 and 2018, more than 500 variants of GANs were proposed by researchers around the world, showing their insatiable interest as well as their impressive creativity in what is now called the GAN Zoo [2].

References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.

[2] Hindupur Avinash. The gan zoo, 2018. URL `https://github.com/hindupuravinash/the-gan-zoo`.

[3] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks, 2016. URL `http://arxiv.org/abs/1701.00160`.

[4] Ian Goodfellow. Iclr 2019 invited talk on adversarial machine learning, 2019. URL `http://www.iangoodfellow.com/slides/2019-05-07.pdf`.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014. URL `http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`.

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

[7] Hui Jonathan. Gan - why it is so hard to train generative adversarial networks!, June 2018. URL `https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b`.

[8] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *J. Artif. Int. Res.*, 4(1):237–285, May 1996. ISSN 1076-9757.

[9] John F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49, 1950. ISSN 0027-8424. doi: 10.1073/pnas.36.1.48. URL `https://www.pnas.org/content/36/1/48`.

[10] Z. Pan, W. Yu, X. Yi, A. Khan, F. Yuan, and Y. Zheng. Recent progress on generative adversarial networks (gans): A survey. *IEEE Access*, 7:36322–36333, 2019.

Appendix

In this appendix, we explain how to derive the cost function of the discriminator (Section 3.2) from the binary cross-entropy (BCE) loss function.

When training a binary classifier, the BCE loss function is defined as follows:

$$L(\hat{y}, y) = y \log \hat{y} + (1 - y) \log(1 - \hat{y}) \tag{A.1}$$

where y is the true label and $\hat{y}$ is the predicted label.

While training the discriminator, the label coming from the real data is $y = 1$, whereas the predicted label is $\hat{y} = D(x)$. Thus, substituting those two values in the equation (A.1), we obtain:

$$L(D(x), 1) = \log(D(x)) \tag{A.2}$$

Regarding the generated data, the label is $y = 0$ and the predicted label is $\hat{y} = D(G(z))$. Again, substituting those two values in the equation (A.1), we get:

$$L(D(G(z)), 0) = \log(1 - D(G(z))) \tag{A.3}$$

Since the discriminator wants to correctly classify the generated and the real data, equations (A.2) and (A.3) should be maximized, leading to the following combined loss function :

$$L^{(D)} = \log(D(x)) + \log(1 - D(G(z))) \tag{A.4}$$

Finally, since the above function is only valid for a single data point, to consider the whole dataset, we need to take the expectation of the above equation as:

$$J^{(D)} = \mathbb{E}_{\boldsymbol{x}} \log D(\boldsymbol{x}) + \mathbb{E}_{\boldsymbol{z}}[\log(1 - D(G(\boldsymbol{z})))] \tag{A.5}$$

where $\mathbb{E}_{\boldsymbol{x}}$ is the expectation according to the true distribution and $\mathbb{E}_{\boldsymbol{z}}$ is the expectation with respect to the generator's distribution. Equation (A.5) is exactly the cost function of the discriminator from Section 3.2.