Generative Adversarial Networks

Alan Zheng*

March 2019

1 Introduction

Generative Adversarial Networks (GANs) are a seminal type of generative model, introduced in 2014 by the University of Montreal. GANs have been heavily used in various generative tasks with impressive results. GANs are most actively used for image generation tasks: plain image generation, image inpainting, super-resolution image generation, and text-to-image. Over these few years, however, they have also been used in real-life applications for text/image/video generation, drug discovery and text-to-image synthesis. Since GAN's first release, there have been multiple iterations on different types of GANs; here, we'll cover the basics only.

2 What do it do

In short, they belong to the set of algorithms named **generative models**. These algorithms belong to the field of **unsupervised learning**, a sub-set of ML which aims to study algorithms that learn the underlying structure of the given data, without specifying a target value. Generative models learn the intrinsic distribution function of the input data ***p(x)*** (or ***p(x,y)*** if there are multiple targets/classes in the dataset), allowing them to generate both synthetic inputs x' and outputs/targets y', typically given some hidden parameters.

In contrast, supervised learning algorithms learn to map a function ***y'=f(x)***, given labeled data y. An example of this would be classification, where one could use customer purchase data (x) and the customer respective age (y) to classify new customers. Most of the supervised learning algorithms are inherently **discriminative**, which means they learn how to model the conditional probability distribution function (p.d.f) ***p(y|x)*** instead, which is the probability of a target (age=35) given an input (purchase=milk). Despite the fact that one could make predictions with this p.d.f, one is not allowed to sample new instances (simulate customers with ages) from the input distribution directly.

3 Intuition

Although the GAN architecture can take on several possible forms, the vanilla GAN consists of two neural networks.

The first model is called a Generator and it aims to generate new data similar to the expected one. The Generator could be assimilated to a human art forger who creates fake works of art.

The second model is named the Discriminator. This model's goal is to recognize if an input data is 'real' — belongs to the original dataset - or if it is 'fake'—generated by a forger. In this scenario, a Discriminator is analogous to an art expert who tries to detect works as truthful or fraud.

*Based off Justin Zhang's lecture

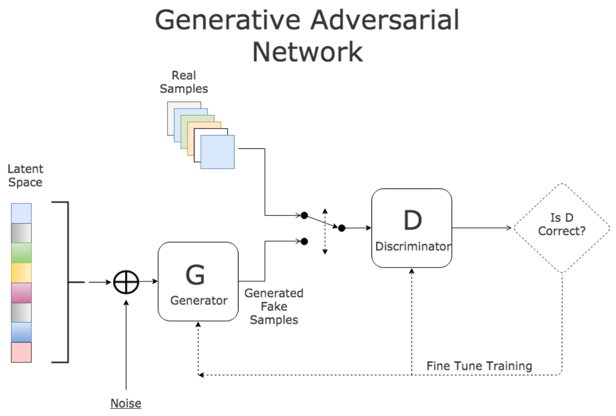


Figure 1: The GAN pipeline.

How do these models interact? Paraphrasing the original paper which proposed this framework, it can be thought of the Generator as having an adversary, the Discriminator. The Generator (forger) needs to learn how to create data in such a way that the Discriminator isn't able to distinguish it as fake anymore. The competition between these two teams is what improves their knowledge, until the Generator succeeds in creating realistic data.

4 Math

A neural network G(z, θ_1) is used to model the Generator mentioned above. It's role is mapping **random input noise variables z** to the **desired data space x** (say images). Conversely, a second neural network D(x, θ_2) models the discriminator and outputs the probability that the data came from the real dataset, in the range (0,1). In both cases, θ_i represents the weights or parameters that define each neural network.

As a result, the Discriminator is trained to correctly classify the input data as either real or fake. This means its weights are updated as to maximize the probability that any real data input x is classified as belonging to the real dataset, while minimizing the probability that any fake image is classified as belonging to the real dataset. In more technical terms, the loss/error function used **maximizes the function D(x), and it also minimizes D(G(z))**.

Furthermore, the Generator is trained to fool the Discriminator by generating data as realistic as possible, which means that the Generator's weights are optimized to maximize the probability that any fake image is classified as belonging to the real data. Formally this means that the loss/error function used for this network **maximizes D(G(z))**.

In practice, the logarithm of the probability (e.g. log D(...)) is used in the loss functions instead of the raw probabilities, since using a log loss heavily penalizes classifiers that are confident about an incorrect classification.

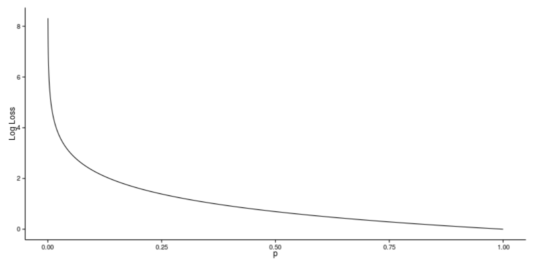


Figure 2: Log Loss Visualization: Low probability values are penalized significantly more.

After training for a while, if the Generator and Discriminator have enough capacity (if the networks can approximate the objective functions), they will reach a point at which 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.

Since during training both the Discriminator and Generator are trying to optimize opposite loss functions, they can be thought of two agents playing a minimax game with value function V(G,D). In this minimax game, the generator is trying to maximize it's probability of having it's outputs recognized as real, while the discriminator is trying to minimize this same value.

$$min_G \, max_D V(D, G) = E_{x \ p_{data}(x)}[\log D(x)] + E_{z \ p_z(z)}[\log(1 - D(G(z))]$$

Thus, the gradient expression we train the discriminator on is as follows:

$$\nabla \frac{1}{m} \sum_{i=1}^{m} [\log D(x_i) + \log(1 - D(G(z_i))]$$

where x is the real data in a given batch, z is the noise for the generator for the given batch, D is the discriminator function, and G is the generator. We want to maximize this expression. The first expression in the summation, $\log D(x_i)$, corresponds to the discriminator output on real data; we clearly want to maximize this probability. The second is a bit more complicated: the $D(G(z_i))$ corresponds to the discriminator's probability estimate that the generated data is real. We want to minimize this, so we make the term $\log(1 - D(G(z_i))$. If you haven't taken multivariable calculus yet, think about the ∇ as a derivative; we simply want to move in the direction that maximizes the summation.

The gradient expression we train the generator on is as follows:

$$\nabla \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(z_i))$$

We want to minimize this expression (i.e. we want to maximize $D(G(z_i))$, the probability of the generated data being real, as determined by the discriminator).

Note that GANs are notoriously difficult to train. This is because GANs are highly unstable; in order to train correctly, we need the generator and discriminator to be roughly on equal levels throughout the training process. If the discriminator overpowers the generator, there will be little gradient for the generator to learn upon; vice-versa, and we run into *mode collapse*, where the generator produces outputs with extremely low variety.

5 Applications

GANs are not yet frequently used in applications at the high school level, so finding a decent (novel) application for them could be a great project idea.

Here are some examples:

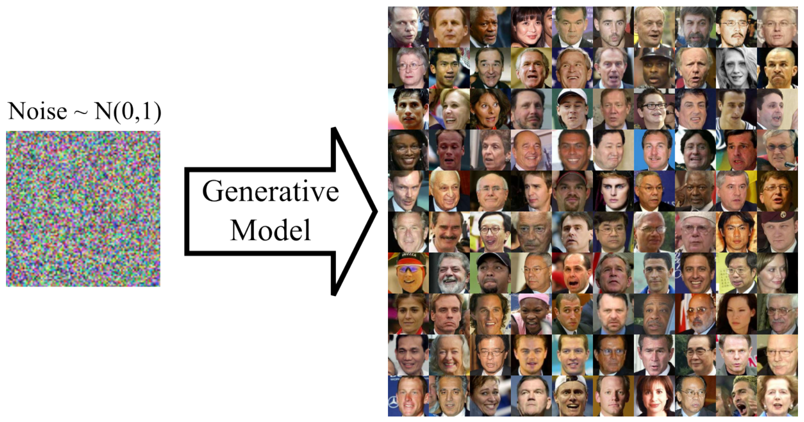


Figure 3: Image generation.

This has been done with a variety of GANs (e.g. Wasserstein GAN), and can be done (with limited success) with vanilla GANs. There are also architectures that convert a caption to an image, with remarkable success (e.g. StackGAN++).

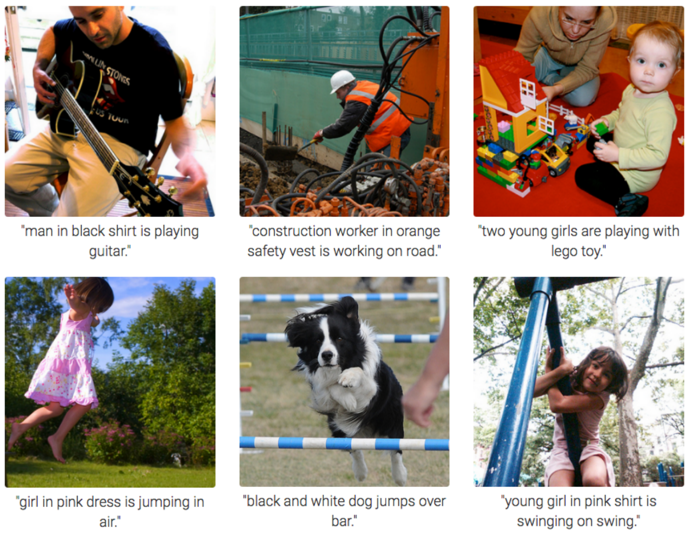

"man in black shirt is playing guitar."
"construction worker in orange safety vest is working on road."
"two young girls are playing with lego toy."
"girl in pink dress is jumping in air."
"black and white dog jumps over bar."
"young girl in pink shirt is swinging on swing."

Figure 4: Caption generation.

Conditional GANs (as well as RNNs) have been used, to much success, for this task.

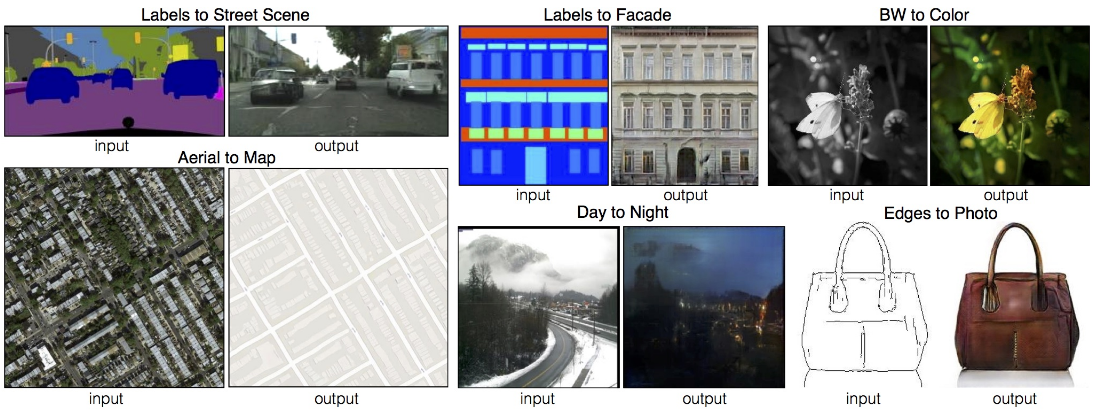


Figure 5: Image mapping.

Two prominent networks for this task are the pix2pix network (Conditional GAN) and the CycleGAN. The results, as you can see, are quite impressive, and can be extended to a variety of image-based tasks.

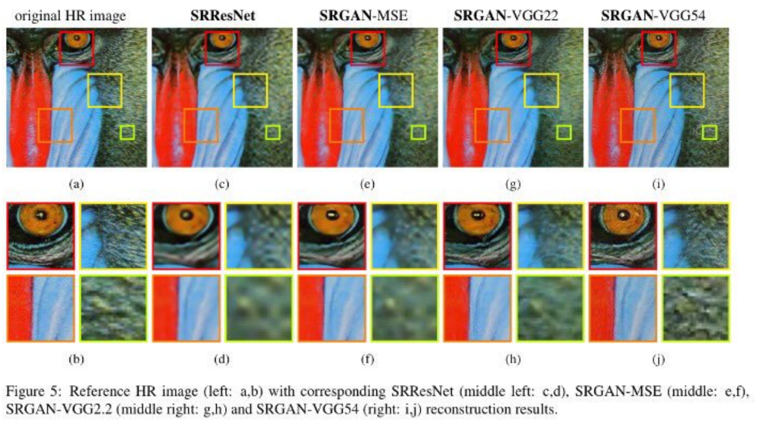


Figure 5: Reference HR image (left: a,b) with corresponding SRResNet (middle left: c,d), SRGAN-MSE (middle: e,f), SRGAN-VGG2.2 (middle right: g,h) and SRGAN-VGG54 (right: i,j) reconstruction results.

Figure 6: Super-resolution.

GANs have been used in the task of super-resolution, interpolating finer texture details that are lost in a low-res image. The most prominent architecture for this task is the SRGAN. The SRGAN has been used to moderate success (though MOS scores are subjective and difficult to validate).

6 CycleGAN

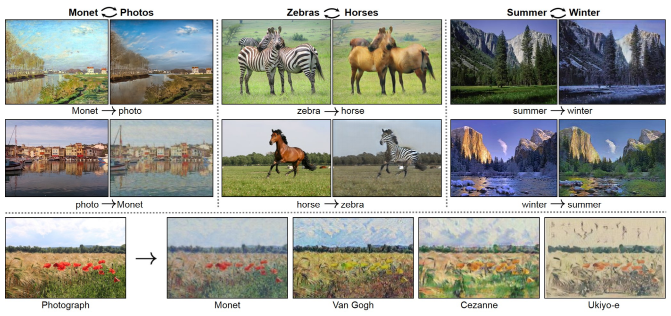


Figure 7: The CycleGAN.

The goal of the CycleGAN is to learn a mapping G that translates a source domain X to a target domain Y given unpaired images.

However, the problem of learning G such that $G(X) \to Y$ is that the mapping G is very under-constrained (in other words, there are many ways that G can minimize loss of the dataset as a whole while producing qualitatively not-very-good individual images). The basic approach of the CycleGAN, thus, is to ensure *cyclic consistency* by introducing an inverse mapping F (a third network) and a cyclic consistency loss that enforces $F(G(X)) \approx X$ and $G(F(Y)) \approx Y$.

As usual, the loss for G is

$$min_G \, max_{D_Y} \log D_Y(x_i) + \log(1 - D_Y(G(z_i))$$

An analogous loss is used for F.

However, a new term is introduced for cyclic consistency:

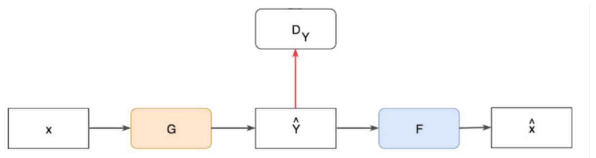
$$\lambda||F(G(x)) - x||_1 + ||G(F(y)) - y||_1$$

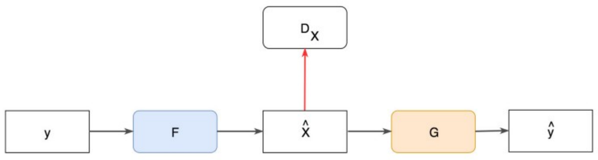
The L1 norm was selected based on empirical data.

The final loss, then, is the sum of these three individual losses.

Let's say we want to convert real images into Van Gogh paintings and vice versa. Overall, the network design would look like this:

- A generator G to convert a real image to a Van Gogh style picture.
- A generator F to convert a Van Gogh style picture to a real image.
- A discriminator D_y to identify real or generated Van Gogh pictures.
- For the other direction, we reverse the data flow and build an additional discriminator D_x to identify real images.





7 Conclusion

GANs are incredibly useful models; however, training them can be very difficult. GANs are continuously being improved to increase stability and to accommodate different types of input. Stay tuned for more lectures later on GAN improvements and such.

We hope to see more GAN projects in the coming months, and we're confident that there are many unexplored uses for them.

For more tips on training a GAN, see this link [https://github.com/soumith/ganhacks]. Seriously. It'll be useful.

For an example of a GAN, see this link. [https://github.com/jacobgil/keras-dcgan/blob/master/dcgan.py]