Part 1 – Intro to Generative Adversarial Networks (GANs)

Hi there! Today's post is about Generative Adversarial Networks (GANs). If you have been keeping an eye on deep learning, or artificial intelligence, you must have heard about it. GAN has become a buzzword in recent years (Figure 1), even when Yann LeCun talked about it, he praised it as "adversarial training is the coolest thing since sliced bread".

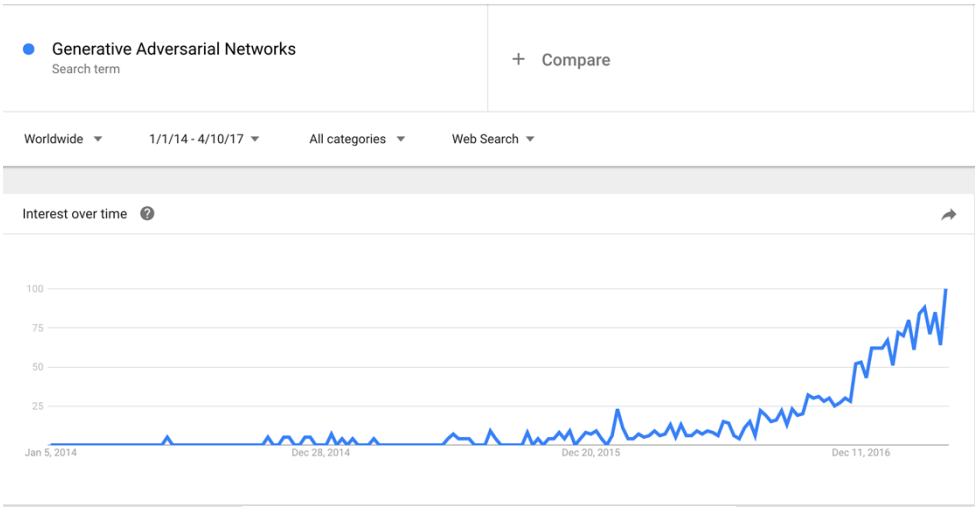
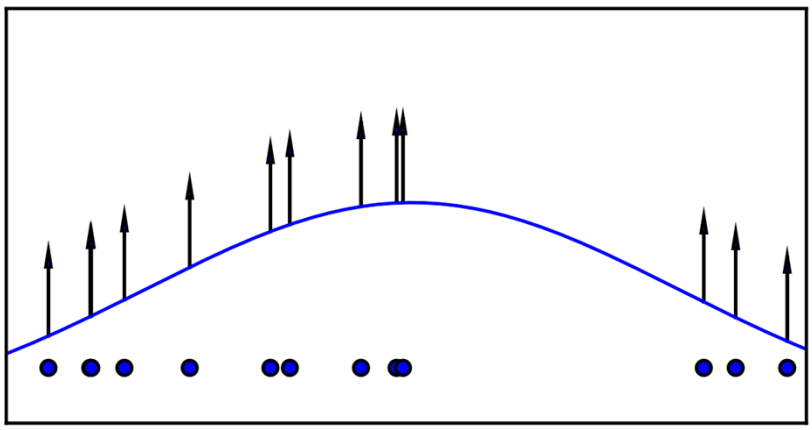


Figure 1: GANs on Google Trends

What are GANs?

Well, so what's the magic about GANs? GANs is a framework for producing a generative model designed to produce realistic samples by way of a two-player minmax game, first introduced by Goodfellow et al.(2014) in 2014. Before going into details about GANs, let's quickly review what a generative model(GM) is. A generative model(GM) is a model for randomly generating observable data values, typically given some hidden parameters. It specifies a joint probability distribution over observation and label sequences. GMs are based on the Maximum Likelihood method (Figure 2, 3), and are important because of their ability of modeling the probability density of data (such as images, Figure 4), and forming conditional probability density functions. In some cases, generative models perform density estimations, while in some other cases, they generate samples from the model distribution.



$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{x \sim p_{\text{data}}} \log p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{\theta})$$

The maximum likelihood process consists of taking several samples from the data generating distribution to form a training set, then pushing up on the probability the model assigns to those points, in order to maximize the likelihood of the training data. This illustration shows how different data points push up on different parts of the density function for a Gaussian model applied to 1-D data. The fact that the density function must sum to 1 means that we cannot simply assign infinite likelihood to all points; as one point pushes up in one place it inevitably pulls down in other places. The resulting density function balances out the upward forces from all the data points in different locations.

Figure 2. The maximum likelihood process (Goodfellow (2016)).

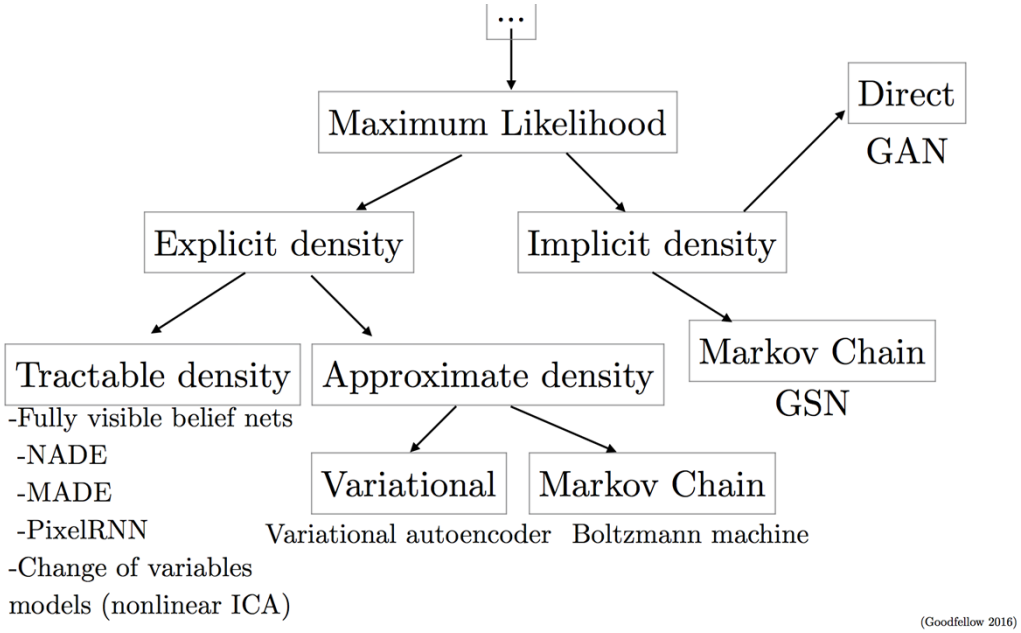


Figure 3. Taxonomy of generative models (Goodfellow (2016)).

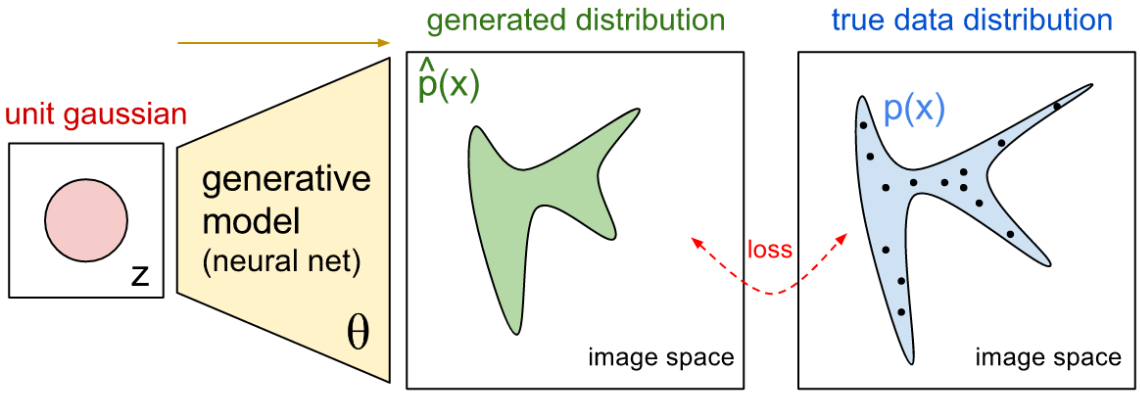


Figure 4. Generative Models

Generative adversarial networks (GANs) focus primarily on generating contents. It is a new method of training deep generative models. The key idea behind it is that it has two players - a generator vs. a discriminator - and it pits them against each other. In mathematical terms, one player, the generator, attempts to generate realistic data samples by transforming noisy samples, z, drawn from a simple distribution (e.g., $z \sim N(0, 1)$) using a transformation function $G_\theta(z)$ with learned weights, θ. On the other hand, the generator receives feedback as to how realistic its synthetic sample is from another player, the discriminator, which attempts to discern between synthetic data samples produced by the generator and samples drawn from the actual dataset using a function $D_\omega(x)$ with learned weights, ω (Ishan et al (2016)). **In short, from the perspective of the Game Theory, the generator generates samples that are as similar as the input data, while the discriminator tries to tell if this sample is fake (made by the generator) or from the real world. The two players take steps in turn in the game.**

In a GAN model, both discriminator and generator are deep networks (differentiable functions), and can be trained with backpropagation.

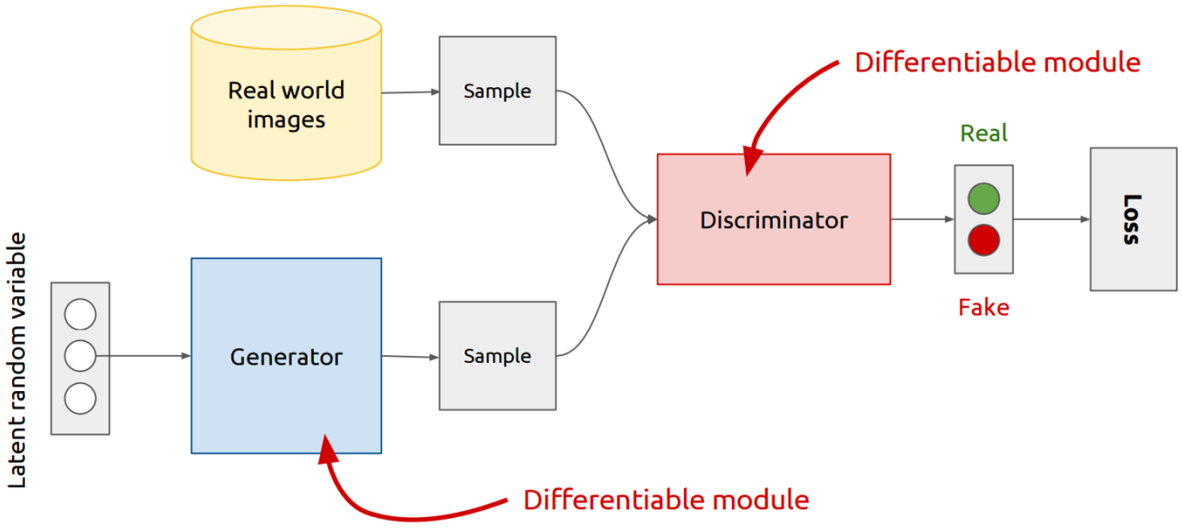


Figure 5. Conceptual Model of GANs: alternating between the training of two modules

So what sets GANs apart from other generative modeling approaches? Goodfellow, the father of GANs, demonstrated some of its features at the NIPS 2016, which include:

- Use a latent code
- Asymptotically consistent
- No Markov chains needed
- Often regarded as producing the best samples

However, problem remain that there is currently no good way to quantify this.

Now, let's take a closer look at the GAN model. To train the two 'players', we need an objective function. Here comes in the famous two-player minmax value function:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

In Goodfellow's original paper, he provided some theoretical analyses of the adversarial nets. I won't go into details in this blog post, please check out the paper if interested. Generally, in the minmax function, G(z) represents a generated sample from distribution z by the generator(G), and D(x) is the estimated (by the discriminator(D)) probability that x is a real sample. Apparently, D(x)=1 when D decides x is a real sample, and D(x)=0 when D decides x is a generated (fake) sample.

An intrinsic way of understanding it is that if x is sampled from the real data, then D(x) tries to be near 1, while if x is sampled from the model generation from input noises by G, we train D to be smarter who would try to make D(G(z)) approach 0, and train G to be cunny such that it tries to make D(G(z)) near 1. Both players have cost functions that are defined in terms of both players' parameters.

Here is a live demo that might help you understand the principle ideas and algorithms of GANs: http://cs.stanford.edu/people/karpathy/gan/

With these in mind, now we move to the numerical training procedure. This would be a problem of finding Nash Equilibria. The generative procedure is to use SGD-like algorithm of choice (Adam) on two minibatches simultaneously: one minibatch from training examples, and another from generated samples. (It is optional to run k steps of one player for every step of the other player. As a matter of fact, as of today, 1 step for each player is favorable.) Now we design a vanilla procedure to train the GANs which later we will modify it a little bit and apply it on a real problem in Part II of this blog:

Iterate through:

- Freeze generator weights, draw samples from both real-world samples and generated samples. Backpropagate error to update discriminator weights. Train discriminator to distinguish between real and fake (generated) data.
- Freeze the discriminator weights, and sample from the generator. Backpropogate error through discriminator to update generator weights.

We iterate these two steps until convergence (which does not always happen). During this process, the discriminator and generator both get improved so eventually, the ideal situation is that the generator gets so good that it is impossible for the discriminator to tell the difference between the real and fake, and hence the discriminator accuracy approaches 0.5 (half-half).

Alternative GANs and Architectures

We have spent the majority of the previous section walking through the foundations of the vanilla GAN model since other complex GAN architectures and algorithms are all built on top of the basic GAN concepts and framework. Ever since the first GAN model came out, there have been a lot of modified GAN architectures that have been proposed. In this section, I will briefly go over some interesting GAN models, and provide you some reference. Hope you will also find GAN models interesting as I do.

1. DCGANs (Radford et al 2015) (code)

One most popular application of GANs right now is for image generation using convolutional neural networks(CNNs). This kind of network is called a deep convolutional generative adversarial network (DCGANs). DCGANs are able to hallucinate original photo-realistic pictures by using a clever combination of two deep neural networks that compete with each other. Figure 6 is an example of such an image-producing GAN. The generator which takes a vector input (z), and produces a 64x64x3 image. The discriminator then takes both real images (x) and generated images (G(z)), and produces a probability P(x) for them. Once the network is trained and we would like to generate new images from it, we simply call G(z) on a new batch of randomized z vectors.

Figure 7 is from Radford's original paper which composes bedroom images 'dreamed up' by his DCGAN model. You may have already seen Figure 7 in other places as an excellent example of the power of DCGANs.

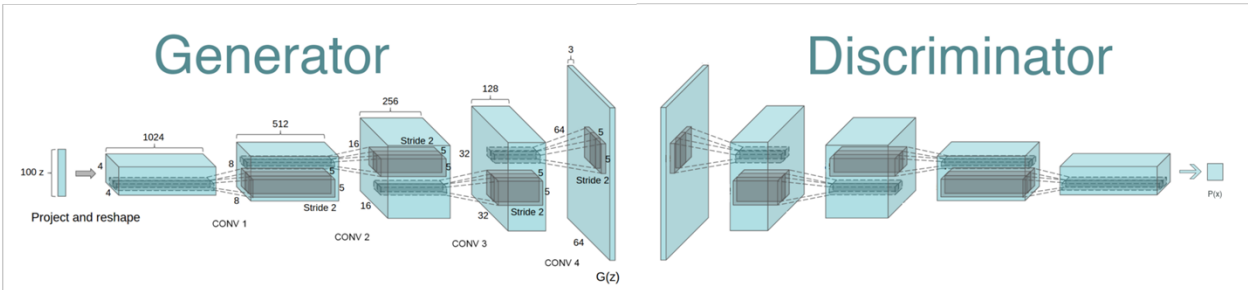


Figure 6. An example architecture for DCGANs.

Figure 7 Bedrooms generated by a DCGAN (Radford et al 2015)

2. InfoGAN (Chen et. al (2016)) (code)

InfoGAN is a generative adversarial network that also maximizes the mutual information between a small subset of the latent variables and the observation. An InfoGAN learns disentangled and interpretable representations for images. It imposes additional structure on this space by adding new objectives that involve maximizing the mutual information between small subsets of the representation variables and the observation. This approach provides exciting results. The example below shows that the resulting dimensions from InfoGAN code captures interpretable dimensions.

(a) Azimuth (pose) (b) Elevation

(c) Lighting (d) Wide or Narrow

Figure 8. InfoGAN implementation on mutual information identification of 3D faces

3. **Wasserstein-GAN (Arjovsky et.al. (2017)) (Code)**

Wasserstein-GAN (WGAN) is a new alternative model to the traditional GAN that came out earlier this year, but has already attracted a lot of attention. In the authors' paper, they showed that there are significant practical benefits to using it over the formulation used in standard GANs. They claimed two main benefits of a meaningful loss metric that correlates with the generator's convergence and sample quality, and improved stability of the optimization process.

Some other resources and alternative GANs you may be interested in:

4. CGAN - Mehdi Mirza, arXiv:1411.1784v1
5. LAPGAN - Emily Denton & Soumith Chintala, arxiv: 1506.05751
6. PPGAN - Anh Nguyen, arXiv:1612.00005v1
7. LS-GAN - Guo-Jun Qi, arxiv: 1701.06264 (code)
8. SeqGAN - Lantao Yu, arxiv: 1609.05473 (code)
9. EBGAN - Junbo Zhao, arXiv:1609.03126v2 (code)
10. VAEGAN - Anders Boesen Lindbo Larsen, arxiv: 1512.09300

Applications & Current Issues

As stated on the OpenAI's webpage, with the development of GAN models, they are expected to eventually generate samples that depict entirely plausible images or

videos. This may by itself find use in multiple applications, such as on-demand generated art, or Photoshop++ commands. Additional presently known applications include image denoising, inpainting, super-resolution, structured prediction, exploration in reinforcement learning, and neural network in cases where labeled data is expensive.

Of course, training a GAN model is not that easy partially due to its unstable dynamics: it is hard to keep generator and discriminator in balance. In practice, optimization can oscillate between solutions. In many cases, generator can collapse. Possible to use supervised labels to help prevent this: https://arxiv.org/abs/1606.03498. Common tips and tricks to make GANs work can be found at: https://github.com/soumith/ganhacks

References

Goodfellow, Ian, et al. "Generative adversarial nets." *Advances in neural information processing systems*. 2014.

Durugkar, Ishan, Ian Gemp, and Sridhar Mahadevan. "Generative Multi-Adversarial Networks." *arXiv preprint arXiv:1611.01673* (2016).

Chen, Xi, et al. "Infogan: Interpretable representation learning by information maximizing generative adversarial nets." *Advances in Neural Information Processing Systems*. 2016.

Goodfellow, Ian. "NIPS 2016 Tutorial: Generative Adversarial Networks." *arXiv preprint arXiv:1701.00160* (2016).

Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." *arXiv preprint arXiv:1511.06434* (2015).

Salimans, Tim, et al. "Improved techniques for training gans." *Advances in Neural Information Processing Systems*. 2016.