Deep Learning

MInDS @ Mines

Generative Adversarial Networks (GANs) are a fairly new and exciting method in deep learning. GANs work by pinning two models against each other resulting in very powerful generative abilities that can produce realistic results. GANs can be used in a variety of creative applications to generate new samples of data that appear indistinguishable from the originals.

So far we have discussed many machine learning models with a focus on their applications. We grouped models based on the primary type of machine learning they fall under, such as supervised / unsupervised learning or their application, such as feature learning. Traditionally, we define models based on a separate characteristic and that is the approach they take to solving a problem. There are two main types of machine learning models; generative models and discriminative models.

Generative Models

Generative models are those that aim to learn the probability distribution of the underlying cause of data appearing a particular way. This ”underlying cause” can be a seen class, in the case of supervised learning or an unseen one, in the case of unsupervised learning. Generative models have the ability to then generate new data based on the learned distribution of classes or groups. Generative models are considered general models. In mathematical terms, a generative model can provide the probability of a data point x existing given that it is from class $k, p(\mathsf{x}|C_k)$. In summary, generative models study the attributes that exist in all points within a group. We've already discussed several generative models such as Naive Bayes for classification, and Gaussian Mixture models for clustering.

Discriminative Models

Discriminative models are those that aim to learn the differences between the classes or groups within the data. They do not care about the attributes that a particular class possesses but only focus on the attributes that can be used to differentiate, or discriminate, between the groups within the data. Discriminative models are often effective at particular tasks since they are trained to do just that but may be difficult to use in a general sense since they only possess targeted knowledge. In mathematical terms, a discriminative mode provides the probability of a particular class k given a particular data point x, $p(C_k|\mathsf{x})$. In summary, discriminative models study the attributes that differ between groups. We've already discussed several discriminative models such as linear regression, support vector machines and traditional neural networks.

Generative Adversarial Networks

Generative Adversarial Networks (GANs) consist of two models, a discriminator and a generator, that are trained against each other via adversarial training. Given a data set, the discriminator is trained to determine whether a new sample it hasn't seen is in fact a member of this data set or not. The generator is trained to generate data points that appear as if they are from the dataset. The fully trained generator is able to produce data samples that are indistinguishable from the real original dataset.

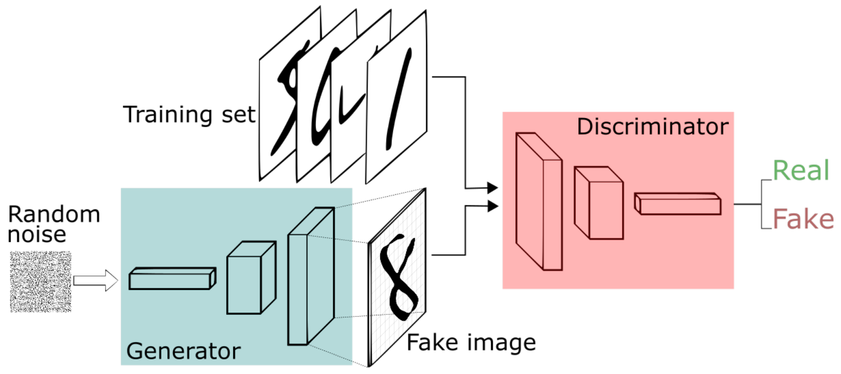

Figure 1: An illustration of a GAN applied to the MNIST dataset. Source

We use p_{data} and p_{gen} to represent the probability distribution for the original data and the probability distribution for the generated samples respectively. The goal of the discriminator is to maximize the difference between those two distributions and the goal of the generator is to minimize this difference. This leads us to a *minmax* problem. A minmax problem is a problem with competing adversaries where the goal is to minimize the loss assuming the adversary will maximize their gain.

A *maxmin* problem is the flip side of minmax where the goal is to maximize the gain assuming the adversary will minimize their loss.

We can measure the similarity between two probability distributions using cross-entropy. In our case, we have two classes so we will use binary cross entropy which is defined as,

$$H(p,q) = -(y \log p + (1-y)\log(1-p)), \tag{1}$$

where y is the indicator for the class and we assume that $1-p=q$ since the data sample can only be one of the classes.

We represent the discriminator model's output as $D(\mathsf{x})$ which is the probability that x belongs to the original data. We represent a random input to the generator as z and the generator's output as $G(\mathsf{z})$. We can represent the objective for GANs as the minimax problem,

$$\min_G \max_D E_{x\sim p_{data}}[\log D(\mathsf{x})] + E_{z\sim p_z}[1-\log D(G(\mathsf{z}))]. \tag{2}$$

This can be broken down into two portions that are more clear. The discriminator aims to separate between two classes and so it uses the usual binary cross entropy formulation minimization. Here the discriminator aims to minimize the similarity between the two distributions,

$$\min_D -E_{x\sim p_{data}}[\log D(\mathsf{x})] - E_{z\sim p_z}[1 - \log D(G(\mathsf{z}))], \tag{3}$$
$$= \max_D E_{x\sim p_{data}}[\log D(\mathsf{x})] + E_{z\sim p_z}[1 - \log D(G(\mathsf{z}))]. \tag{4}$$

The generator aims to produce samples that have a high similarity with the original data and so the goal for the generator is to maximize the binary cross entropy,

$$\max_G -E_{x\sim p_{data}}[\log D(\mathsf{x})] - E_{z\sim p_z}[1 - \log D(G(\mathsf{z}))], \tag{5}$$
$$= \min_G E_{x\sim p_{data}}[\log D(\mathsf{x})] + E_{z\sim p_z}[1 - \log D(G(\mathsf{z}))]. \tag{6}$$

By combining these two separate formulations we get the previous min-max problem which can be solved in conjunction. By iterating over the solution to this formulation from each model's perspective, both models get to be very efficient. The final optimal model for our generator is when the discriminator can not distinguish between it and the real data. This means that the prediction is about 0.5 data and 0.5 generation.

GANs have a simple approach; train two opposing models that push each other to improve, but in practice they have many issues. GANs are often difficult to train, their results may not converge, the discriminator may be too good that the generator doesn't learn to beat it and many other problems. We will now discuss some of the problems that arise and variations on GANs that attempt to alleviate some of the problems or improve their performance.

Mode Collapse

Since in practice we iterate over updating the discriminator and the generator there isn't a clear distinction between solving the minmax problem or the maxmin problem. There is a problem with solving the maxmin problem instead of the minmax problem however since,

$$\min_G \max_D V(G,D) \neq \max_D \min_G V(G,D). \tag{7}$$

If we are updating the discriminator first, the generator has to learn new creative ways to manipulate it. If we update the generator first then it will maintain the result that managed to bypass the discriminator. The resulting output is that the generator will only focus on some samples within the data to generate and will not generate all instances provided in the dataset. This issue is called *mode collapse*.

Variations on GANs

There are many variations on GANs that have been developed since the publication of the original paper. Some variations focus on solving the core issue with GANs regarding training whereas others aim to focus on applying them to specific application areas. Here are some of the introduced variations,

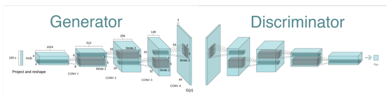

Figure 2: An illustration of DCGANs.

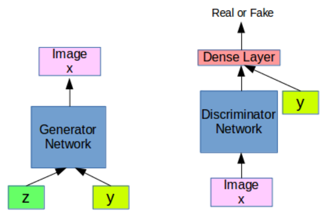

Figure 3: An illustration of CGANs and how they incorporate the condition y into the input.

- Deep Convolutional GAN (DCGAN) - Optimized for applications that benefit from the introduction of convolutional layers. They simply add convolutional layers to the discriminator and generator networks with the generators' convolutional layers functioning in reverse. The generator's convolutional layers start with a deep and low dimensionality representation and increase dimensionality and lower depth to create an image similar to a real data sample.
- Unrolled GAN - Performs multiple generator gradient updates while using updated discriminator weights without applying them to updating the discriminator itself. This reduces likelihood of mode collapse and improves training ability.[1] This approach is simply a training methodology and can be combined with other variations.
- Wasserstein GAN (WGAN) - Updates the loss function to use the Wasserstein metric[2] instead of binary cross entropy. This approach improves training ability and likelihood of mode collapse. This approach updates the loss function and so it can be combined with other variations that modify other aspects of the neural network.
- Conditional GAN (CGAN) - Provide an additional input to the GAN that is a "condition" that is applied to create an expected output type.

[1] In addition to the original paper, the following tutorial on Unrolled GANs may be useful.

[2] The Wasserstein metric is defined as

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)]$$

For more variations of GANs, check out the GAN zoo.

Example GAN Applications

There are many examples of applications of GANs and here are some of them,

- Image style transfer
- Generating photos of nonexistent objects, animals or people
- Music composition
- Poetry / text authorship

GANs are still a new and exciting field in machine learning with more applications soon to be discovered.

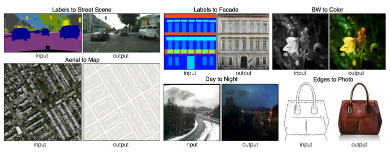

Figure 4: An illustration of pix2pix. Source

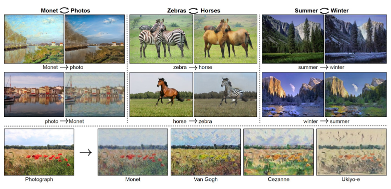

Figure 5: An illustration of CycleGAN. Source

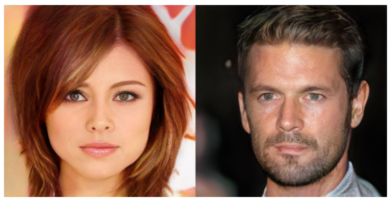

Figure 6: Example of generated celebrity photos. Source