What’s a generative adversarial network? If you haven’t yet heard of generative adversarial networks, don’t worry, you will. The hottest topic in deep learning, GANs, as they’re called, have the potential to create systems that learn more with less help from humans.

Traditionally stating, *Generative adversarial networks (GANs) are an exciting recent innovation in machine learning. GANs are generative models: they create new data instances that resemble your training data.*

The field of Machine Learning is something where everything is explained by their name. So, whenever we hear the name *Generative*, we instantly imagine a scenario that an object has the capability of having the power or function of generating, originating, producing, or reproducing. And this is exactly what we are trying to do in GAN’s. In Machine Learning, "Generative" describes a class of statistical models that contrasts with discriminative models.

To start this discussion, we can informally describe the architecture of GAN’s. Informally **Generative** models can generate new data instances whereas **Discriminative** models discriminate between different kinds of data instances. A generative model could generate new photos of animals that look like real animals, while a discriminative model could tell a dog from a cat. GANs are just one kind of generative model.

Moving towards a more formal definition of GAN’s by introducing some concepts of Probability, wherein given a set of data instances X and a set of labels Y:

* **Generative** models capture the joint probability p(X, Y), or just p(X) if there are no labels.
* **Discriminative** models capture the conditional probability p(Y | X).

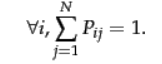
To understand GAN’s, we need to understand that the concept of generating new data is not something new. Hence, it is very important to know the traditional statistical and probabilistic approach towards generating data. We have Markov Chains that explains a somewhat similar concept.

A Markov chain is a *discrete-time stochastic process:* a process that occurs in a series of time-steps in each of which a random choice is made.

A Markov chain consists of  *states*. Each web page will correspond to a state in the Markov chain we will formulate.

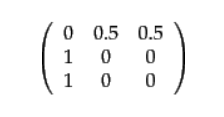
A Markov chain is characterized by an  transition probability matrix  each of whose entries is in the interval [0,1]; the entries in each row of add up to 1. The Markov chain can be in one of the  states at any given time-step; then, the entry   tells us the probability that the state at the next time-step is j, conditioned on the current state being i. Each entry  is known as a transition probability and depends only on the current state i; this is known as the Markov property. Thus, by the Markov property,

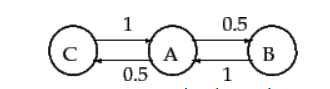


and, 

A matrix with non-negative entries that satisfies the above equation is known as a *stochastic matrix*. A key property of a stochastic matrix is that it has a *principal left eigenvector* corresponding to its largest eigenvalue, which is 1.

In a Markov chain, the probability distribution of next states for a Markov chain depends only on the current state, and not on how the Markov chain arrived at the current state. Figure 1 shows a simple Markov chain with three states. From the middle state A, we proceed with (equal) probabilities of 0.5 to either B or C. From either B or C, we proceed with probability 1 to A. The transition probability matrix of this Markov chain is then





***Figure 1:*** *A simple Markov chain with three states; the numbers on the links indicate the transition probabilities.*

A Markov chain's probability distribution over its states may be viewed as a *probability vector*: a vector all whose entries are in the interval [0,1], and the entries add up to 1. An  -dimensional probability vector each of whose components corresponds to one of the  states of a Markov chain can be viewed as a probability distribution over its states. For our simple Markov chain of Figure 1, the probability vector would have 3 components that sum to 1.

We can view a random surfer on the web graph as a Markov chain, with one state for each web page, and each transition probability representing the probability of moving from one web page to another. The teleport operation contributes to these transition probabilities. The adjacency matrix  of the web graph is defined as follows: if there is a hyperlink from page i to page j, then , otherwise . We can readily derive the transition probability matrix   for our Markov chain from the  matrix  :

* If a row of   has no 1's, then replace each element by 1/N. For all other rows proceed as follows.
* Divide each 1 in   by the number of 1's in its row. Thus, if there is a row with three 1's, then each of them is replaced by 1/3.
* Multiply the resulting matrix by .
* Add α/N to every entry of the resulting matrix, to obtain  .

If a Markov chain can run for many time steps, each state is visited at a (different) frequency that depends on the structure of the Markov chain. In our running analogy, the surfer visits certain web pages (say, popular news home pages) more often than other pages. We now make this intuition precise, establishing conditions under which such the visit frequency converges to fixed, steady-state quantity. Following this, we set the PageRank of each node   to this steady-state visit frequency and show how it can be computed.

After a careful study of Markov Chains, we are now in a very comfortable situation to study more about GAN’s. *A generative model includes the distribution of the data itself and tells you how likely a given example is. For example, models that predict the next word in a sequence are typically generative models (usually much simpler than GANs) because they can assign a probability to a sequence of words.*

*A discriminative model ignores the question of whether a given instance is likely, and just tells you how likely a label is to apply to the instance.*

*\*NOTE: This is a very generalized definition of GAN as they are many other variant of Generative Adversarial Network.*

Neither kind of model has to return a number representing a probability. We can model the distribution of data by imitating that distribution.

Surprisingly, there is a general intuition between decision tree classifier and GAN’s as well. a discriminative classifier like a decision tree can label an instance without assigning a probability to that label. Such a classifier would still be a model because the distribution of all predicted labels would model the real distribution of labels in the data.

Similarly, a generative model can model a distribution by producing convincing "fake" data that looks like it's drawn from that distribution.

**GENERATIVE MODELS ARE HARD:**

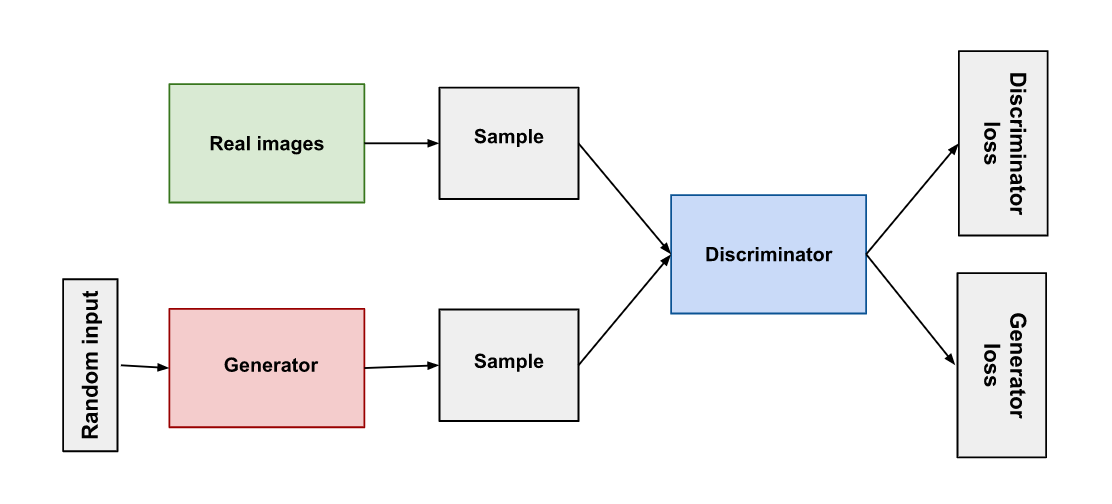
Generative models tackle a more difficult task than analogous discriminative models. Generative models have to model more. A generative model for images might capture correlations like "things that look like boats are probably going to appear near things that look like water" and "eyes are unlikely to appear on foreheads." These are very complicated distributions. In contrast, a discriminative model might learn the difference between "sailboat" or "not sailboat" by just looking for a few tell-tale patterns. It could ignore many of the correlations that the generative model must get right. Discriminative models try to draw boundaries in the data space, while generative models try to model how data is placed throughout the space. For example, the following diagram shows discriminative and generative models of handwritten digits:

Two graphs, one labelled 'Discriminative Model'
          and the other labelled 'Generative Model'. Both graphs show
          the same four datapoints. Each point is labeled with the image
          of the handwritten digit that it represents. In the discriminative
          graph there's a dotted line separating two data points from the
          remaining two. The region above the dotted line is labelled 'y=0' and
          the region below the line is labelled 'y=1'. In the generative graph
          two dotted-line circles are drawn around the two pairs of points. The
          top circle is labelled 'y=0' and the bottom circle is labelled 'y=1

***Figure 2: Discriminative and generative models of handwritten digits.***

The discriminative model tries to tell the difference between handwritten 0's and 1's by drawing a line in the data space. If it gets the line right, it can distinguish 0's from 1's without ever having to model exactly where the instances are placed in the data space on either side of the line. In contrast, the generative model tries to produce convincing 1's and 0's by generating digits that fall close to their real counterparts in the data space. It has to model the distribution throughout the data space. GANs offer an effective way to train such rich models to resemble a real distribution. To understand how they work we'll need to understand the basic structure of a GAN.

**GENERAL OVERVIEW OF GAN:**

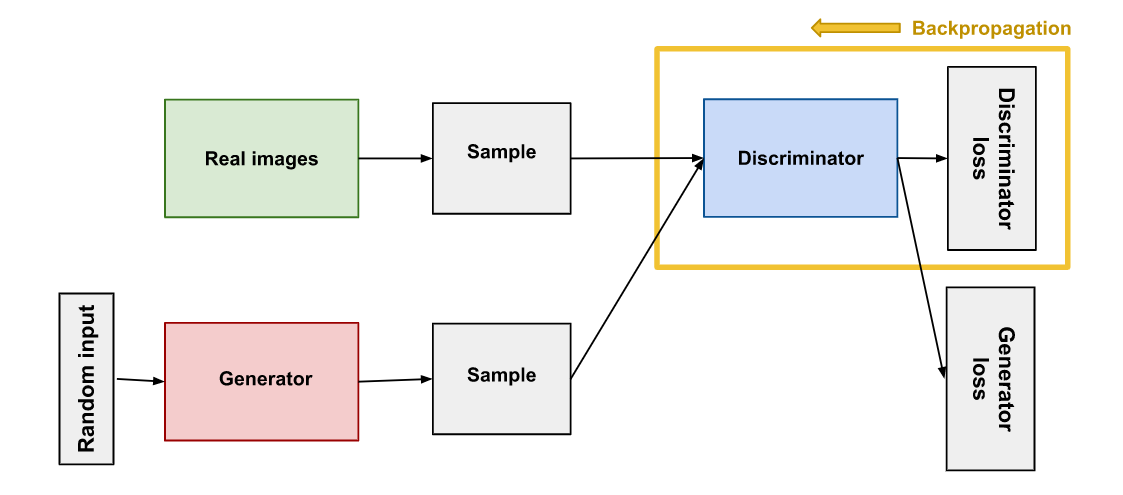


***Figure 3***

Both the generator and the discriminator are neural networks. The generator output is connected directly to the discriminator input. Through backpropagation, the discriminator's classification provides a signal that the generator uses to update its weights.

**DISCRIMINATOR:**

The discriminator in a GAN is simply a classifier. It tries to distinguish real data from the data created by the generator. It could use any network architecture appropriate to the type of data it's classifying.



***Figure 4***

The discriminator's training data comes from two sources:

* **Real data** instances, such as real pictures of people. The discriminator uses these instances as positive examples during training.
* **Fake data** instances created by the generator. The discriminator uses these instances as negative examples during training.

From figure 4 we can see that the two "Sample" boxes represent these two data sources feeding into the discriminator. During discriminator training the generator does not train. Its weights remain constant while it produces examples for the discriminator to train on.

The discriminator connects to two loss functions. During discriminator training, the discriminator ignores the generator loss and just uses the discriminator loss.

During discriminator training:

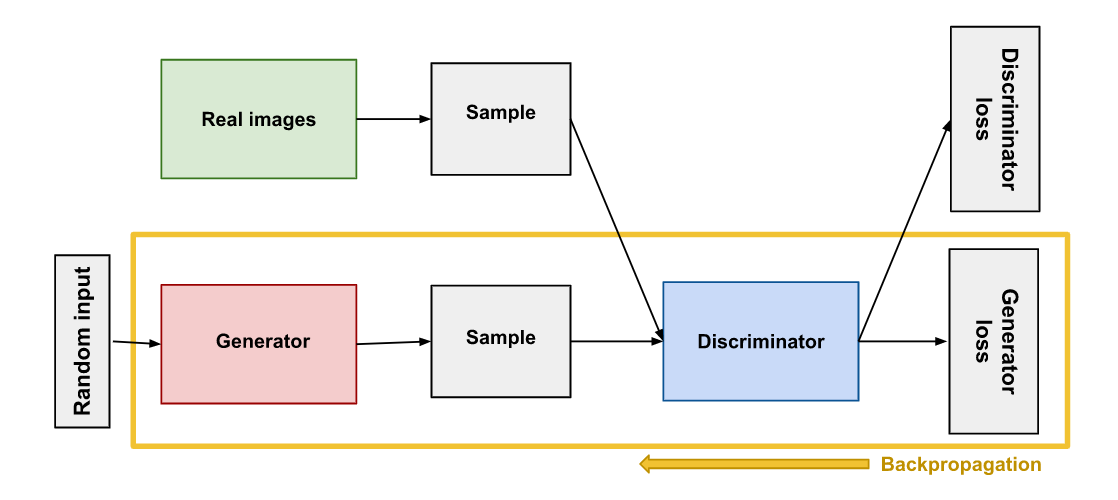
1. The discriminator classifies both real data and fake data from the generator.
2. The discriminator loss penalizes the discriminator for misclassifying a real instance as fake or a fake instance as real.
3. The discriminator updates its weights through backpropagation from the discriminator loss through the discriminator network.

**GENERATOR:**

The generator part of a GAN learns to create fake data by incorporating feedback from the discriminator. It learns to make the discriminator classify its output as real.

Generator training requires tighter integration between the generator and the discriminator than discriminator training requires. The portion of the GAN that trains the generator includes:

* random input
* generator network, which transforms the random input into a data instance
* discriminator network, which classifies the generated data
* discriminator output
* generator loss, which penalizes the generator for failing to fool the discriminator



***Figure 5***

Neural networks need some form of input. Normally we input data that we want to do something with, like an instance that we want to classify or make a prediction about. But what do we use as input for a network that outputs entirely new data instances?

In its most basic form, a GAN takes random noise as its input. The generator then transforms this noise into a meaningful output. By introducing noise, we can get the GAN to produce a wide variety of data, sampling from different places in the target distribution.

Experiments suggest that the distribution of the noise doesn't matter much, so we can choose something that's easy to sample from, like a uniform distribution. For convenience the space from which the noise is sampled is usually of smaller dimension than the dimensionality of the output space.

To train a neural net, we alter the net's weights to reduce the error or loss of its output. In our GAN, however, the generator is not directly connected to the loss that we're trying to affect. The generator feeds into the discriminator net, and the *discriminator* produces the output we're trying to affect. The generator loss penalizes the generator for producing a sample that the discriminator network classifies as fake.

This extra chunk of network must be included in backpropagation. Backpropagation adjusts each weight in the right direction by calculating the weight's impact on the output — how the output would change if you changed the weight. But the impact of a generator weight depends on the impact of the discriminator weights it feeds into. So, backpropagation starts at the output and flows back through the discriminator into the generator.

At the same time, we don't want the discriminator to change during generator training. Trying to hit a moving target would make a hard problem even harder for the generator.

So, we train the generator with the following procedure:

1. Sample random noise.
2. Produce generator output from sampled random noise.
3. Get discriminator "Real" or "Fake" classification for generator output.
4. Calculate loss from discriminator classification.
5. Backpropagate through both the discriminator and generator to obtain gradients.
6. Use gradients to change only the generator weights.

This is one iteration of generator training. In the next section we'll see how to juggle the training of both the generator and the discriminator.

**CONVERGENCE:**

As the generator improves with training, the discriminator performance gets worse because the discriminator can't easily tell the difference between real and fake. If the generator succeeds perfectly, then the discriminator has a 50% accuracy. In effect, the discriminator flips a coin to make its prediction.

This progression poses a problem for convergence of the GAN as a whole: the discriminator feedback gets less meaningful over time. If the GAN continues training past the point when the discriminator is giving completely random feedback, then the generator starts to train on junk feedback, and its own quality may collapse.

For a GAN, convergence is often a fleeting, rather than stable, state.

**LOSS FUNCTIONS:**

GANs try to replicate a probability distribution. They should therefore use loss functions that reflect the distance between the distribution of the data generated by the GAN and the distribution of the real data.

Hence, we do come across a scenario where the question is that how do we capture the difference between two distributions in GAN loss functions? This question is an area of active research, and many approaches have been proposed.

In this article we will be addressing the two main loss functions that are addressed in GAN’s.

A GAN can have two loss functions: one for generator training and one for discriminator training. But now we have a question worth asking, “*How can two loss functions work together to reflect a distance measure between probability distributions?”*

In the loss schemes we'll look at here, the generator and discriminator losses derive from a single measure of distance between probability distributions. In both of these schemes, however, the generator can only affect one term in the distance measure: the term that reflects the distribution of the fake data. So, during generator training we drop the other term, which reflects the distribution of the real data.

The generator and discriminator losses look different in the end, even though they derive from a single formula.

In the paper that introduced GANs, the generator tries to minimize the following function while the discriminator tries to maximize it:

In this function:

* D(x) is the discriminator's estimate of the probability that real data instance x is real.
* Ex is the expected value over all real data instances.
* G(z) is the generator's output when given noise z.
* D(G(z)) is the discriminator's estimate of the probability that a fake instance is real.
* Ez is the expected value over all random inputs to the generator (in effect, the expected value over all generated fake instances G(z)).
* The formula derives from the cross-entropy between the real and generated distributions.

The generator can't directly affect the log(D(x)) term in the function, so, for the generator, minimizing the loss is equivalent to minimizing log(1 - D(G(z))).

But if we look at TF-GAN’s, we encounter that it uses Wasserstein Loss.

This loss function depends on a modification of the GAN scheme (called "Wasserstein GAN" or "WGAN") in which the discriminator does not actually classify instances. For each instance it outputs a number. This number does not have to be less than one or greater than 0, so we can't use 0.5 as a threshold to decide whether an instance is real or fake. Discriminator training just tries to make the output bigger for real instances than for fake instances.

Because it can't really discriminate between real and fake, the WGAN discriminator is actually called a "critic" instead of a "discriminator". This distinction has theoretical importance, but for practical purposes we can treat it as an acknowledgement that the inputs to the loss functions don't have to be probabilities.

The loss functions themselves are deceptively simple:

* **Critical Loss:** *D(x) – D(G(z))*

The discriminator tries to maximize this function. In other words, it tries to maximize the difference between its output on real instances and its output on fake instances.

* **Generator Loss:** *D(G(z))*

The generator tries to maximize this function. In other words, it tries to maximize the discriminator's output for its fake instances.

Wasserstein GANs are less vulnerable to getting stuck than minimax-based GANs, and avoid problems with vanishing gradients. The earth mover distance also has the advantage of being a true metric: a measure of distance in a space of probability distributions. Cross-entropy is not a metric in this sense.

For a much practical approach I am attaching the results from WGAN’s on the dataset that I performed.

**REFERENCE:**

* Material related to Markov Chains : <https://nlp.stanford.edu/IR-book/html/htmledition/markov-chains-1.html>