Generative Adversarial Networks

- For hosted blog post with animations, venture to https://littlepegs.github.io/ (https://littlepegs.github.io/)

Where does it come from? Who are its ancestors?

Generative Adversarial Networks (GANs) were first introduced to the public in June of 2014. Ian Goodfellow and his team at University of Montreal were working on a new approach to generative models, and what started as an interesting idea Goodfellow had at a bar resulted in a seminal paper that would leave much of the machine learning community in some mode of astonishment.

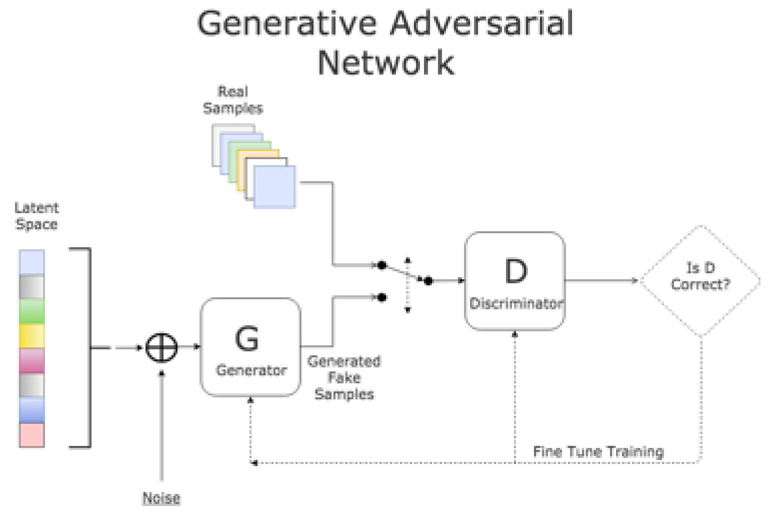
To understand what GANs are and what they do, it is useful to know what a Generative Model is. A generative model is a unsupervised learning technique which takes a sequence of input samples and attempts to predict the most likely outcome given that input. An example is the word prediction feature on modern smart phones. Given a set of words typed by the user, the model will predict a a word most likely to be the user's next choice.

The basic process of a generative model can be seen as one that takes in a random seed from some distribution and transforms it into an output that looks and feels like a sample from the real data. These models were historically popular in realms like classical statistics and statistical physics, where one application included modeling phase transitions in thermodynamics. During the 1980's and 90's, these models began making their way to machine learning and neuroscience, leading to new developments in generative-style neural nets such as as variational and denoising autoencoders.

What do they do differently, and how?

GANs apply this baseline concept to a neural network architecture to achieve results superior to the traditional neural networks. The GAN framework implements two nets with adversarial roles. The first net, called the Generator (G), creates 'fake' samples that resemble the true data, while the Discriminator (D) is fed both samples from the real dataset and 'fake' ones from the generator. The discriminator's chief task is then to distinguish if the sample is from the real set or if it is 'fake'. With its guess in place, D is rewarded if it makes a correct prediction. Running this process iteratively, D is rewarded with the total number of correct predictions it makes, while G is rewarded every time D makes a mistake.

To understand this intuitively, the classic analogy of counterfeiters comes to mind: counterfeiters have made some false currency and would like to convince the authorities that, indeed, their cash is legitimate. The investigators' mission is then to distinguish real cash from forged, and to punish the counterfeiters accordingly.



Lets look a slightly deeper into this novel framework. Let Preal and Psynth be the distribution over real data and synthetic data, respectively. The discriminator D can be trained such that it maps inputs, such as images, to numbers in the range [0,1] and attempts to discern which distribution the image came from. Then, its expected output Ex[D(x)] is as high as possible when x is a member of Preal and as low as possible when it is a member of Psynth. Now we can use the training data to train D using backpropagation. Once D has been trained, the adversaries may begin their game.

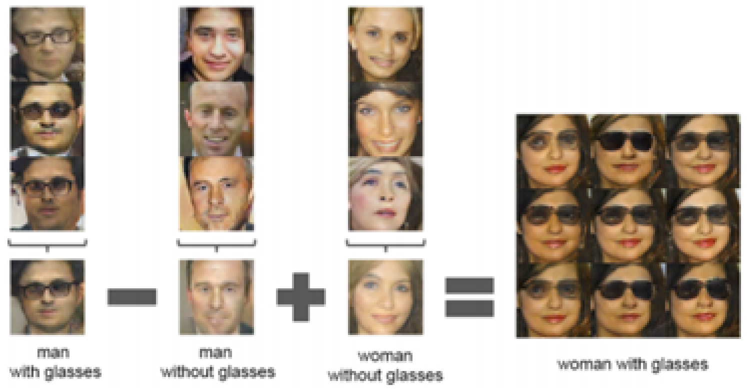
The goal of the generator G is to create instances z in Psynth such that Ex[D(z)] is as high as possible in order to fool D into believing they are real. Once these base goals have been laid out, the training of G begins. On each 'move', D will take a samples from both P distributions and improve its capability to differentiate between. Meanwhile, G produces some instances from its forgery plant and updates its parameters so that the expected discriminator output (given G's samples) increases slightly. This training continues until the expected output on instances from Preal and Psynth become equal. That is, when G has trained enough so that its generated fakes samples can no longer be differentiated from the real data by the discriminator. Returning to the analogy, the game stops when the forger prevails.

Applications and Future Promise

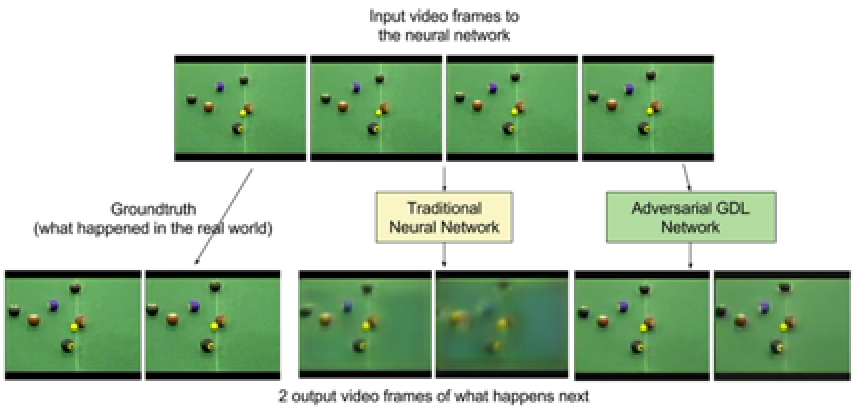
One of the prominent applications in which GANs have differentiated themselves is in the generation of novel images based on the natural world. Given a large dataset to train on, the images generated by GANs approach realism and are difficult to differentiate from photographs. Here is a set of cat images synthesized by GANs:

It has also been shown that GANs are able to learn object hierarchies. Researchers at Facebook AI experimented with a bedroom image dataset by removing the representation of windows from all images. What they found was that the GAN was able to replace this space in the bedroom with objects such as televisions and doors. This insight corroborates the idea that the network is able to differentiate representations of scenes from those of objects.

Further experiments on faces data, for example, has found that GANs are able to represent features such as smiles, and learn objects attributes like scale, rotation, and position. In one experiment, arithmetic applied to face images reveals the various relationships that a GAN is able to capture.



The better GANs are able to learn how the world looks and behaves, the better they are able to predict what future renditions and instances will look like. Work into this field has already begun in the form of video frame prediction. In this application, a network is trained using video frames and is asked to predict the next frame.


Input video frames to the neural network
Groundtruth (what happened in the real world)
Traditional Neural Network
Adversarial GDL Network
2 output video frames of what happens next

The results are of noticeably higher quality than those produced by the conventional neural network. The images predicted are crisp, contain well defined edges, and are immediately recognizable as ‘real’ to the unsuspecting observer. Without ever being briefed on the laws of motion or the physical properties of the context, the adversarial network recognizes patterns from its training data and is able to generate realistic scenes as predictions. With continued improvements, we could see GANs completely change what is possible in the realms of video scene enhancement, text-to-image synthesis, generative visual manipulation, and countless others.

Using GANs to generate MNIST digits

Let us see if we can implement what we have thus far discussed. We will download the popular MNIST digit dataset, and use TensorFlow to create some adversarial nets. Using the MNIST data, we will train the discriminator (D) and then run the alternation game, where the generator G will seek to improve its digit 'imposters' until D can (and hopefully the reader) no longer differentiate whether it is a real or a fake.

First, we create two networks. This is a fairly open task, allowing the experimenter to build nets as complex or as simple as they desire. In order to get a clear picture of the process, we will build two simple 2-layer nets for D and G.

Of course, we'll start by importing some dependencies.

```
In [ ]: import tensorflow as tf
        from tensorflow.examples.tutorials.mnist import input_data
        import numpy as np
        import matplotlib.pyplot as plt
        import matplotlib.gridspec as gridspec
        import os
```

```
In [ ]: # the Discriminator
        X = tf.placeholder(tf.float32, shape=[None, 784])

        D_W1 = tf.Variable(xavier_init([784, 128]))
        D_b1 = tf.Variable(tf.zeros(shape=[128]))

        D_W2 = tf.Variable(xavier_init([128, 1]))
        D_b2 = tf.Variable(tf.zeros(shape=[1]))

        theta_D = [D_W1, D_W2, D_b1, D_b2]

        # the Generator
        Z = tf.placeholder(tf.float32, shape=[None, 100])

        G_W1 = tf.Variable(xavier_init([100, 128]))
        G_b1 = tf.Variable(tf.zeros(shape=[128]))

        G_W2 = tf.Variable(xavier_init([128, 784]))
        G_b2 = tf.Variable(tf.zeros(shape=[784]))

        theta_G = [G_W1, G_W2, G_b1, G_b2]
```

The generator will take as input a 100-dimensional vector and transform it into an output of 768 dimensions. That is, a vector representing a standard MNIST image of size 28x28.

The discriminator will take as input a MNIST image and return its prediction in the form of a scalar representing the probability it is a true image from the original dataset.

```
In [ ]: def generator(z):
            G_h1 = tf.nn.relu(tf.matmul(z, G_W1) + G_b1)
            G_log_prob = tf.matmul(G_h1, G_W2) + G_b2
            G_prob = tf.nn.sigmoid(G_log_prob)

            return G_prob


        def discriminator(x):
            D_h1 = tf.nn.relu(tf.matmul(x, D_W1) + D_b1)
            D_logit = tf.matmul(D_h1, D_W2) + D_b2
            D_prob = tf.nn.sigmoid(D_logit)

            return D_prob, D_logit
```

Now we are ready to define the loss function. Following Goodfellow's formulation in his 2014 paper presented at NIPS, we have:

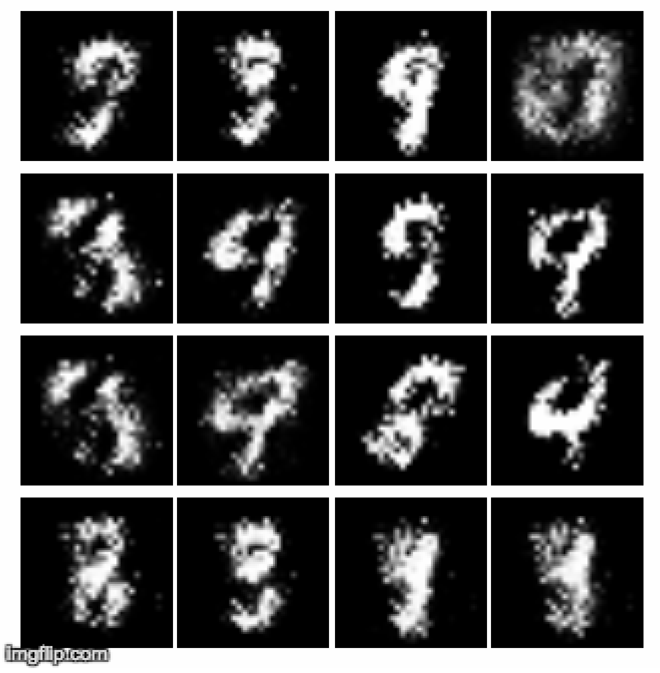
```
In [ ]: G_sample = generator(Z)
        D_real, D_logit_real = discriminator(X)
        D_fake, D_logit_fake = discriminator(G_sample)

        D_loss = -tf.reduce_mean(tf.log(D_real) + tf.log(1. - D_fake))
        G_loss = -tf.reduce_mean(tf.log(D_fake))
```

Since TensorFlow's optimizer can only hanlde minimizations, we take the opposite of the loss and attempt to minimize it.

Now we can let the adversaries tango, if you will. We run the training process for 100,000 iterations locally (given computational constraints, namely 4GB RAM).

```
In [ ]: def sample_Z(m, n):
            return np.random.uniform(-1., 1., size=[m, n])

        for it in range(100000):
            if it % 1000 == 0:
                samples = sess.run(G_sample, feed_dict={Z: sample_Z(16, Z_dim)})

                fig = plot(samples)
                plt.savefig('out2/{}.png'.format(str(i).zfill(3)),
        bbox_inches='tight')
                i += 1
                plt.close(fig)

            X_mb, _ = mnist.train.next_batch(mb_size)

            _, D_loss_curr = sess.run([D_solver, D_loss], feed_dict={X: X_mb, Z:
         sample_Z(mb_size, Z_dim)})
            _, G_loss_curr = sess.run([G_solver, G_loss], feed_dict={Z:
        sample_Z(mb_size, Z_dim)})
```

By sampling the generator every 1000 iterations, we can get a glimpse of the training process. Amazingly, in only 6000 iterations, the generated images go from random blurr to semi-destinguishable digits (Don't worry, we will see the result of 100k iterations soon...).

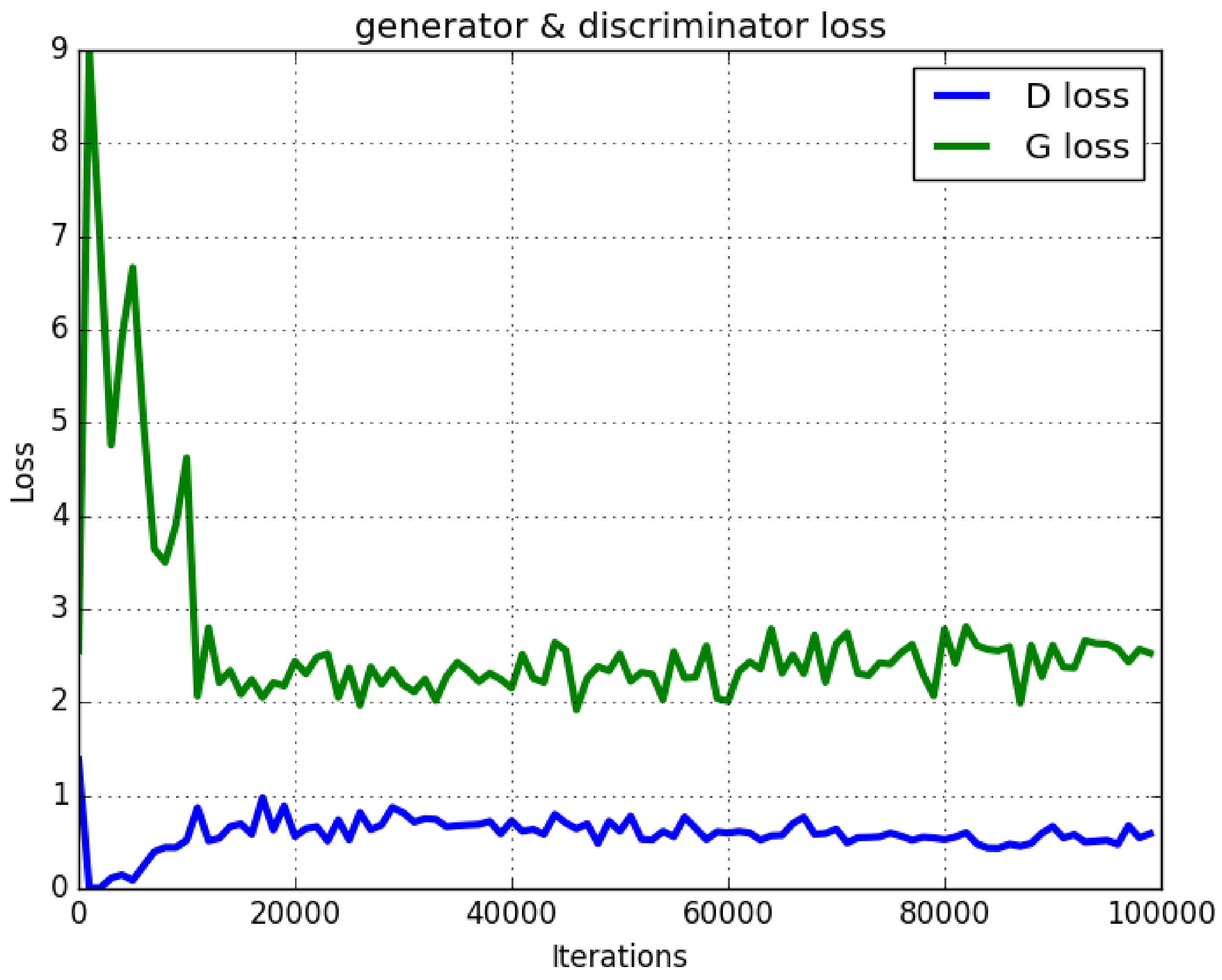


We can also try defining the loss functions slightly differently and see if the results are still in line with what we want. Considering D(x) given some input x, D wishes to minimize D(x) when x is fake, and maximize it when x is real. In other words, the discriminator seeks to minimize the expression D(G(x)). On the other hand, the generator wants to maximize the probability it is accepted as a 'real' sample: max D(G(x)).

Using this alternative definition, we have the following loss functions:

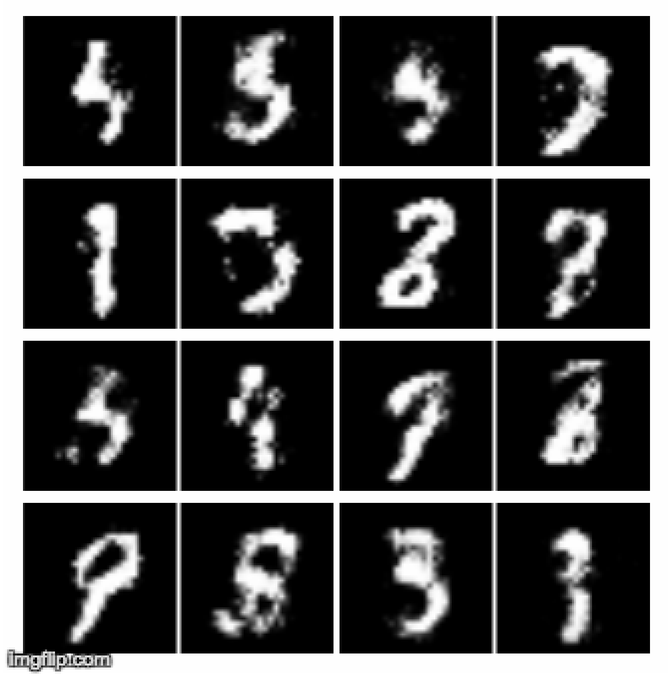
```
In [ ]: D_loss_real = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(log
        its=D_logit_real, labels=tf.ones_like(D_logit_real)))
        D_loss_fake = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(log
        its=D_logit_fake, labels=tf.zeros_like(D_logit_fake)))
        D_loss = D_loss_real + D_loss_fake
        G_loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=D
        _logit_fake, labels=tf.ones_like(D_logit_fake)))
```

Training the GAN for 100,000 iterations again, we observe loss as a function of iterations in the below graph.



From this graph we can see the generator loss drops dramatically in the first 15,000 iterations and then stabilizes with some fluctuation until the end of training. Similarly, we see discriminator loss drop sharply in the first few iterations, but then return to a homeostasis between 0 and 1.

Sampling the generator every 1000 iterations, we see well defined digits born out of what started as a nebulous zygote. In this animation, we see a sequence of 100 frames, one for every 1000 iterations.



The results are pretty staggering, considering the generator's first images and how rapidly they became viable MNIST candidates.

Concluding Remarks

In this blog post, we brought generative adversarial networks to the fore and discussed their defining properties, as defined by their creator - Ian Goodfellow. Looking at the dynamic between the generative and discriminative nets, we saw what makes the adversarial training unique and so effective.

With these fundementals, we were able to see how GANs are producing such realistic results in the field of image generation and how they might be improved to create convincing video frame prediction in the futre.

Lastly, we implemented a GAN using TensorFlow and the used MNIST digit dataset to produce digit images very comparable to the originals. Two different loss functions were applied, and each produced results of admirable quality.

We have seen the potential for GANs in producing realistic images. However, open questions remain about an appropriate metric with which to assess the GAN's results. In the case of images, the results are easy to intuit and compare. In other settings, such as text analysis, the results produced by these networks are more difficult to describe and to compare. In future work, we will likely see developments addressing this difficulty and applications whose beginnings we have only gotten a glimpse of.

References

- https://code.facebook.com/posts/1587249151575490/a-path-to-unsupervised-learning-through-adversarial-networks/ (https://code.facebook.com/posts/1587249151575490/a-path-to-unsupervised-learning-through-adversarial-networks/)
- http://blog.evjang.com/2016/06/generative-adversarial-nets-in.html (http://blog.evjang.com/2016/06/generative-adversarial-nets-in.html)
- http://wiseodd.github.io/techblog/2016/09/17/gan-tensorflow/ (http://wiseodd.github.io/techblog/2016/09/17/gan-tensorflow/)
- http://www.offconvex.org/2017/03/15/GANs/ (http://www.offconvex.org/2017/03/15/GANs/)
- https://arxiv.org/pdf/1406.2661.pdf (https://arxiv.org/pdf/1406.2661.pdf)