The competition **played out(逐渐发生，展开)** in public on Kaggle, and online community owned by Google that allows data scientists and machine learners to find and publish data sets, …

A bag of tricks

Training GANs and tuning GAN implementations is notoriously difficult. There are a number of known "tricks" that **one should keep in mind**. Like most things in deep learning, it is more alchemy than science: **these tricks are really just heuristics, not theory-backed guidelines**. They are backed by some level of intuitive understanding of the phenomenon at hand, and they are known to work well empirically, albeit not necessarily in every context.

Here are a few of the tricks that we leverage in our own implementation of a GAN generator and discriminator below. It is **not an exhaustive list** of GAN-related tricks; you will find many more across the GAN literature.

* We use tanh as the last activation in the generator, instead of sigmoid, which would be more commonly found in other types of models.
* We sample points from the latent space using a normal distribution (Gaussian distribution), not a uniform distribution.
* Stochasticity is good to induce robustness. Since GAN training results in a dynamic equilibrium, GANs are likely to get "stuck" in all sorts of ways. Introducing randomness during training helps prevent this. We introduce randomness in two ways: 1) we use dropout in the discriminator, 2) we add some random noise to the labels for the discriminator.
* Sparse gradients can hinder GAN training. In deep learning, sparsity is often a desirable property, but not in GANs. There are two things that can induce gradient sparsity: 1) max pooling operations, 2) ReLU activations. Instead of max pooling, we recommend using strided convolutions for downsampling, and we recommend using a LeakyReLU layer instead of a ReLU activation. It is similar to ReLU but it relaxes sparsity constraints by allowing small negative activation values.
* In generated images, it is common to see "checkerboard artifacts" caused by unequal coverage of the pixel space in the generator. To fix this, we use a kernel size that is divisible by the stride size, whenever we use a strided Conv2DTranpose or Conv2D in both the generator and discriminator.

First, we develop a generator model, which turns a vector into a candidate image. One of the many issues that commonly arise with GANs is that the generator gets stuck with generated images that look like noise. A possible solution is to use dropout on both the discriminator and generator.

Then, we develop a discriminator model, that takes as input a candidate image (real or synthetic) and classifies it into one of two classes, either "generated image" or "real image that comes from the training set".

Finally, we setup the GAN, which chains the generator and the discriminator.

Alex attributes his success in the competition to antother variant of GBM algorithm called XGBoost. Howver despite its massive popularity