Sameera Ramasinghe Follow
Graduate Student in Deep Learning@ANU, Co-Founder@ConscientAI
Feb 14 · 12 min read

Generative Adversarial Networks — A Theoretical Walk-Through.

Since Ian Goodfellow first proposed the idea of GANs (https://arxiv.org/abs/1406.2661), it has become a buzz word within ML community, simply because it works stunningly well (given that you came up with a perfect architecture). Many people, specially Yann LeCun, a who is considered as one of the giants in Deep Learning, stated at some point that GANs are a significant breakthrough in deep learning.

One thing I have noticed is that many people who claim to be familiar with GANs lack the theoretical foundation that lies beneath it, which is important.

READER ALERT: This post is not for absolute noobs. I assume you have a basic understanding on what GANs actually do in a practical point of view, as I will not discuss them here. I will use terms such as Discriminator and Generator without much explanation, assuming that you know it already. In a nutshell, GANs are a set of generative models, which can learn to generate data (ideally) identical to a given data distribution. At present, it is mainly used in computer vision domain to generate semantically meaningful images for various purposes.

Here, I will go through each important theoretical point mentioned in the original paper and try to explain it in simple terms along with derivations, whenever necessary.

Enough talk…So here we go…

First, we'll focus on the most important equation in the whole paper.

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

Let's carefully pay attention to each of the terms in this equation.

D(x): The discriminator function. Simply put, if you input a '*x*' data point (generated or from the original dataset) through *D*, it will output a scaler value between 0 and 1. This value is the probability that '*x*' is from the original dataset. Let's repeat. Keep in mind that *D(x)* outputs the **probability that '*x*' is from the original dataset. Not the other way around.** Ideally, we want the *D(x)* (at the equilibrium point) to output 0.5 to every data point of *x* distribution, whether it's from the generator or from the original dataset. Intuitively this means that the *D(x)* cannot distinguish between generated data and original data, which implies that generator is generating data perfectly matching with the original distribution.

G(z): The generator function. In here, *z* is the noise vector, which is the input to the generator function. The output of *G(z)* is a matrix whose dimensions are equal to *x*'s. Ideally, we want *G(z)* to output matrices which are indistinguishable from the original data (*x*) distribution.

If you look closely at the Equation 1, there are two loops. The objective of the inner loop is to maximize the right hand side expression as far as possible (By only tweaking *D*'s parameters). The objective of the outer loop is to minimize the right hand side expression as far as possible (By only tweaking *G*'s parameters).

Let's see what does this mean intuitively.

To the whole function to be maximized, the first term *E(log(D(x))* needs to be maximized. Which means *D(x)* needs to be maximized. If you can remember the *log (x)* plot, you will figure out that when *D(x)* becomes close to 1, *E(log(D(x))* becomes close to 0. When *D(x)* becomes close to 0 *E(log(D(x))* becomes close to *-infinity*. Which means that when maximizing the first term, the *D(x)* will try to output values close to 1, for original data. Which is the purpose of *D(x)*.

Then let's look at the second term. The maximum value of *1-log(D(G(z))* is positive *infinity*, and it gets that value when *D(G(z)) = 0*. Which means that when maximizing the second term, the *D(G(z))* will try to output values close to 0, which is the purpose of *D(x)*.

Now let's have a look at the outer loop.

The objective of the outer loop is to minimize the right hand side equation by only tweaking *G* parameters. Which means we will only be changing *G*. Now the first term of the right hand side does not depend on *G*. This means that we can ignore the first term when considering outer loop. When trying to minimize the second term, lowest value it could have is *-infinity*, which is achieved when *D(G(z)) = 1*. But *D* outputs 1 only when the input is from the original data. This means that *G(z)* will have to tweak its parameters in such a way that the output of *G(z)* will have to be as close as possible to original data distribution.

Right…Now after this analysis, you should understand that playing this minimax game results in making *G(z)* as close as possible to the original data distribution, make *D(x) = 1* when x is from original data, and make *D(G(z)) = 0*. **Now this the point where most of the people stop reading the analysis and assume that they have understood the whole paper**. Seems pretty straightforward right?

No…It's not straightforward. It's the rest of the analysis which is more interesting and contains the whole idea of GANs (specially regarding the convergence of GANs).

Now let's see what is missing in the above explanation. Observe that the inner loop of the minimax game tries to push *P_g~D(G(z))* towards 0, while outer loop tries to push *P_g~D(G(z))* towards 1. So, where would this game end? What would happen if the inner loop wins ? Or if outer loop wins? Is there an equilibrium point at all? If so, does that equilibrium point produce the optimum generator? The rest of the analysis is purely there to answer these questions.

I am going to repeat the two key questions again.

1. Is there an equilibrium point to this minimax game?
2. If there is, does that point produce the optimum generator?

We need 'yes' as the answer to both these questions. Let's see if that's the case.

By definition, *E(f(x)* of some function *f(x)* with respect to a probability distribution *p(x)* is the average value of *f(x)* when *x* is drawn from *p(x)*. Then *E(x)* is calculated as,

$$E_{x\sim p}[f(x)] = \int p(x)f(x)dx$$

Therefore, we can rewrite *V(D,G)* as,

$$V(G,D) = \int_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}) \log(D(\boldsymbol{x}))dx + \int_{z} p_{\boldsymbol{z}}(\boldsymbol{z}) \log(1 - D(g(\boldsymbol{z})))dz$$

Now comes the tricky part. LOTUS theorem that comes with statistics states that if *g(x)* = *x* and one knows *p(x)* but not *p(g(x)), E(g(x))* can be still found using,

$$E(g(x)) = \int g(x)p(x)dx$$

Now let,

$$F(x) = \int p_g(x)log(1 - D(x))dx$$

We know that,

$$x_g = g(z)$$

Then by LOTUS theorem,

$$F(x) = \int p(z) log(1 - D(g(z)))dx$$

Therefore, we can rewrite *V(D,G)* as,

$$= \int_{\boldsymbol{x}} p_{\text{data}}(\boldsymbol{x}) \log(D(\boldsymbol{x})) + p_g(\boldsymbol{x}) \log(1 - D(\boldsymbol{x}))dx$$

Consider function,

$$f(p_{data}, p_g) = p_{data}(x) log(D(x)) + p_g(x) log(1 - D(x))dx$$

Now visualize a 3D plot,

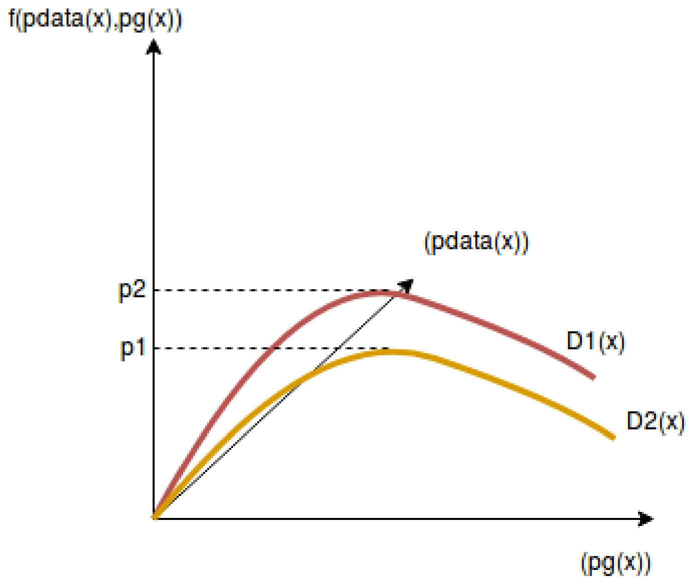


Figure 1

As you can see in the plot, different *D(x)* functions will give different *f(pdata, pg)* curves, for same data points of *pdata* and *pg*. *V(G,D)* is the area under the curve *f(pdata,pg)*. So, if we can find a *D*(x)* for every *(pdata,pg)* point, which gives the maximum *f(pdata,pg)* value for each of those points, integrating along *D*(x)* curve will give us the highest area under *f(pdata,pg)* curve.

How do we find the maximum of a function? easy. We differentiate it and find the locations where it is equal to zero. Considering each constant data point *(pdata,pg)* and differentiating w.r.t. *D(x)*,

$$f(p_{data}, p_g) = p_{data}(x) log(D(x)) + p_g(x) log(1 - D(x)) dx$$

$$f' = p_{data}(x)\frac{1}{D(x)ln(C)} - p_g(x)\frac{1}{(1-D(x))ln(C)}dx = 0$$

$$D(x) = \frac{p_{data}}{p_{data} + p_g}$$

This equation is quite intuitive. It says that for given two distributions, *pdata(original)* , *pg (generated)*, ideal discriminator should be able to identify the original data portion.

So as you can see, for any data point *(pdata,pg)*, if we choose *D(x) = D*(x)*, we will get the highest possible value for *f(pdata,pg)*. So integrating along that curve will give us the maximum value for *V(G,D)*. Note that *D*(x)* is not a static function. It will try to change it's value towards *D*(X)* in each *(pg, pdata)* point during the maximization of *V(G,D)*. Practically, that's what gradient descent tries to do to *D(x)* while training. Trying to bring it closer to *D*(x)* at each *(pg, pdata)* point, by tweaking its parameters through backpropagation. You should also notice that *pg* and *pdata* distributions are static during this procedure, as we do not change the generator (*G*) parameters.

So we can rewrite the equation,

$$\begin{aligned} C(G) =& \max_D V(G,D) \\ =& \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}[\log(1 - D_G^*(G(\boldsymbol{z})))] \\ =& \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}[\log D_G^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[\log(1 - D_G^*(\boldsymbol{x}))] \\ =& \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log \frac{p_{\text{data}}(\boldsymbol{x})}{P_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] + \mathbb{E}_{\boldsymbol{x}\sim p_g}\left[\log \frac{p_g(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right] \end{aligned}$$

Right…Now we focus on minimizing game on *C(G)* by tweaking *G(Z)* parameters where *C(G)* is,

$$=\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\left[\log\frac{p_{\text{data}}(\boldsymbol{x})}{P_{\text{data}}(\boldsymbol{x})+p_g(\boldsymbol{x})}\right]+\mathbb{E}_{\boldsymbol{x}\sim p_g}\left[\log\frac{p_g(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x})+p_g(\boldsymbol{x})}\right]$$

Some of you might be confused now, as you do not see a *G(z)* in the equation now, and must be wondering how are we going to minimize equation by tweaking *G(Z)*. Note that *pg* and *pdata* are both dependent on *G(Z)*. **So *G(Z)* is embedded in the equation.** Also, by tweaking *G(Z)* we are changing *D*(x)* too. That's why we are interested in finding out in the whole space of x, is there a point, which satisfies the condition for *D*(x)* (which is the goal of maximizing game), and also the minimum of *C(G)* (which is the goal of minimizing game). In other words, if there is such a point, then there is an equilibrium point to this minimax game. Otherwise, the two players will play this game forever without an agreement. Also, another important point is, does that point give us the optimum generator? (As you may see, I am just re-phrasing our two main questions)

We ask ourselves the question, "What constraint does the optimal generator impose upon *pg* and *pdata* ?". Obviously the answer is pg = pdata. In other words, the generated data distribution needs to be identical to the original data distribution. So we will start from that point.

Let *pg = pdata*. Then, *D*(x) = 1/2*, by maximizing game. Then, *C(G) = log(1/2) + log(1/2) = -log4*. To see that this is also the global minimum point of C(G) we subtract,

$$\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}[-\log 2]+\mathbb{E}_{\boldsymbol{x}\sim p_g}[-\log 2]=-\log 4$$

from *C(G)* and get,

$$C(G)-(-log(4)) = \int p_{data}[log\frac{p_{data}}{p_{data}+p_g}+log2] + \int p_g[log\frac{p_g}{p_{data}+p_g}+log2]$$

$$C(G)-(-log(4)) = \int p_{data}[log\frac{2p_{data}}{p_{data}+p_g}] + \int p_g[log\frac{2p_g}{p_{data}+p_g}]$$

$$C(G)-(-log(4)) = \int p_{data}[log\frac{p_{data}}{(p_{data}+p_g)/2}] + \int p_g[log\frac{p_g}{(p_{data}+p_g)/2}]$$

I assume you have heard of KL divergence. If you haven't, it's a measure of how much a given distribution differs from a second distribution. The definition of KL divergence is,

$$D_{KL}(P||Q) = \int p(x)log\frac{p(x)}{q(x)}$$

By applying this on above equation we get,

$$C(G)-(-log(4)) = D_{KL}(p_{data}||(p_{data}+p_g)/2) + D_{KL}(p_g||(p_{data}+p_g)/2)$$

By definition, *Jenson- Shanon divergence* between two distributions is given as,

$$JSD(P||Q) = \frac{1}{2}D_{KL}(P||M) + \frac{1}{2}D_{KL}(Q||M)$$

Using this we get,

$$C(G)-(-log(4)) = 2JSD(p_{data}||p_g)$$

Now, *Jenson- Shanon divergence* between two distributions is a non negative value, Therefore we get,

$$C(G) - (-log(4)) \geq 0$$

$$C(G) \geq (-log(4))$$

$$C(G)_{min} = (-log(4))$$

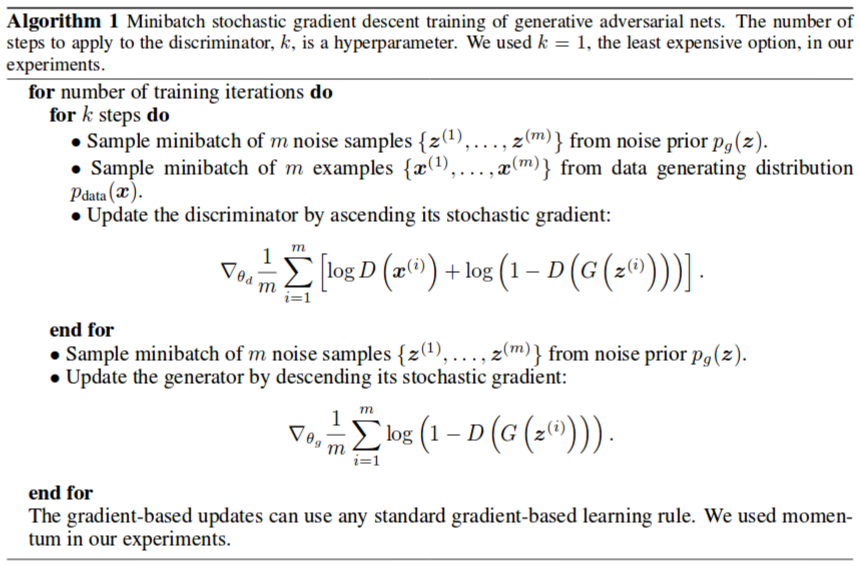
Therefore it is proven that the global minimum value *C(G)* can have is -*log(4)*, and it is achieved with *D*(x) = 1/2.*

So it should be clear now, that at the point of *pg=pdata*, both maximizing player and minimizing player will achieve there goals. In other words, that is our equilibrium point! Which answers 'yes' our first main question. And most importantly, in that equilibrium point, *pg=pdata*, which gives us the 'yes' answer for the second question and the OPTIMUM GENERATOR. HURRAAH!!!

So all done? Seems yes doesn't it? Well a one small glitch remains.

How are we going to reach this optimum point? True, the minimizing, maximizing game will get us there, but it's all just words and numbers. What is the practical algorithm we are going to use to get there.

What the authors have done to answer this question is that coming up with an algorithm proving that this algorithm will get us to the optimal point. Let's have a look at this algorithm.



Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

```
for number of training iterations do
    for $k$ steps do
        • Sample minibatch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
        • Sample minibatch of $m$ examples $\{x^{(1)}, \dots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
        • Update the discriminator by ascending its stochastic gradient:
              $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right].$
    end for
    • Sample minibatch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
    • Update the generator by descending its stochastic gradient:
          $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right).$
end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.
```

Convergence Algorithm

Let's have a quick thought on what does this global optima looks like. Remember that *D* needs to maximize *V(G,D)* for a given *G*, and again *G* will try to minimize *V(G,D)* at the optimal *D*. So what does this give us?

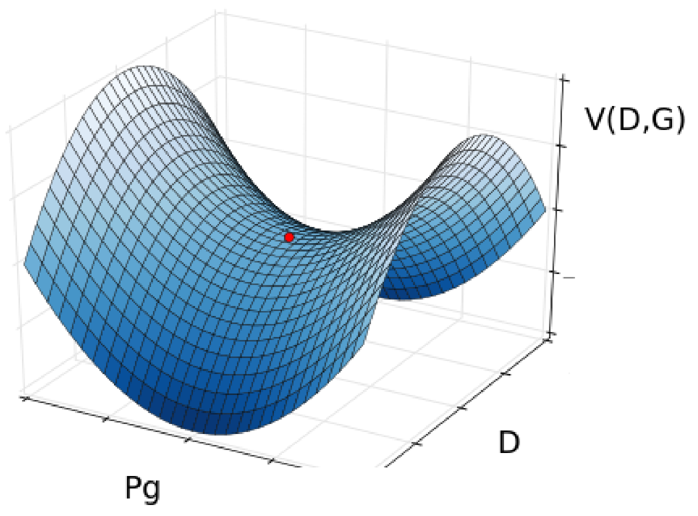


Figure 2

If you guessed it correctly, it will be a saddle point. Something close to above plot. I used *Pg* instead of *G* for easier understanding of the next part of this article. As *Pg* is solely dependent on *G*, we can use *Pg* instead of *G*. Note that this is not a 100% accurate plot, as *D* will not be optimal at the same value for every given *Pg*.

Our goal is to reach the red point in the plot. The paper then says something like this.

> ***"The subderivatives of a supremum of convex functions include the derivative of the function at the point where the maximum is attained.".***

What does this mean in plain English? Let's see.

I will draw another plot *Pg* vs *V(G,D)*, making *D(x) = D*(x)* (optimal *D*) at every point of *Pg*. We will reach this point by applying gradient descent for *D*, for each fixed point of *Pg* as in the inner loop of the Convergence Algorithm (see the algorithm). This simply means, in the above plot, at each *Pg*, I will pick the maximum value of *V(G,D)* by varying *D(x)* and plot *Pg* vs *V(G,D)*. This will give me a plot like below.

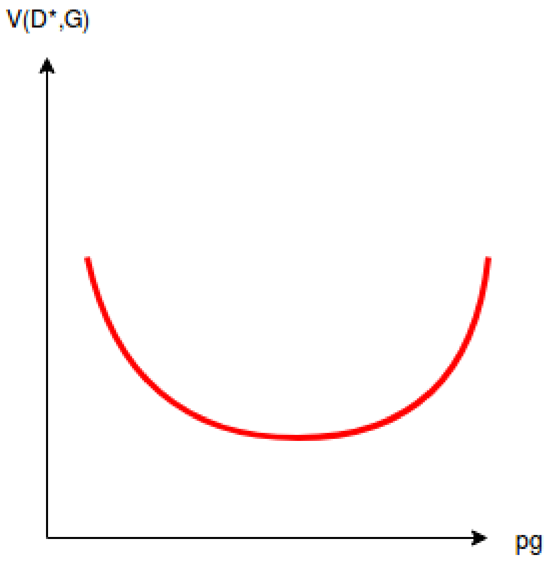


Figure 3

The above statement simply says, if I get the set of derivative of *V(D*,g)* w.r.t. pg, it will include the all the partial derivatives of *V(D,G)* w.r.t. *pg*, at the locations where *V(D,G)* is maximum for a given *Pg* (see the plot in Figure 2). **This means that the derivative at the red dot in Figure 2 is somewhere in the set of derivatives of the function in Figure 3.** And by applying gradient descent on *G(z)* parameters (partial differentiating *V(D*,G)* w.r.t. *pg*), we can update *pg* and reach that red dot. Furthermore by previous proofs, we know at that global optimum, *pdata = pg* and *D(X) = 1/2*, which is exactly what we want.

If this last part is not very clear to you, let's think of it in simple terms. Look again at Figure 2.

> *By applying gradient descent to* D, *for a given* pg, *we get to the optimum* D *for that* pg. *(Inner loop of the convergence algorithm).*

> *Then keeping D(X) fixed, we apply gradient descent to pg, and get closer to the red dot. (Outer loop of the convergence algorithm)*

> *Since partial derivatives of pg at optimum D points include red dot,* ***given enough capacity to D and G****, we will eventually reach the red point by using the convergence algorithm .*

Thats it!! That's the complete theoretical explanation of the Generative Adversarial Networks paper by Goodfellow et al. Before concluding, for the sake of completeness I will quickly go through few things mentioned in the paper. I intentionally kept these at last, as it is easier to understand them after going through the whole explanation. The paper contains the following figure,

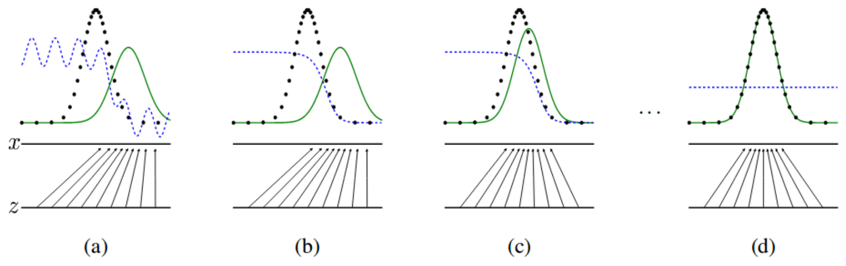


Figure 5

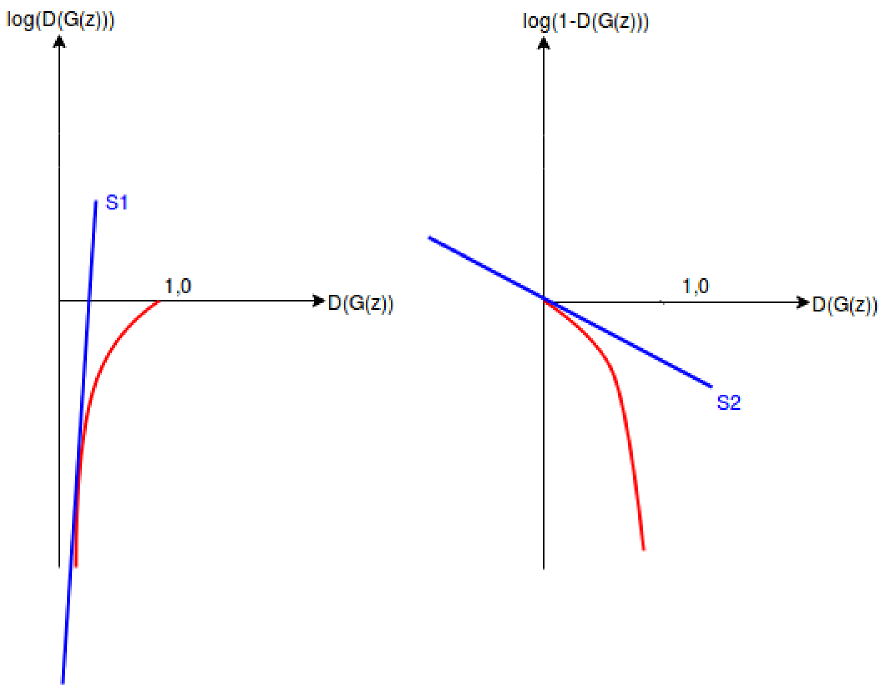
Green line is *pg*, black line is *pdata* and blue line is *D(x)*. Let's go through each of these one-by-one. (a) is the initial situation. *pg* is distributed far from *pdata*, and *D* can roughly distinguish between them (Low value at *pg* and high value at *pdata*). The we update *D* using gradient decent, which will give us (b). Now *D* can distinguish between *pg* and *pdata* better. Then in (c), we update *G*, which will move *pg* closer to *pdata*. After few iterations, *pg=pdata* and *D = 1/2* everywhere as in (d).

Paper then mentions that in practice,

> ***rather than training G to minimize (log(1-D(G(z)))), it's better to train to G to maximize log(D(G(z)))***.

How do we make sense of this theoretically?

Let's have a look at *log(D(G(z)))* and *log(1-D(G(z)))* plots.



In practice, initially G is lousy and produce data which are clearly distinguishable from original data. Therefore initially, *D(G(z)) ~ 0*. Now look at the two plots. S1 and S2 are the gradients of the two plots where *D(G(z))* is close to 0. As $|S1| >> |S2|$, *log(D(G(z)))* will initially have much stronger gradients than *log(1-D(G(z)))*. Remember our goal is to bring *D(G(z))* to *1/2*. While minimizing game tries to push *D(G(z))* towards 1, maximizing game will try to push *D(G(z))* towards zero. If maximizing payer wins initially (which means separately identify generated and original data with a very high accuracy), *D(G(z))* will never reach *1/2*. Therefore for minimizing player, it is important to have stronger gradients initially to keep up with the maximizing player and reach the equilibrium point.

Well…that's all there is to it!! :) Hope you enjoyed the post :D

If there are any questions or suggestions, don't hesitate to leave a comment. Have a good day!!