Generative Adversarial Nets

•••

Hanna Landrus and David Pitman

A Talk in Two Acts

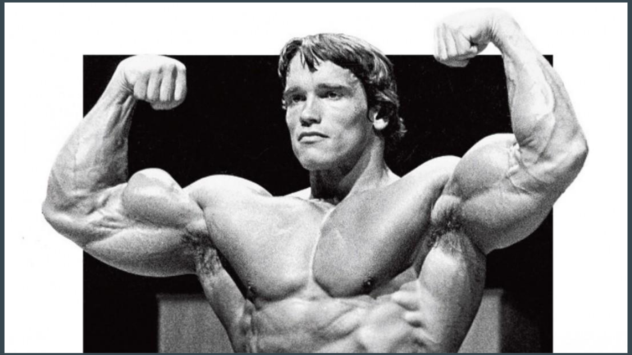
Act I:

Hanna talks about the paper.

Act II:

David talks about advancements since the paper, the challenges of using GANs and his preliminary results.

Why GANs?

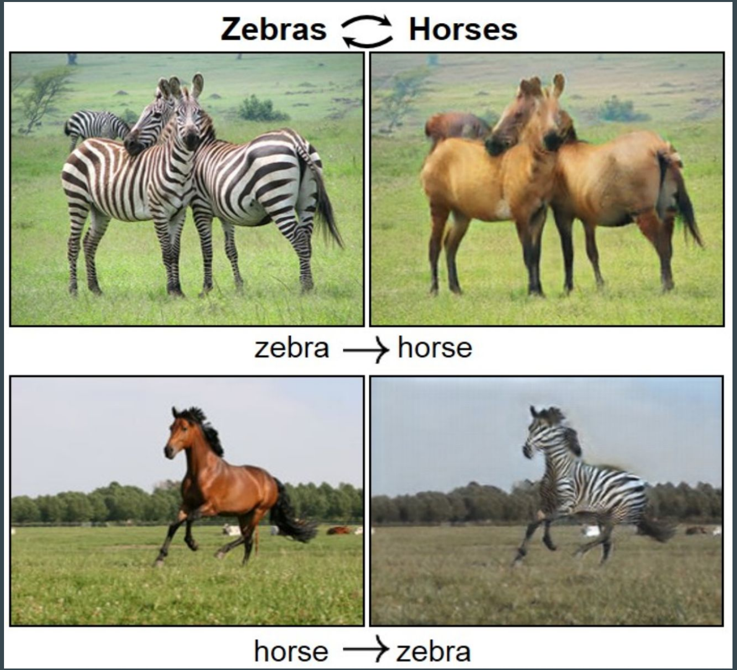


Why not gains?

Or why not gans?

GANs is a technique used to mimic a distribution of data.

https://thispersondoesnotexist.com/

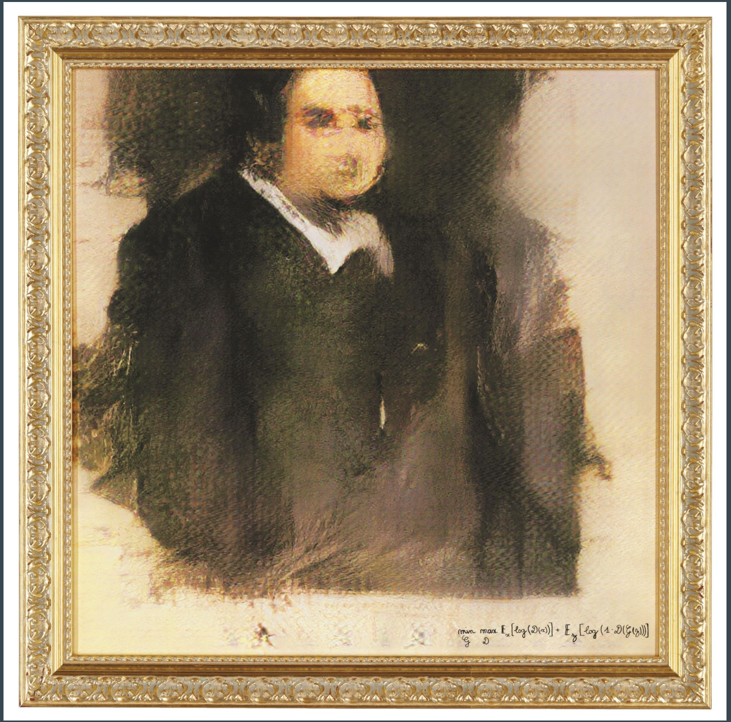


GANs in the News

Modified GANs generated art recently sold for $432k

Code that generated the art is based on an open source project

The creator of the project did not receive any $ or credit.

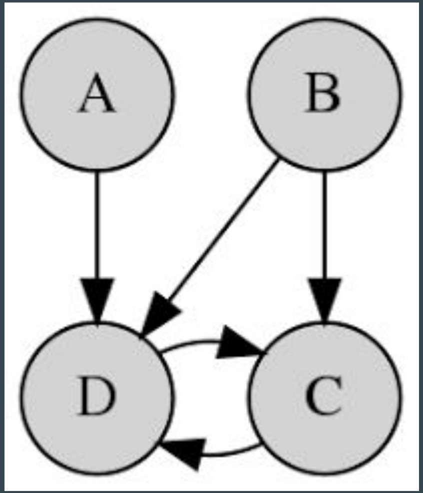


Discussion: Who “owns” art generated from open source code?

Now that I got you talking…
Let's talk about math!

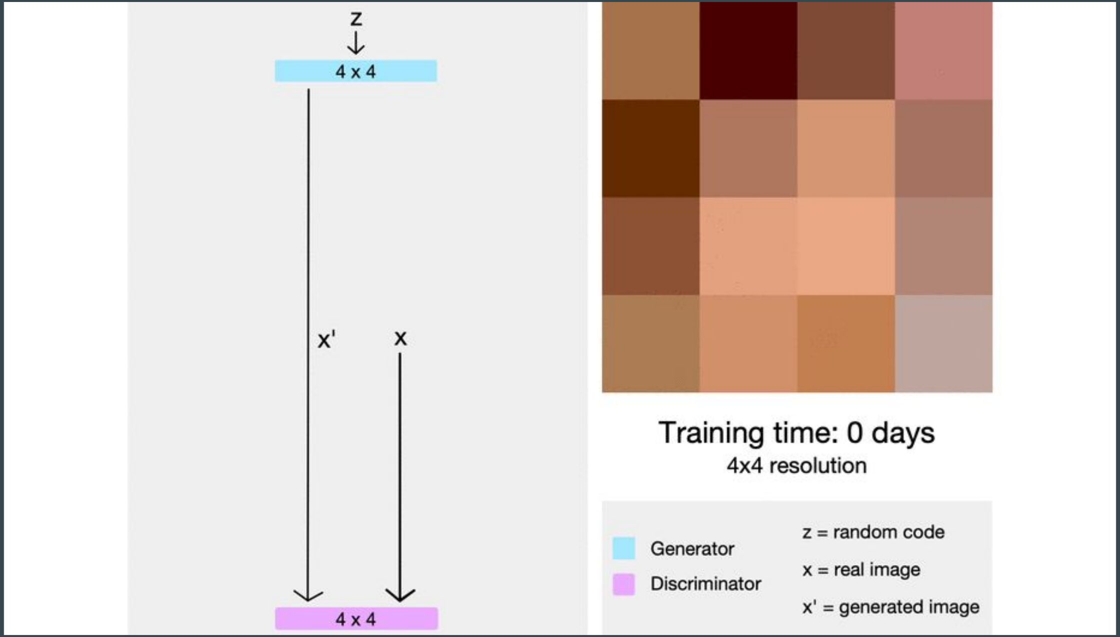
Background: Generative Models

- Previous work focused on heavily on graphical models
- Focus was often on explicitly defining a probability distribution
- Many techniques had computational difficulties
- Often relied on Markov chain Monte Carlo methods for sampling
- Some generative models did not rely on explicitly defining probability distributions like generative stochastic networks (GSN)



Conceptual Formulation

- Two models work against each other.
- The model G generates forgeries of a data set.
- The model D is trained to classify fakes from originals.
- They go back and forth each getting better.

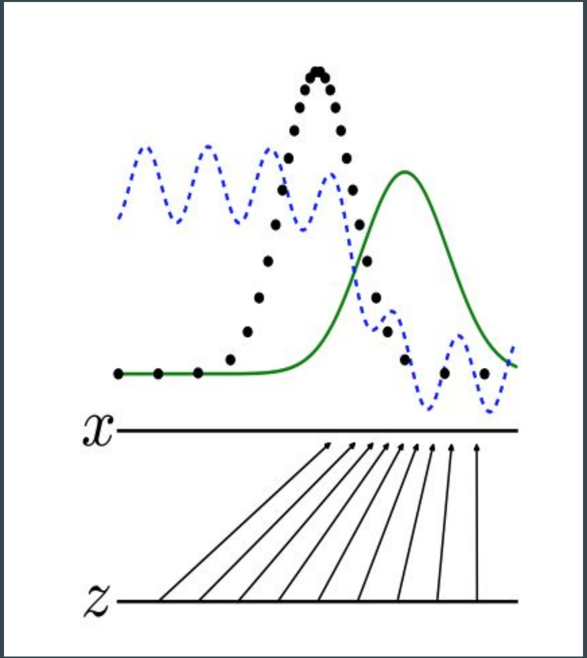


Variable Definition

- x - sample data we are trying to mimic.
- p_{data} - distribution of data
- z - Noisy samples to feed to generative model
- p_z - distribution of z
- $G(z; \theta_g)$ - Generative model that takes samples for z as input and learns parameters θ_g.
- p_g - the distribution generated by G.
- $D(x;\theta_d)$ - Discriminative model that learns parameters θ_d in order to predict if an input is from the data or p_g. Output of model in $[0,1]$.

Mathematical Formulation: Two-player Minimax Game

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



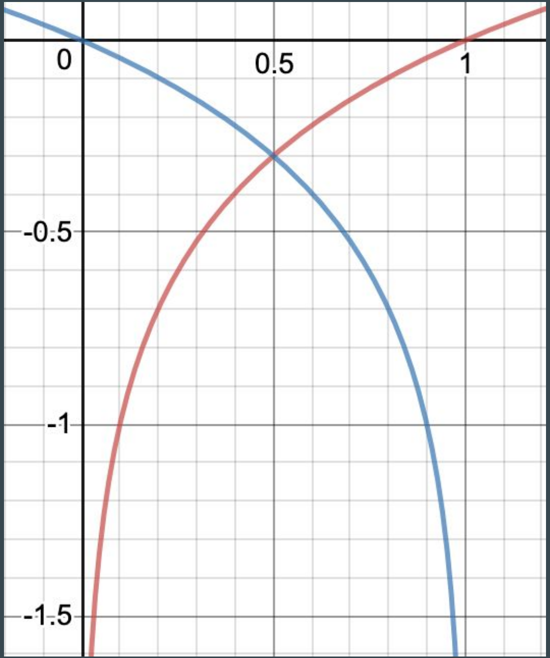
Dotted black line - p_{data}

Dotted blue line - D(x)

Green line - p_g

Blue line - log(1-x)

Red line - log(x)



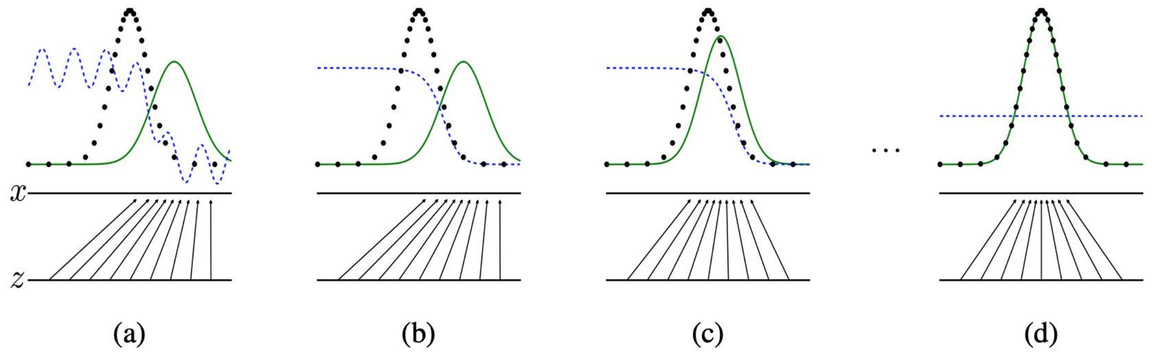
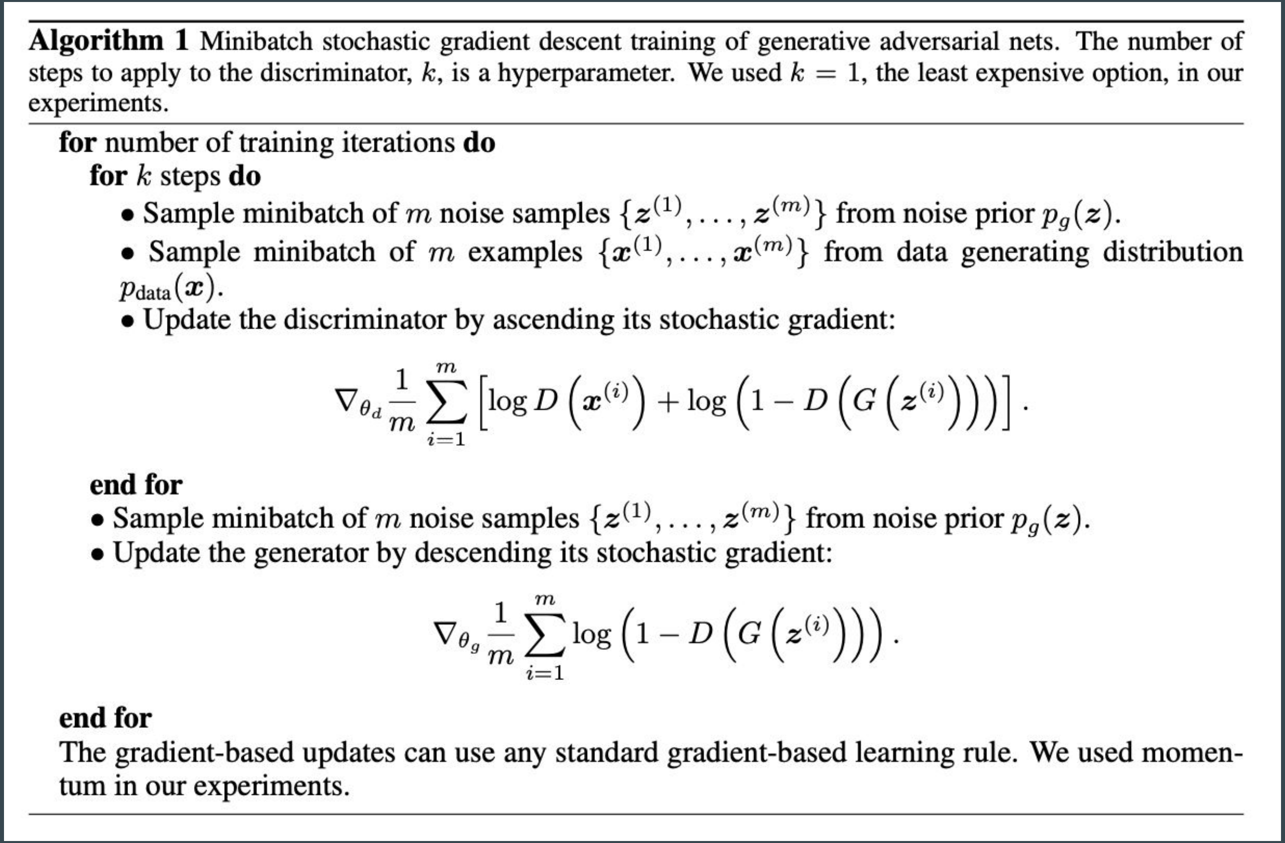


Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which $\boldsymbol{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\boldsymbol{x}$. The upward arrows show how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x})+p_g(\boldsymbol{x})}$. (c) After an update to G, gradient of D has guided $G(\boldsymbol{z})$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(\boldsymbol{x}) = \frac{1}{2}$.

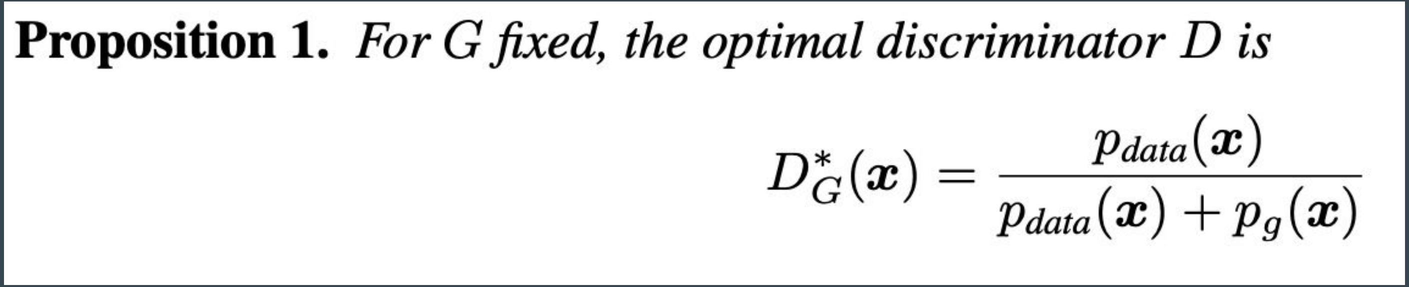
Discuss: If a model can directly mimic an underlying data source what are the implications?

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.

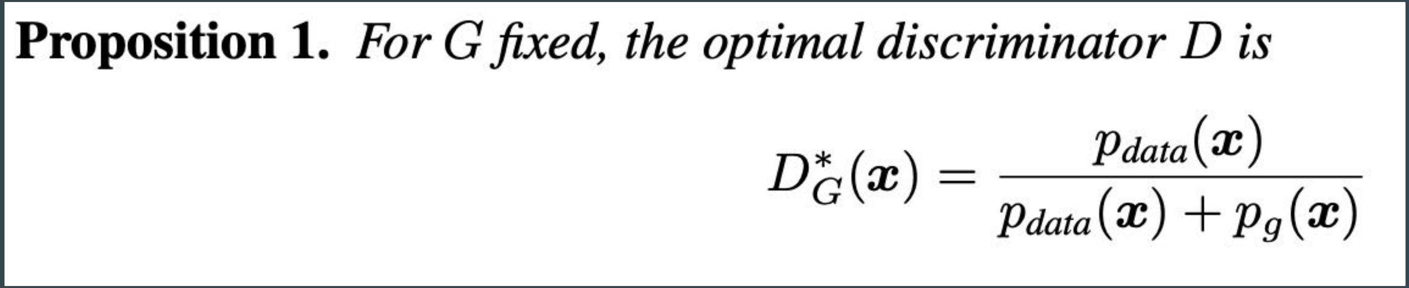
```
for number of training iterations do
    for $k$ steps do
        • Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
        • Sample minibatch of $m$ examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$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


Proposition 1. *For G fixed, the optimal discriminator D is*

$$D^*_G(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$$

The optimal guess for the discriminator at x

is the “weight” of p_{data} at x.



Proposition 1. *For G fixed, the optimal discriminator D is*

$$D^*_G(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$$

$x = 0 \text{ -> } p_g(0) = 0$

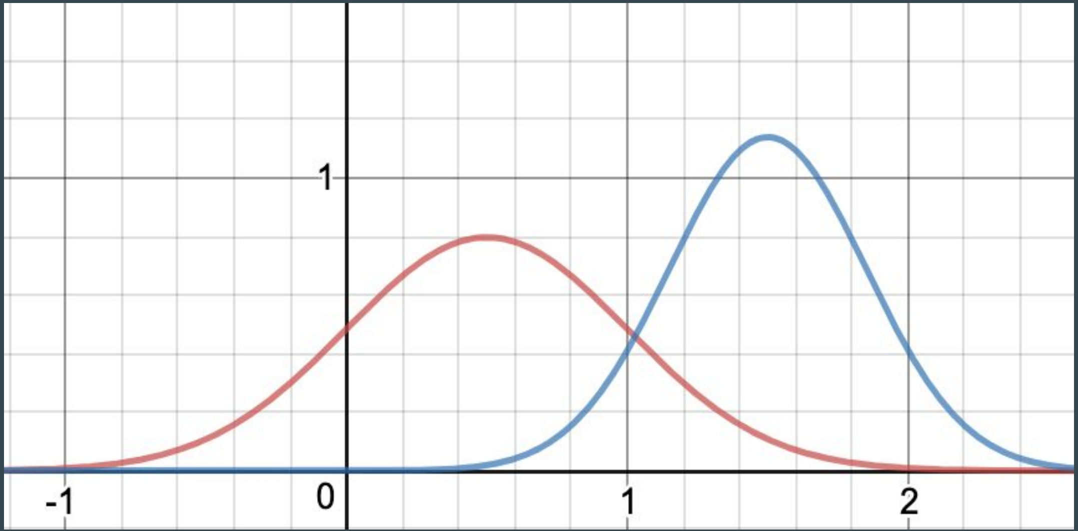
$p_{data}(0)/p_{data}(0) = 1$

$x = 1.1 \text{ -> } p_g(1.1) = p_{data}(1.1)$

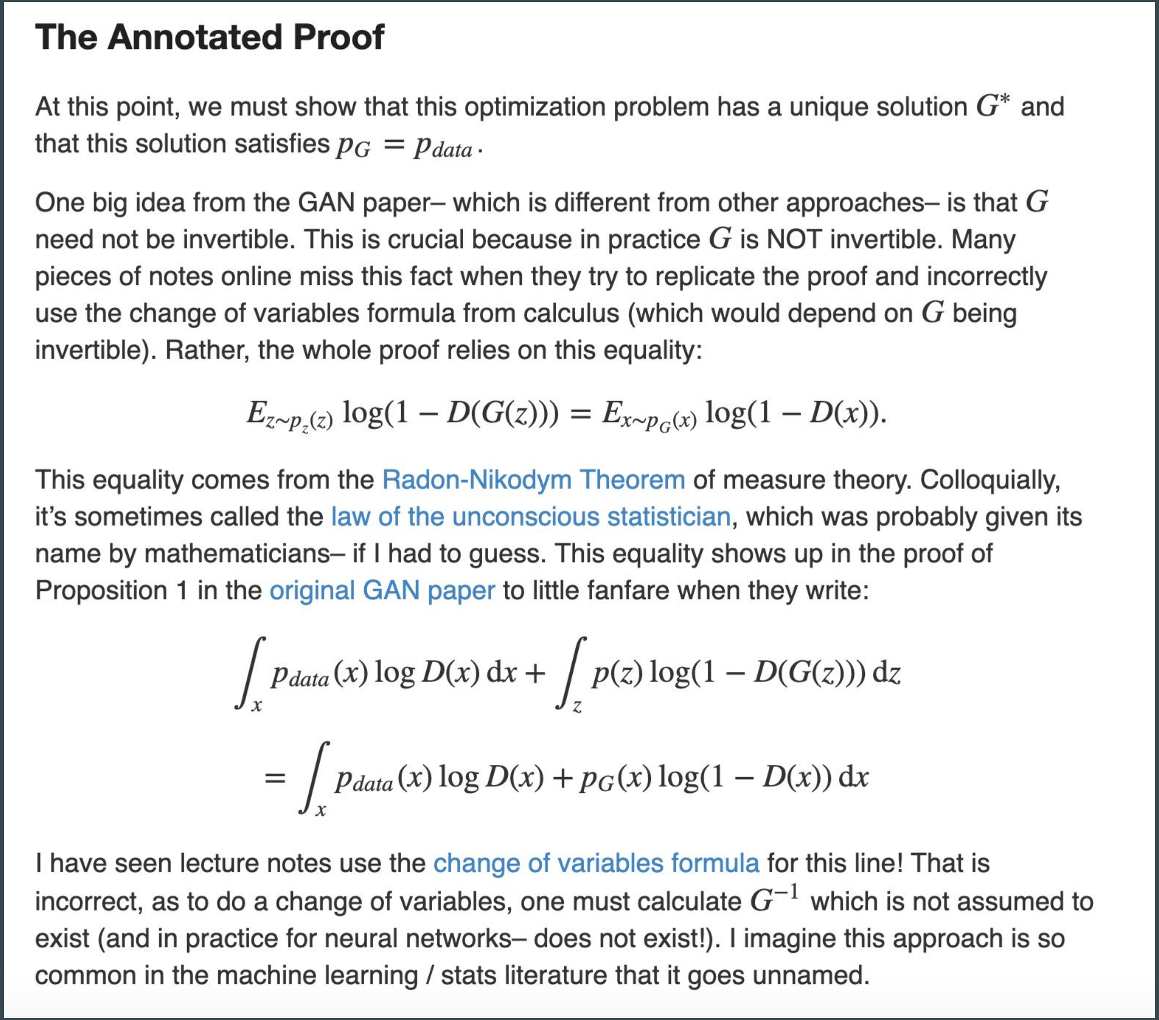
$p_{data}(1.1)/2p_{data}(1.1) = ½$

$x = 2.2 \text{ -> } p_{data}(2.2) = 0$

$0 / p_g(2) = 0$



Red = p_{data} Blue = p_g



The Annotated Proof

At this point, we must show that this optimization problem has a unique solution G^* and that this solution satisfies $p_G = p_{data}$.

One big idea from the GAN paper– which is different from other approaches– is that G need not be invertible. This is crucial because in practice G is NOT invertible. Many pieces of notes online miss this fact when they try to replicate the proof and incorrectly use the change of variables formula from calculus (which would depend on G being invertible). Rather, the whole proof relies on this equality:

$$E_{z \sim p_z(z)} \log(1 - D(G(z))) = E_{x \sim p_G(x)} \log(1 - D(x)).$$

This equality comes from the Radon-Nikodym Theorem of measure theory. Colloquially, it's sometimes called the law of the unconscious statistician, which was probably given its name by mathematicians– if I had to guess. This equality shows up in the proof of Proposition 1 in the original GAN paper to little fanfare when they write:

$$\int_x p_{data}(x) \log D(x) \, \mathrm{d}x + \int_z p(z) \log(1 - D(G(z))) \, \mathrm{d}z$$

$$= \int_x p_{data}(x) \log D(x) + p_G(x) \log(1 - D(x)) \, \mathrm{d}x$$

I have seen lecture notes use the change of variables formula for this line! That is incorrect, as to do a change of variables, one must calculate G^{-1} which is not assumed to exist (and in practice for neural networks– does not exist!). I imagine this approach is so common in the machine learning / stats literature that it goes unnamed.

Scott Rome

https://srome.github.io/An-Annotated-Proof-of-Generative-Adversarial-Networks-with-Implementation-Notes/

Can simplify

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

To solve for the max of

$$f(y) = a \log(y) + b \log(1\text{-}y)$$

$$f'(y) = a/y + b/(1\text{-}y) = 0$$

$$y = a/(a+b)$$

Now the Other Way: Finding the Optimal Solution for G

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

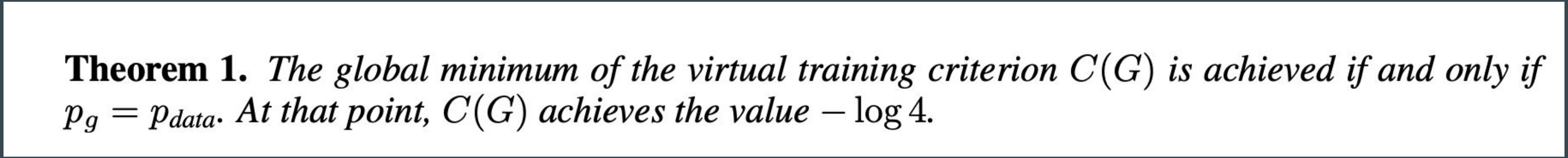
C(G) - Virtual training criterion - Finding the G that will provide the optimal solution

If we know the best strategy for D then the best solution to G would be the solution that best exploits this strategy.

$$D_G^*(\boldsymbol{x}) = \frac{p_{data}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$$

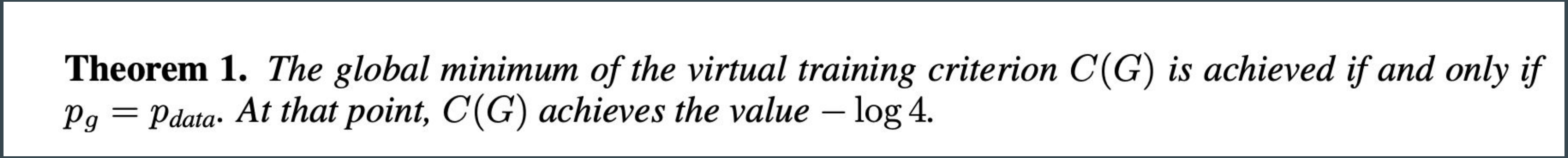
To exploit the above formula it make sense that we want $p_{data} = p_g$ because then best guess is 50/50

Note: Plugging this into C(G) gives you - log 4



Theorem 1. *The global minimum of the virtual training criterion $C(G)$ is achieved if and only if $p_g = p_{data}$. At that point, $C(G)$ achieves the value $-\log 4$.*

But how do we know this is the only minimum?

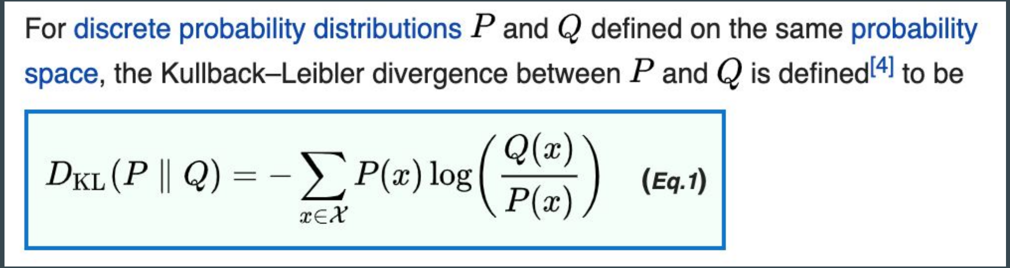


Theorem 1. *The global minimum of the virtual training criterion $C(G)$ is achieved if and only if $p_g = p_{data}$. At that point, $C(G)$ achieves the value $-\log 4$.*

To prove this is the only minimum we just need to do some algebra and calculus tricks and then use two double name divergences.

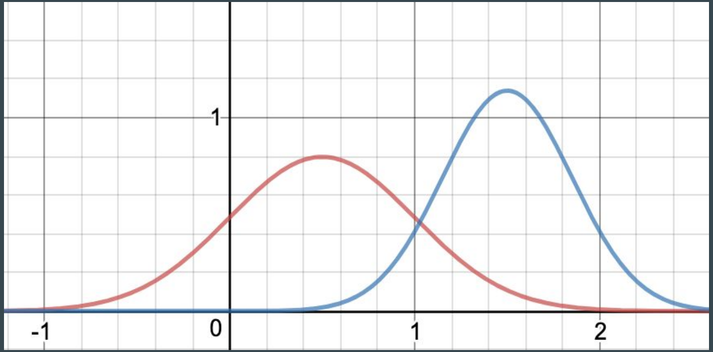
Let’s get a cracking....

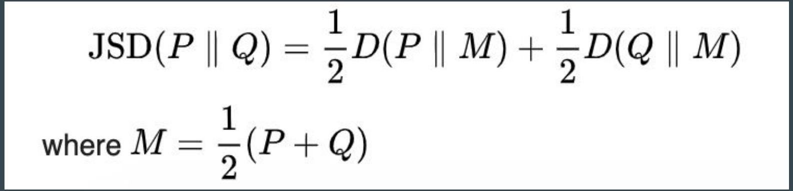
The Kullback–Leibler divergence and Jensen–Shannon divergence provide ways to quantify the similarity between two probability distributions.



For discrete probability distributions P and Q defined on the same probability space, the Kullback–Leibler divergence between P and Q is defined[4] to be

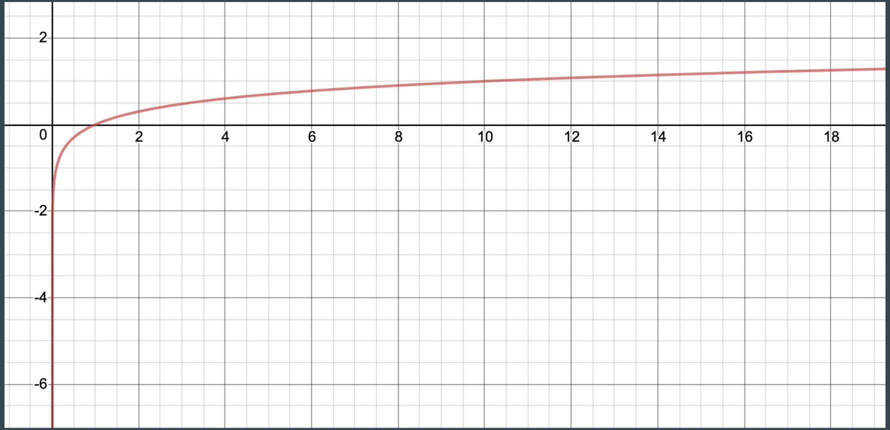
$$D_{\mathrm{KL}}(P \parallel Q) = -\sum_{x\in\mathcal{X}} P(x)\log\left(\frac{Q(x)}{P(x)}\right) \quad \textit{(Eq.1)}$$





$$\mathrm{JSD}(P \parallel Q) = \frac{1}{2}D(P \parallel M) + \frac{1}{2}D(Q \parallel M)$$

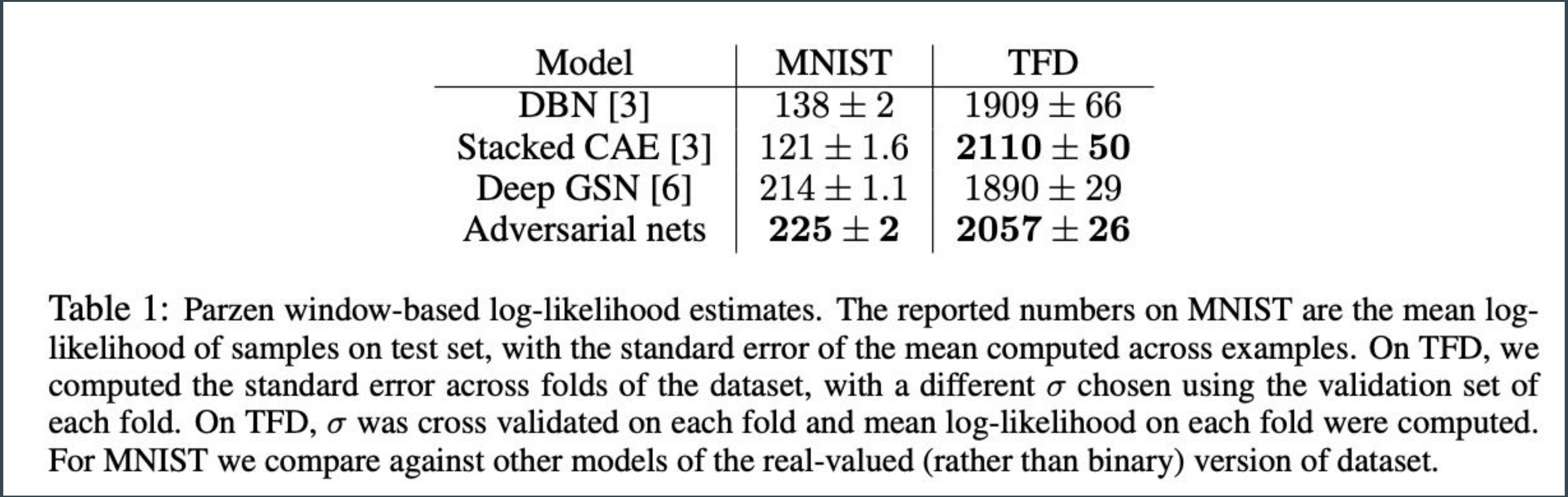
where $M = \frac{1}{2}(P+Q)$



If Jensen–Shannon is zero

Then P = Q

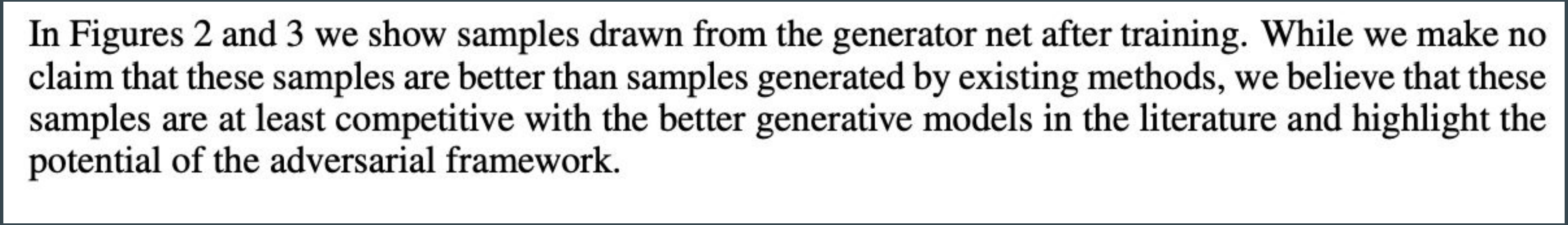
In Practice



Model	MNIST	TFD
DBN [3]	138 ± 2	1909 ± 66
Stacked CAE [3]	121 ± 1.6	$\mathbf{2110 \pm 50}$
Deep GSN [6]	214 ± 1.1	1890 ± 29
Adversarial nets	$\mathbf{225 \pm 2}$	$\mathbf{2057 \pm 26}$

Table 1: Parzen window-based log-likelihood estimates. The reported numbers on MNIST are the mean log-likelihood of samples on test set, with the standard error of the mean computed across examples. On TFD, we computed the standard error across folds of the dataset, with a different σ chosen using the validation set of each fold. On TFD, σ was cross validated on each fold and mean log-likelihood on each fold were computed. For MNIST we compare against other models of the real-valued (rather than binary) version of dataset.

Any opinions on the choice of gaussian parzen windows to estimate the log-likelihood?


In Figures 2 and 3 we show samples drawn from the generator net after training. While we make no claim that these samples are better than samples generated by existing methods, we believe that these samples are at least competitive with the better generative models in the literature and highlight the potential of the adversarial framework.

Discuss: Is there room for more gut evaluations of models in a space like generative models?

References

https://arxiv.org/pdf/1406.2661.pdf

https://github.com/junyanz/CycleGAN

https://github.com/robbiebarrat/art-DCGAN

https://towardsdatascience.com/progan-how-nvidia-generated-images-of-unprecedented-quality-51c98ec2cbd2

https://skymind.ai/wiki/generative-adversarial-network-gan

https://srome.github.io/An-Annotated-Proof-of-Generative-Adversarial-Networks-with-Implementation-Notes/