Generative Adversarial Nets

GANs: don't work with any explicit density function!
we give up on explicitly modeling density, and just want ability to sample.

Now what are explicit and implicit density functions?

Explicit density estimation: explicitly define and solve for pmodel(x)
(i.e. maximize the log likelihood of that explicit distribution either by approximating it as in VAE(we define a lower bound on that) or without approximating whenever it is tractable as in PixelRNN.)

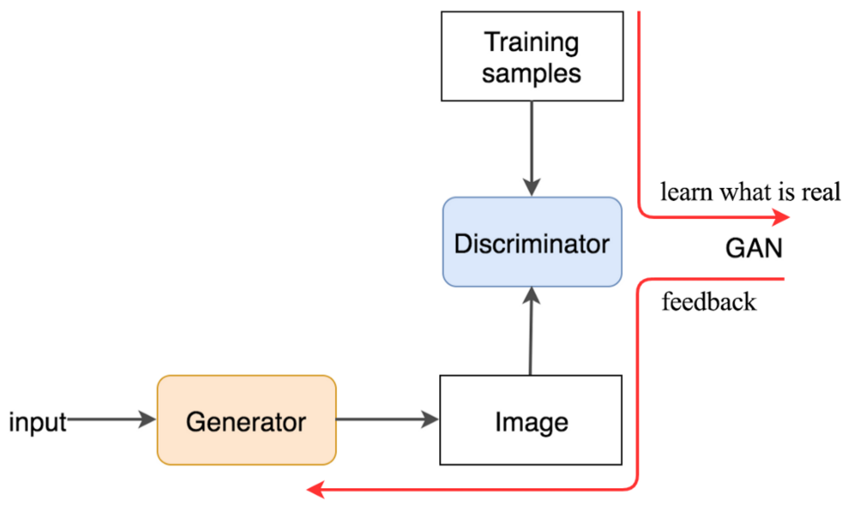
Implicit density estimation: learn model that can sample from pmodel(x) w/o explicitly defining it as in case of GAN

Problem:Want to sample from complex high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution

And we use Neural Networks for that!!
INPUT(random noise) => Generator Network => output

Training GANs: Two-player game

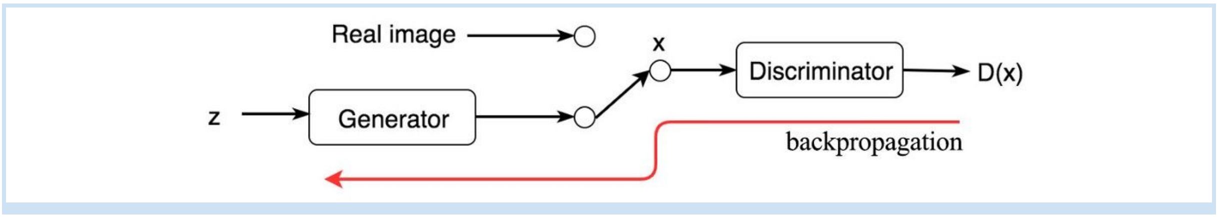


So how is it done technically? The discriminator looks at real images (training samples) and generated images separately. It distinguishes whether the input image to the discriminator is real or generated. The output *D(X)* is the probability that the input *x* is real, i.e. *P(class of input = real image)*.

So basically
Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images

We train the discriminator just like a deep network classifier. If the input is real, we want *D(x)=1*. If it is generated, it should be zero. Through this process, the discriminator identifies features that contribute to real images

On the other hand, we want the generator to create images with *D(x) = 1* (matching the real image). So we can train the generator by backpropagation this target value all the way back to the generator, i.e. we train the generator to create images that towards what the discriminator thinks it is real.



Objective function

Alternate between:

1. Gradient ascent on discriminator

$$\max_D V(D) = \underbrace{\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})]}_{\text{recognize real images better}} + \underbrace{\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]}_{\text{recognize generated images better}}$$

Discriminator wants to maximize objective such that D(x) is close to 1 (real) and D(G(z)) is close to 0 (fake)

2. Gradient descent on generator

$$\min_G V(G) = \underline{\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]}$$

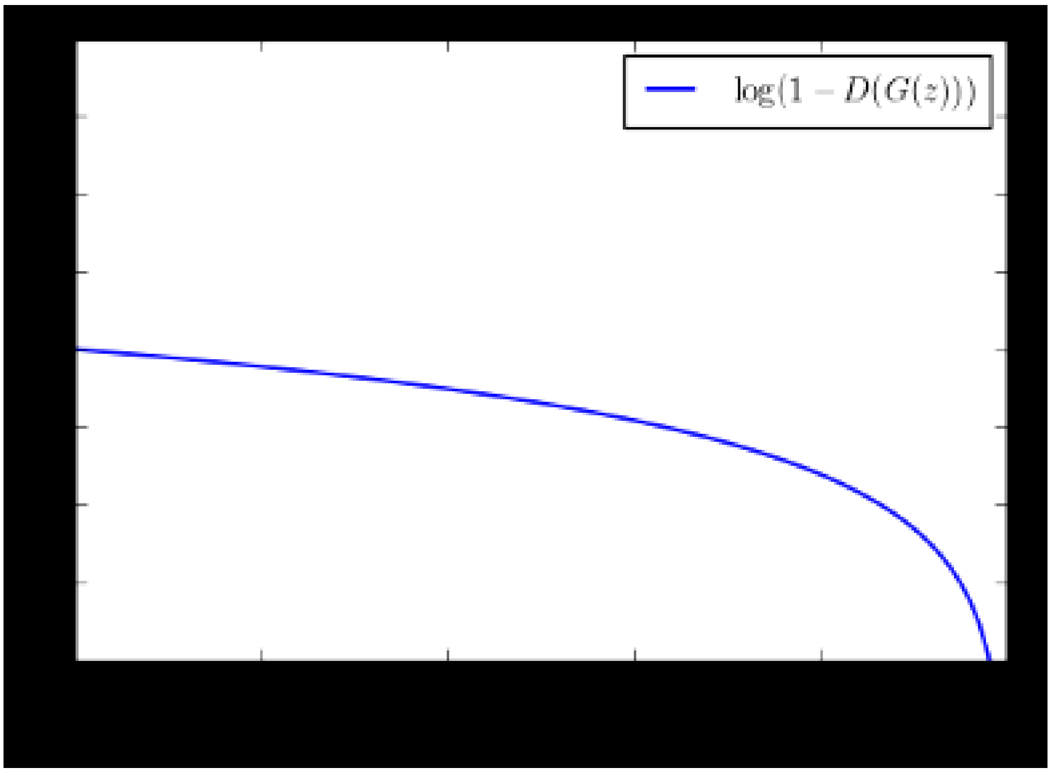
Optimize G that can fool the discriminator the most.

Generator wants to minimize objective such that D(G(z)) is close to 1(discriminator is fooled into thinking generated G(z) is real)

Generator diminished gradient

The discriminator usually wins early against the generator. It is always easier to distinguish the generated images from real images in early training

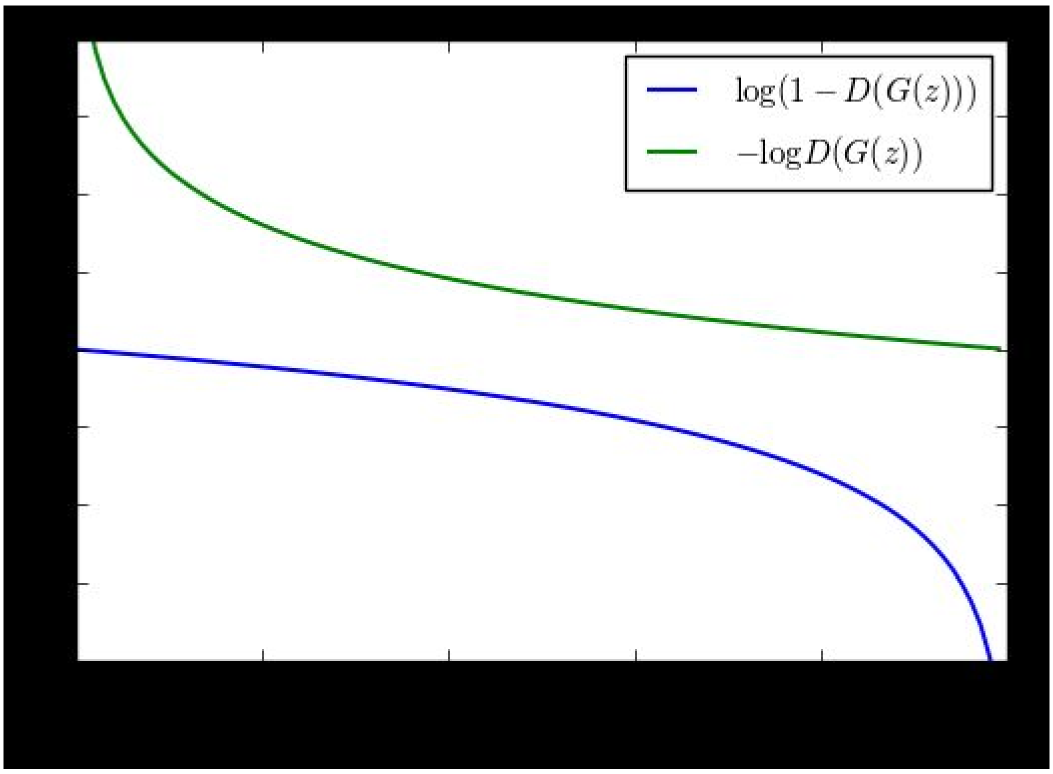
That makes V approaches 0. i.e. - log(1 -D(G(z))) → 0. The gradient for the generator will also vanish which makes the gradient descent optimization very slow.



Plot of log(1-D(G(z))) vs G(z)

When sample is likely fake, model wants to learn from it to improve generator. But gradient in this region is relatively flat!
Gradient signal dominated by region where sample is already good

Hence ,In practice optimizing this generator objective does not work well!



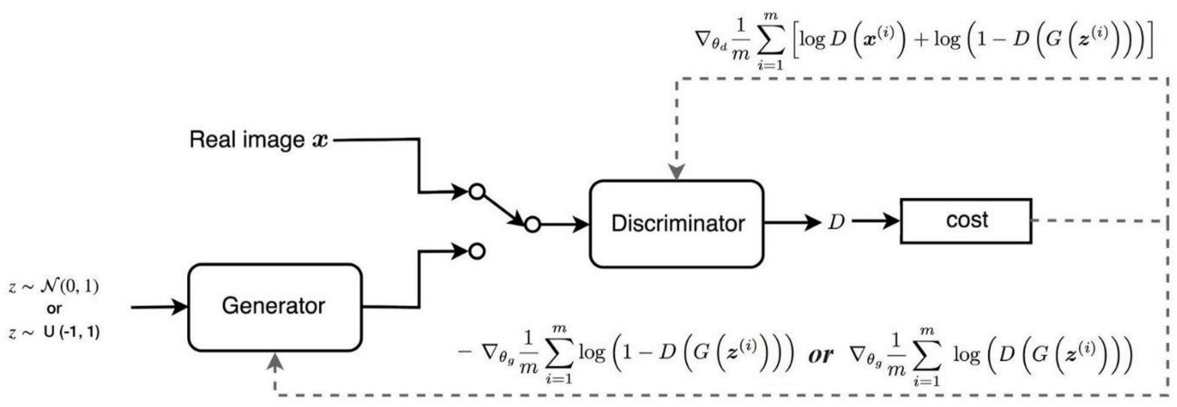
Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong. Same objective of fooling discriminator, but now higher gradient signal for bad samples => works much better! Standard in practice.

To improve that, the GAN provides an alternative function to backpropagate the gradient to the generator.

$$-\nabla_{\theta_g} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \to 0 \quad \textit{change to} \quad \nabla_{\theta_g} \log\left(D\left(G\left(z^{(i)}\right)\right)\right)$$

Once both objective functions are defined, they are learned jointly by the alternating
We fix the generator model's parameters and perform a single iteration of gradient descent on the discriminator using the real and the generated images

Then we switch sides. Fix the discriminator and train the generator for another single iteration. We train both networks in alternating steps until the generator produces good quality images.
The following summarizes the data flow and the gradients used for the backpropagation.

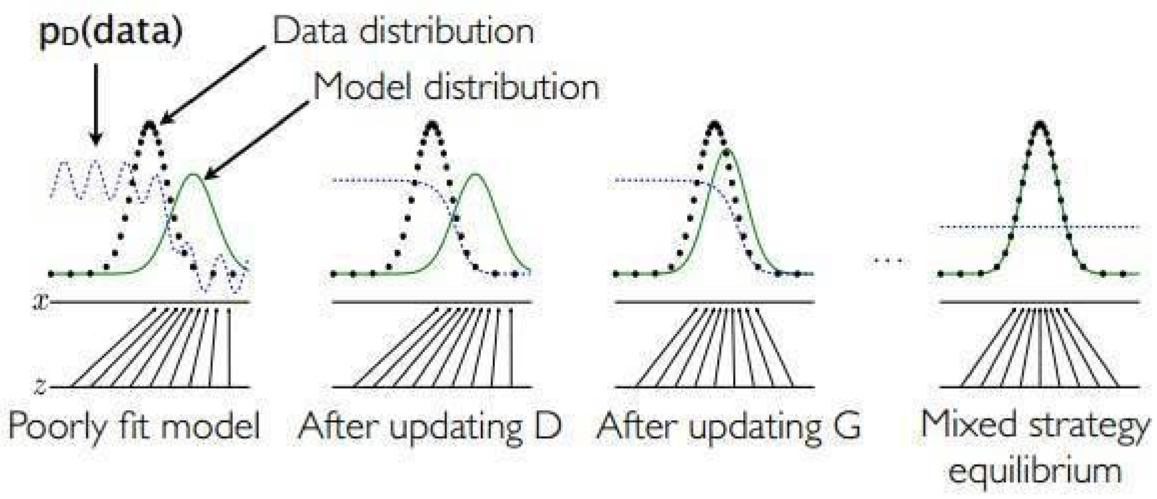


$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right]$
Real image x
Discriminator
D
cost
$z \sim \mathcal{N}(0, 1)$
or
$z \sim$ U (-1, 1)
Generator
$-\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)$ or $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(D\left(G\left(z^{(i)}\right)\right)\right)$

Here is the pseudo code that puts all things together and shows how GAN is trained.

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 $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
    • Sample minibatch of $m$ examples $\{x^{(1)}, \ldots, x^{(m)}\}$ 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 $\{z^{(1)}, \ldots, z^{(m)}\}$ 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.
```



1.The lower horizontal line is the domain from which z is sampled, in this case uniformly
2.The horizontal line above is part of the domain of x.
3.The upward arrows show how the mapping x = G(z) imposes the non-uniform distribution pg ontransformed sample.

Let’s jump into some proofs

Proof (GAN optimal point)

Optimizing GAN is optimizing JS-divergence(Jensen–Shannon divergence). This is obviously not clear from the cost function

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

So modify the loss function

$$\begin{aligned}\min_G \max_D V(D, G) &= \mathbb{E}_{x\sim p_r(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \\ &= \mathbb{E}_{x\sim p_r(x)}[\log D(x)] + \mathbb{E}_{x\sim p_g(x)}[\log(1 - D(x)] \\ &= \int_x \Big(p_r(x)\log D(x) + p_g(x)\log(1 - D(x))\Big)dx\end{aligned}$$

If ***G*** is fixed, the optimal Discriminator ***D**** is

(proof in next section)

if $y = a\log(y) + b\log(1 - y)$, the optimal y is

$$\Longrightarrow y^* = \frac{a}{a+b}$$

Optimize $D(x) = p_r(x)\log D(x) + p_g(x)\log(1 - D(x))$, we get

$$\Longrightarrow D^*(x) = \frac{p_r(x)}{p_r(x) + p_g(x)}$$

Find the optimal value for ***V***:

$$\begin{aligned}\min_G V(D^*, G) &= \int_x \Big(p_r(x)\log D^*(x) + p_g(x)\log(1 - D^*(x))\Big)dx \\ &= \int_x \Big(p_r(x)\log\frac{p_r(x)}{p_r(x)+p_g(x)} + p_g(x)\log\frac{p_g(x)}{p_r(x)+p_g(x)}\Big)dx\end{aligned}$$

$$\begin{aligned}D_{JS}(p_r\|p_g) &= \frac{1}{2}D_{KL}(p_r||\frac{p_r+p_g}{2}) + \frac{1}{2}D_{KL}(p_g||\frac{p_r+p_g}{2}) \\ &= \frac{1}{2}\Big(\int_x p_r(x)\log\frac{2p_r(x)}{p_r(x)+p_g(x)}dx\Big) + \frac{1}{2}\Big(\int_x p_g(x)\log\frac{2p_g(x)}{p_r(x)+p_g(x)}dx\Big) \\ &= \frac{1}{2}\Big(\log 2 + \int_x p_r(x)\log\frac{p_r(x)}{p_r(x)+p_g(x)}dx\Big) + \\ &\quad \frac{1}{2}\Big(\log 2 + \int_x p_g(x)\log\frac{p_g(x)}{p_r(x)+p_g(x)}dx\Big) \\ &= \frac{1}{2}\Big(\log 4 + \min_G V(D^*, G)\Big)\end{aligned}$$

i.e.

$$\min_G V(D^*, G) = 2D_{JS}(p_r\|p_g) - 2\log 2$$

Since the Jensen–Shannon divergence between two distributions is always non-negative and zero only when they are equal,
Hence the only solution is pg = pdata(i.e. pr).......

With $p = q$, the optimal value for D and V is

$$D^*(x) = \frac{p}{p+q} = \frac{1}{2}$$

$$\begin{aligned}\min_G \max_D V(D, G) &= \mathbb{E}_{x\sim p_r(x)}[\log\frac{1}{2}] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - \frac{1}{2})] \\ &= -2\log 2\end{aligned}$$

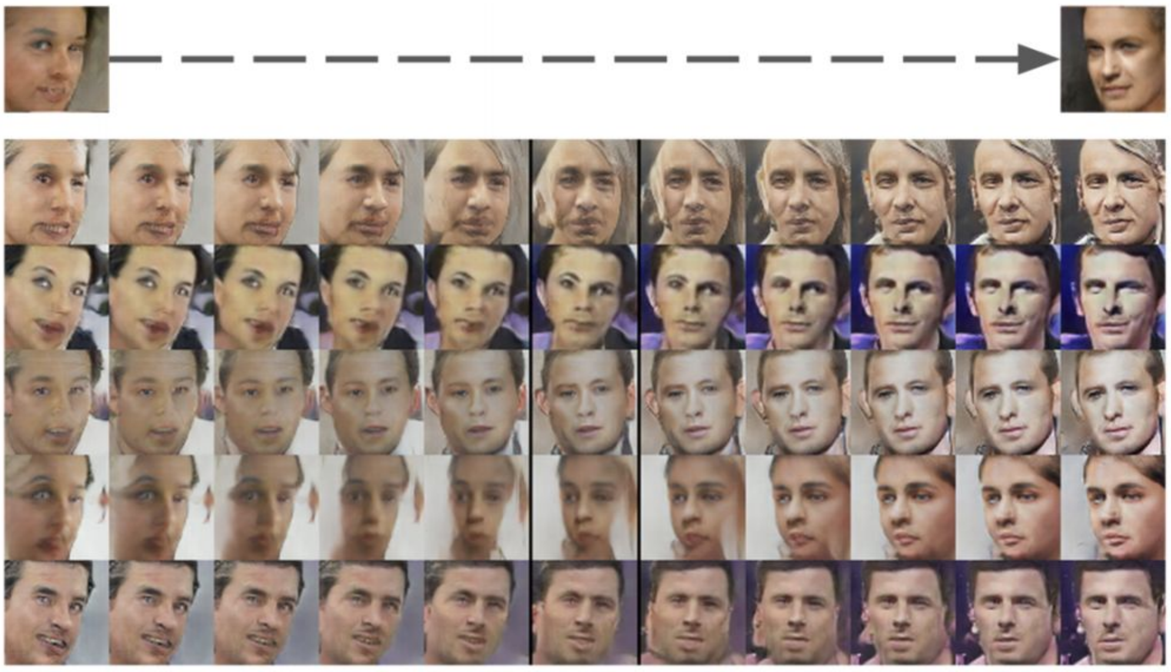
Exploring the latent space

The generator model in the GAN architecture takes a point from the latent space as input and generates a new image.

Through training, the generator learns to map points into the latent space with specific output images and this mapping will be different each time the model is trained.
Typically, new images are generated using random points in the latent space. Taken a step further, points in the latent space can be constructed (e.g. all 0s, all 0.5s, or all 1s) and used as input or a query to generate a specific image.

A series of points can be created on a linear path between two points in the latent space, such as two generated images. These points can be used to generate a series of images that show a transition between the two generated images.

The transition between two generated faces, specifically by creating a linear path through the latent dimension between the points that generated two faces and then generating all of the faces for the points along the path.



Exploring the structure of the latent space for a GAN model is both interesting for the problem domain and helps to develop an intuition for what has been learned by the generator model

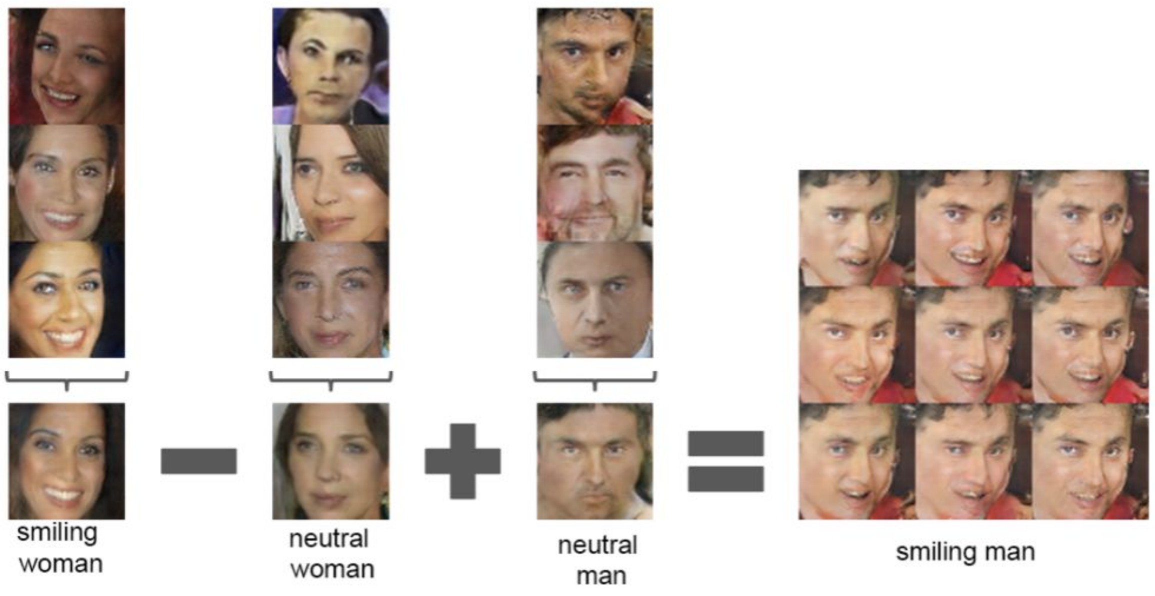
Finally, the points in the latent space can be kept and used in simple vector arithmetic to create new points in the latent space that, in turn, can be used to generate images. This is an interesting idea, as it allows for the intuitive and targeted generation of images.

The vector arithmetic with faces

For example, a face of a smiling woman minus the face of a neutral woman plus the face of a neutral man resulted in the face of a smiling man.

```
smiling woman - neutral woman + neutral
man = smiling man
```

Specifically, the arithmetic was performed on the points in the latent space for the resulting faces. Actually on the average of multiple faces with a given characteristic, to provide a more robust result.



DISADVANTAGES:

1.
There is no explicit representation of pg(x)

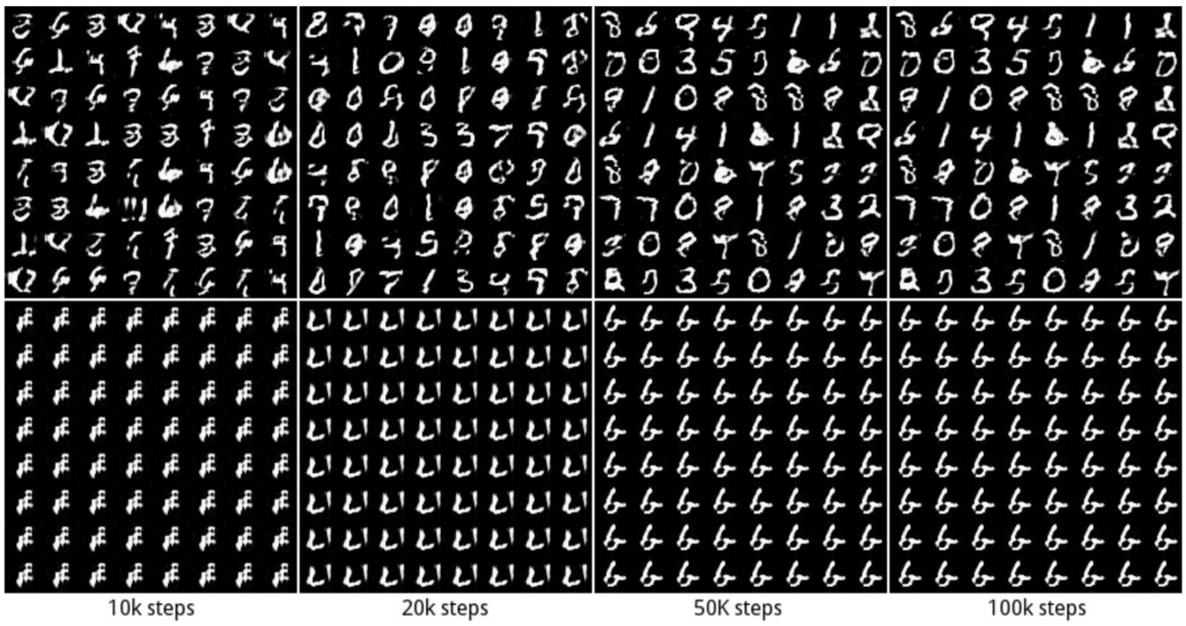
2.
 D must be synchronized well with G during training (in particular, G must not be trained too much without updatingD, in order to avoid “the Helvetica scenario” in which G collapses too many values of z to the same value of x to have enough diversity to model pdata),

This is also known as MODE COLLAPSE   i.e. the generator collapses which produces limited varieties of samples,

Mode

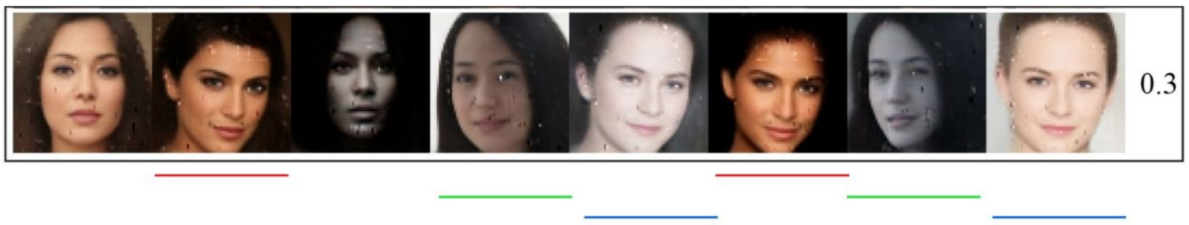
Real-life data distributions are multimodal.

For example, in MNIST, there are 10 major **modes** from digit ‘0’ to digit ‘9’. The samples below are generated by two different GANs. The top row produces all 10 modes while the second row creates a single mode only (the digit “6”). This problem is called **mode collapse** when only a few modes of data are generated.



Why mode collapse in GAN?

Mode collapse is one of the hardest problems to solve in GAN. A complete collapse is not common but a partial collapse happens often. The images below with the same underlined color look similar and the mode starts collapsing.



Let's consider one extreme case where ***G*** is trained extensively without updates to ***D***. The generated images will converge to find the optimal image ***x**** that fool ***D*** the most, the most realistic image from the discriminator perspective. In this extreme, ***x**** will be independent of ***z***.

$$x^* = argmax_x D(x)$$

This is bad news. The mode collapses to a **single point**. The gradient associated with ***z*** approaches zero.

$$\frac{\partial J}{\partial z} \approx 0$$

When we restart the training in the discriminator, the most effective way to detect generated images is to detect this single mode. Since the generator desensitizes the impact of ***z*** already, the gradient from the discriminator will likely push the single point around for the next most vulnerable mode
Now, both networks are overfitted to exploit short-term opponent weakness. This turns into a cat-and-mouse game and the model will not converge.

During training, the discriminator is constantly updated to detect adversaries. Therefore, the generator is less likely to be overfitted. In practice, our understanding of mode collapse is still limited. Our intuitive explanation above is likely oversimplified....

3.
Highly sensitive to the hyperparameter selections.
Hyperparameter tuning needs patience. No cost functions will work without spending time on the hyperparameter tuning.

4.
Cost v.s. image quality
In a discriminative model, the loss measures the accuracy of the prediction and we use it to monitor the progress of the training. However, the loss in GAN measures how well we are doing compared with our opponent. Often, the generator cost

increases but the image quality is actually improving. We fall back to examine the generated images manually to verify the progress. This makes model comparison harder which leads to difficulties in picking the best model in a single run. It also complicates the tuning process.

ADVANTAGES:
advantages are primarily computational

1.
Adversarial models may also gain some statistical advantage from the generator network not being updated directly with data examples,
but only with gradients flowing through the discriminator. This means that components of the input are not copied directly into the generator's parameters.

2.
Another advantage of adversarial networks
is that they can represent very sharp, even degenerate distributions, while methods based on
Markov chains require that the distribution be somewhat blurry in order for the chains to be able to mix between modes.

Tips for training GANS:GANHACKS