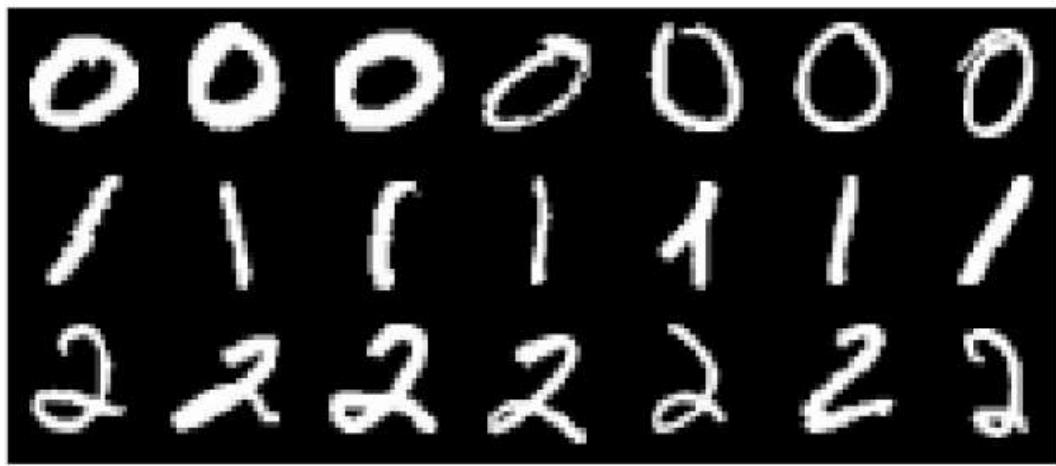


- **We've only seen discriminative models so far**
  - Given an image  $X$ , predict a label  $Y$
  - Estimates  $P(Y|X)$
- **Discriminative models have several key limitations**
  - Can't model  $P(X)$ , i.e. the probability of seeing a certain image
  - Thus, can't sample from  $P(X)$ , i.e. **can't generate new images**
- **Generative models (in general) cope with all of above**
  - Can model  $P(X)$
  - Can generate new images

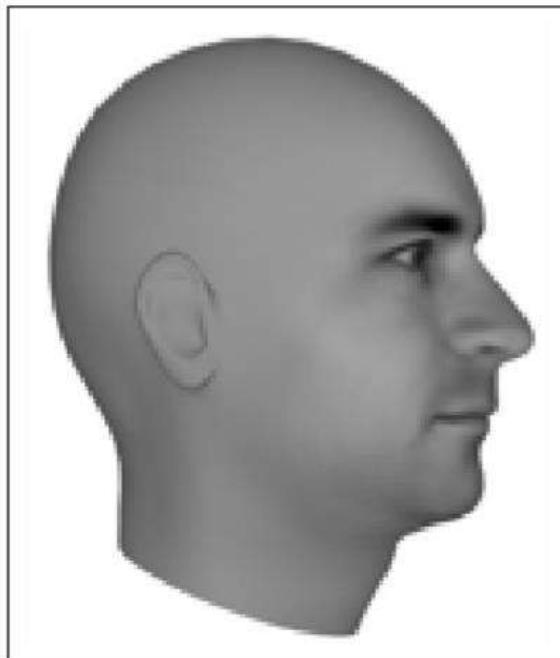
## Implicit density estimation problem

- What if we are only interested to sample from the complex high dimensional intractable data distribution without caring to know  $P(\text{data})$  at all?

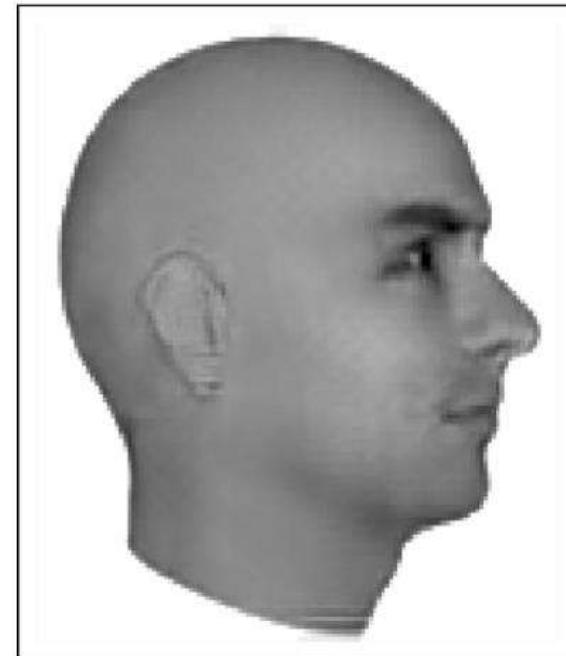


# Magic of GAN

Ground Truth



Adversarial



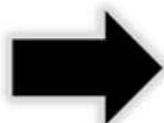
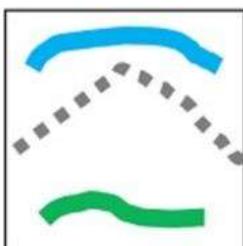
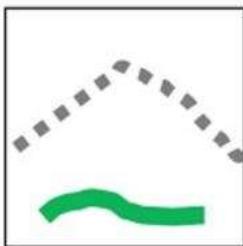
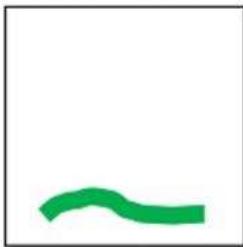
Lotter, William, Gabriel Kreiman, and David Cox. "Unsupervised learning of visual structure using predictive generative networks." *arXiv preprint arXiv:1511.06380* (2015).

# Magic of GAN

Which one is Computer generated?



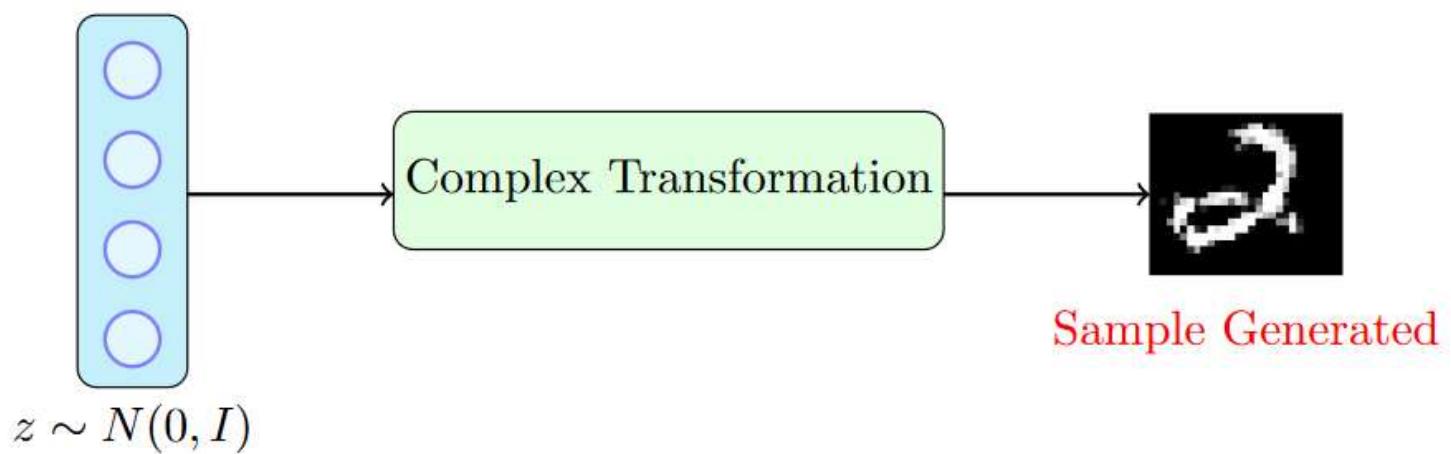
User edits

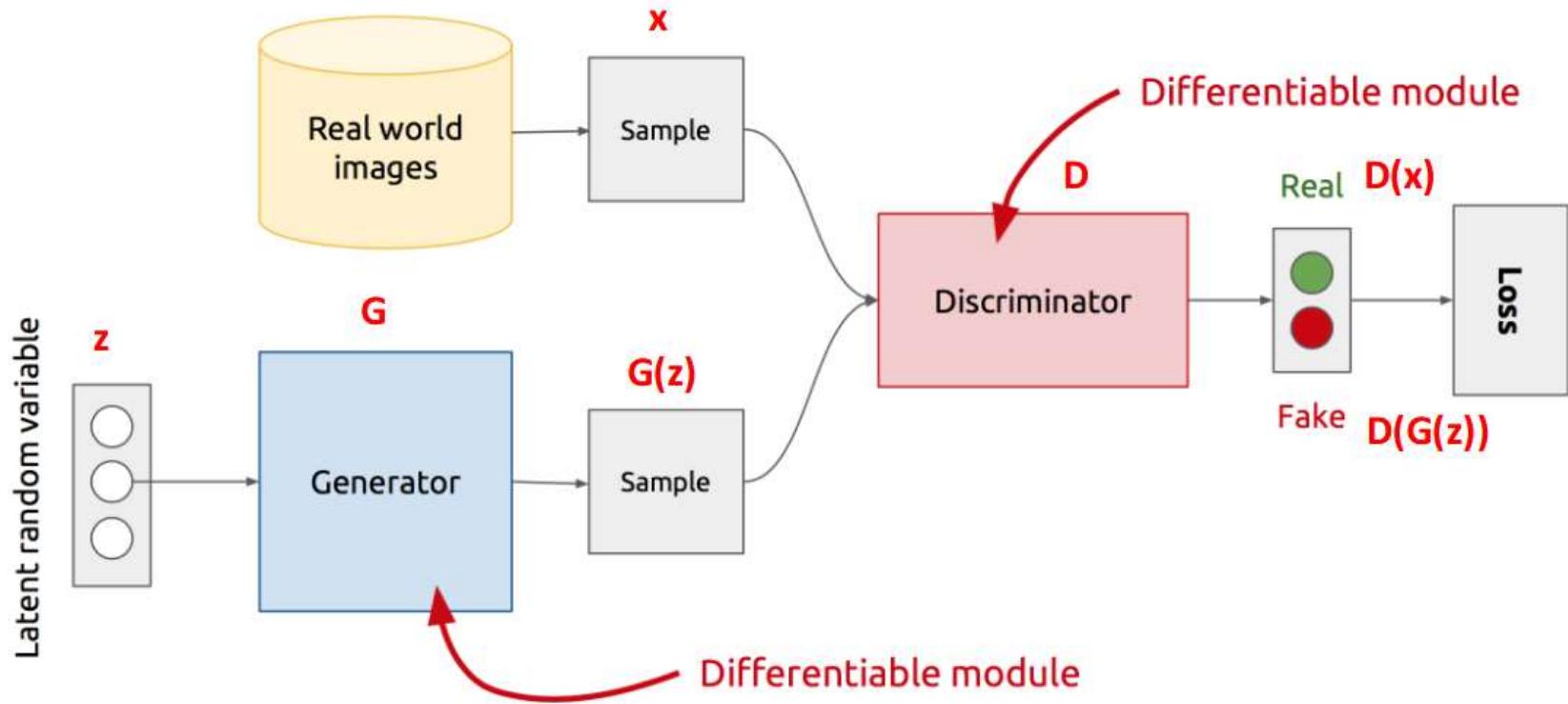


Generated images



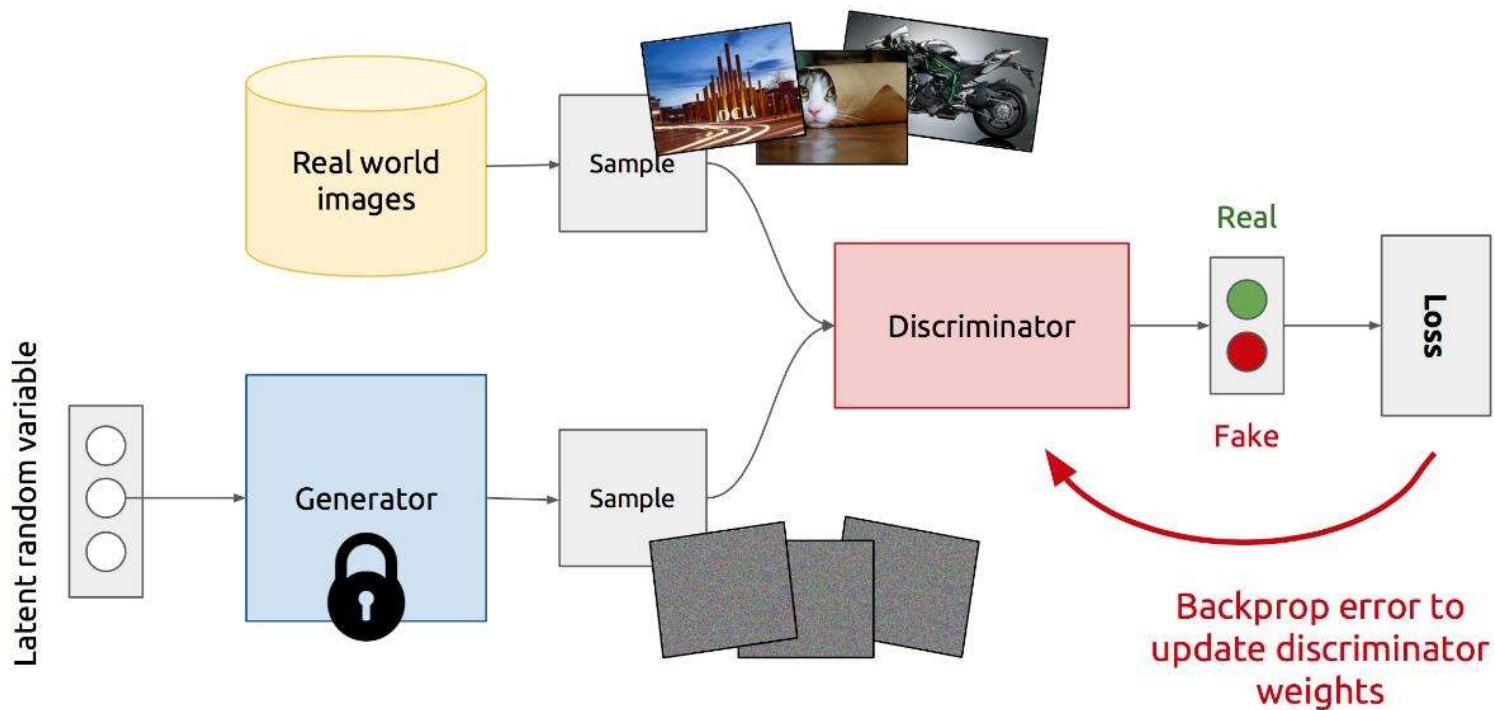
What about



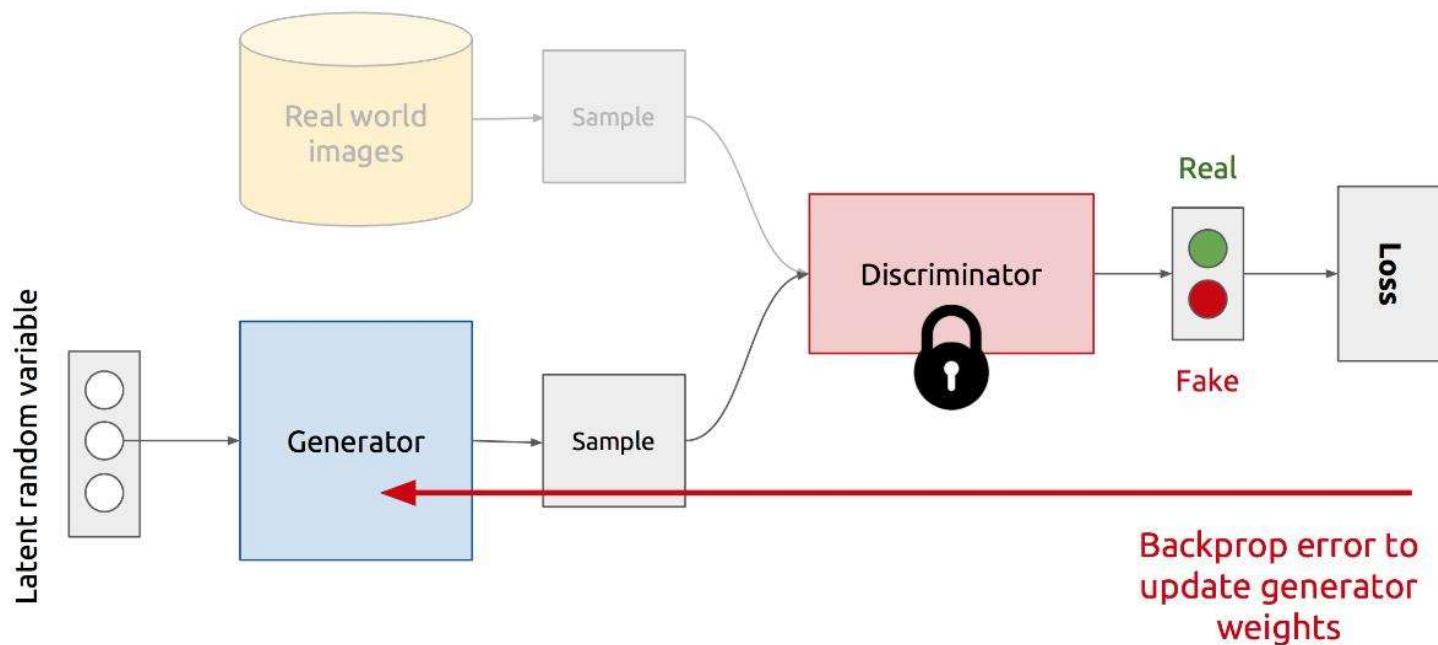


- $\mathbf{Z}$  is some random noise (Gaussian/Uniform).
- $\mathbf{Z}$  can be thought as the latent representation of the image.

# Training discriminator



# Training generator



Let  $G_\phi$  be the generator and  $D_\theta$  be the discriminator ( $\phi$  and  $\theta$  are the parameters of  $G$  and  $D$ , respectively)

We have a neural network based generator which takes as input a noise vector  $z \sim N(0, I)$  and produces  $G_\phi(z) = X$

We have a neural network based discriminator which could take as input a real  $X$  or a generated  $X = G_\phi(z)$  and classify the input as real/fake

Given an image generated by the generator as  $G_\phi(z)$  the discriminator assigns a score  $D_\theta(G_\phi(z))$  to it

This score will be between 0 and 1 and will tell us the probability of the image being real or fake

For a given  $z$ , the generator would want to maximize  $\log D_\theta(G_\phi(z))$  (log likelihood) or minimize  $\log(1 - D_\theta(G_\phi(z)))$

The task of the discriminator is to assign a high score to real images and a low score to fake images  
And it should do this for all possible real images and all possible fake images

# Loss functions

- For generator

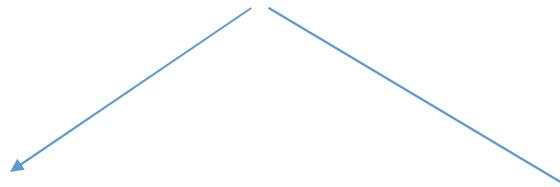
$$\min_{\phi} E_{z \sim p(z)} [\log(1 - D_{\theta}(G_{\phi}(z)))]$$

- For discriminator

$$\max_{\theta} E_{x \sim p_{data}} [\log D_{\theta}(x)] + E_{z \sim p(z)} [\log(1 - D_{\theta}(G_{\phi}(z)))]$$

# Loss function

$$\begin{aligned} \min_{\phi} \max_{\theta} & [\mathbb{E}_{x \sim p_{data}} \log D_{\theta}(x) \\ & + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta}(G_{\phi}(z)))] \end{aligned}$$



Generator minimizes it

Discriminator maximizes it

# Training a vanilla GAN

**Step 1:** Gradient Ascent on Discriminator

$$\max_{\theta} [\mathbb{E}_{x \sim p_{data}} \log D_{\theta}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta}(G_{\phi}(z)))]$$

**Step 2:** Gradient Descent on Generator

$$\min_{\phi} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta}(G_{\phi}(z)))$$

---

```

1: procedure GAN TRAINING
2:   for number of training iterations do
3:     for k steps do
4:       • Sample minibatch of  $m$  noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ 
5:       • Sample minibatch of  $m$  examples  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  from data generating distribution  $p_{data}(\mathbf{x})$ 
6:       • Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta} \frac{1}{m} \sum_{i=1}^m \left[ \log D_{\theta} \left( x^{(i)} \right) + \log \left( 1 - D_{\theta} \left( G_{\phi} \left( z^{(i)} \right) \right) \right) \right]$$

7:     end for
8:     • Sample minibatch of  $m$  noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ 
9:     • Update the generator by ascending its stochastic gradient

$$\nabla_{\phi} \frac{1}{m} \sum_{i=1}^m \left[ \log \left( D_{\theta} \left( G_{\phi} \left( z^{(i)} \right) \right) \right) \right]$$

10:   end for
11: end procedure

```

---

# Tackling vanishing gradients

$$V(D, G) = \mathbb{E}_{x \sim p(x)} [\log D(x)] + \boxed{\mathbb{E}_{z \sim q(z)} [\log(1 - D(G(z)))]}$$

$$\nabla_{\theta_G} V(D, G) = \nabla_{\theta_G} \mathbb{E}_{z \sim q(z)} [\log(1 - D(G(z)))]$$

- $\nabla_a \log(1 - \sigma(a)) = \frac{-\nabla_a \sigma(a)}{1 - \sigma(a)} = \frac{-\sigma(a)(1 - \sigma(a))}{1 - \sigma(a)} = -\sigma(a) = -D(G(z))$
- Gradient goes to 0 if  $D$  is confident, i.e.  $D(G(z)) \rightarrow 0$
- Minimize  $-\mathbb{E}_{z \sim q(z)} [\log D(G(z))]$  for **Generator** instead (keep Discriminator as it is)

# Optimal D

- The value function

$$L(G, D) = \int_x \left( p_r(x) \log(D(x)) + p_g(x) \log(1 - D(x)) \right) dx$$

- What is the gradient of L with respect to D?

$$D^*(x) = \frac{p_r(x)}{p_r(x) + p_g(x)} \in [0, 1]$$

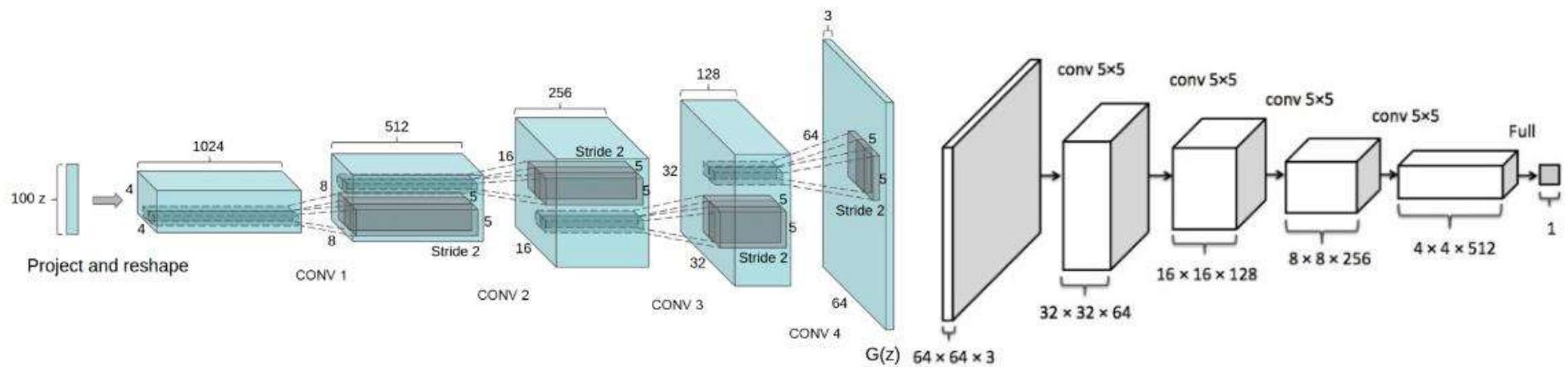
# Global optima

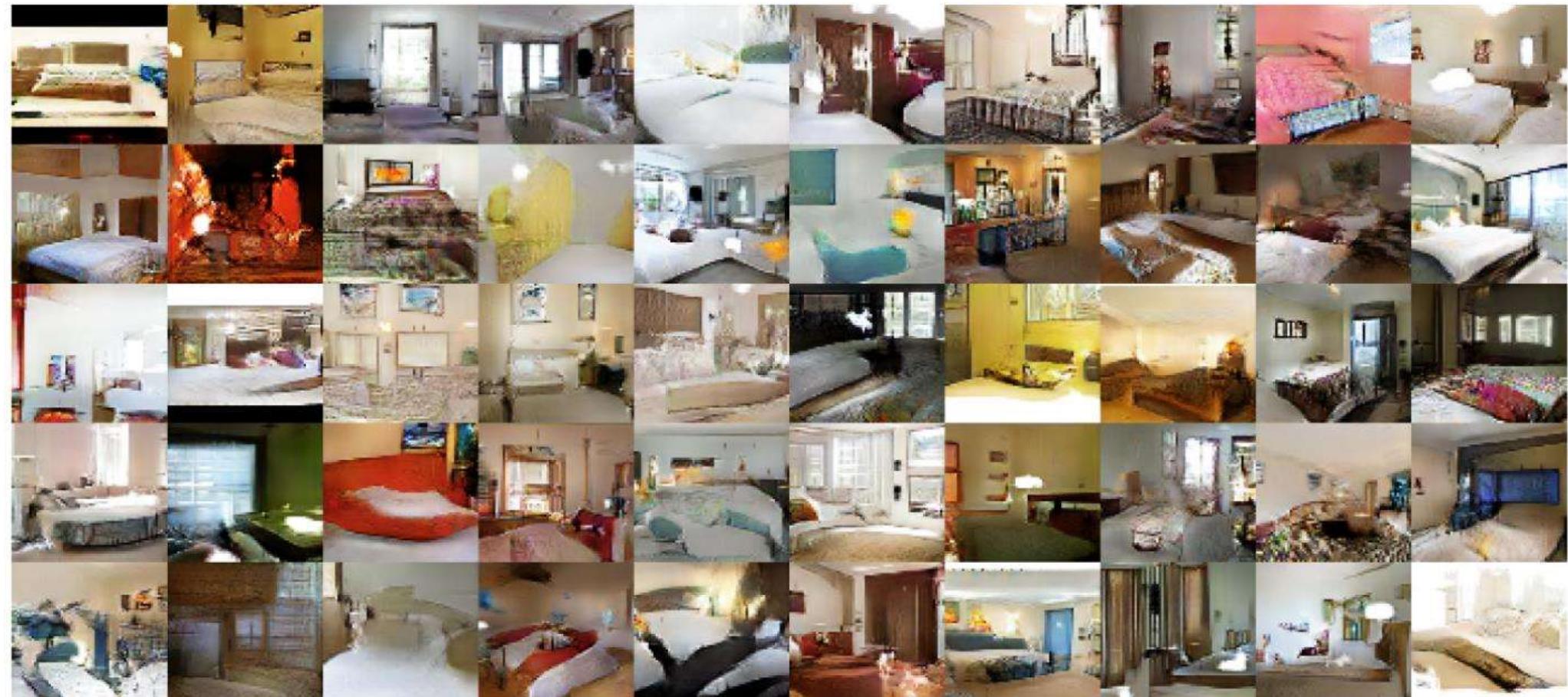
$$\begin{aligned}\min_G V(D^*, G) &= \int_x \left( p_r(x) \log D^*(x) + p_g(x) \log(1 - D^*(x)) \right) dx \\ &= \int_x \left( p_r(x) \log \frac{p_r(x)}{p_r(x) + p_g(x)} + p_g(x) \log \frac{p_g(x)}{p_r(x) + p_g(x)} \right) dx\end{aligned}$$

Which is equal to the JSD between real and fake distributions

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

# DC GAN



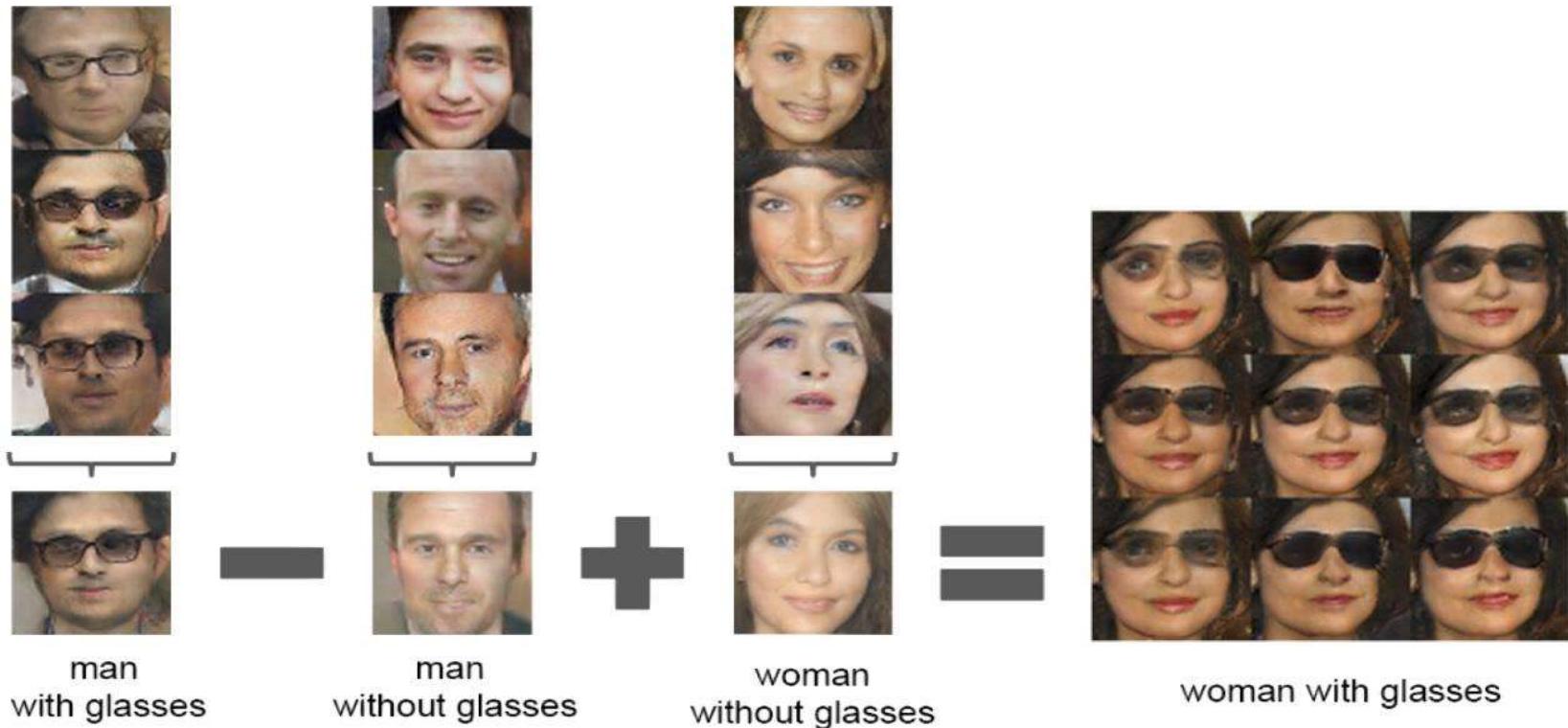


Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." arXiv:1511.06434 (2015).



Goodfellow, Ian, et al. "Generative adversarial nets." *Advances in neural information processing systems*. 2014.

# Latent vectors capture meaningful information



## Why GANs?

- Sampling (or generation) is straightforward.
- Training doesn't involve Maximum Likelihood estimation.
- Robust to Overfitting since Generator never sees the training data.
- Empirically, GANs are good at capturing the modes of the distribution.

# Some tips for GAN network design

- Replace pooling by strided convolution and unpooling by fractional strided convolution
- Use batch-norm
- Avoid FC layers in deeper architectures
- Use ReLU in all the generator layers except output where use tanh
- Use leaky-ReLU in the discriminator layers

\*\* See GAN-hacks (by soumith chintala) for more details

# Problems in training a GAN

*Training GANs consists in finding a Nash equilibrium to a two-player non-cooperative game. [...] Unfortunately, finding Nash equilibria is a very difficult problem. Algorithms exist for specialized cases, but we are not aware of any that are feasible to apply to the GAN game, where the cost functions are non-convex, the parameters are continuous, and the parameter space is extremely high-dimensional*

# Non convergence of GAN training

- Deep Learning models (in general) involve a single player
  - The player tries to maximize its reward (minimize its loss).
  - Use SGD (with Backpropagation) to find the optimal parameters.
  - SGD has convergence guarantees (under certain conditions).
  - **Problem:** With non-convexity, we might converge to local optima.

$$\min_G L(G)$$

- GANs instead involve two (or more) players

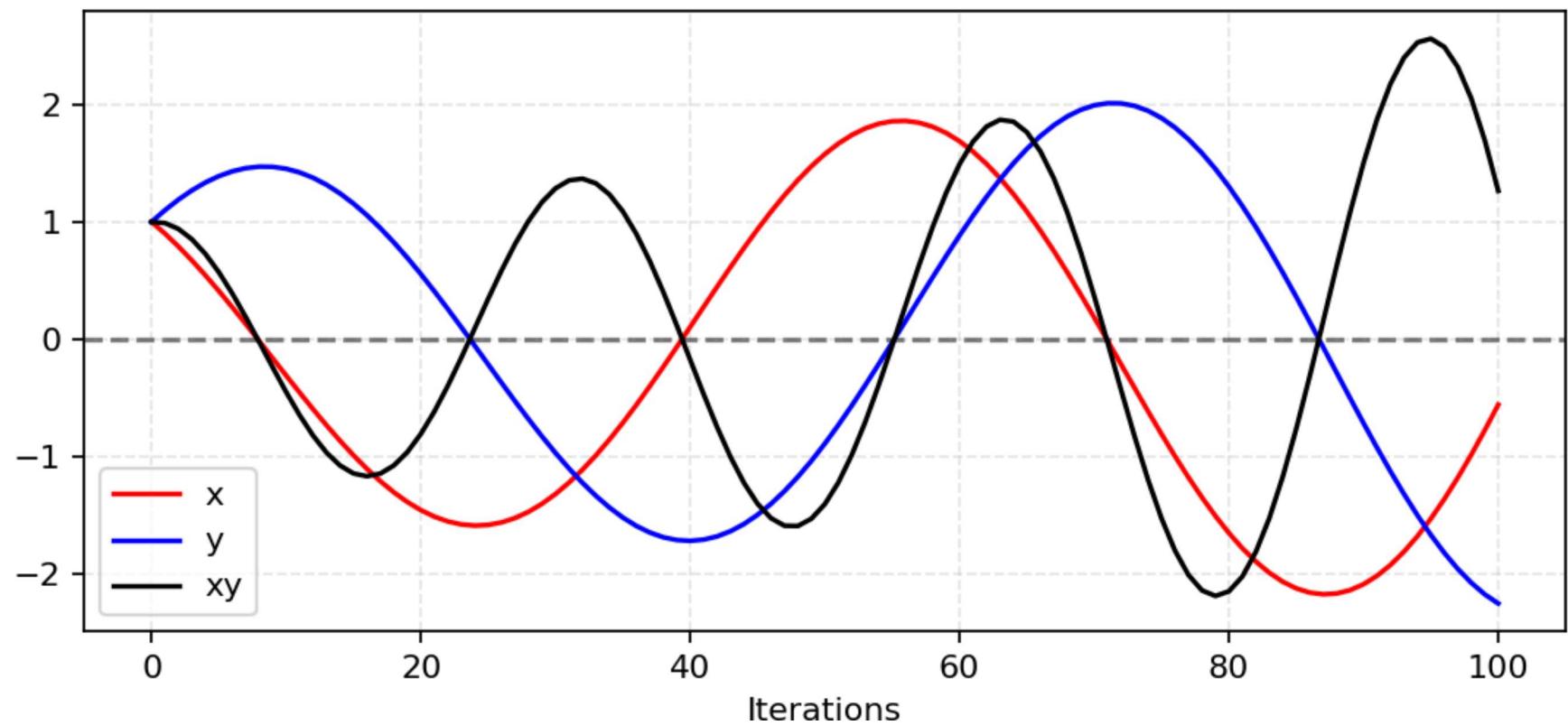
- Discriminator is trying to maximize its reward.
- Generator is trying to minimize Discriminator's reward.

$$\min_G \max_D V(D, G)$$

- SGD was not designed to find the Nash equilibrium of a game.
- **Problem:** We might not converge to the Nash equilibrium at all.

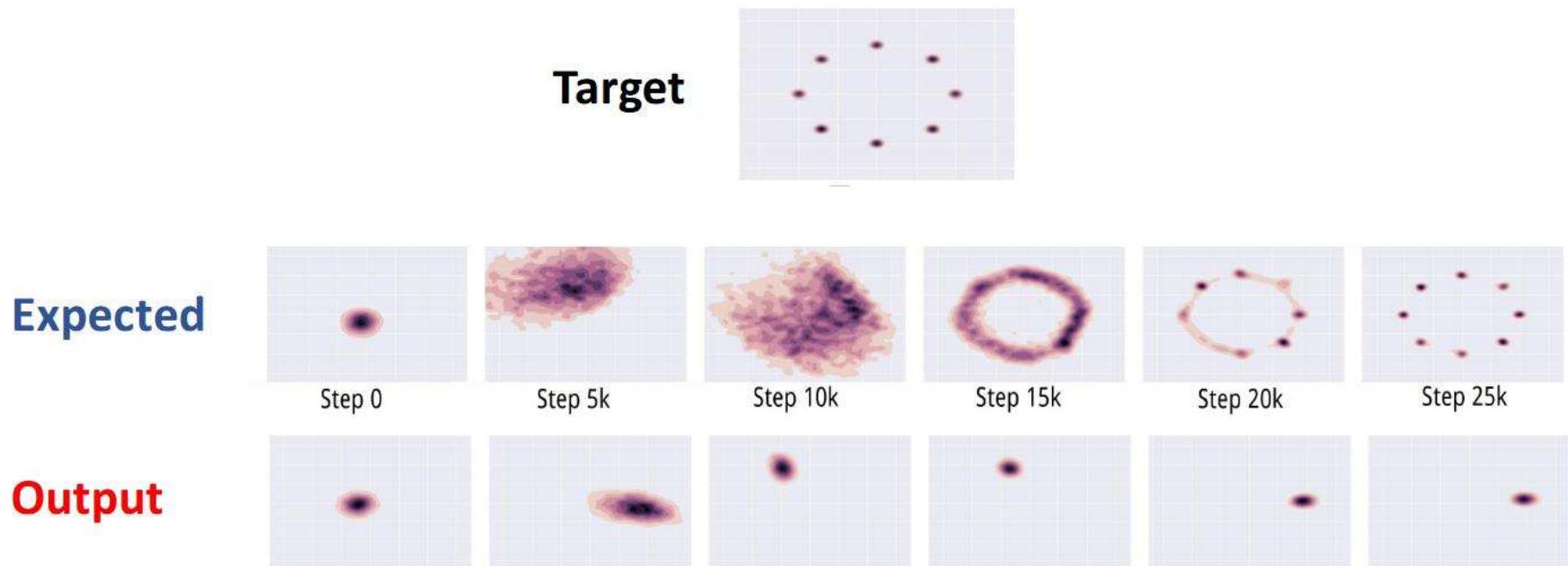
# Problem in obtaining the equilibrium

$$\min_x \max_y xy$$

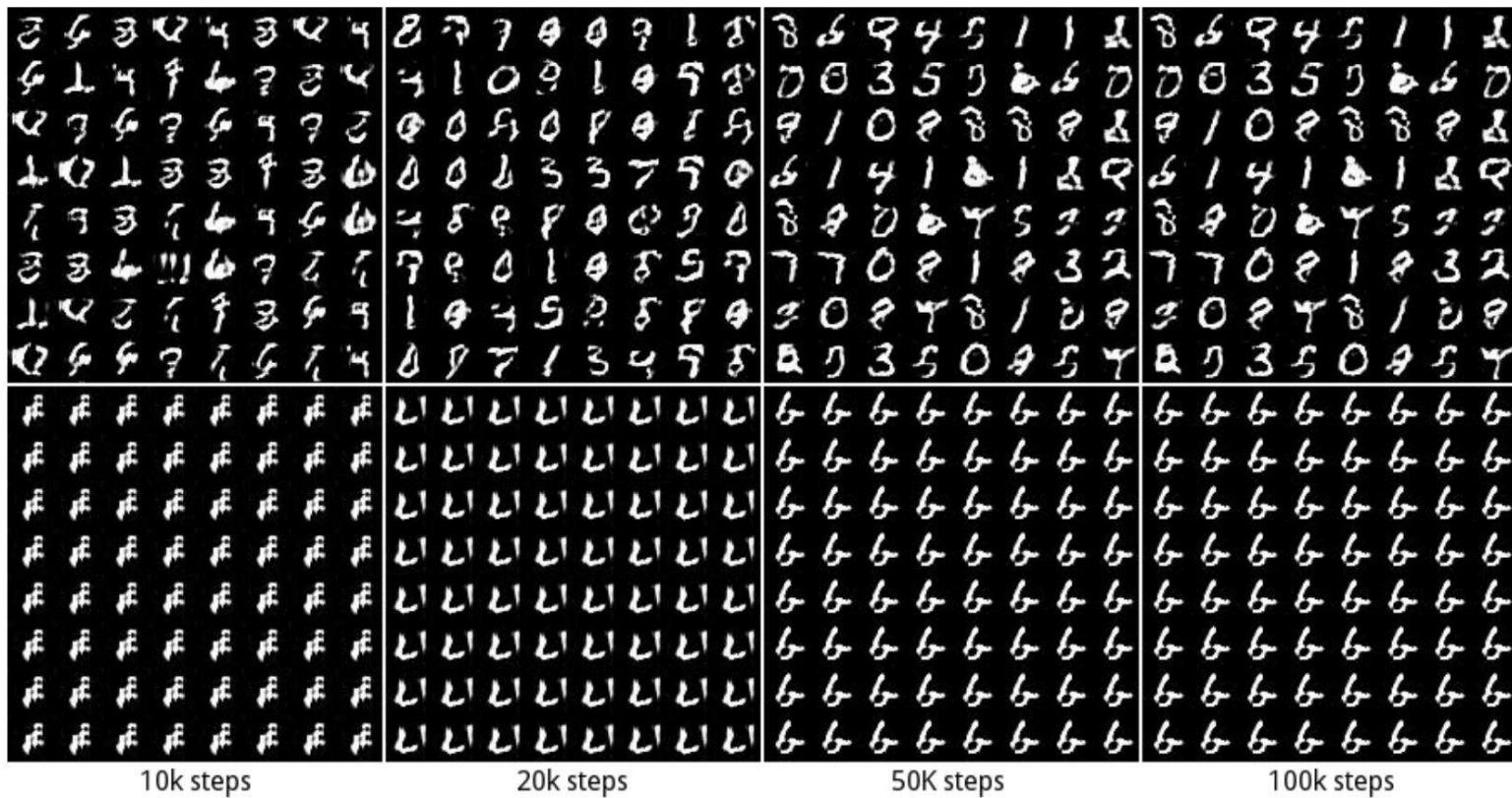


# Mode collapse

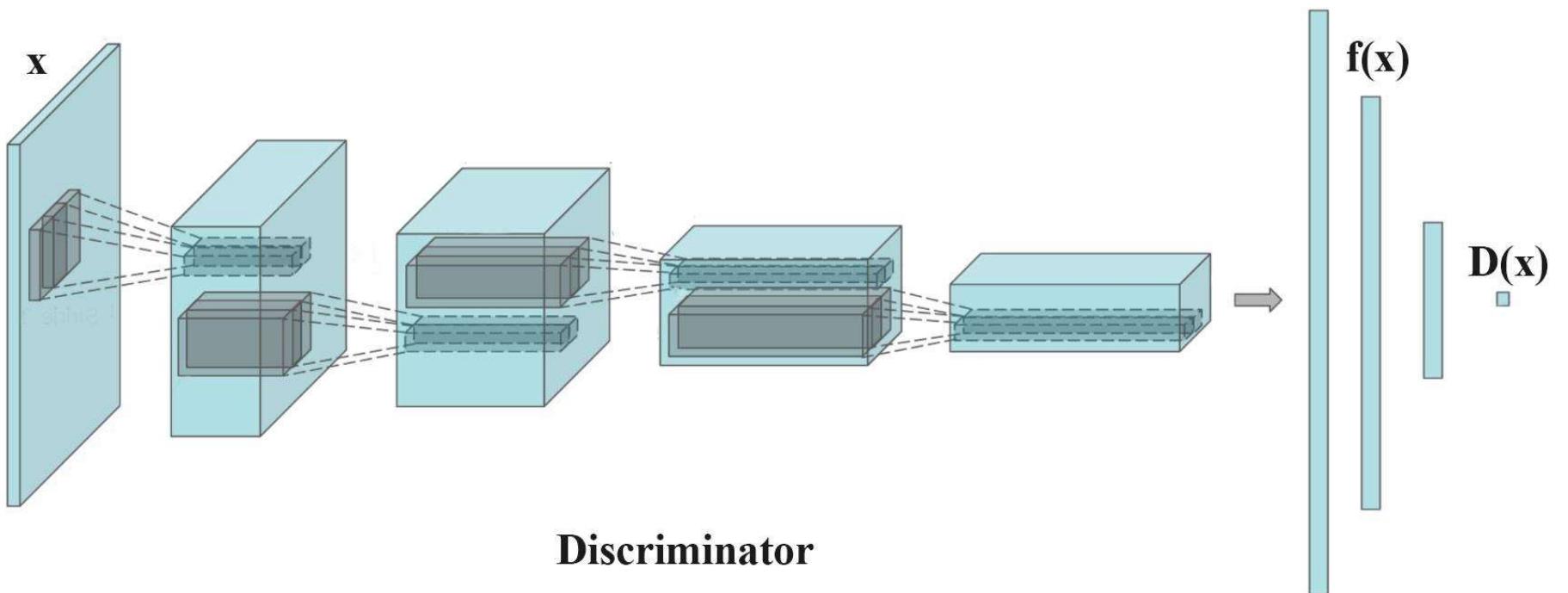
- Generator fails to output diverse samples



# Mode collapse in MNIST



# Feature matching



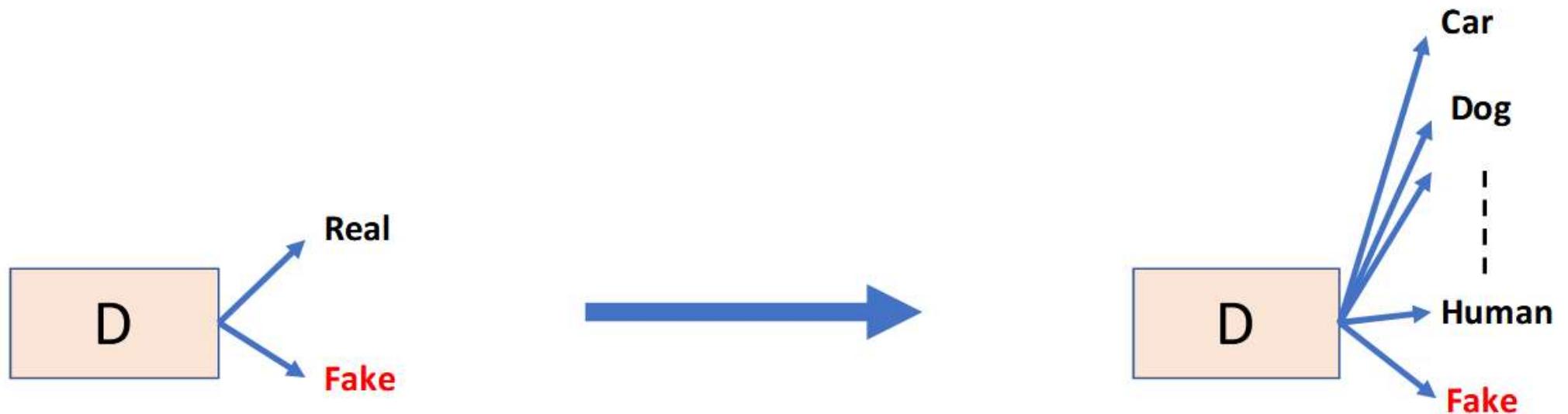
$$\|\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \mathbf{f}(\mathbf{x}) - \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} \mathbf{f}(G(\mathbf{z}))\|_2^2$$

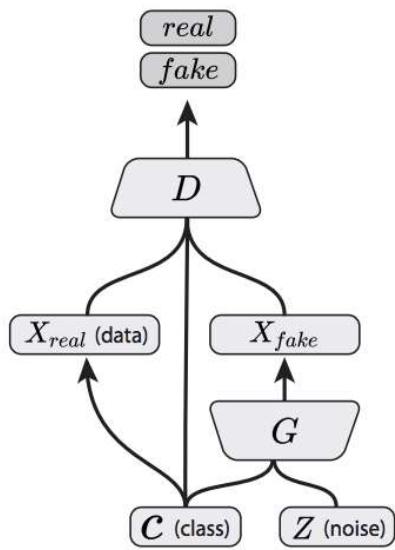
- Extract features that capture diversity in the mini-batch
  - For e.g. L2 norm of the difference between all pairs from the batch
- Feed those features to the discriminator along with the image
- Feature values will differ b/w diverse and non-diverse batches
  - Thus, Discriminator will rely on those features for classification
- This in turn,
  - Will force the Generator to match those feature values with the real data
  - Will generate diverse batches

# Mini-batch discrimination

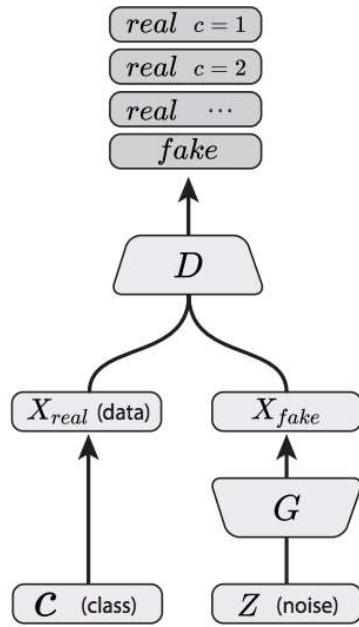
- **At Mode Collapse,**
  - Generator produces good samples, but a very few of them.
  - Thus, Discriminator can't tag them as fake.
- **To address this problem,**
  - Let the Discriminator know about this edge-case.
- **More formally,**
  - Let the Discriminator look at the entire batch instead of single examples
  - If there is lack of diversity, it will mark the examples as fake
- **Thus,**
  - Generator will be forced to produce diverse samples.

# Supervision with labels

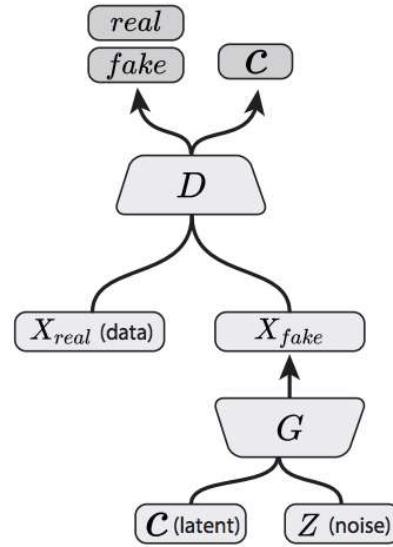




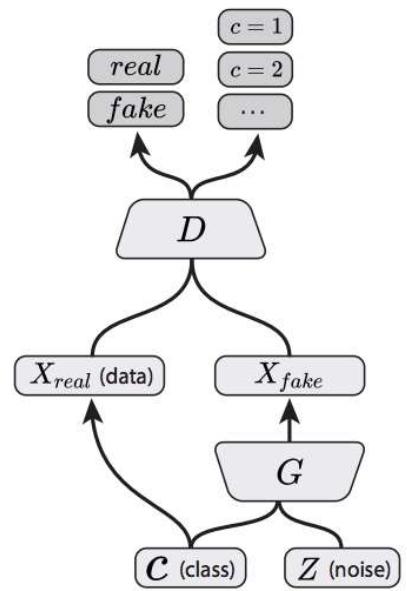
Conditional GAN  
(Mirza & Osindero, 2014)



Semi-Supervised GAN  
(Odena, 2016; Salimans, et al., 2016)



InfoGAN  
(Chen, et al., 2016)



AC-GAN  
(Present Work)

# An alternate view of GAN

$$V(D, G) = \mathbb{E}_{x \sim p(x)}[\log D(x)] + \mathbb{E}_{z \sim q(z)}[\log(1 - D(G(z)))]$$

$$D^* = \operatorname{argmax}_D V(D, G)$$

$$G^* = \operatorname{argmin}_G V(D, G)$$

- In this formulation, Discriminator's strategy was  $D(x) \rightarrow 1, D(G(z)) \rightarrow 0$
- Alternatively, we can flip the binary classification labels i.e. **Fake = 1, Real = 0**

$$V(D, G) = \mathbb{E}_{x \sim p(x)}[\log(1 - D(x))] + \mathbb{E}_{z \sim q(z)}[\log(D(G(z)))]$$

- In this new formulation, Discriminator's strategy will be  $D(x) \rightarrow 0, D(G(z)) \rightarrow 1$

We can use this

$$D^* = \operatorname{argmin}_D \mathbb{E}_{x \sim p(x)} \log(D(x)) + \mathbb{E}_{z \sim q(z)} [\log(1 - D(G(z)))]$$

- Now, we can replace cross-entropy with any loss function (**Hinge Loss**)

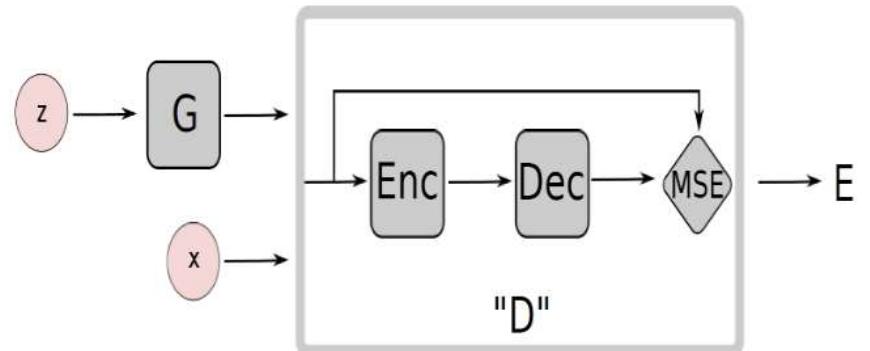
$$D^* = \operatorname{argmin}_D \mathbb{E}_{x \sim p(x)} D(x) + \mathbb{E}_{z \sim q(z)} \max(0, m - D(G(z)))$$

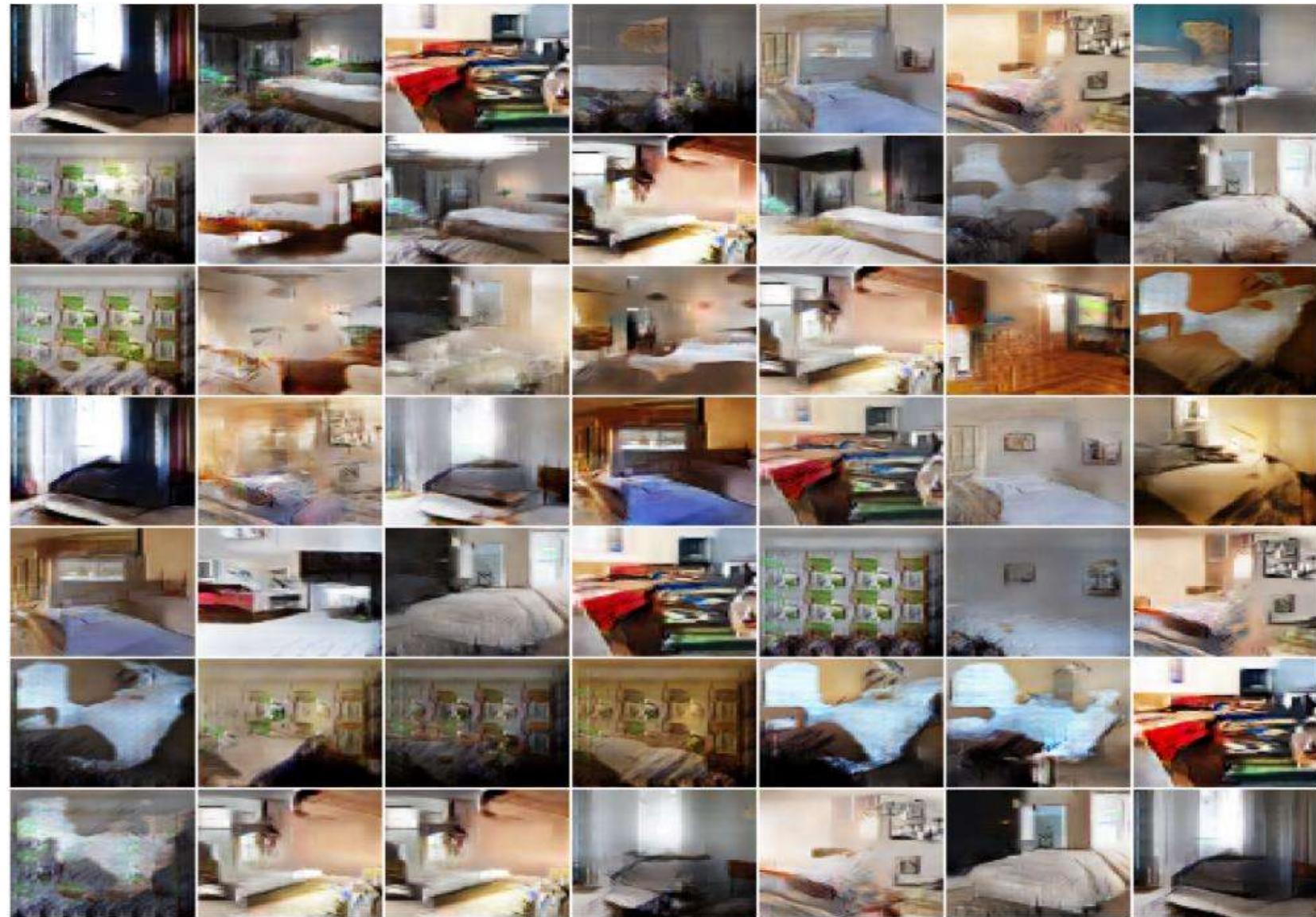
- And thus, instead of outputting probabilities, Discriminator just has to output:-
  - High values for fake samples
  - Low values for real samples

# Energy based GAN

- Modified game plans
  - **Generator** will try to generate samples with low values
  - **Discriminator** will try to assign high scores to fake values
- Use AutoEncoder inside the Discriminator
- Use Mean-Squared Reconstruction error as  $D(x)$ 
  - High Reconstruction Error for Fake samples
  - Low Reconstruction Error for Real samples

$$D(x) = ||Dec(Enc(x)) - x||_{MSE}$$



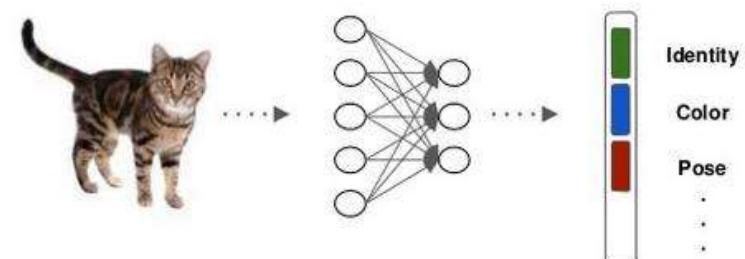


# One issue so far

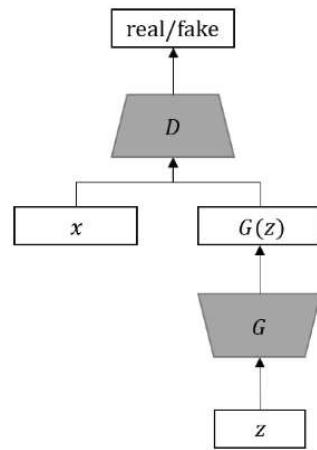
- How can we ensure the generation of images with a specific characteristics?
- Since the input to G is noise, we do not have explicit control over which images are generated.
- C-GAN solves to some extent by conditioning on the class labels
- A learning based unsupervised way is called INFOGAN
- It provides disentangled representations

# Disentangled representations

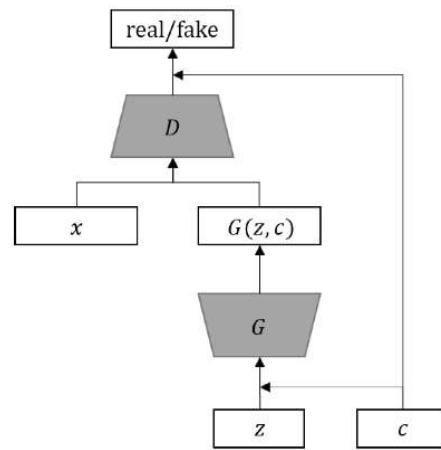
The ability to understand high-dimensional data, and to distill that knowledge into useful representations in an [unsupervised](#) manner, remains a key challenge in [deep learning](#). One approach to solving these challenges is through *disentangled representations*, models that capture the independent features of a given scene in such a way that if one feature changes, the others remain unaffected. If done successfully, a machine learning system that is designed to navigate the real world, such as a self driving car or a robot, can disentangle the different factors and properties of objects and their surroundings, enabling the generalization of knowledge to previously unobserved situations. While, unsupervised disentanglement methods have already been used for [curiosity driven exploration](#), [abstract reasoning](#), [visual concept learning](#) and [domain adaptation for reinforcement learning](#), recent progress in the field makes it difficult to know how well different approaches work and the extent of their limitations.



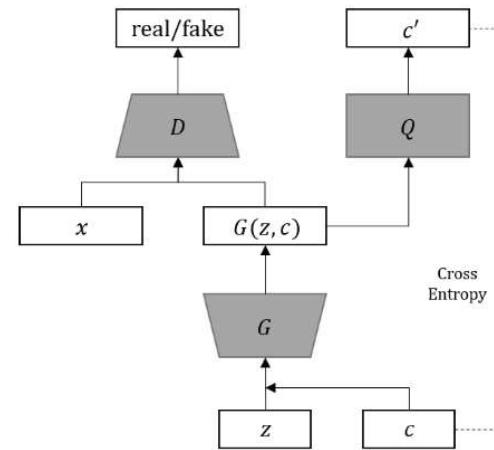
# The evolution so far



(a) GAN, DCGAN, LSGAN, WGAN



(b) CGAN



(c) InfoGAN

# Idea of mutual information

- Mutual Information captures the mutual dependence between two variables
- Mutual information between two variables  $X, Y$  is defined as:

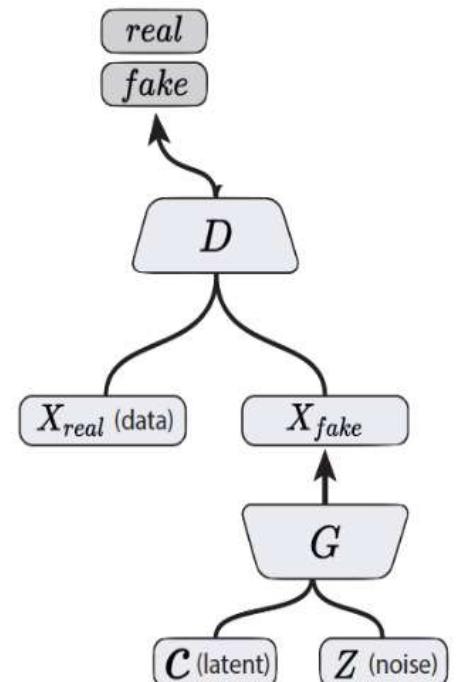
$$I(X; Y) = \sum_{x,y} p(x, y) \log\left(\frac{p(x, y)}{p(x)p(y)}\right)$$

$$I(X; Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

# InfoGAN

- We want to maximize the mutual information  $I$  between  $\mathbf{c}$  and  $\mathbf{x} = \mathbf{G}(\mathbf{z}, \mathbf{c})$
- Incorporate in the value function of the minimax game.

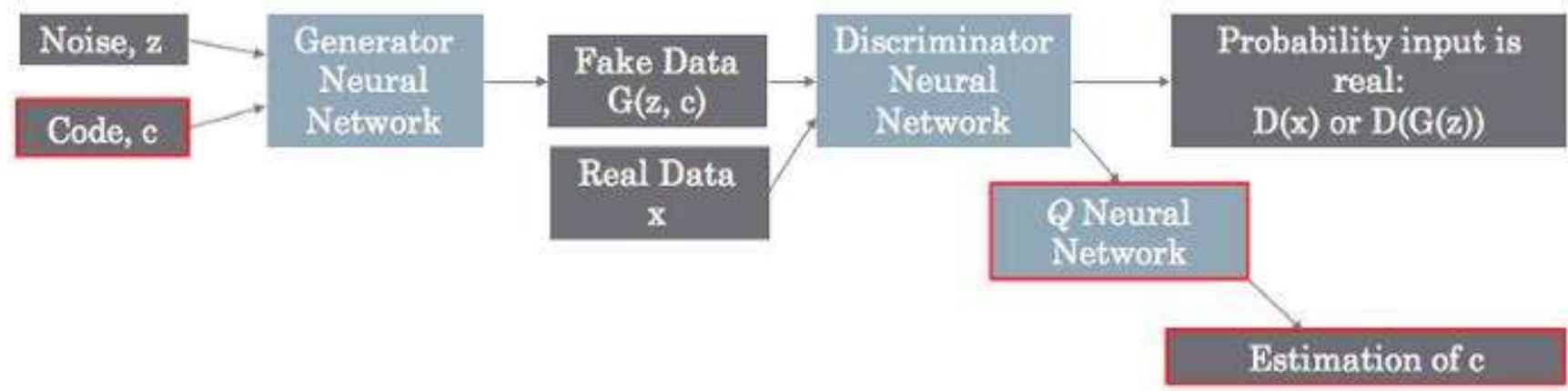
$$\min_G \max_D V_I(D, G) = V(D, G) - \lambda I(c; G(z, c))$$



## Mutual Information's Variational Lower bound

$$\begin{aligned} I(c; G(z, c)) &= H(c) - H(c|G(z, c)) \\ &= \mathbb{E}_{x \sim G(z, c)} \left[ \mathbb{E}_{c' \sim P(c|x)} [\log P(c'|x)] \right] + H(c) \\ &= \mathbb{E}_{x \sim G(z, c)} \left[ D_{KL}(P||Q) + \mathbb{E}_{c' \sim P(c|x)} [\log Q(c'|x)] \right] + H(c) \\ &\geq \mathbb{E}_{x \sim G(z, c)} \left[ \mathbb{E}_{c' \sim P(c|x)} [\log Q(c'|x)] \right] + H(c) \\ &\geq \mathbb{E}_{c \sim P(c), x \sim G(z, c)} [\log Q(c|x)] + H(c) \end{aligned}$$

# InfoGAN



## Example (CGAN)

# Examples (InfoGAN)



(a) Azimuth (pose)

(b) Presence or absence of glasses



(c) Hair style

(d) Emotion

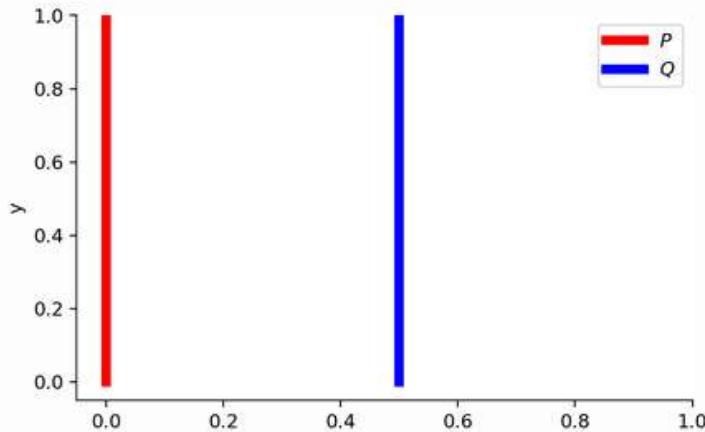


(a) Rotation



(b) Width

# Calculate KL, JS for non-overlapping distributions



$$D_{KL}(P\|Q) = \sum_{x=0, y \sim U(0,1)} 1 \cdot \log \frac{1}{0} = +\infty$$

$$D_{KL}(Q\|P) = \sum_{x=\theta, y \sim U(0,1)} 1 \cdot \log \frac{1}{0} = +\infty$$

$$D_{JS}(P, Q) = \frac{1}{2} \left( \sum_{x=0, y \sim U(0,1)} 1 \cdot \log \frac{1}{1/2} + \sum_{x=0, y \sim U(0,1)} 1 \cdot \log \frac{1}{1/2} \right) = \log 2$$

Always constant

[

Let us first look at a simple case where the probability domain is *discrete*. For example, suppose we have two distributions  $P$  and  $Q$ , each has four piles of dirt and both have ten shovelfuls of dirt in total. The numbers of shovelfuls in each dirt pile are assigned as follows:

$$\begin{aligned}P_1 &= 3, P_2 = 2, P_3 = 1, P_4 = 4 \\Q_1 &= 1, Q_2 = 2, Q_3 = 4, Q_4 = 3\end{aligned}$$

In order to change  $P$  to look like  $Q$ , as illustrated in Fig. 7, we:

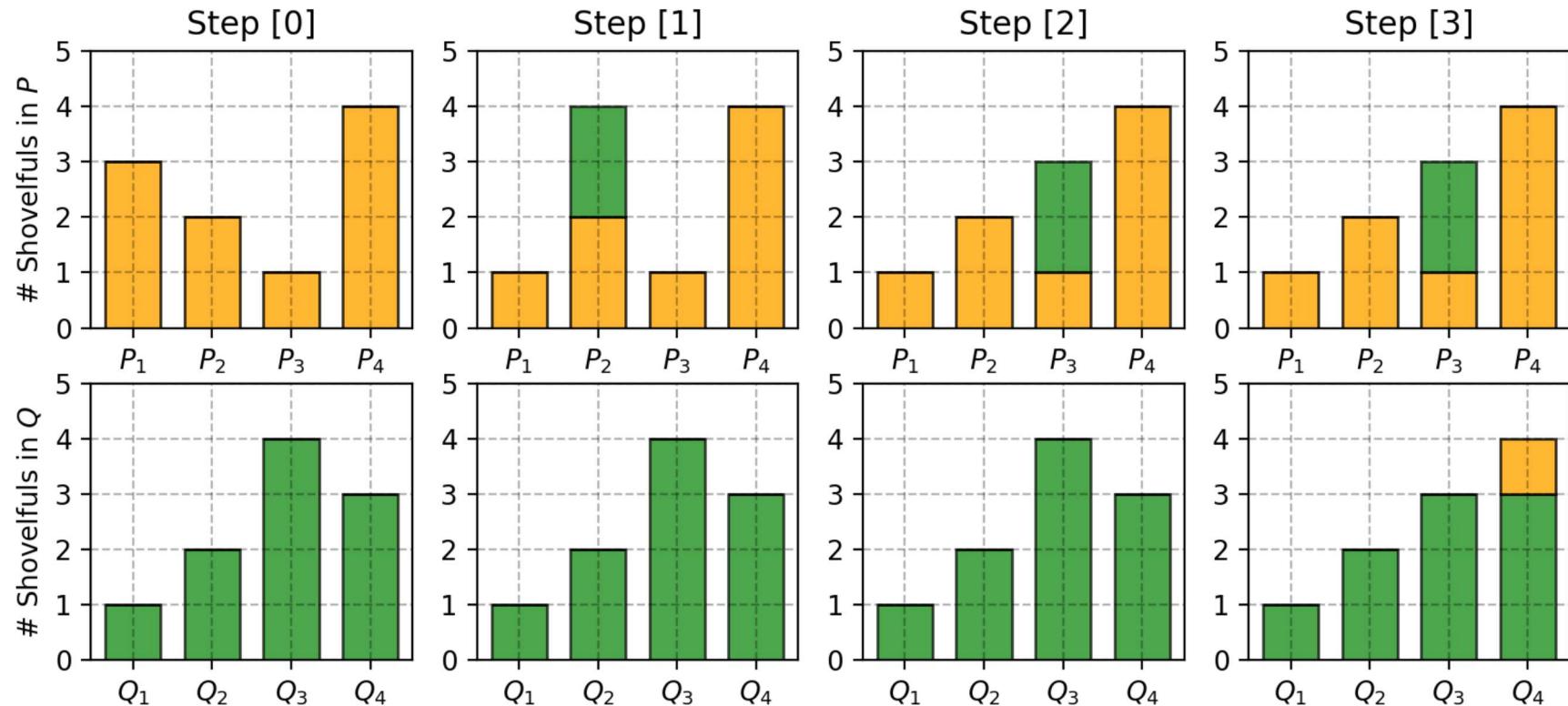
- First move 2 shovelfuls from  $P_1$  to  $P_2 \Rightarrow (P_1, Q_1)$  match up.
- Then move 2 shovelfuls from  $P_2$  to  $P_3 \Rightarrow (P_2, Q_2)$  match up.
- Finally move 1 shovelfuls from  $Q_3$  to  $Q_4 \Rightarrow (P_3, Q_3)$  and  $(P_4, Q_4)$  match up.

If we label the cost to pay to make  $P_i$  and  $Q_i$  match as  $\delta_i$ , we would have  $\delta_{i+1} = \delta_i + P_i - Q_i$  and in the example:

$$\begin{aligned}\delta_0 &= 0 \\ \delta_1 &= 0 + 3 - 1 = 2 \\ \delta_2 &= 2 + 2 - 2 = 2 \\ \delta_3 &= 2 + 1 - 4 = -1 \\ \delta_4 &= -1 + 4 - 3 = 0\end{aligned}$$

Finally the Earth Mover's distance is  $W = \sum |\delta_i| = 5$ .

# EM distance



# Wasserstein distance

$$W(p_r, p_g) = \inf_{\gamma \sim \Pi(p_r, p_g)} \mathbb{E}_{(x,y) \sim \gamma} [\|x - y\|]$$

$$\sum_{x,y} \gamma(x,y) \|x - y\| = \mathbb{E}_{x,y \sim \gamma} \|x - y\|$$

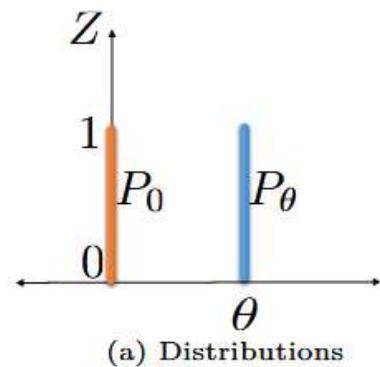
Kantorovich rubinson duality

$$W(p_r, p_g) = \frac{1}{K} \sup_{\|f\|_L \leq K} \mathbb{E}_{x \sim p_r} [f(x)] - \mathbb{E}_{x \sim p_g} [f(x)]$$

$P_0$ : distribution of  $(0, Z)$ , where  $Z \sim U[0, 1]$

$P_\theta$ : distribution of  $(\theta, Z)$ , where  $\theta$  is a single real parameter

- $KL(P_0||P_\theta) = KL(P_\theta||P_0) = \begin{cases} +\infty & \text{if } \theta \neq 0 \\ 0 & \text{if } \theta = 0 \end{cases}$
- $JS(P_0||P_\theta) = \begin{cases} \log 2 & \text{if } \theta \neq 0 \\ 0 & \text{if } \theta = 0 \end{cases}$
- $W(P_0||P_\theta) = |\theta|$



---

**Algorithm 1** WGAN, our proposed algorithm. All experiments in the paper used the default values  $\alpha = 0.00005$ ,  $c = 0.01$ ,  $m = 64$ ,  $n_{\text{critic}} = 5$ .

---

**Require:** :  $\alpha$ , the learning rate.  $c$ , the clipping parameter.  $m$ , the batch size.  
 $n_{\text{critic}}$ , the number of iterations of the critic per generator iteration.

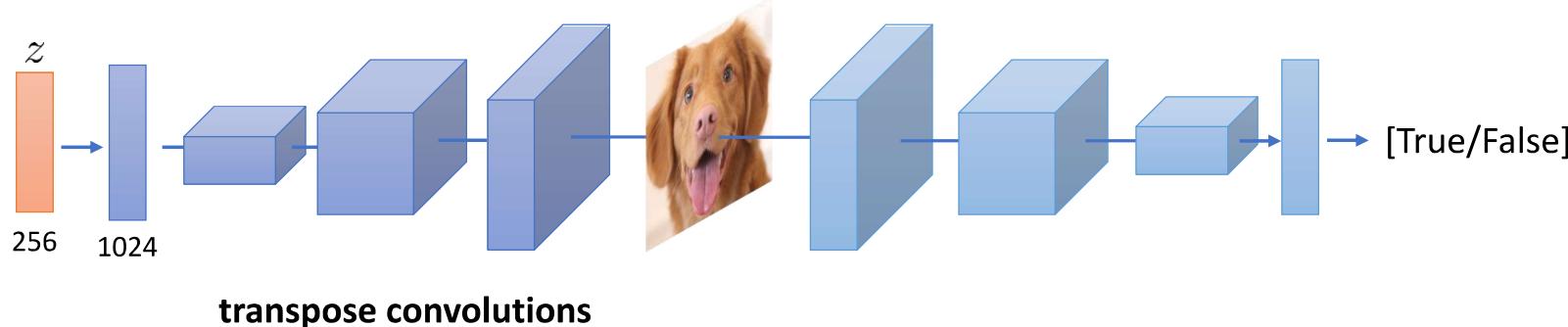
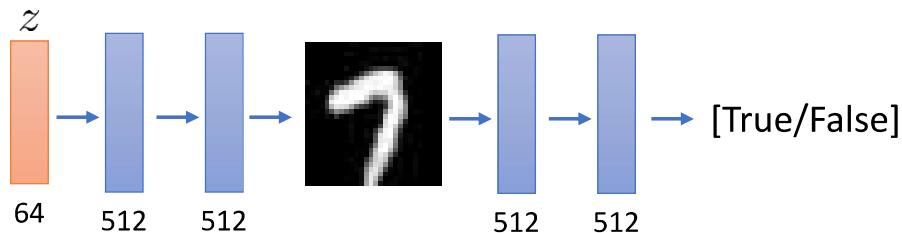
**Require:** :  $w_0$ , initial critic parameters.  $\theta_0$ , initial generator's parameters.

```
1: while  $\theta$  has not converged do
2:   for  $t = 0, \dots, n_{\text{critic}}$  do
3:     Sample  $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$  a batch from the real data.
4:     Sample  $\{z^{(i)}\}_{i=1}^m \sim p(z)$  a batch of prior samples.
5:      $g_w \leftarrow \nabla_w \left[ \frac{1}{m} \sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)})) \right]$ 
6:      $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$ 
7:      $w \leftarrow \text{clip}(w, -c, c)$ 
8:   end for
9:   Sample  $\{z^{(i)}\}_{i=1}^m \sim p(z)$  a batch of prior samples.
10:   $g_\theta \leftarrow -\nabla_\theta \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)}))$ 
11:   $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$ 
12: end while
```

---

# GAN architectures

some made-up architectures



a real architecture (BigGAN)

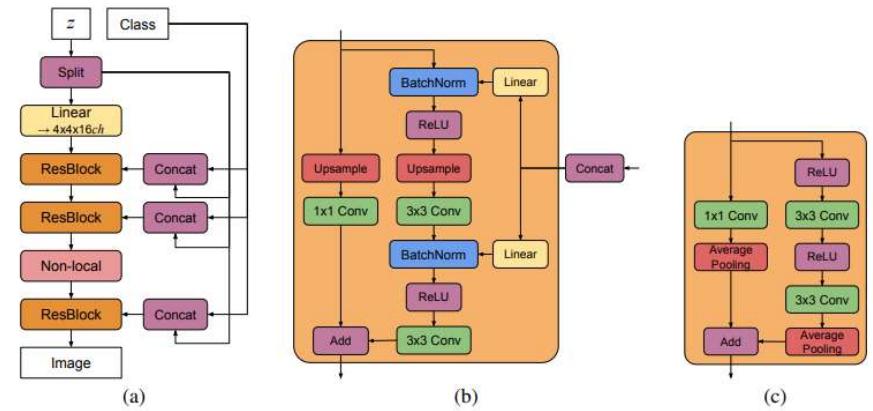
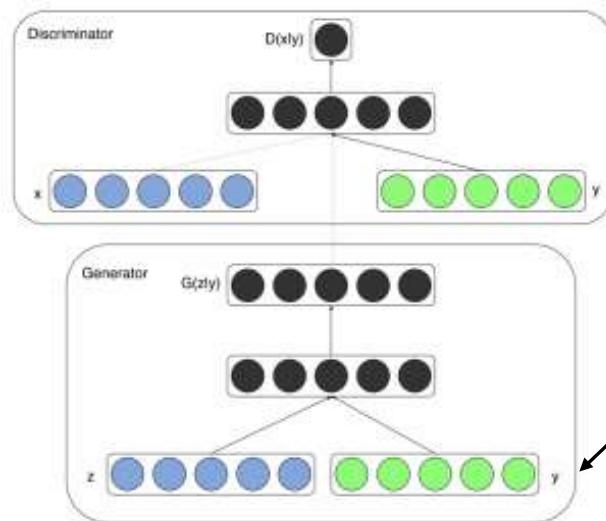


Figure 15: (a) A typical architectural layout for BigGAN's  $\mathbf{G}$ ; details are in the following tables. (b) A Residual Block ( $ResBlock\ up$ ) in BigGAN's  $\mathbf{G}$ . (c) A Residual Block ( $ResBlock\ down$ ) in BigGAN's  $\mathbf{D}$ .

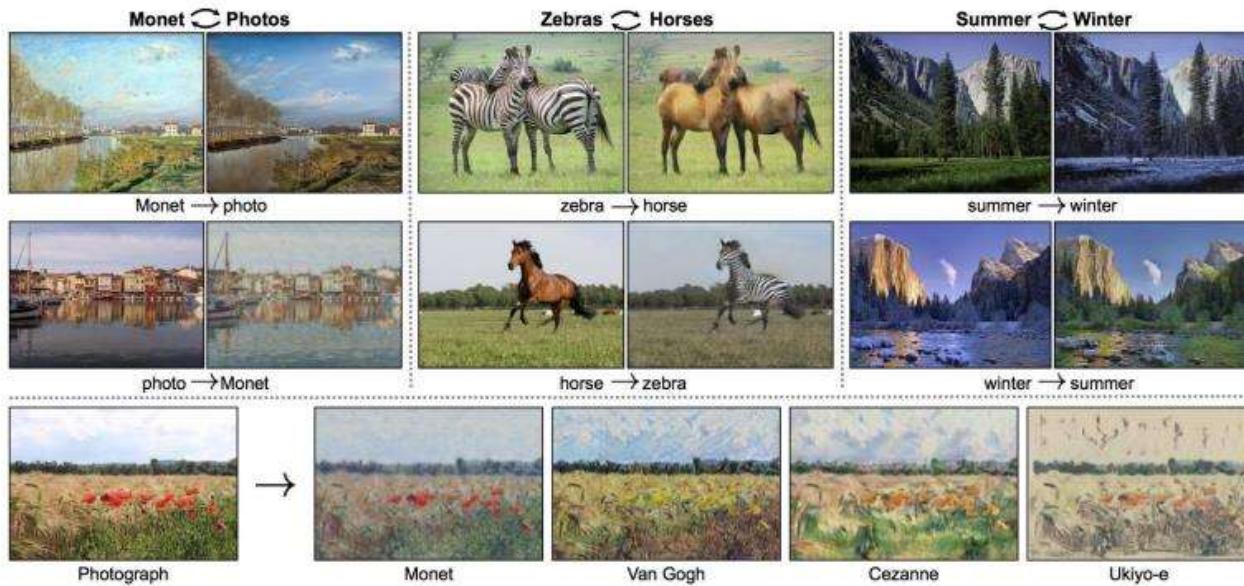
# Conditional GANs



append conditioning (e.g., class label)  
to **both** generator and discriminator



# CycleGAN



**Problem:** we don't know which images "go together"

# CycleGAN

two (conditional) generators:

$G$ : turn  $X$  into  $Y$  (e.g., horse into zebra)

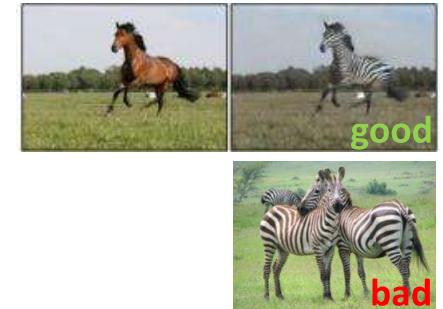
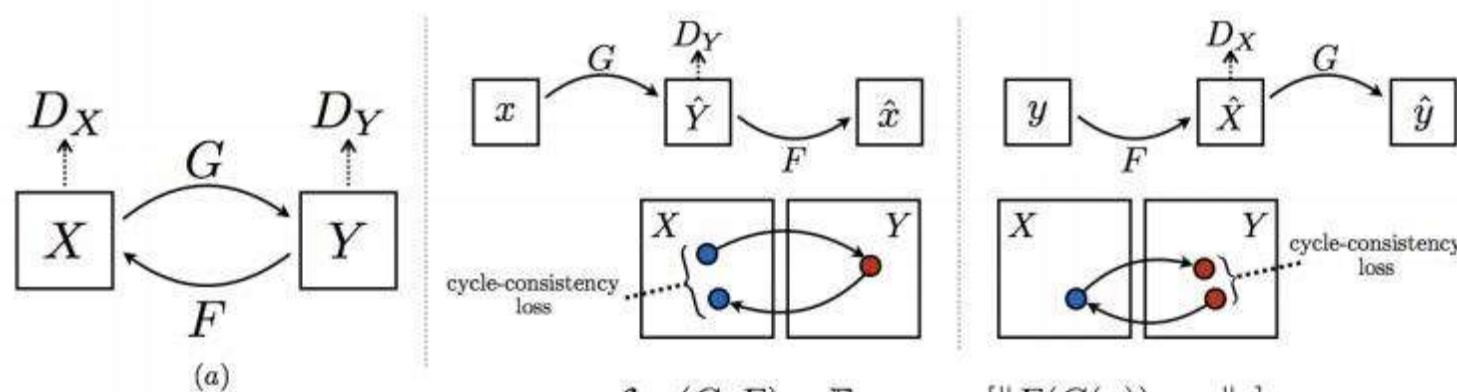
$F$ : turn  $Y$  into  $X$  (e.g., zebra into horse)

two discriminators:

$D_X$ : is it a realistic horse?

$D_Y$ : is it a realistic zebra?

**Problem:** why should the “translated” zebra look anything like the original horse?



If I turn this horse into a zebra, and then turn that zebra back into a horse, I should get the same horse!

$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)} [\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)} [\|G(F(y)) - y\|_1].$$



# Summary

- The GAN is a 2-player game  $\min_G \max_D V(D, G) = E_{x \sim p_{\text{data}}(x)}[\log D(x)] + E_{z \sim p(z)}[\log(1 - D(G(z)))]$
- We can derive the **optimal** discriminator, and from this determine what objective is minimized at the Nash equilibrium
  - Note that this does **not** guarantee that we'll actually find the Nash equilibrium!
- We can train either fully connected **or** convolutional GANs
- We can turn horses into zebras

$$D_G^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_G(x)}$$

$$V(D_G^*, G) = D_{\text{JS}}(p_{\text{data}} \| p_G)$$

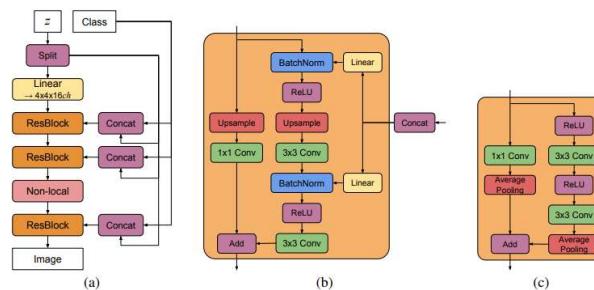
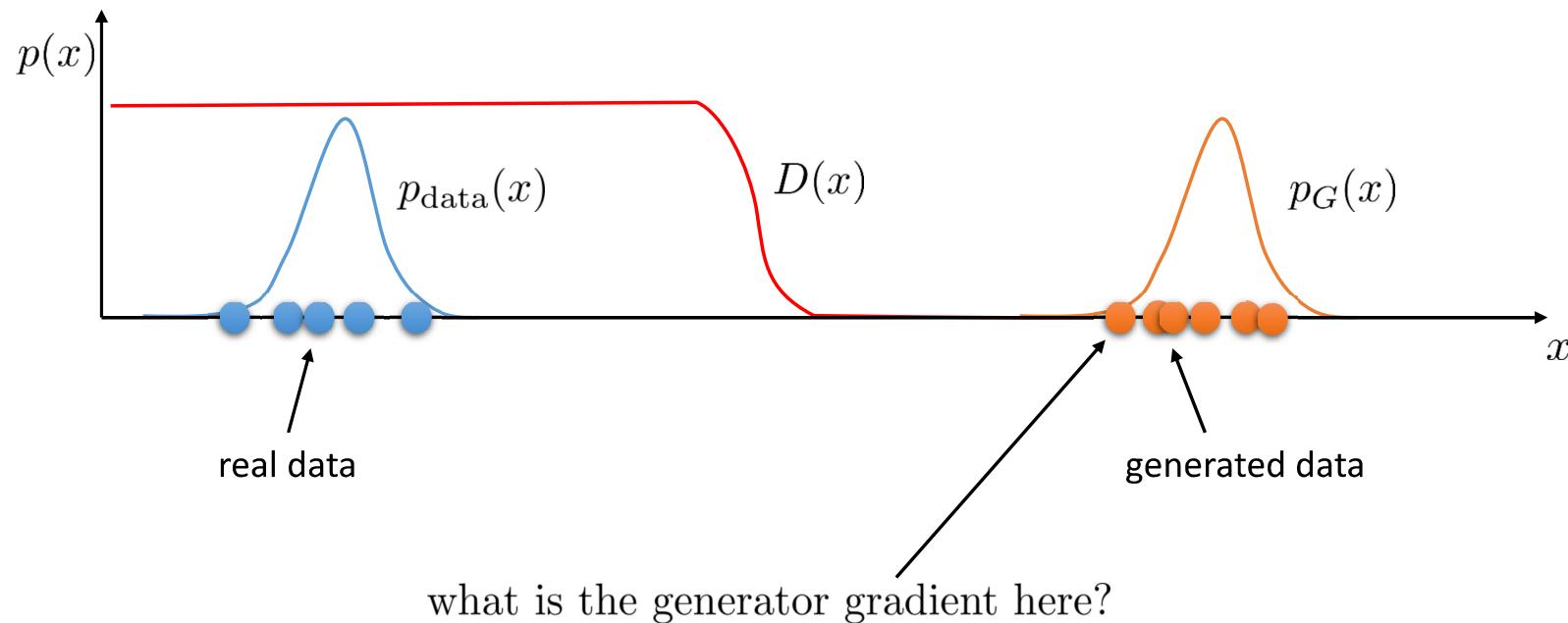


Figure 15: (a) A typical architectural layout for BigGAN's  $G$ ; details are in the following tables.  
 (b) A Residual Block (*ResBlock up*) in BigGAN's  $G$ . (c) A Residual Block (*ResBlock down*) in BigGAN's  $D$ .

# Improved GAN Training

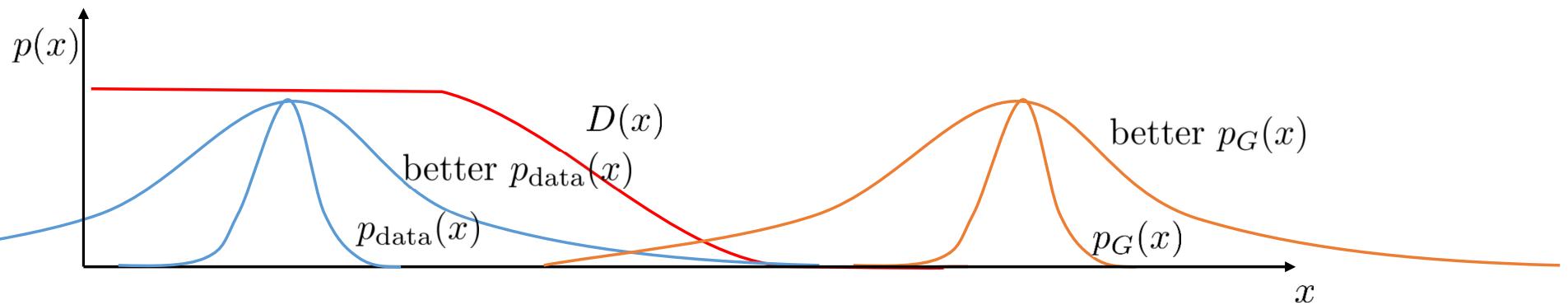
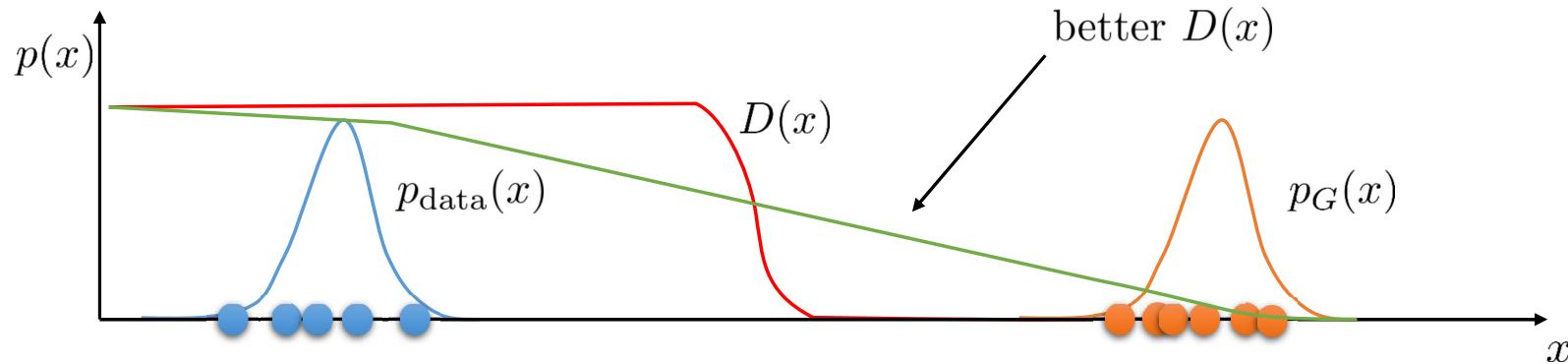
# Why is training GANs hard?



$$\min_{\theta} \max_{\phi} V(\theta, \phi) = E_{x \sim p_{\text{data}}(x)} [\log D_{\phi}(x)] + E_{z \sim p(z)} [\underbrace{\log(1 - D_{\phi}(G_{\theta}(z)))}_{\text{all of these values are basically the same}}]$$

all of these values are basically the same

# How can we make it better?



# Improved GAN techniques

(in no particular order)

**Least-squares GAN (LSGAN)**

discriminator outputs real-valued number

**Wasserstein GAN (WGAN)**

discriminator is constrained to be Lipschitz-continuous

**Gradient penalty**

discriminator is constrained to be continuous even harder

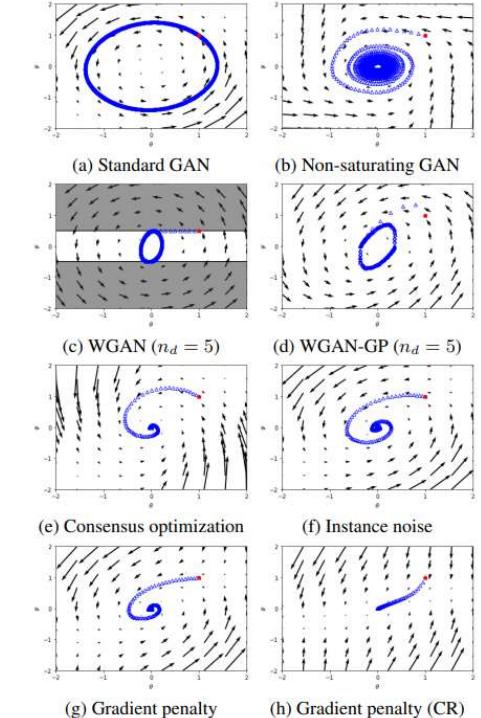
**Spectral norm**

discriminator is **really** constrained to be continuous

**Instance noise**

add noise to the data and generated samples

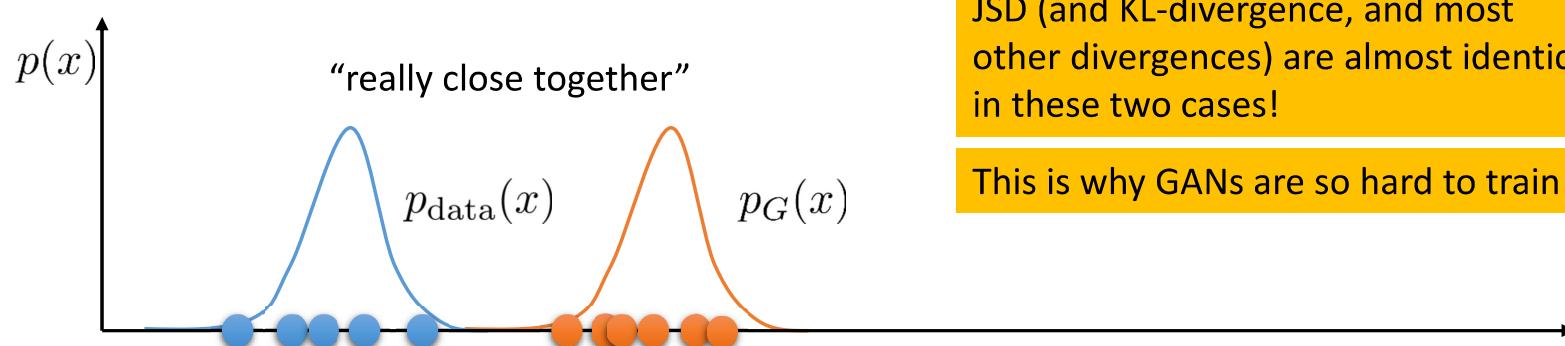
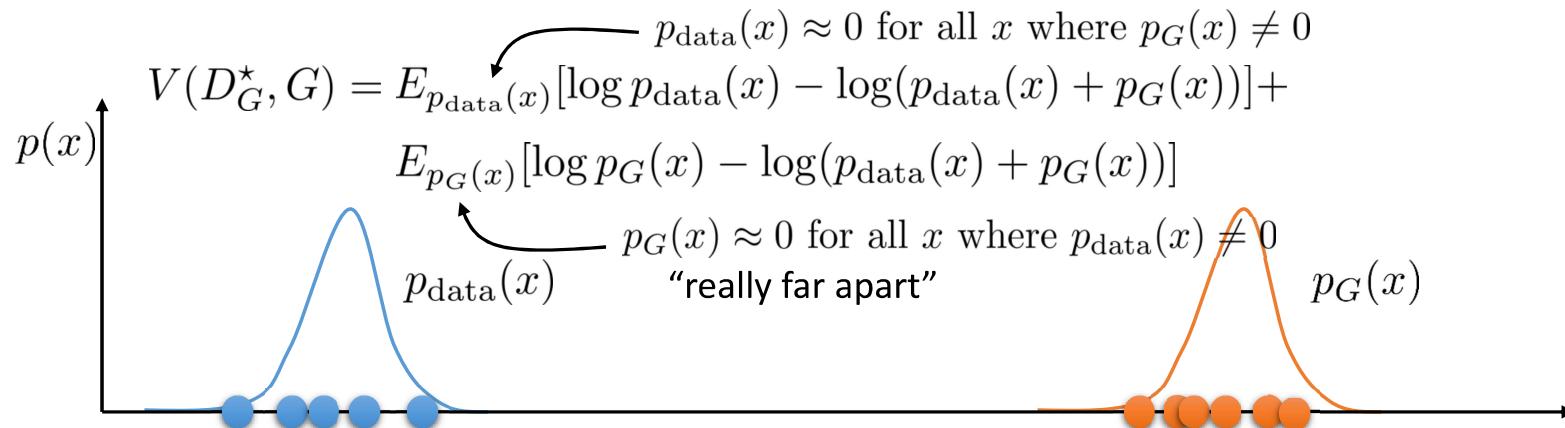
these are pretty good choices today



Mescheder et al. **Which Training Methods for GANs do actually Converge?** 2108.

# Wasserstein GAN (WGAN)

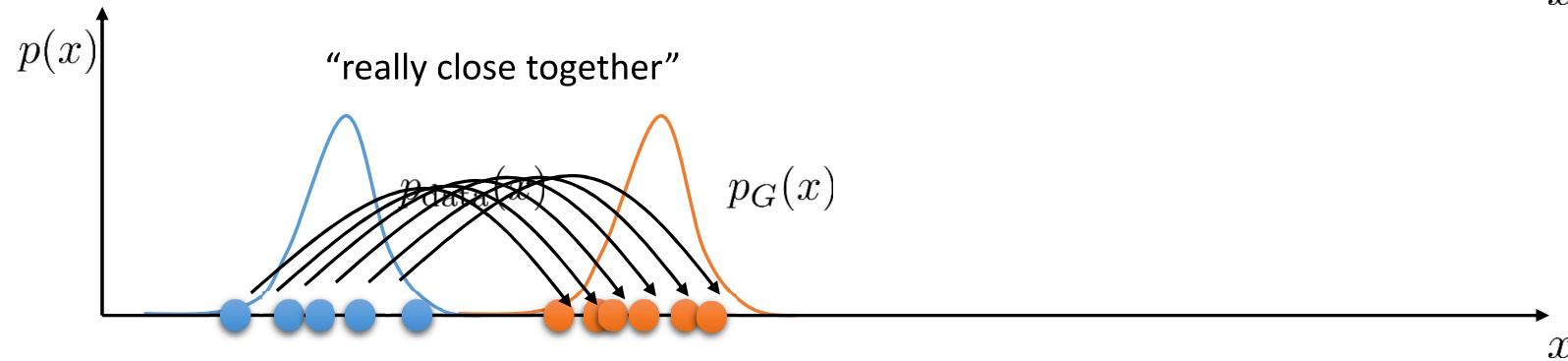
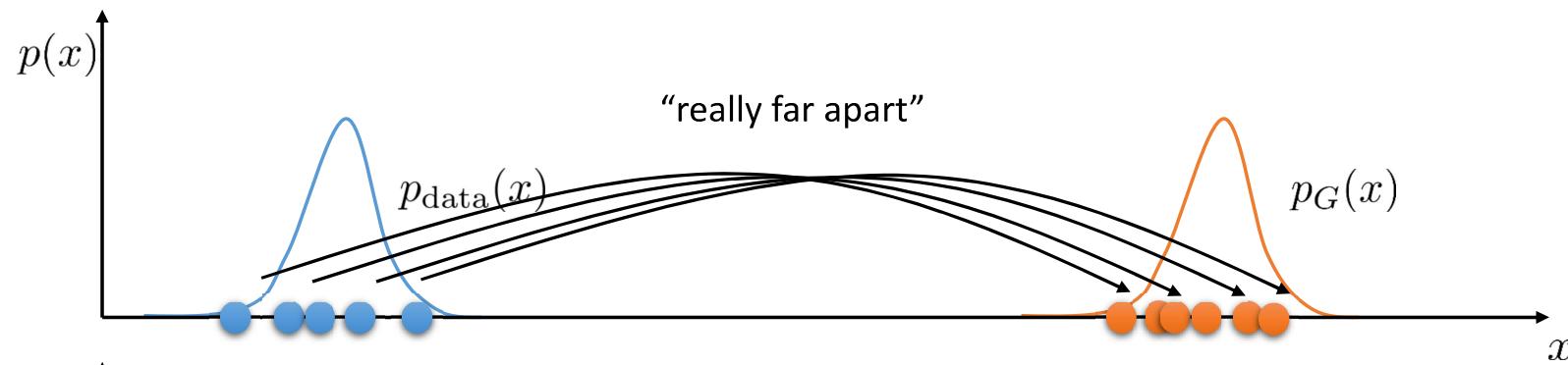
**High-level intuition:** the JS divergence used by the classic GAN doesn't account for "distance"



# Wasserstein GAN (WGAN)

**A better metric:** consider how far apart (in Euclidean space) all the “bits” of probability are

**More precisely:** optimal transport (“Earth mover’s distance”) – how far do you have to go to “transport” one distribution into another



# Wasserstein GAN (WGAN)

**A better metric:** consider how far apart (in Euclidean space) all the “bits” of probability are

**More precisely:** optimal transport (“Earth mover’s distance”) – how far do you have to go to “transport” one distribution into another

Formal definition: (don’t worry too much if this is hard to understand, not actually necessary to *implement* WGAN)

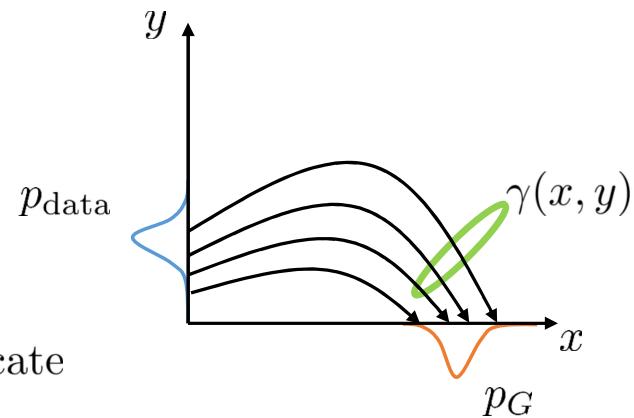
$$W(p_{\text{data}}, p_G) = \inf_{\gamma} E_{(x,y) \sim \gamma(x,y)} [| |x - y| |]$$



$\gamma(x, y)$  is a *distribution* over  $x, y$

with marginals  $\gamma_X(x) = p_{\text{data}}(x)$  and  $\gamma_Y(y) = p_G(y)$

**intuition:** correlations between  $x$  and  $y$  in  $\gamma(x, y)$  indicate which  $x$  should be “transported” to which  $y$



# Wasserstein GAN (WGAN)

**A better metric:** consider how far apart (in Euclidean space) all the “bits” of probability are

**More precisely:** optimal transport (“Earth mover’s distance”) – how far do you have to go to “transport” one distribution into another

$$W(p_{\text{data}}, p_G) = \inf_{\gamma} E_{(x,y) \sim \gamma(x,y)} [| |x - y| |]$$

could learn  $\gamma$  directly       $p_{\text{data}}(x)$  is unknown  
but this is very hard       $\gamma(x, y)$  is really complex

cool theorem based on Kantorovich-Rubinstein duality:

$$W(p_{\text{data}}, p_G) = \sup_{||f||_L \leq 1} E_{p_{\text{data}}}[f(x)] - E_{p_G(x)}[f(x)]$$



set of all 1-Lipschitz scalar functions

$$|f(x) - f(y)| \leq |x - y|$$

equivalent to saying function has bounded slope  
i.e., it should not be too steep

expressed as difference of expectations  
under  $p_G(x)$  and  $p_{\text{data}}(x)$   
just like a regular GAN!

How?

# Wasserstein GAN (WGAN)

**A better metric:** consider how far apart (in Euclidean space) all the “bits” of probability are

**More precisely:** optimal transport (“Earth mover’s distance”) – how far do you have to go to “transport” one distribution into another

$$W(p_{\text{data}}, p_G) = \sup_{\|f\|_L \leq 1} E_{p_{\text{data}}}[f(x)] - E_{p_G(x)}[f(x)]$$



doesn’t guarantee 1-Lipschitz  
unless we pick bounds very carefully

does guarantee K-Lipschitz  
for some finite  $K$

set of all 1-Lipschitz scalar functions

$$|f(x) - f(y)| \leq |x - y|$$

**idea:** if  $f$  is a neural net with ReLU activations, can bound the weights  $W$

$$\text{1-layer: } f_\theta(x) = \text{ReLU}(W_1 x + b_1) \quad \theta = \{W_1, b_1\}$$

if  $W_{1,i,j} \in [-0.01, 0.01]$ , slope can’t be bigger than  $0.01 \times D$

$$\text{2-layer: } f_\theta(x) = W_2 \text{ReLU}(W_1 x + b_1) + b_2 \quad \theta = \{W_1, b_1, W_2, b_2\}$$

if  $W_{\ell,i,j} \in [-0.01, 0.01]$ , slope is bounded (but greater than 0.01!)

# Wasserstein GAN (WGAN)

**A better metric:** consider how far apart (in Euclidean space) all the “bits” of probability are

**More precisely:** optimal transport (“Earth mover’s distance”) – how far do you have to go to “transport” one distribution into another

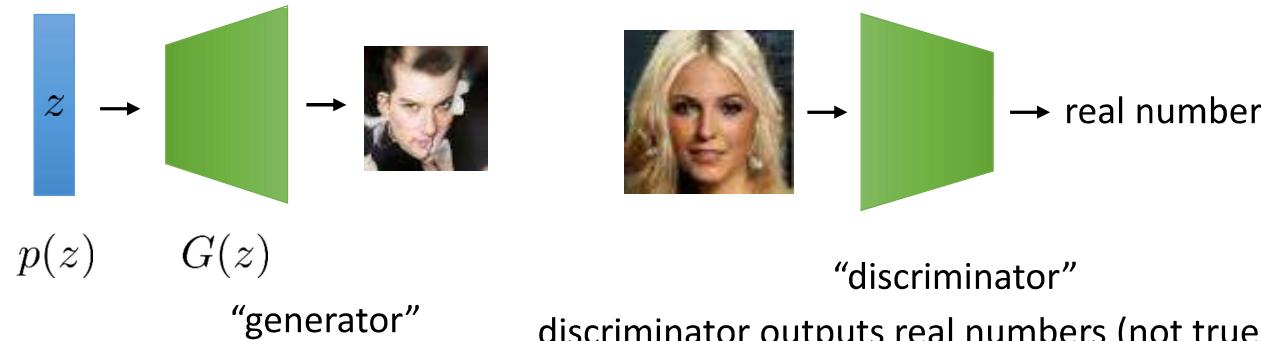
$$W(p_{\text{data}}, p_G) = \sup_{\|f\|_L \leq 1} E_{p_{\text{data}}}[f(x)] - E_{p_G(x)}[f(x)]$$

Weight clipping is a clearly terrible way to enforce a Lipschitz constraint. If the clipping parameter is large, then it can take a long time for any weights to reach their limit, thereby making it harder to train the critic till optimality. If the clipping is small, this can easily lead to vanishing gradients when the number of layers is big, or batch normalization is not used (such as in RNNs). We experimented with simple variants (such as projecting the weights to a sphere) with little difference, and we stuck with weight clipping due to its simplicity and already good performance. However, we do leave the topic of enforcing Lipschitz constraints in a neural network setting for further investigation, and we actively encourage interested researchers to improve on this method.

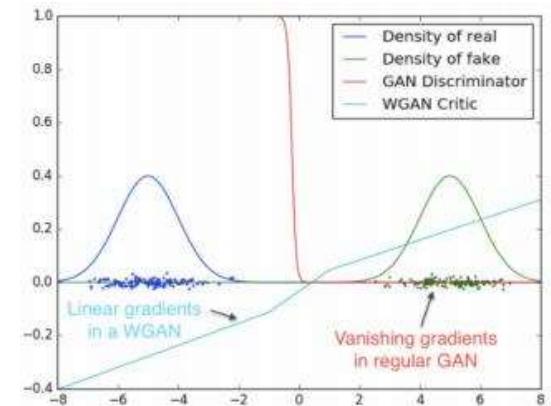
# Wasserstein GAN (WGAN)

$$W(p_{\text{data}}, p_G) = \sup_{\|f\|_L \leq 1} E_{p_{\text{data}}}[f(x)] - E_{p_G(x)}[f(x)]$$

1. update  $f_\theta$  using gradient of  $E_{x \sim p_{\text{data}}}[f_\theta(x)] - E_{z \sim p(z)}[f_\theta(G(z))]$
2. clip all weight matrices in  $\theta$  to  $[-c, c]$
3. update generator to *maximize*  $E_{z \sim p(z)}[f_\theta(G(z))]$



discriminator outputs real numbers (not true/false probability)  
e.g.,  $f_\theta(x) = W_2 \text{ReLU}(W_1 x + b_1) + b_2$   
discriminator uses *weight clipping*



# Better discriminator regularization

Weight clipping is a clearly terrible way to enforce a Lipschitz constraint.

**Gradient penalty:** Want bounded slope? We'll give you bounded slope!

1-Lipschitz:  
 $|f(x) - f(y)| \leq |x - y|$

update  $f_\theta$  using gradient of

$$E_{x \sim p_{\text{data}}}[f_\theta(x) - \underbrace{\lambda(||\nabla_x f_\theta(x)||_2 - 1)^2}_{\text{make norm of gradient close to 1}}] - E_{z \sim p(z)}[f_\theta(G(z))]$$

# Spectral norm

**Idea:** bound the Lipschitz constant in terms of singular values of each  $W_\ell$

$$f(x) = f_3 \circ f_2 \circ f_1(x)$$

neural net layers (e.g., linear, conv, ReLU, etc.)

$$\|f(x)\|_{\text{Lip}} = \|f_3 \circ f_2 \circ f_1\|_{\text{Lip}} \leq \|f_3\|_{\text{Lip}} \cdot \|f_2\|_{\text{Lip}} \cdot \|f_1\|_{\text{Lip}}$$

Lipschitz constant                                  to get intuition for why this is true, imagine these are **linear** functions

$\text{ReLU}(x) = \max(0, x) \Rightarrow$  max slope is 1!    that's easy, how about linear layers?

max slope of  $Wx + b$  is **spectral norm**:

$$\sigma(W) = \max_{h:h \neq 0} \frac{\|Wh\|}{\|h\|} = \max_{\|h\| \leq 1} \|Wh\|$$

largest singular value of  $W$

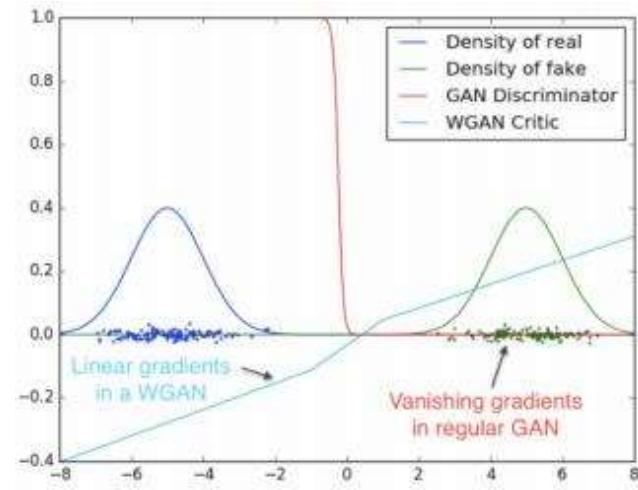
**Method:** after each grad step, force  $W_\ell \leftarrow \frac{W_\ell}{\sigma(W_\ell)}$

See paper for how to  
implement this efficiently

for details, see: Miyato et al. **Spectral Normalization for Generative Adversarial Networks**. 2018.

# GAN training summary

- GAN training is really hard, because the discriminator can provide poor gradients
- Various “tricks” can make this much more practical
  - “Smooth” real-valued discriminators: LSGAN, WGAN, WGAN-GP, spectral norm
  - Instance noise
- The theory behind these tricks is quite complex, but the methods in practice are very simple
- Such GANs are **much** easier to train



# Introduction to Wasserstein GANs (WGANs)

Wasserstein GANs (WGANs) utilize the Wasserstein distance to measure the difference between real and generated data distributions.

- ▶ Improved stability and convergence in training.
- ▶ Addresses issues like mode collapse and vanishing gradients.

# Wasserstein Distance

The Wasserstein distance  $W_p(P, Q)$  between two probability distributions  $P$  and  $Q$  is defined as:

$$W_p(P, Q) = \left( \inf_{\gamma \in \Gamma(P, Q)} \int_{X \times X} d(x, y)^p d\gamma(x, y) \right)^{1/p}$$

where:

- ▶  $\Gamma(P, Q)$  is the set of joint distributions with marginals  $P$  and  $Q$ .
- ▶  $d(x, y)$  is the distance function.
- ▶  $p$  is the order of the Wasserstein distance.

# Kantorovich-Rubinstein Duality

For  $p = 1$ , the Kantorovich-Rubinstein duality provides:

$$W_1(P, Q) = \sup_{\|\phi\|_L \leq 1} \left( \int_X \phi(x) dP(x) - \int_X \phi(x) dQ(x) \right)$$

where:

- ▶  $\phi$  ranges over 1-Lipschitz functions.
- ▶  $\|\phi\|_L$  is the Lipschitz constant of  $\phi$ , which must be  $\leq 1$ .

# Lipschitz Functions

A function  $\phi : X \rightarrow \mathbb{R}$  is Lipschitz with constant  $K$  if:

$$|\phi(x) - \phi(y)| \leq K \cdot d(x, y)$$

For 1-Lipschitz functions,  $K = 1$ :

$$|\phi(x) - \phi(y)| \leq d(x, y)$$

# Example of Lipschitz Function

Consider  $\phi(x) = x$  on the real line:

$$|\phi(x) - \phi(y)| = |x - y|$$

This function is 1-Lipschitz because:

$$|\phi(x) - \phi(y)| \leq d(x, y)$$

# WGAN Loss Function

**\*\*Discriminator Loss:\*\***

$$L_D = \mathbb{E}_{x \sim P_{\text{data}}} [D(x)] - \mathbb{E}_{z \sim P_z} [D(G(z))]$$

**\*\*Generator Loss:\*\***

$$L_G = -\mathbb{E}_{z \sim P_z} [D(G(z))]$$

- ▶  $D(x)$  is the discriminator's score for real data  $x$ .
- ▶  $D(G(z))$  is the score for generated data  $G(z)$ .

# Example of WGAN with Lipschitz Constraints

Let  $P_{\text{data}} \sim \mathcal{N}(0, 1)$  and  $P_{\text{gen}} \sim \mathcal{N}(\mu, \sigma^2)$ :

$$W_1(P_{\text{data}}, P_{\text{gen}}) = |\mu| + |\sigma - 1|$$

**\*\*Training the WGAN:\*\***

- ▶ **\*\*Discriminator\*\*:** Maximize  
 $\mathbb{E}_{x \sim P_{\text{data}}} [D(x)] - \mathbb{E}_{z \sim P_z} [D(G(z))].$
- ▶ **\*\*Generator\*\*:** Minimize  $\mathbb{E}_{z \sim P_z} [D(G(z))]$ .

# Conclusion

Wasserstein GANs use the Wasserstein distance to enhance GAN training stability. The Kantorovich-Rubinstein duality simplifies Wasserstein-1 distance calculation using 1-Lipschitz functions.

- ▶ Provides theoretical and practical advantages.
- ▶ Useful in various applications including generative models.

# References

- ▶ Arjovsky, M., Chintala, S., Bottou, L. (2017). Wasserstein GAN. \*Proceedings of the 34th International Conference on Machine Learning (ICML)\*.
- ▶ Kantorovich, L. V., Rubinstein, G. (1958). On a problem of optimal allocation of resources. \*Econometrica\*.