Generative adversarial networks

Biplab Banerjee

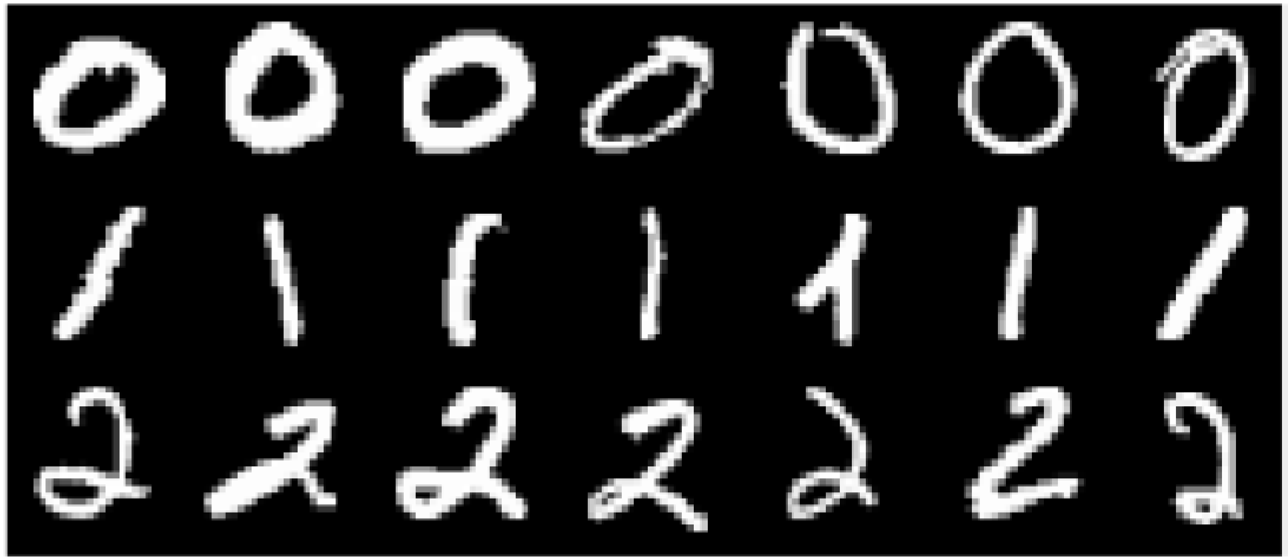
- We've only seen discriminative models so far
  - Given an image **X**, predict a label **Y**
  - Estimates **P(Y|X)**

- 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

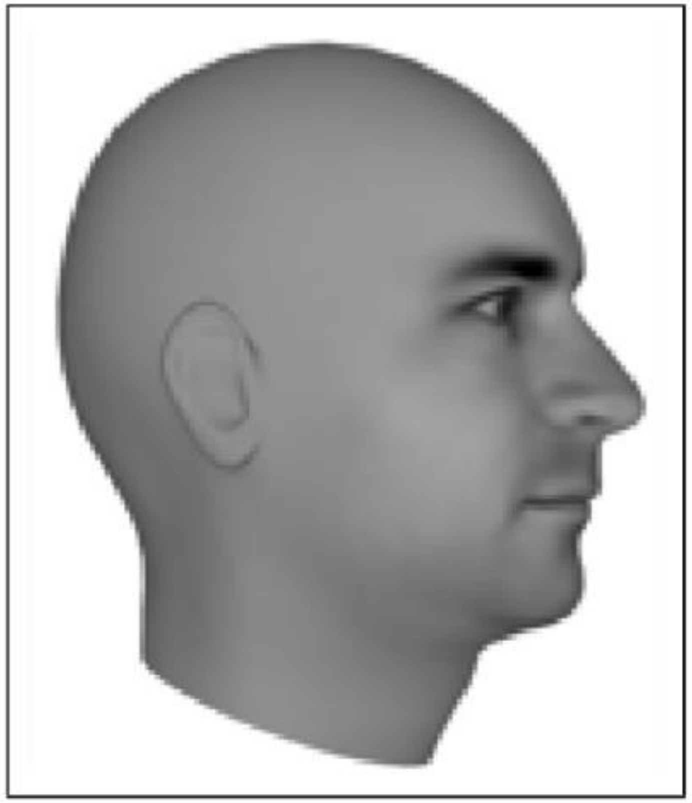
Implicit density estimation problem

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

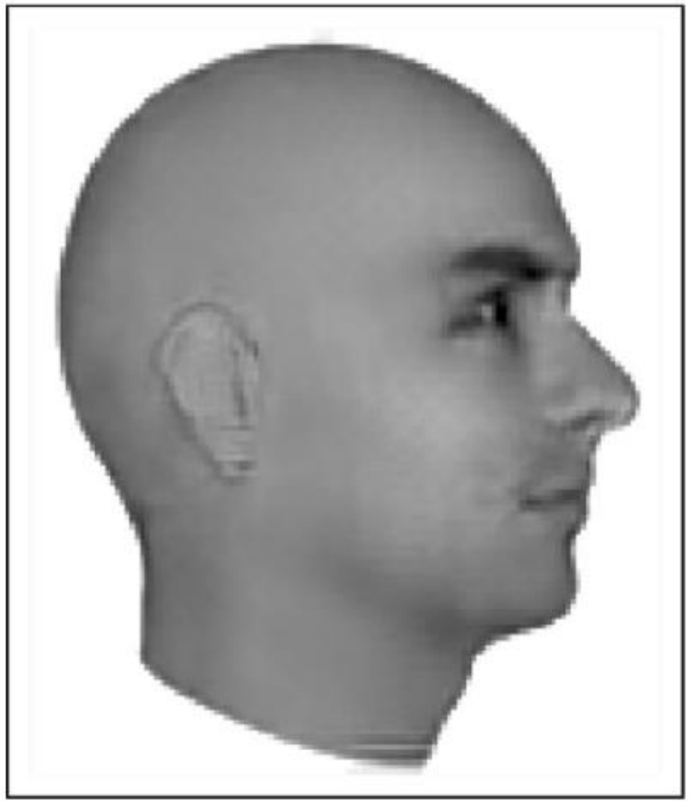


Magic of GAN

Ground Truth



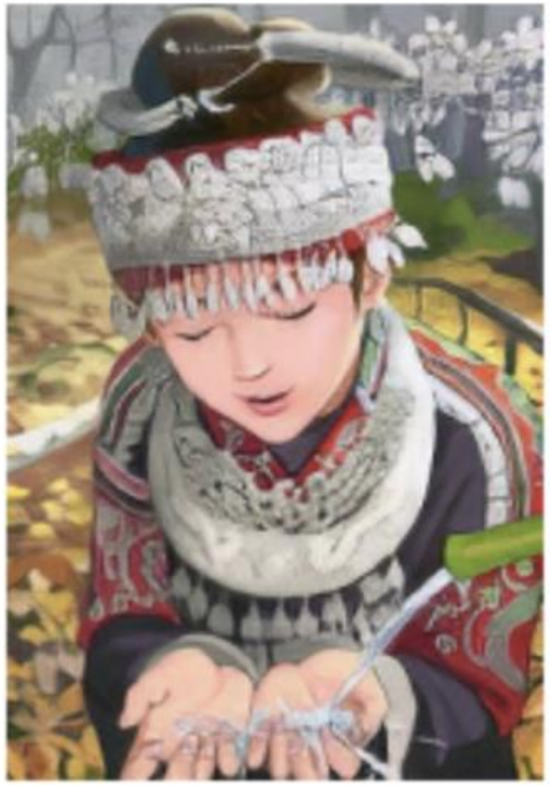
Adversarial

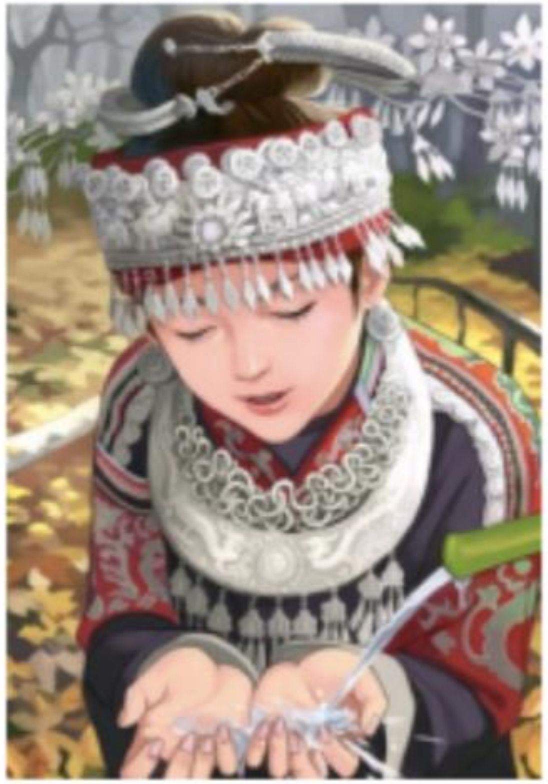


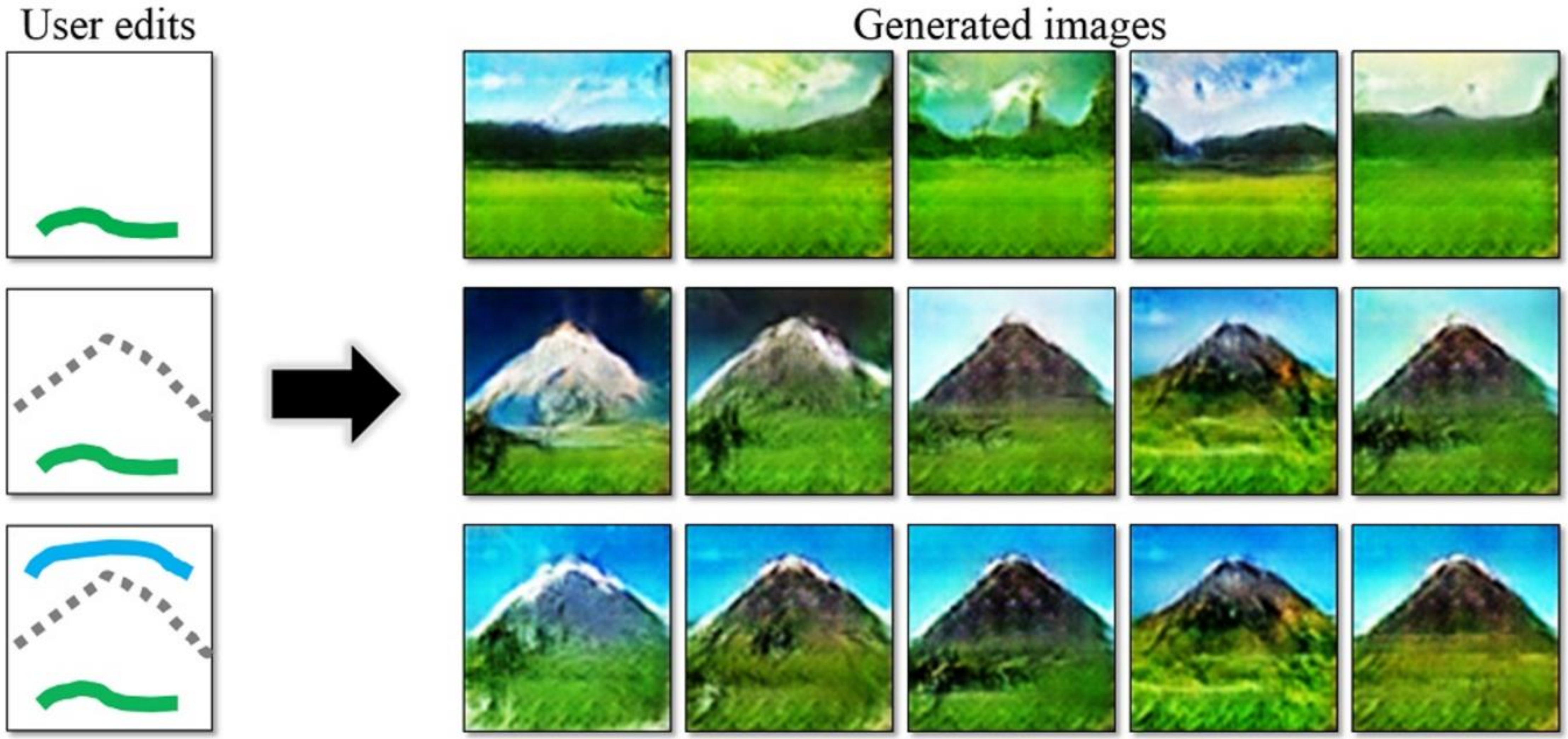
Lotter, William, Gabriel Kreiman, and David Cox. "Unsupervised learning of visual structure using predictive generative networks." *arXiv preprint arXiv:1511.06380* (2015).

Magic of GAN

Which one is Computer generated?

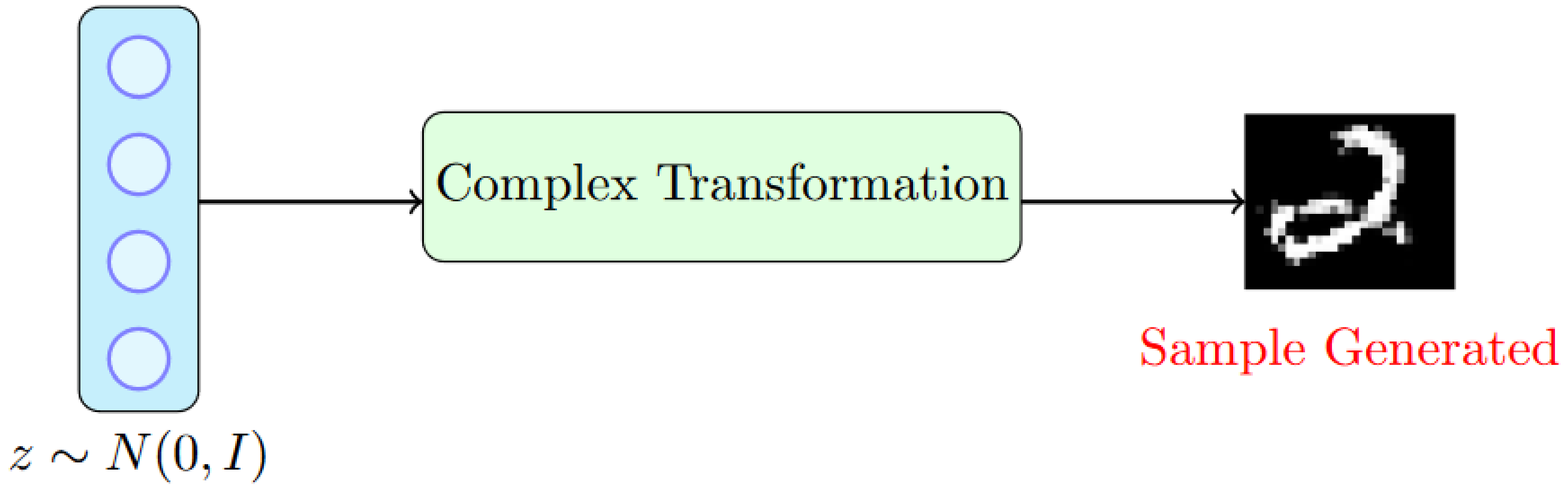


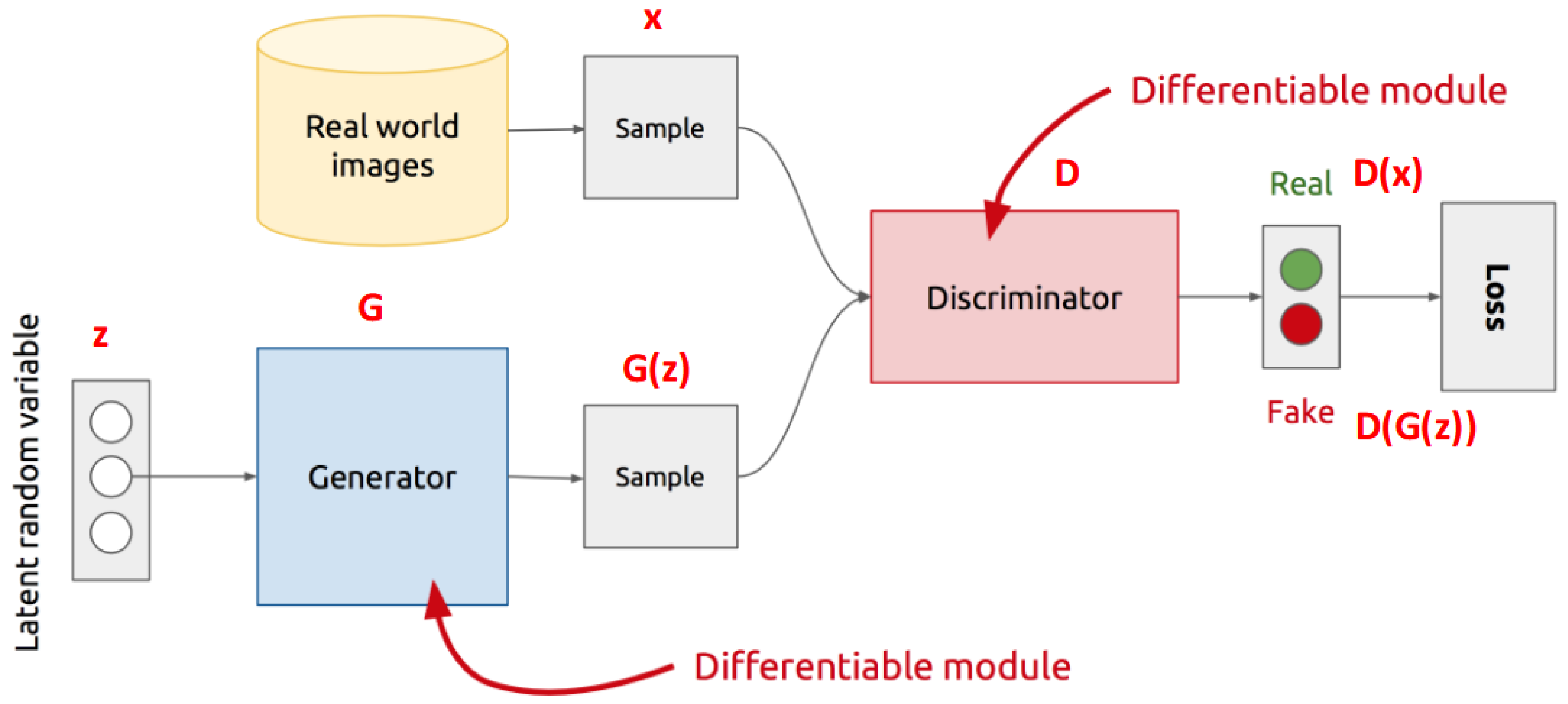




http://people.eecs.berkeley.edu/~junyanz/projects/gvm/

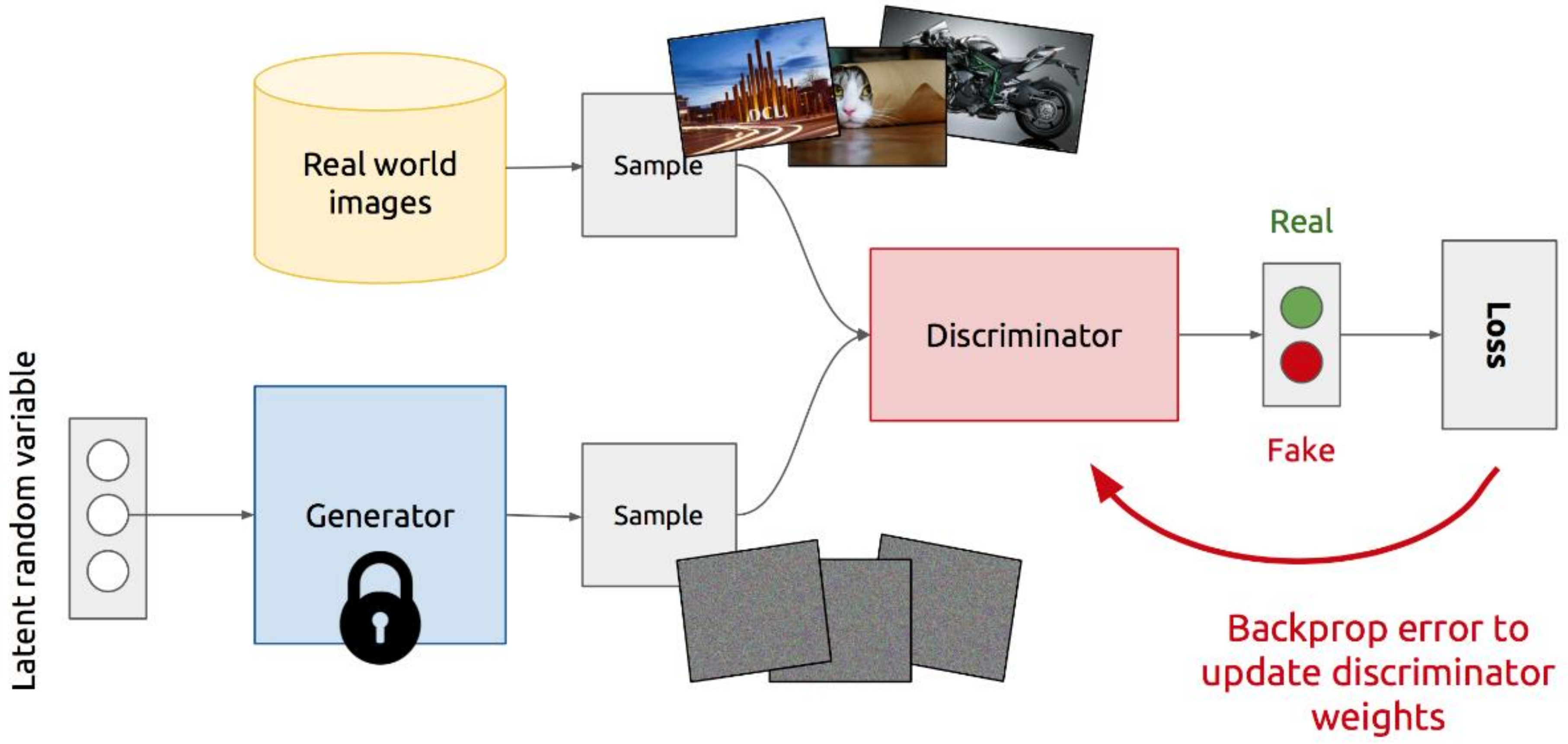
What about



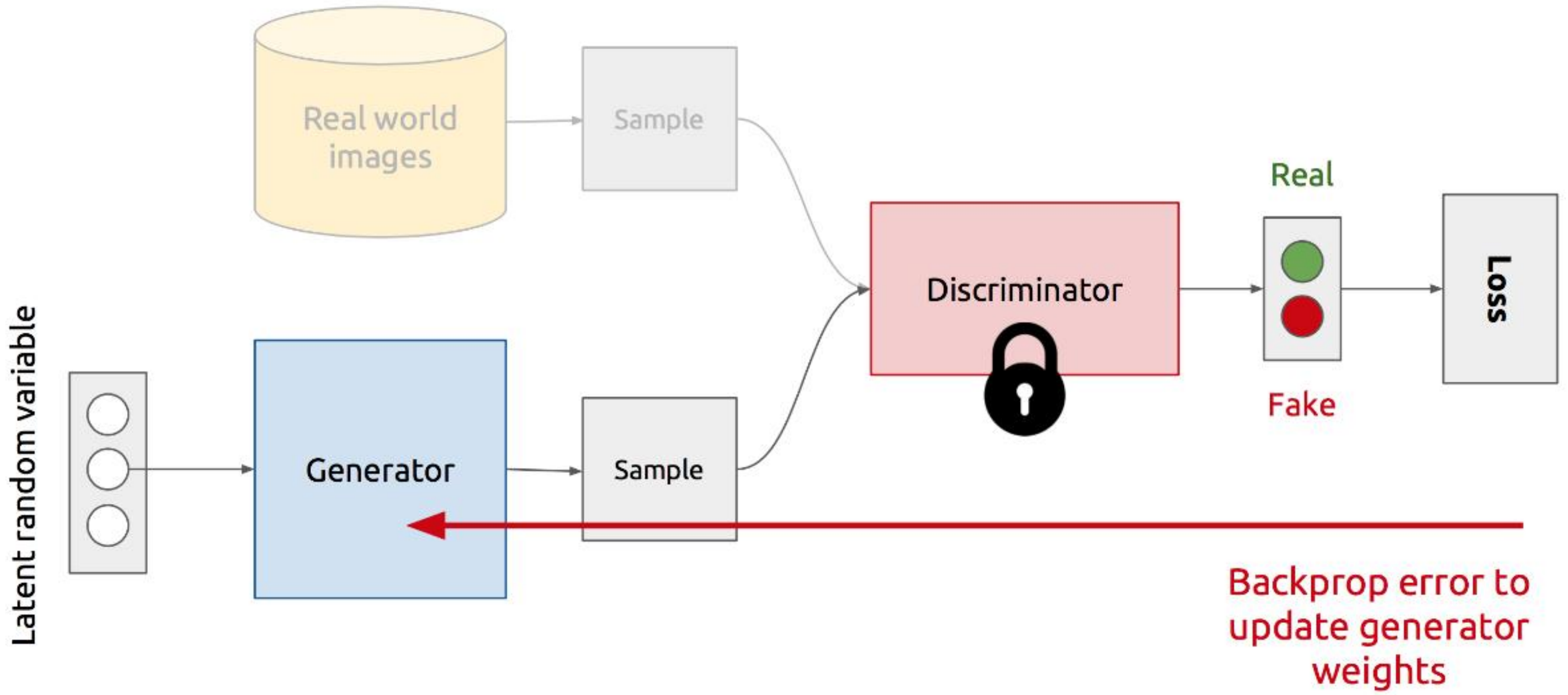


- **Z** is some random noise (Gaussian/Uniform).
- **Z** can be thought as the latent representation of the image.

Training discriminator



Training generator



Let G_ϕ be the generator and D_θ be the discriminator (ϕ and θ 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

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

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

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

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

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

- $\nabla_a \log(1 - \sigma(a)) = \frac{-\nabla_a \sigma(a)}{1 - \sigma(a)} = \frac{-\sigma(a)\left(1 - \sigma(a)\right)}{1 - \sigma(a)} = -\sigma(a) = -D(G(z))$

- Gradient goes to 0 if D is confident, i.e. $D(G(z)) \to 0$

- Minimize $\boxed{-\mathbb{E}_{z \sim q(z)}[\log D(G(z))]}$ for **Generator** instead (keep Discriminator as it is)

Optimal D

- The value function

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

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

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

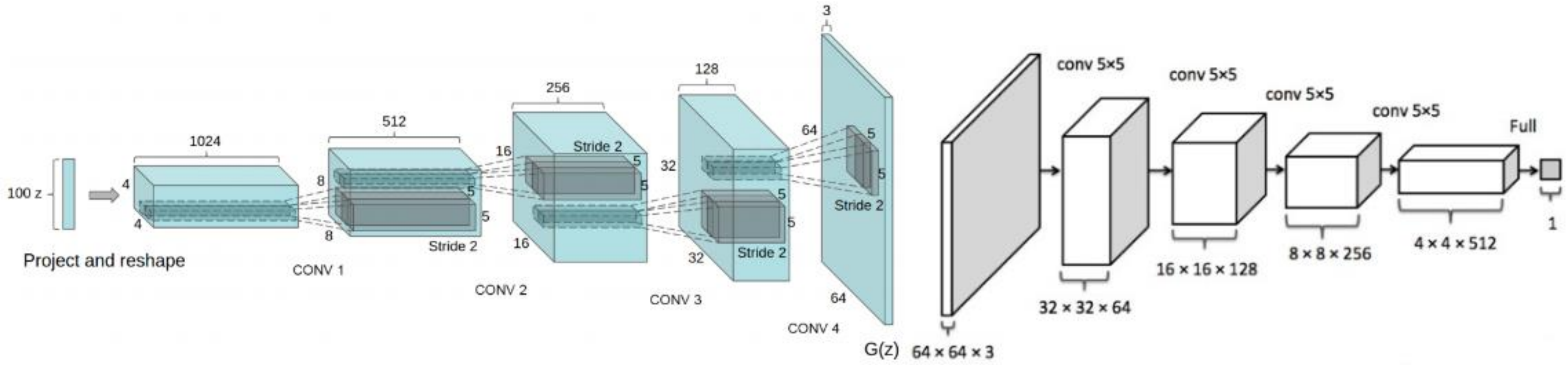
Global optima

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

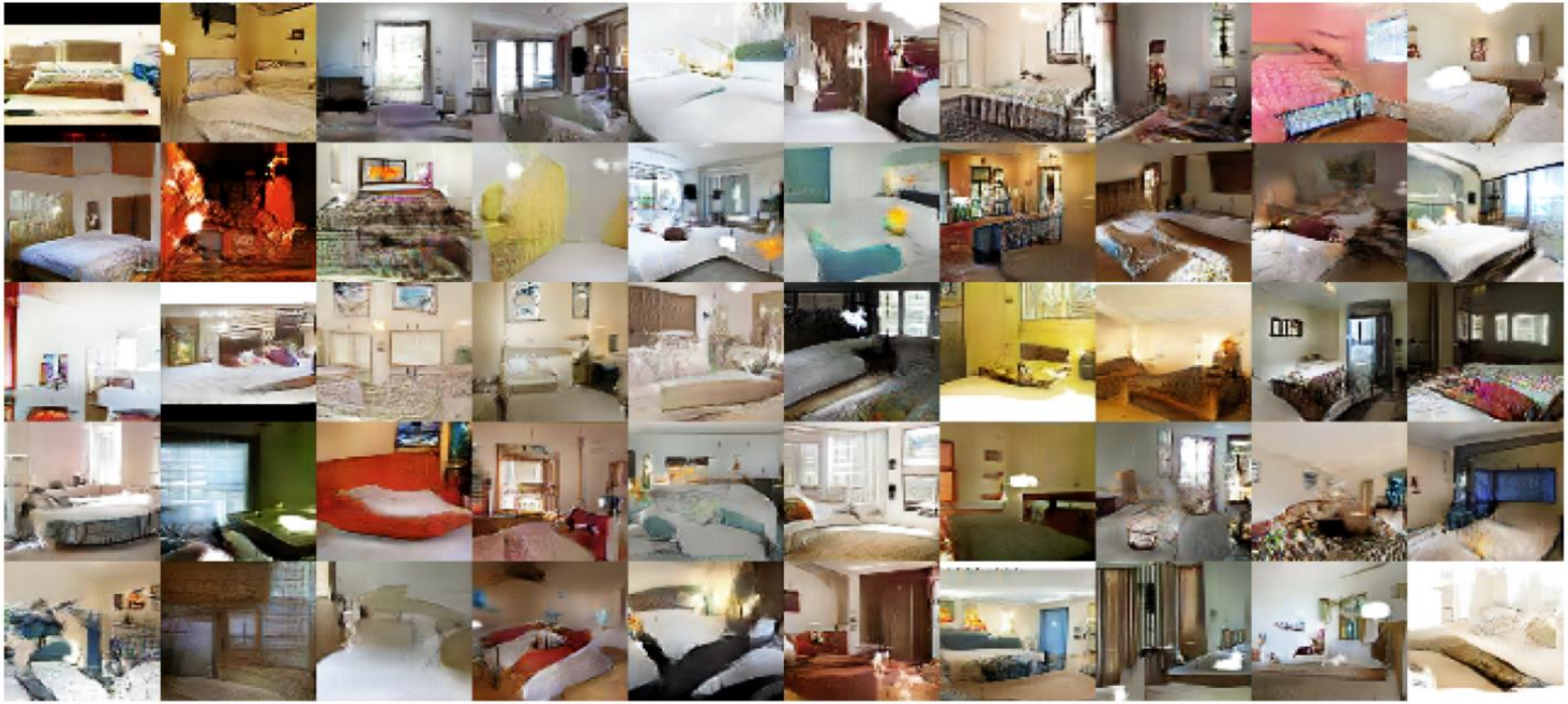
Which is equal to the JSD between real and fake distributions

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

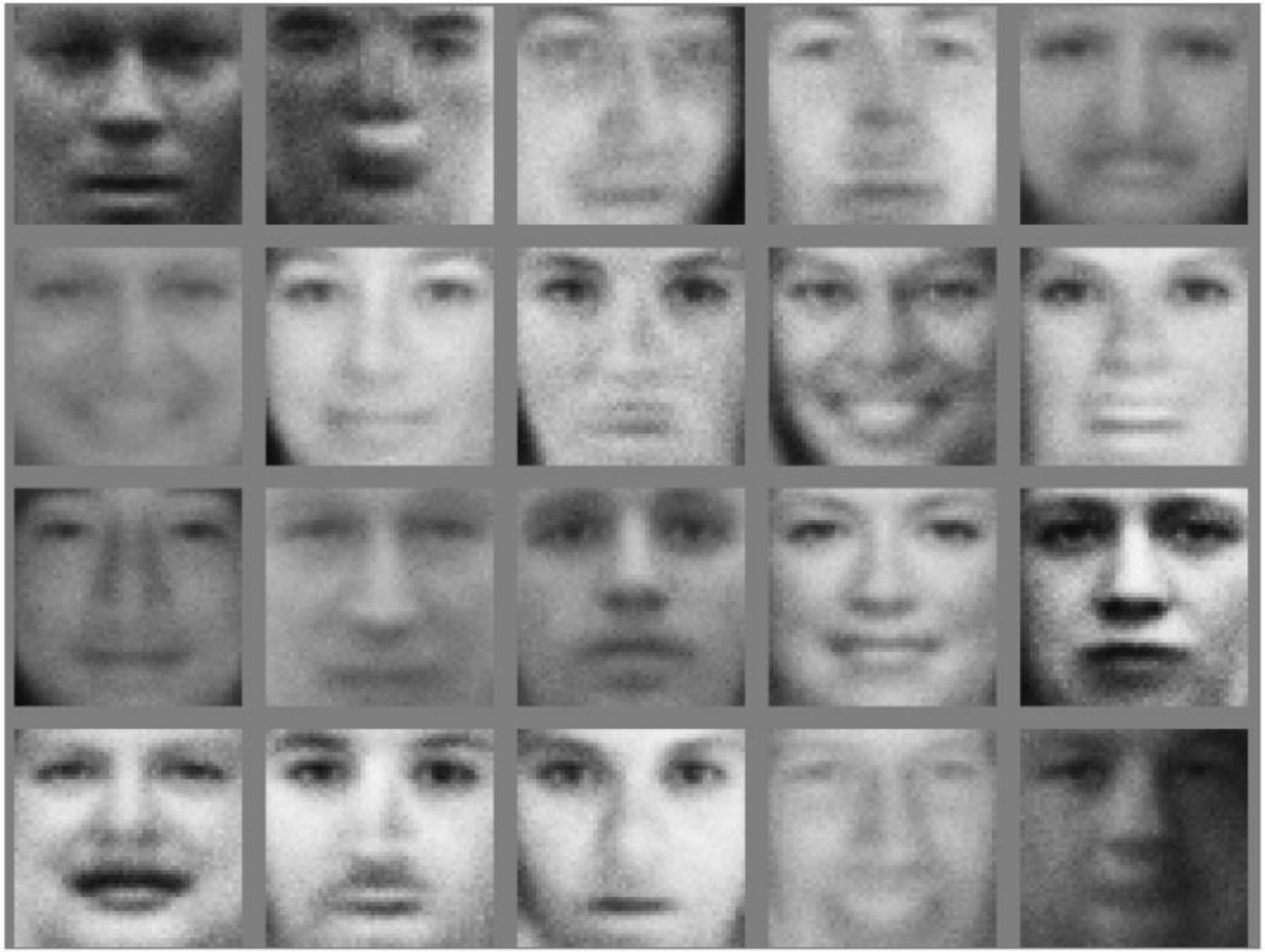
DC GAN



100 z
Project and reshape
1024
4
4
512
8
8
5
5
Stride 2
CONV 1
256
16
16
5
5
Stride 2
CONV 2
128
32
32
5
5
Stride 2
CONV 3
3
64
64
5
5
Stride 2
CONV 4
G(z)
64 × 64 × 3
conv 5×5
32 × 32 × 64
conv 5×5
16 × 16 × 128
conv 5×5
8 × 8 × 256
conv 5×5
4 × 4 × 512
Full
1

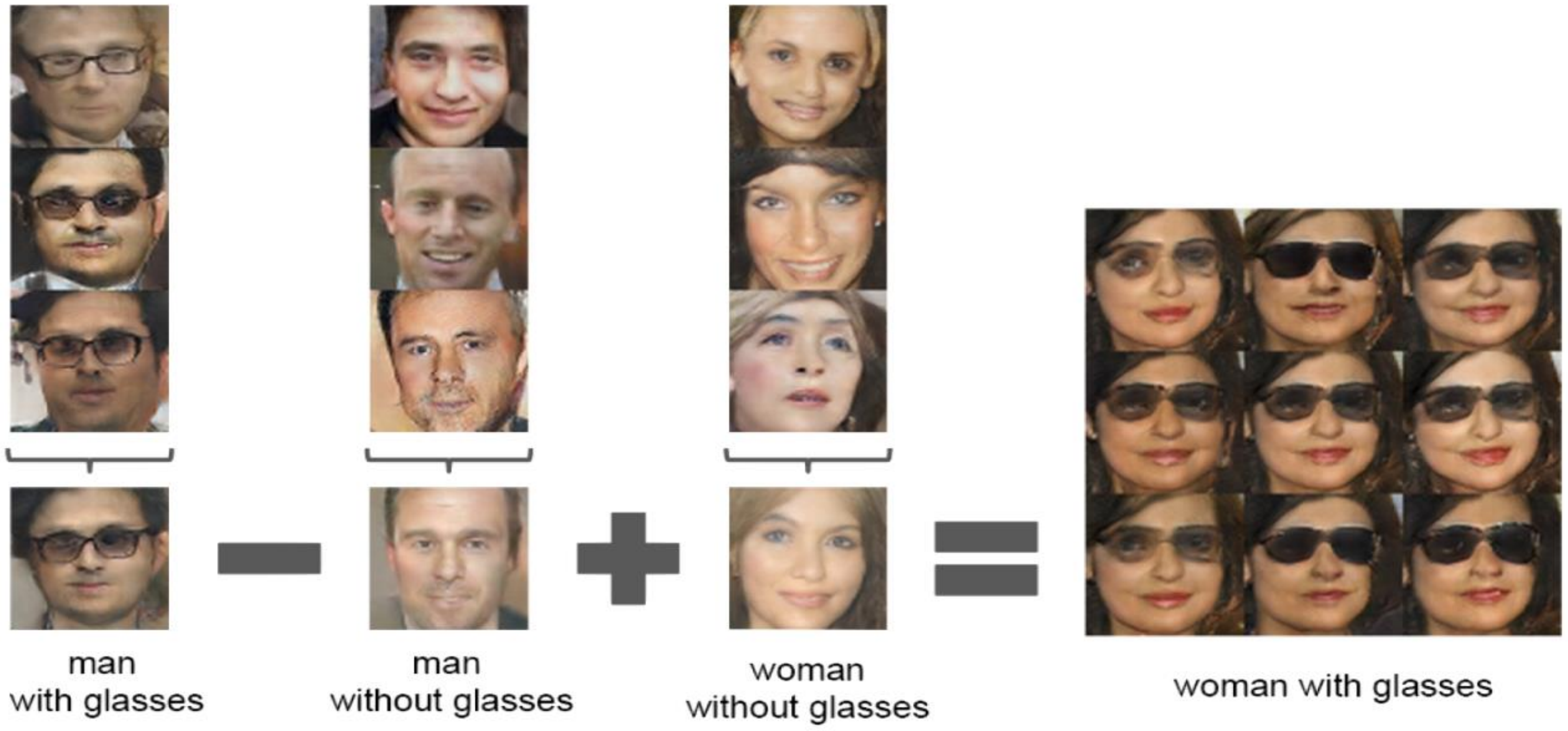


Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." arXiv:1511.06434 (2015).



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

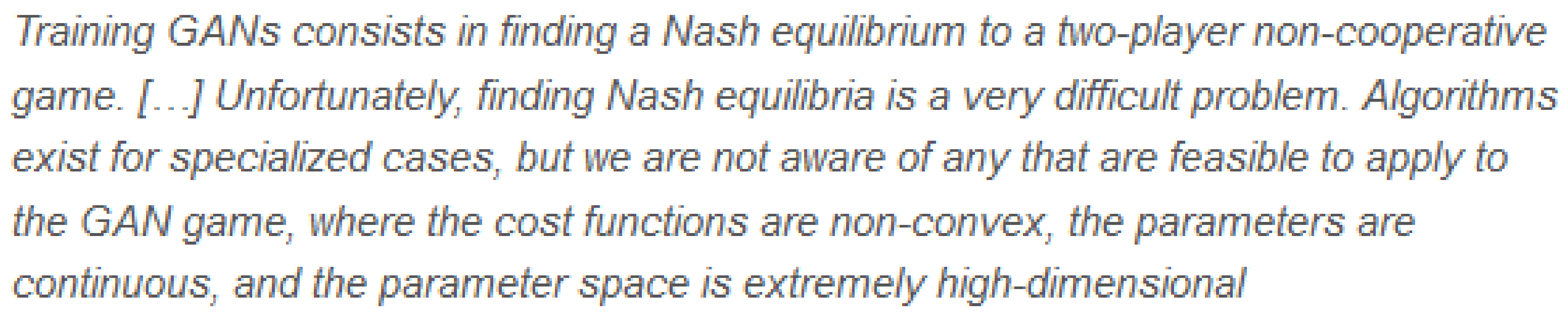
- Replace pooling by strided convolution and unpolling by fractional strided convolution
- Use batch-norm
- Avoid FC layers in deeper architectures
- Use ReLU in all the generator layers except output where

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.

$$\min_{G} L(G)$$

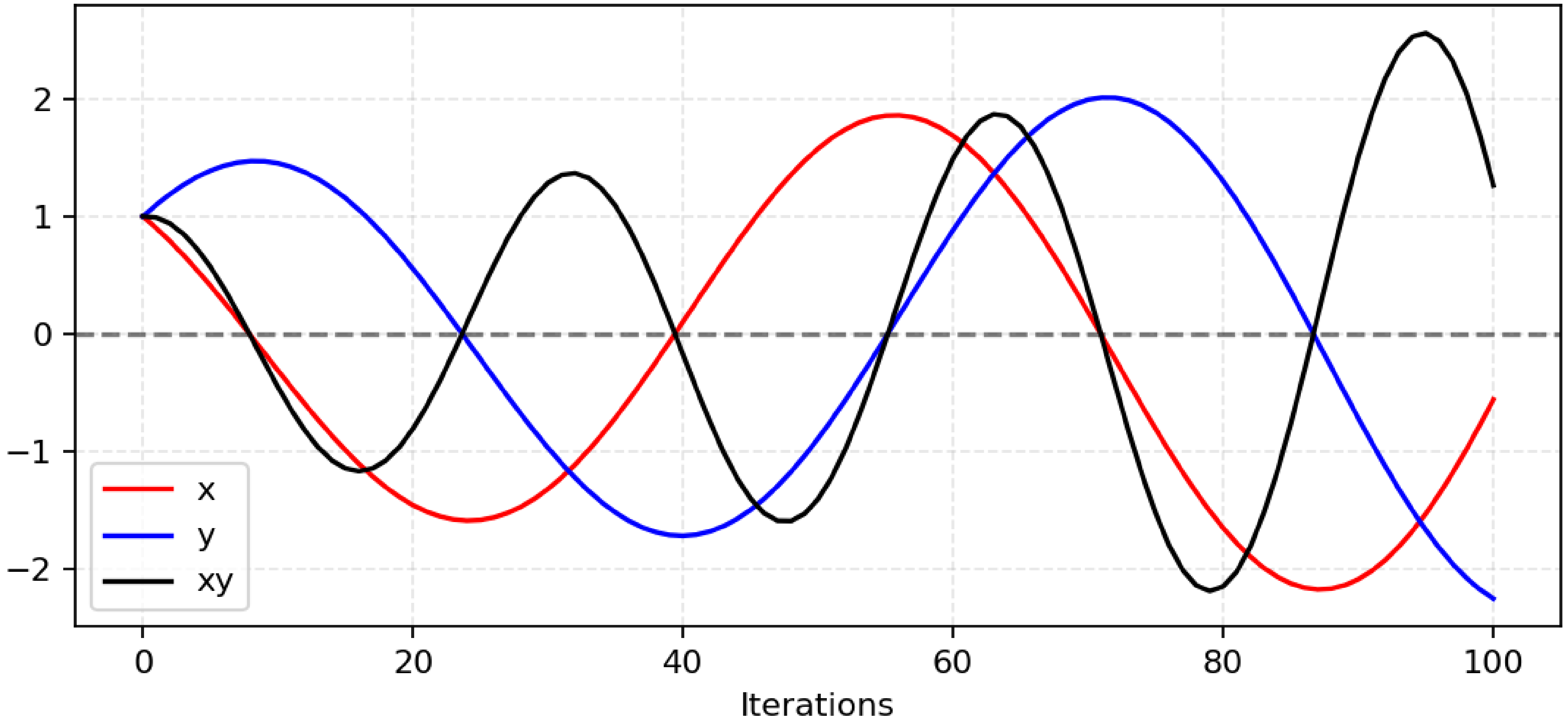
- **GANs instead involve two (or more) players**
  - Discriminator is trying to maximize its reward.
  - Generator is trying to minimize Discriminator's reward.

$$\min_{G} \max_{D} V(D, G)$$

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

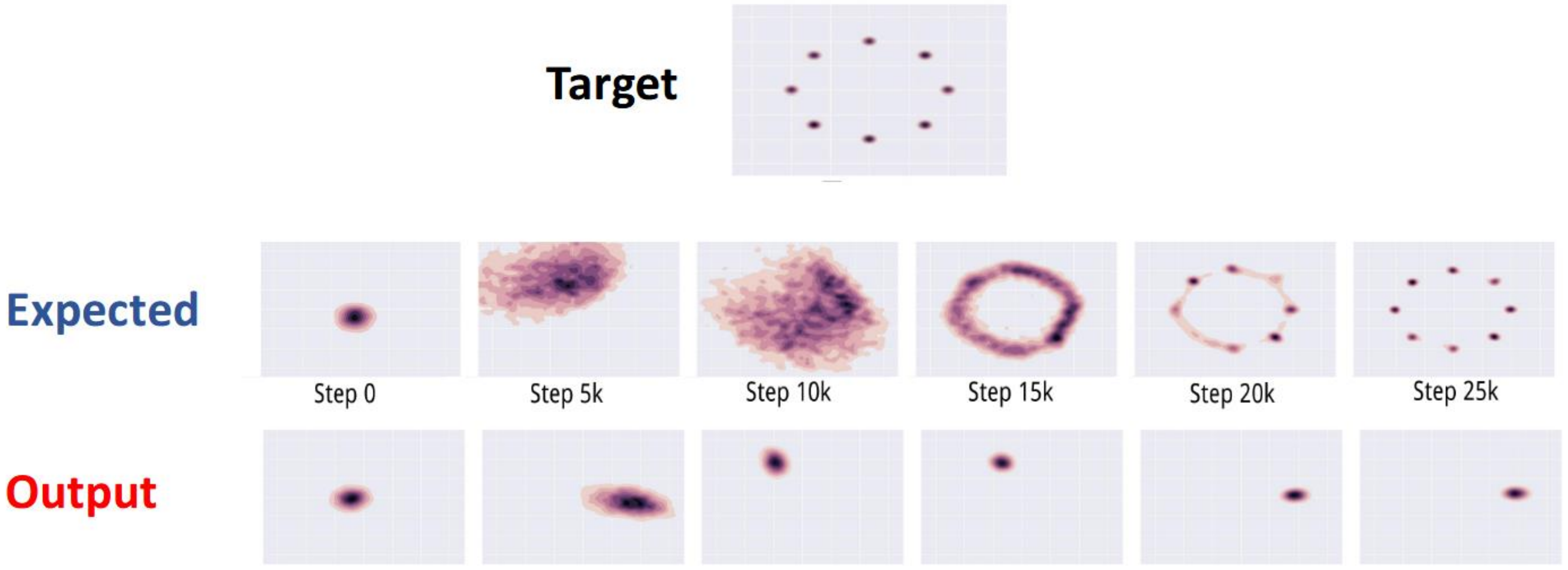
Problem in obtaining the equilibrium

$$\min_x \max_y \; xy$$

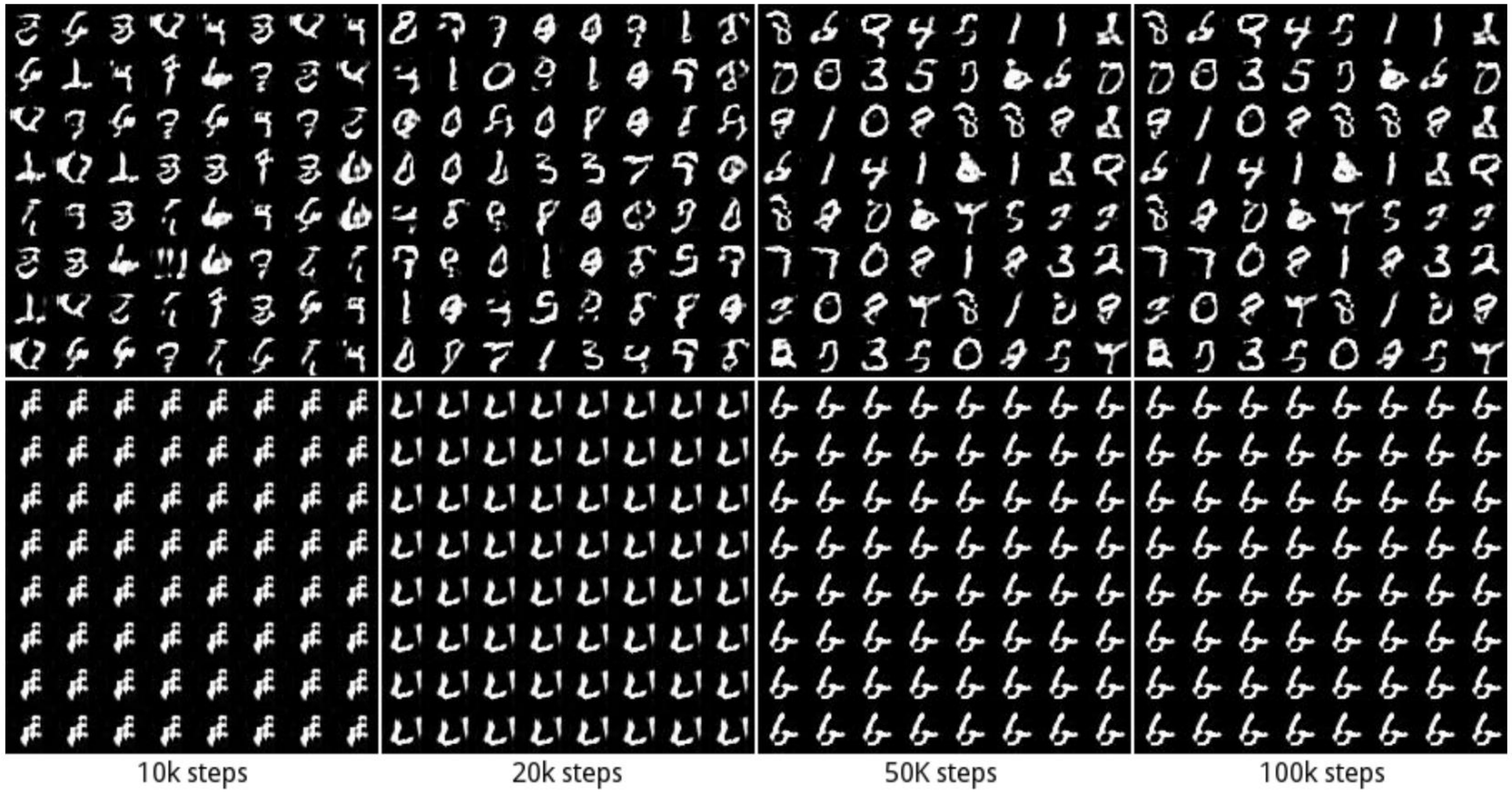


Mode collapse

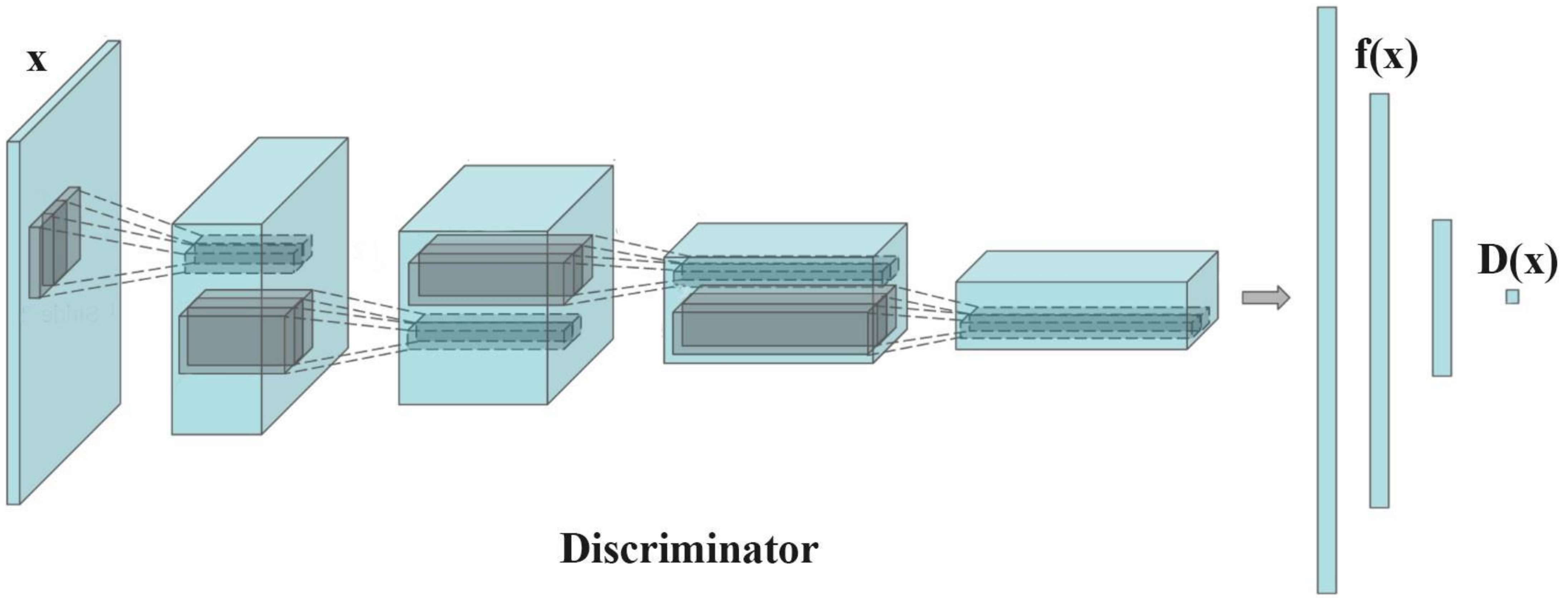
- **Generator fails to output diverse samples**



Mode collapse in MNIST



Feature matching



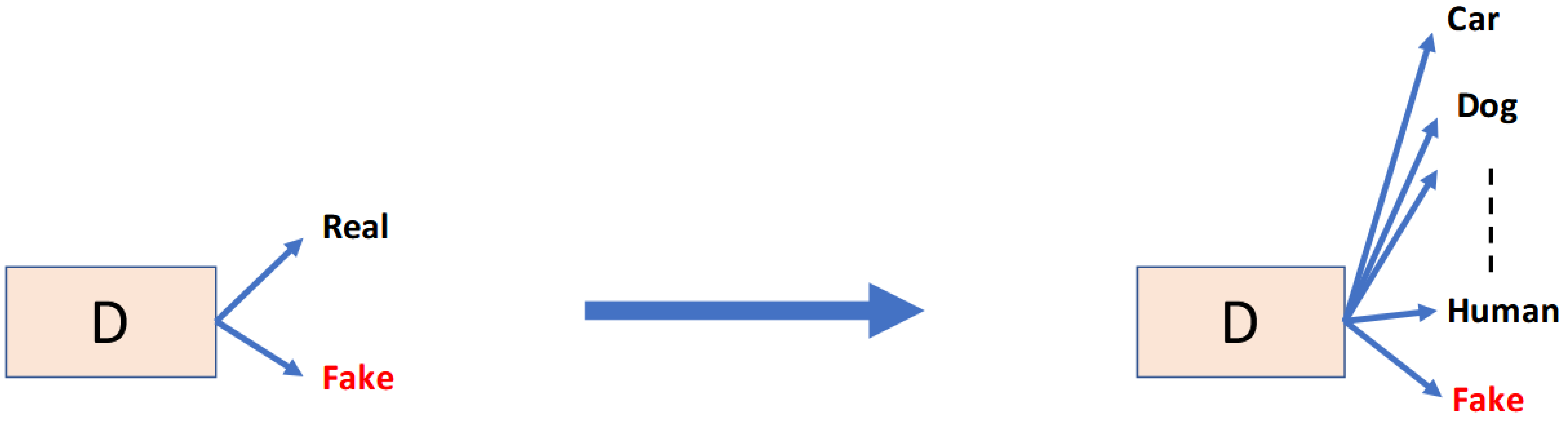
$$||\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\mathbf{f}(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\mathbf{f}(G(\boldsymbol{z}))||_2^2$$

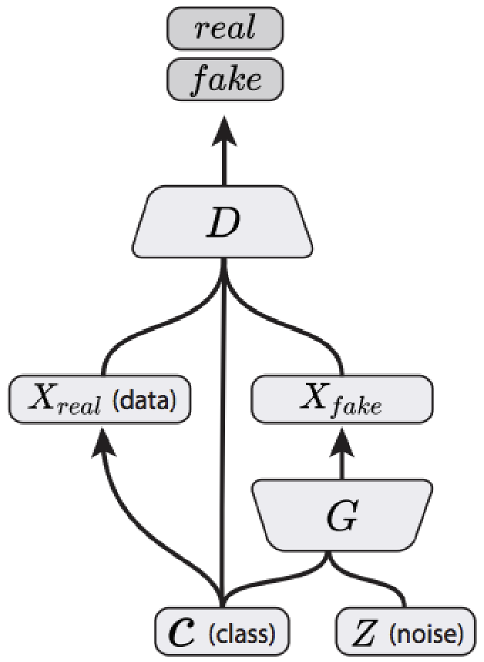
- **Extract features that capture diversity in the mini-batch**
  - For e.g. L2 norm of the difference between all pairs from the batch
- **Feed those features to the discriminator along with the image**
- **Feature values will differ b/w diverse and non-diverse batches**
  - Thus, Discriminator will 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

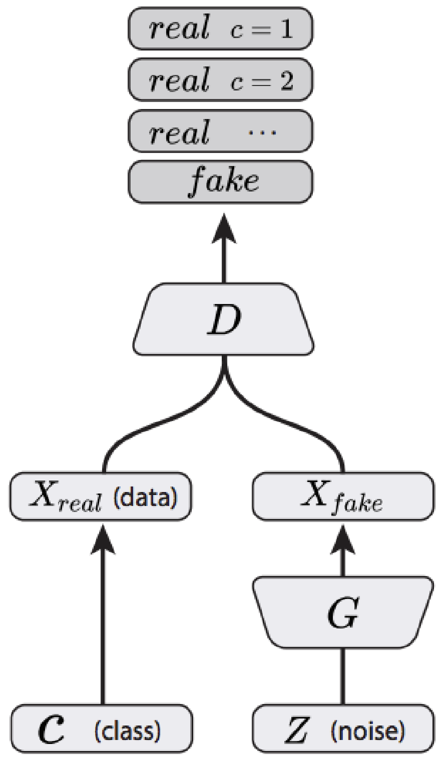
- **At Mode Collapse,**
  - Generator produces good samples, but a very few of them.
  - Thus, Discriminator can't tag them as fake.
- **To address this problem,**
  - Let the Discriminator know about this edge-case.
- **More formally,**
  - Let the Discriminator look at the entire batch instead of single 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

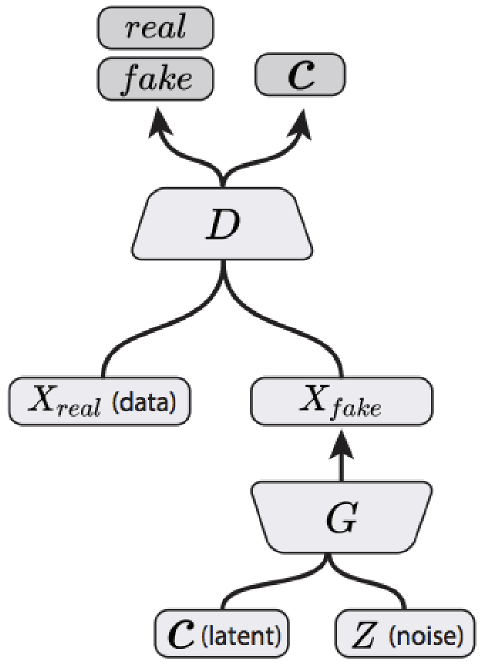




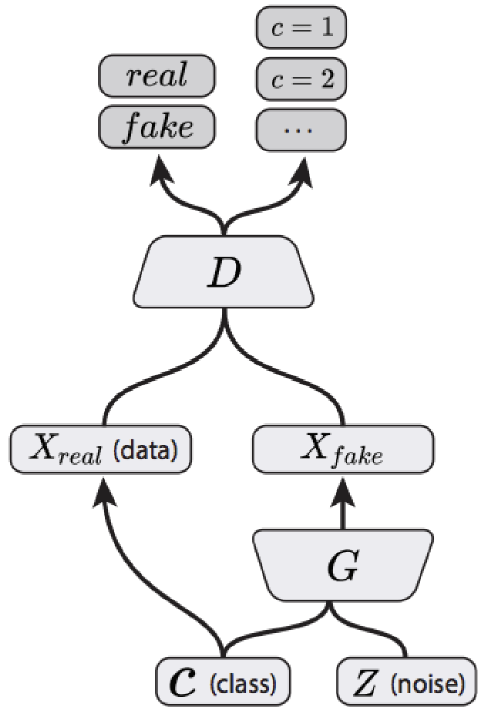
Conditional GAN
(Mirza & Osindero, 2014)



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



InfoGAN
(Chen, et al., 2016)



AC-GAN
(Present Work)

An alternate view of GAN

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

$$\boxed{D^* = \underset{D}{argmax}\, V(D,G)} \quad \boxed{G^* = \underset{G}{argmin}\, V(D,G)}$$

- In this formulation, Discriminator's strategy was $D(x) \to 1,\ D(G(z)) \to 0$

- Alternatively, we can flip the binary classification labels i.e. **Fake = 1**, **Real = 0**

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

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

We can use this $\boxed{D^* = argmin_D \, \mathbb{E}_{x\sim p(x)} \log\big(D(x)\big) + \mathbb{E}_{z\sim q(z)} \left[\log\left(1 - D\big(G(z)\big)\right)\right]}$

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

$$\boxed{D^* = argmin_D \, \mathbb{E}_{x\sim p(x)} D(x) + \mathbb{E}_{z\sim q(z)} \max\left(0, m - D\big(G(z)\big)\right)}$$

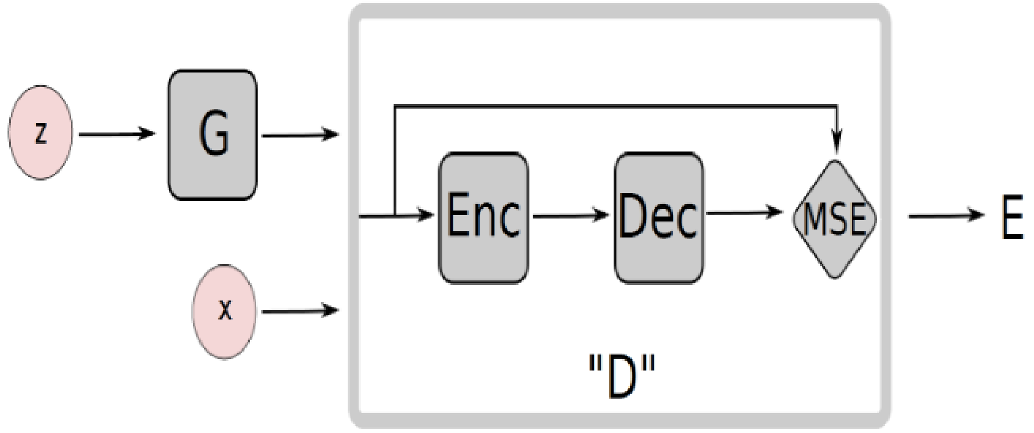
- And thus, instead of outputting probabilities, Discriminator just has to output :-
  - High values for fake samples
  - Low values for real samples

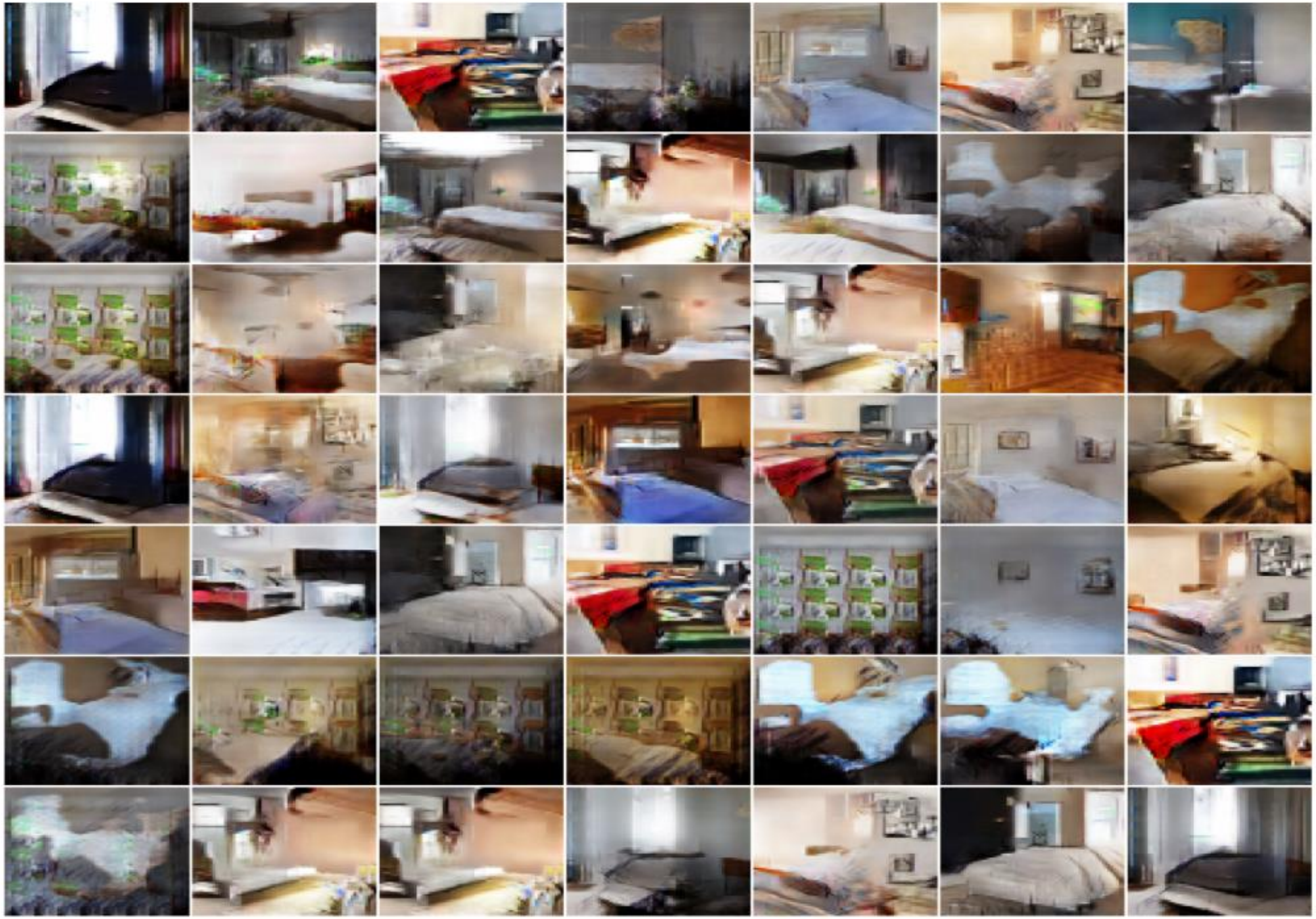
Energy based GAN

- Modified game plans
  - **Generator** will try to generate samples with low values
  - **Discriminator** will try to assign high scores to fake values

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

- **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



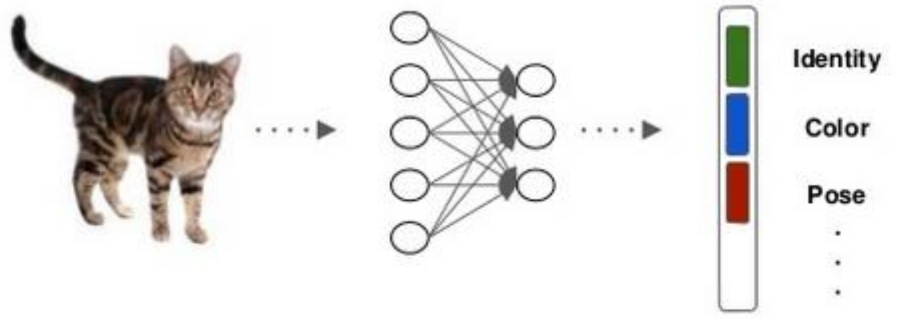


One issue so far

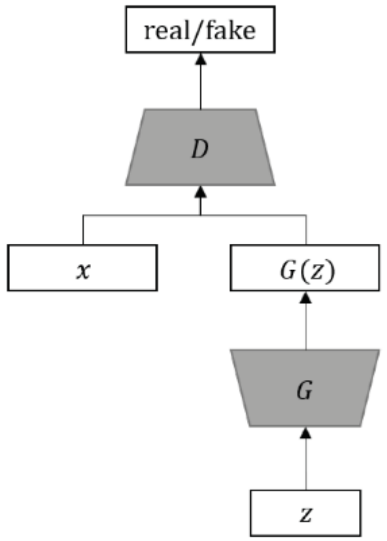
- How can we ensure the generation of images with a specific characteristics?
- Since the input to G is noise, we do not have explicit control over which images are generated.
- C-GAN solves to some extent by conditioning on the class labels
- A learning based unsupervised way is called INFOGAN
- It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

Disentangled representations

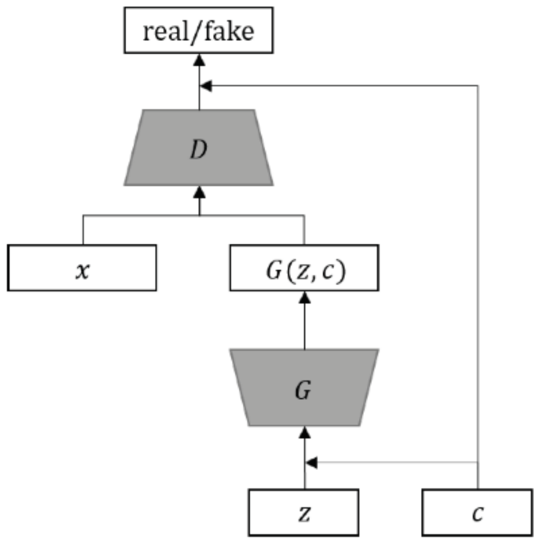
The ability to understand high-dimensional data, and to distill that knowledge into useful representations in an unsupervised manner, remains a key challenge in deep learning. One approach to solving these challenges is through *disentangled representations*, models that capture the independent features 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.



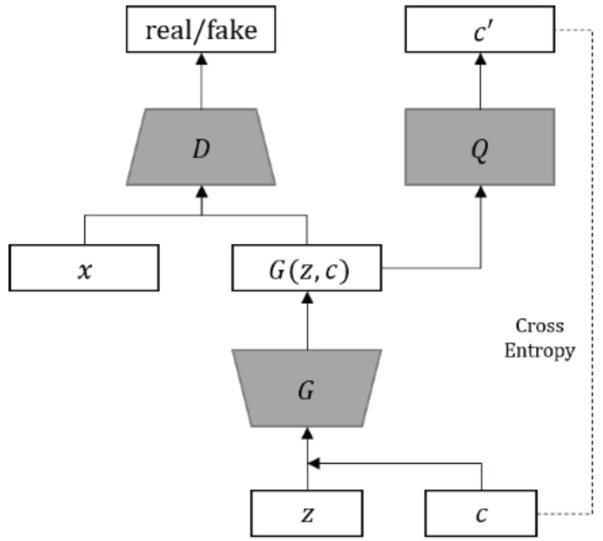
The evolution so far



(**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:**

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

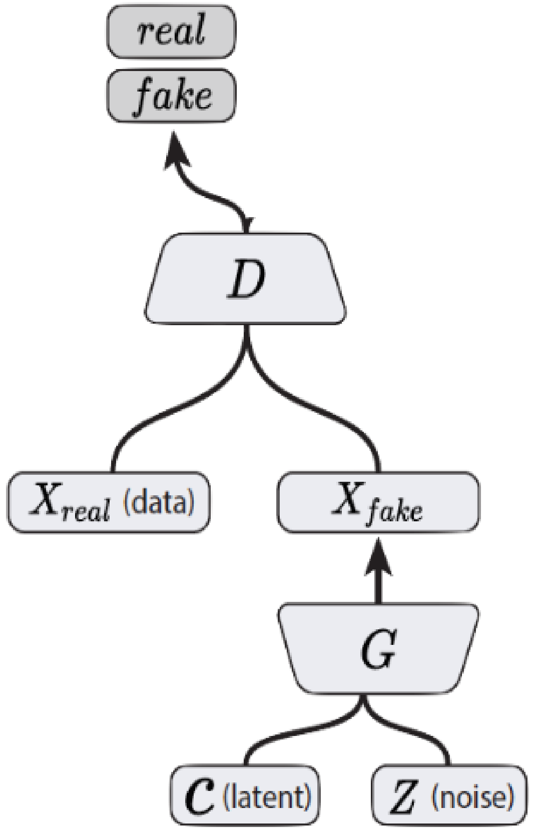
$$I(X;Y) = H(X) - H(X|Y) = \mathrm{H(Y)} - \mathrm{H(Y|X)}$$

InfoGAN

- We want to maximize the mutual information I between $\boldsymbol{c}$ and $\mathbf{x} = \boldsymbol{G(z, c)}$

- Incorporate in the value function of the minimax game.

$$\min_{G} \max_{D} V_I(D, G) = V(D, G) \boxed{- \lambda I\big(c; G(z, c)\big)}$$

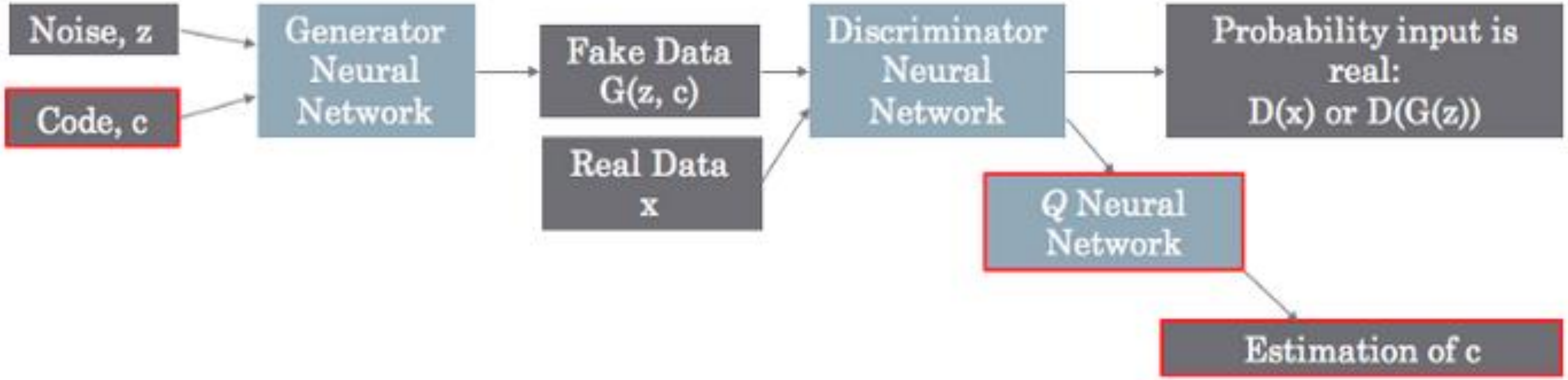


Mutual Information's Variational Lower bound

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

$$\geq \mathbb{E}_{c\sim P(c),\ x \sim G(z,c)}[\log Q(c|x)] + H(c)$$

InfoGAN


Noise, z
Code, c
Generator Neural Network
Fake Data G(z, c)
Real Data x
Discriminator Neural Network
Probability input is real: D(x) or D(G(z))
Q Neural Network
Estimation of c

Example (CGAN)

$[1, 0, 0, 0, 0, 0, 0, 0, 0, 0] \longrightarrow$

$[0, 1, 0, 0, 0, 0, 0, 0, 0, 0] \longrightarrow$

$[0, 0, 1, 0, 0, 0, 0, 0, 0, 0] \longrightarrow$

$[0, 0, 0, 1, 0, 0, 0, 0, 0, 0] \longrightarrow$

$[0, 0, 0, 0, 1, 0, 0, 0, 0, 0] \longrightarrow$

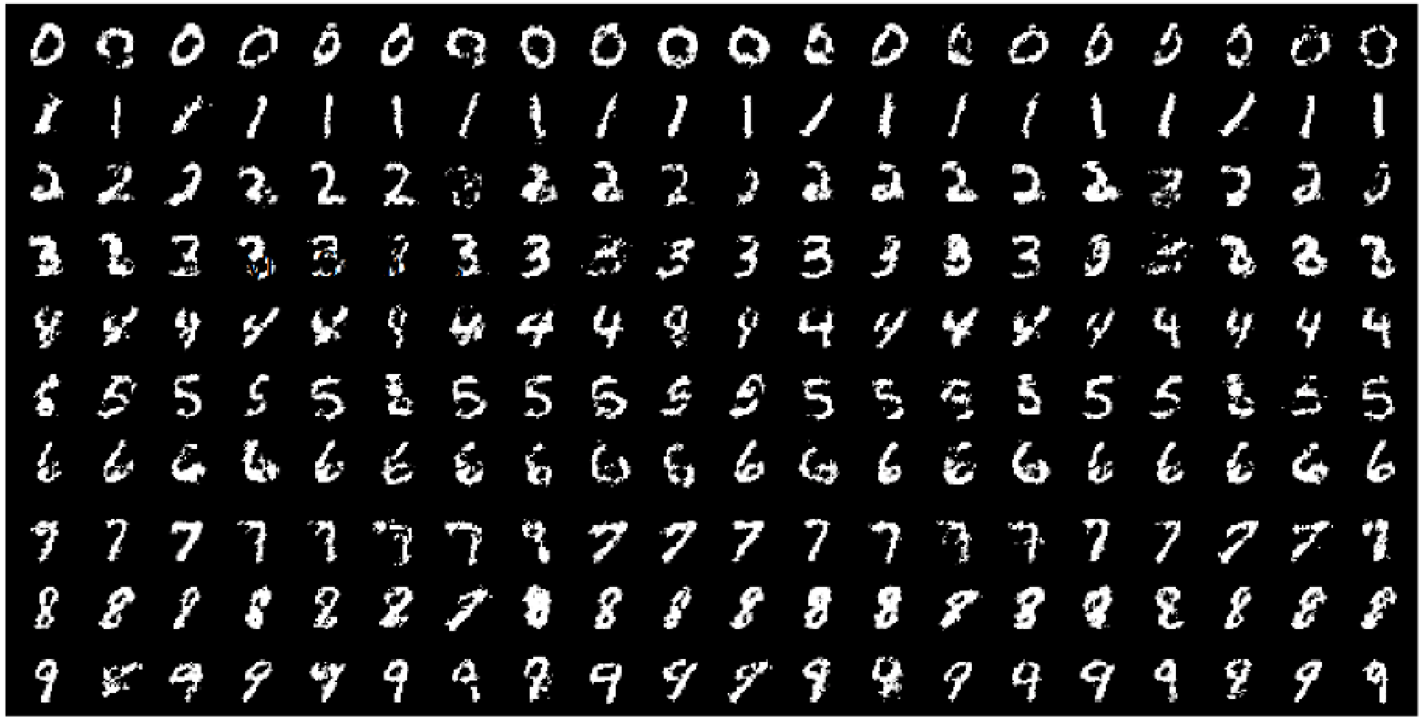
$[0, 0, 0, 0, 0, 1, 0, 0, 0, 0] \longrightarrow$

$[0, 0, 0, 0, 0, 0, 1, 0, 0, 0] \longrightarrow$

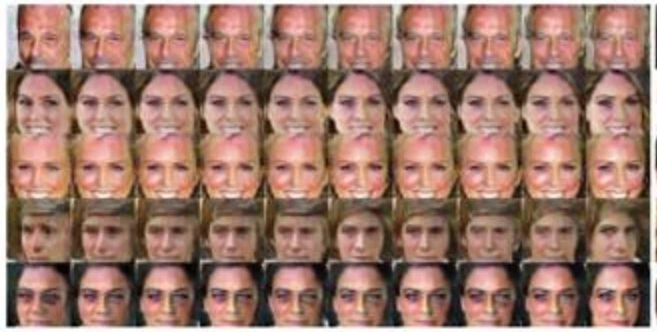
$[0, 0, 0, 0, 0, 0, 0, 1, 0, 0] \longrightarrow$

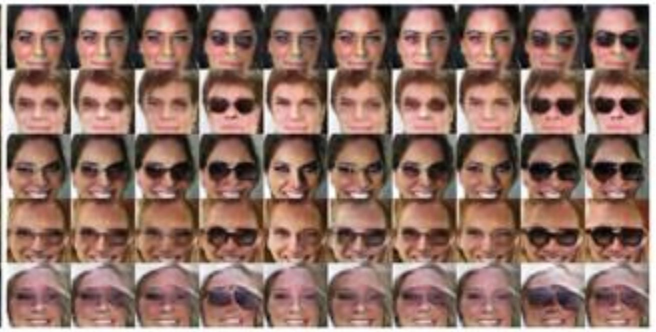
$[0, 0, 0, 0, 0, 0, 0, 0, 1, 0] \longrightarrow$

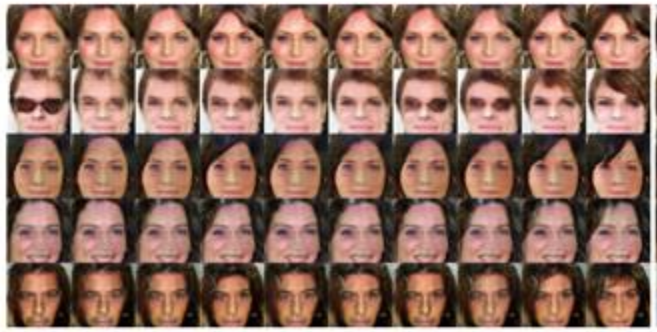
$[0, 0, 0, 0, 0, 0, 0, 0, 0, 1] \longrightarrow$

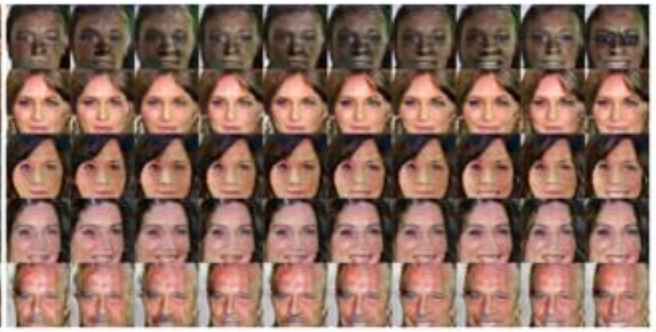


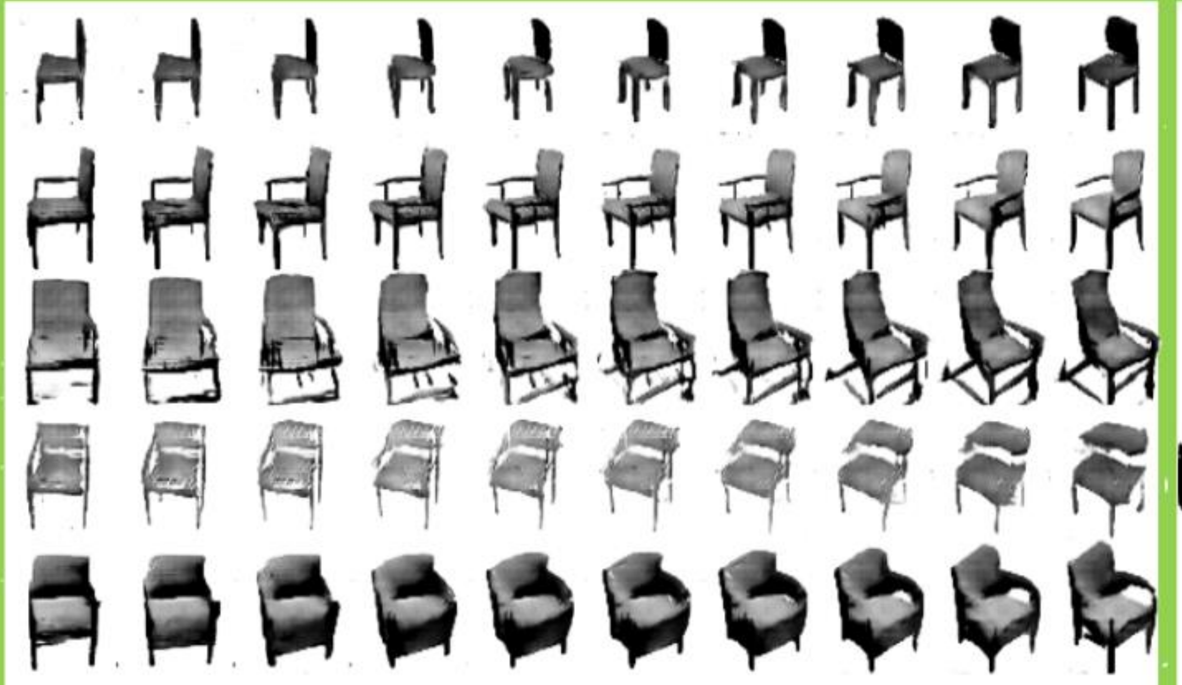
Examples (InfoGAN)

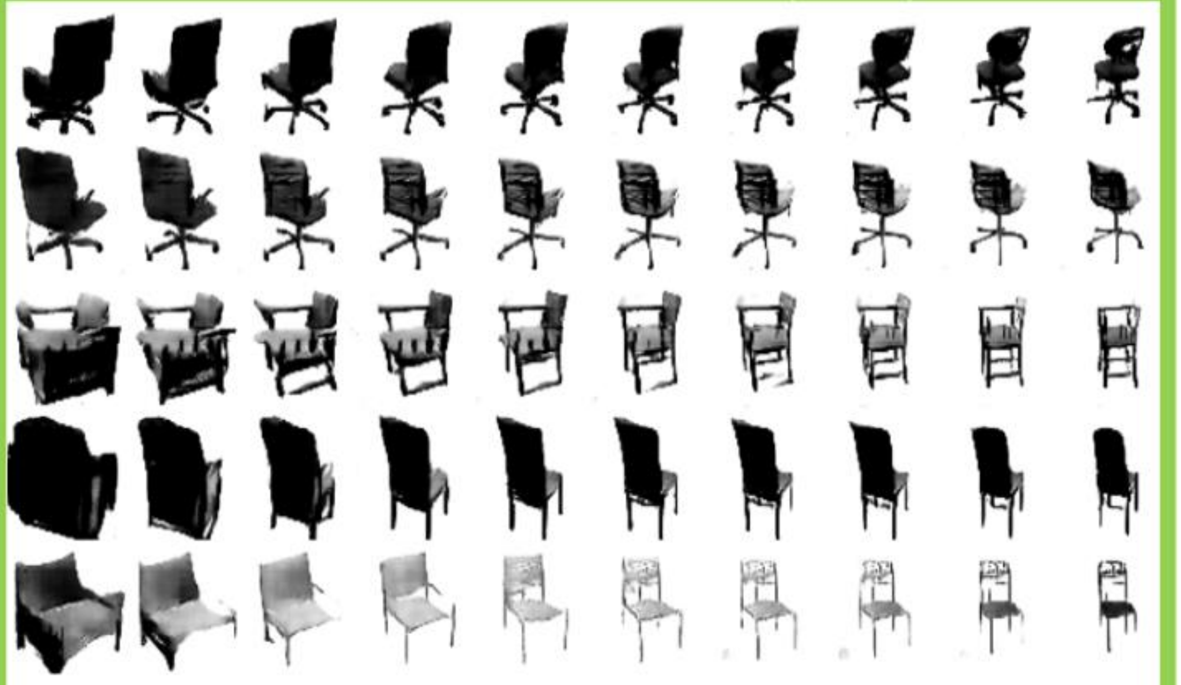

(a) Azimuth (pose)


(b) Presence or absence of glasses

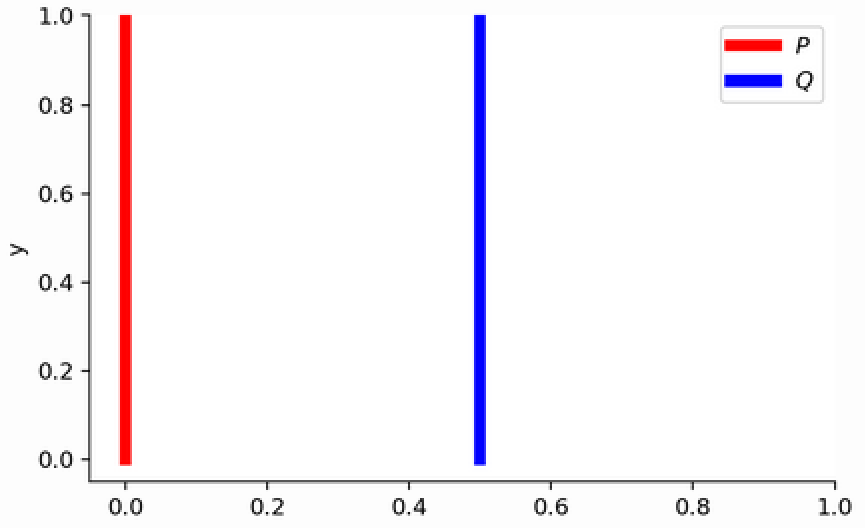

(c) Hair style


(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:

$$P_1 = 3, P_2 = 2, P_3 = 1, P_4 = 4$$
$$Q_1 = 1, Q_2 = 2, Q_3 = 4, Q_4 = 3$$

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

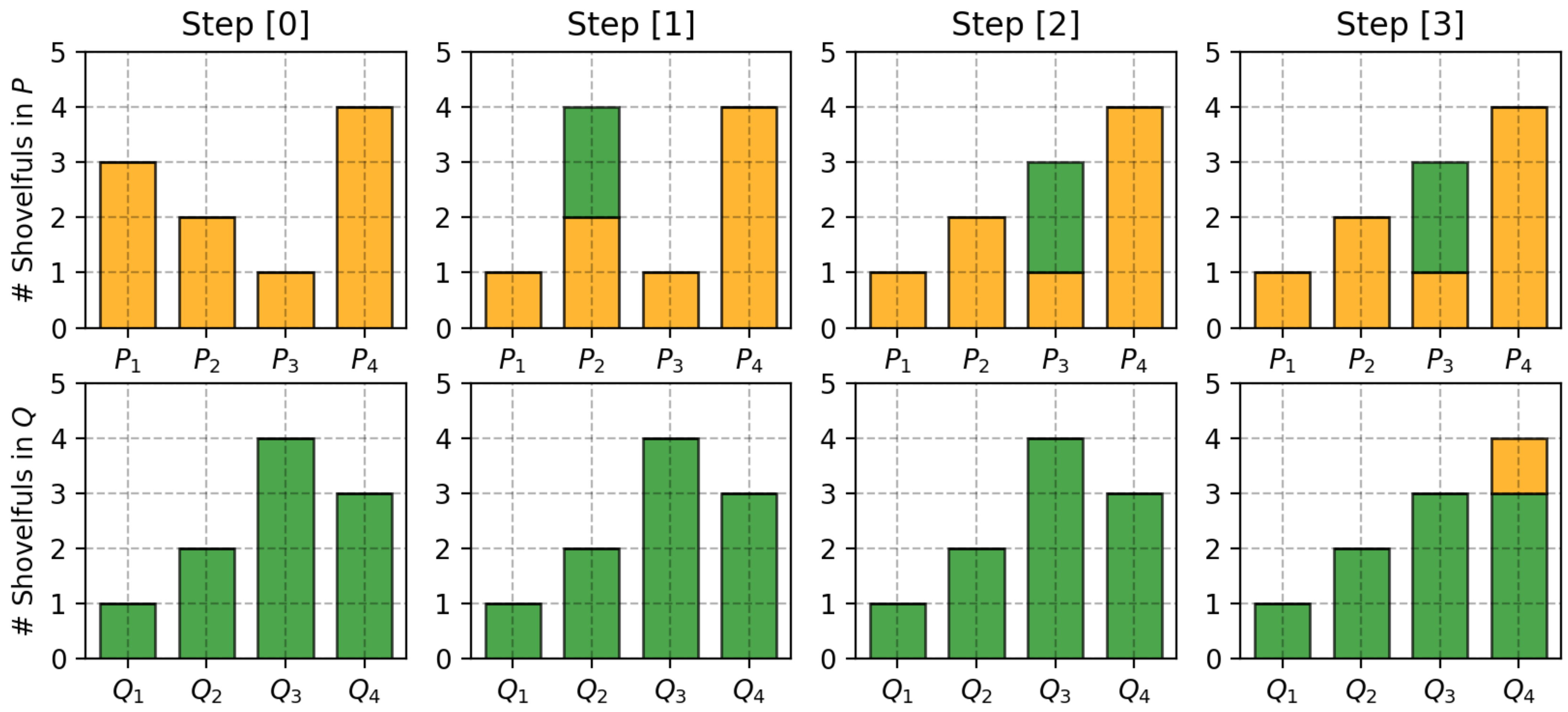
- First move 2 shovelfuls from P_1 to P_2 => (P_1, Q_1) match up.
- Then move 2 shovelfuls from P_2 to P_3 => (P_2, Q_2) match up.
- Finally move 1 shovelfuls from Q_3 to Q_4 => (P_3, Q_3) and (P_4, Q_4) match up.

If we label the cost to pay to make P_i and Q_i match as δ_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



Step [0]
Step [1]
Step [2]
Step [3]
# Shovelfuls in P
# Shovelfuls in Q
P_1 P_2 P_3 P_4
Q_1 Q_2 Q_3 Q_4
0 1 2 3 4 5

Wasserstein distance

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

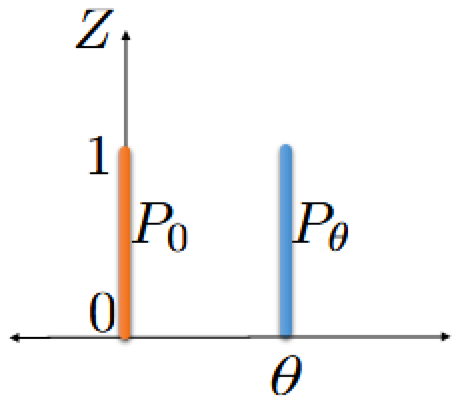
$$\sum_{x,y} \gamma(x, y)\|x - y\| = \mathbb{E}_{x,y\sim\gamma}\|x - y\|$$

Kantorovich rubinson duality

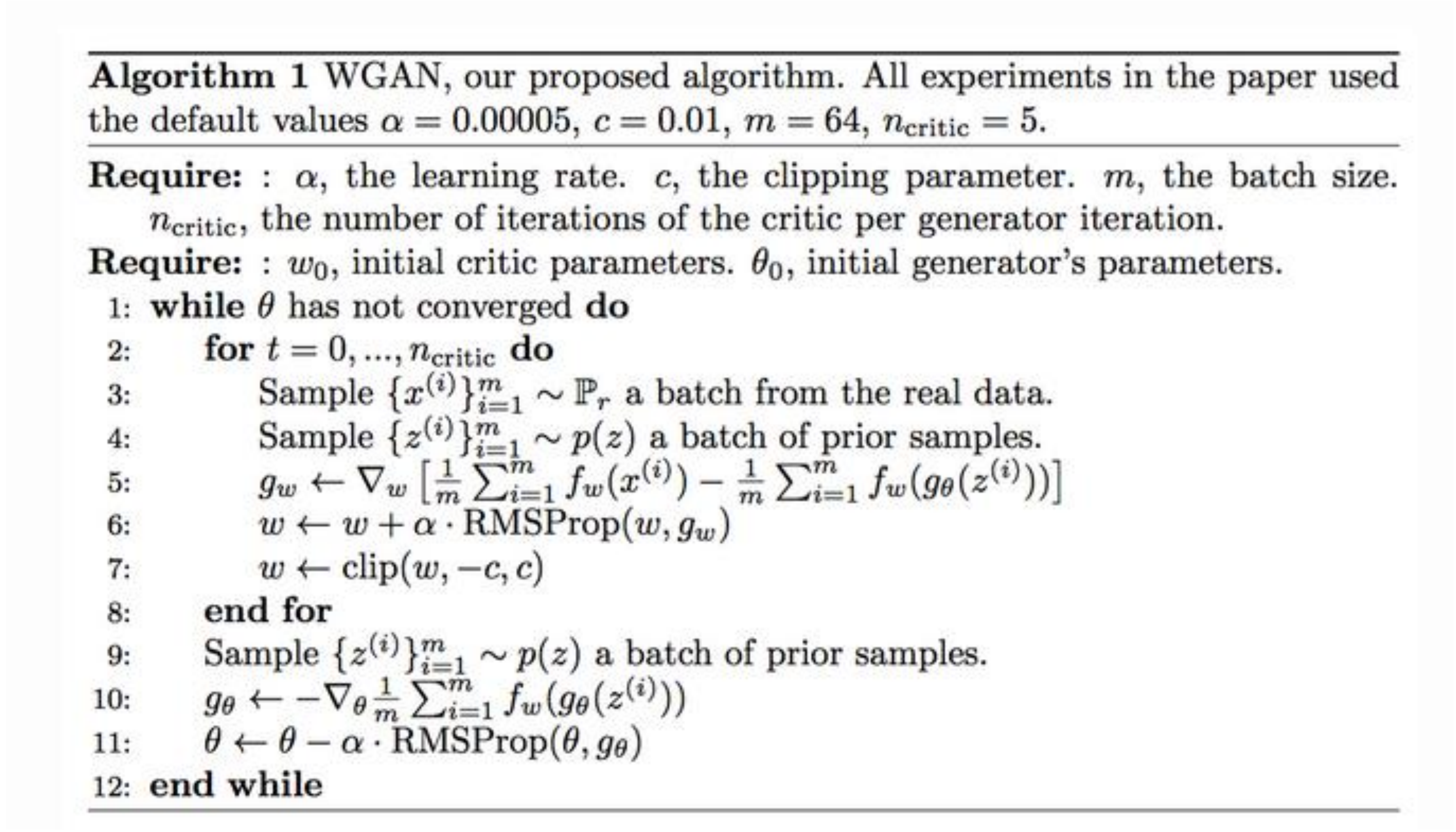
$$W(p_r, p_g) = \frac{1}{K} \sup_{\|f\|_L \leq K} \mathbb{E}_{x\sim p_r}[f(x)] - \mathbb{E}_{x\sim p_g}[f(x)]$$

P_0: distribution of $(0, Z)$, where $Z \sim U[0,1]$
P_θ: distribution of (θ, Z), where θ is a single real parameter

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



(a) Distributions



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 $t = 0, ..., n_{\text{critic}}$ 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
```