

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

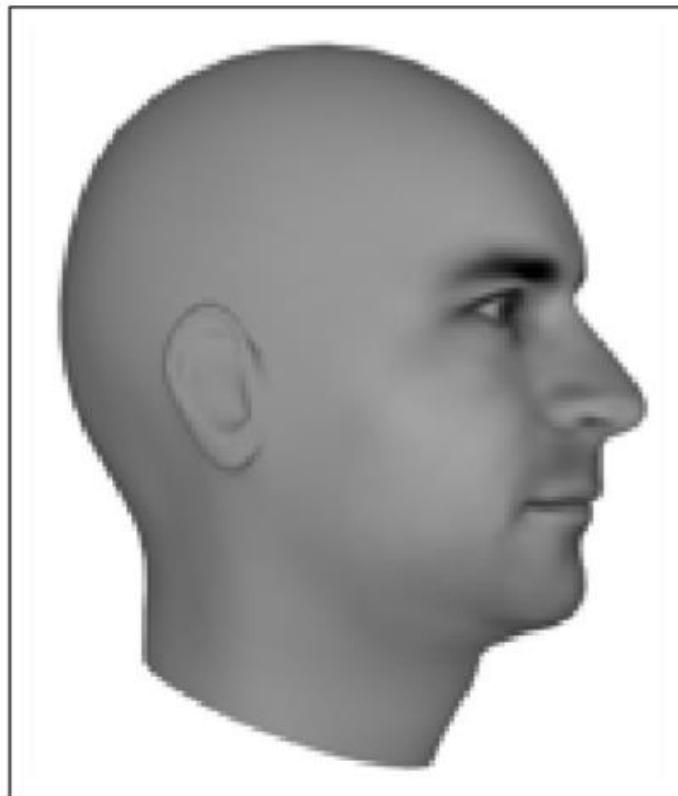
Implicit density estimation problem

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

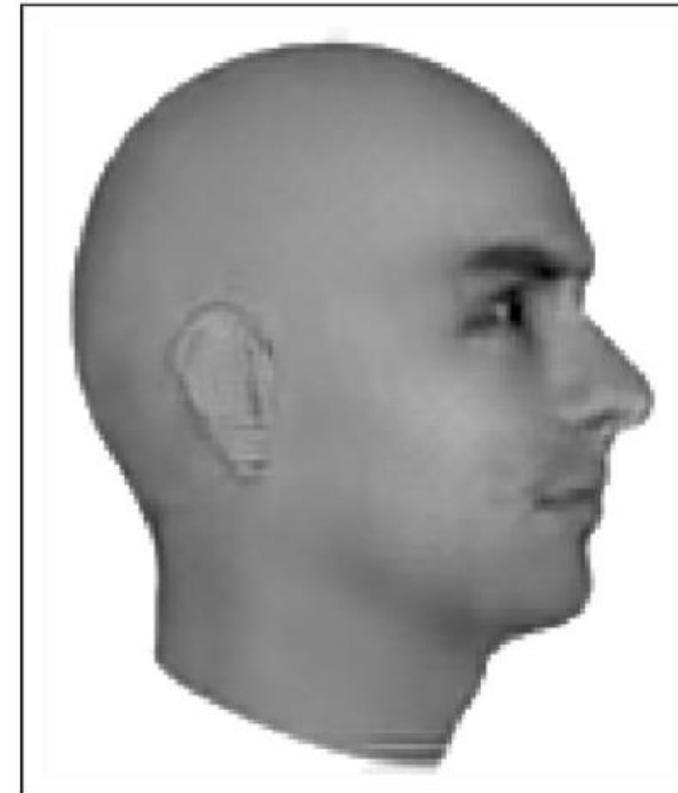


Magic of GAN

Ground Truth



Adversarial

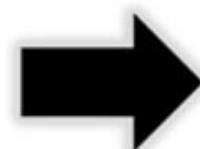
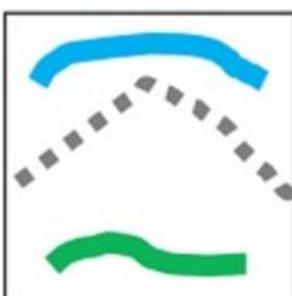
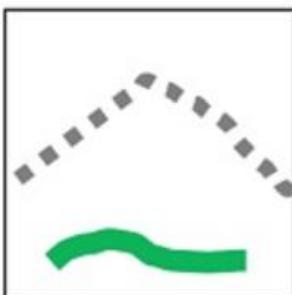


Magic of GAN

Which one is Computer generated?



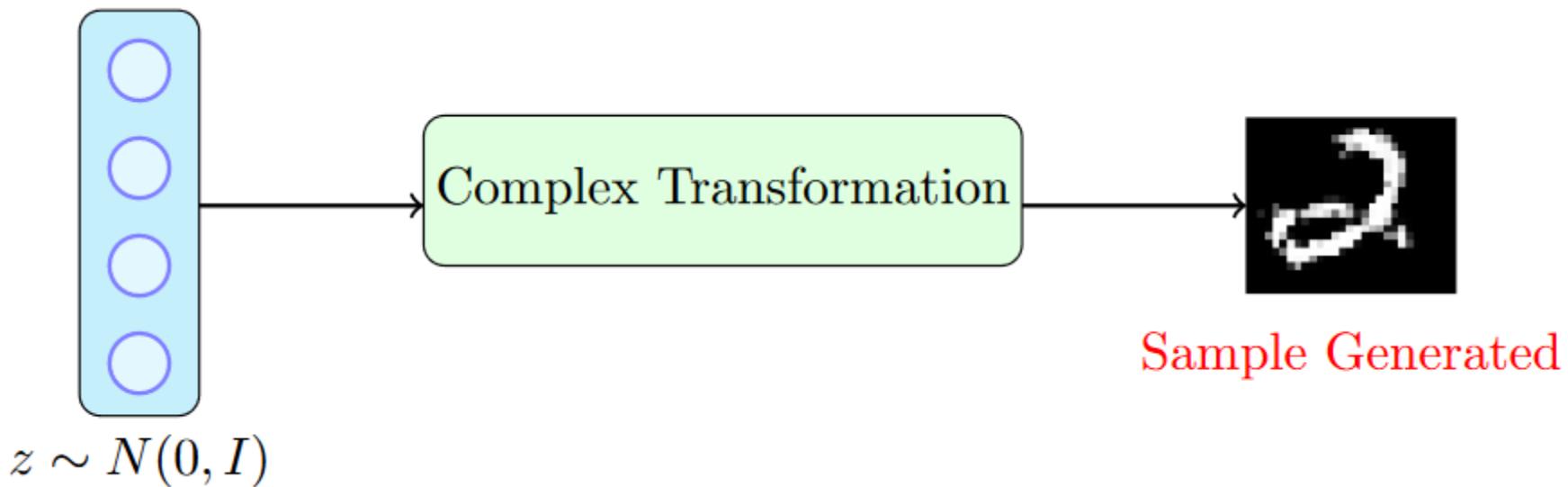
User edits

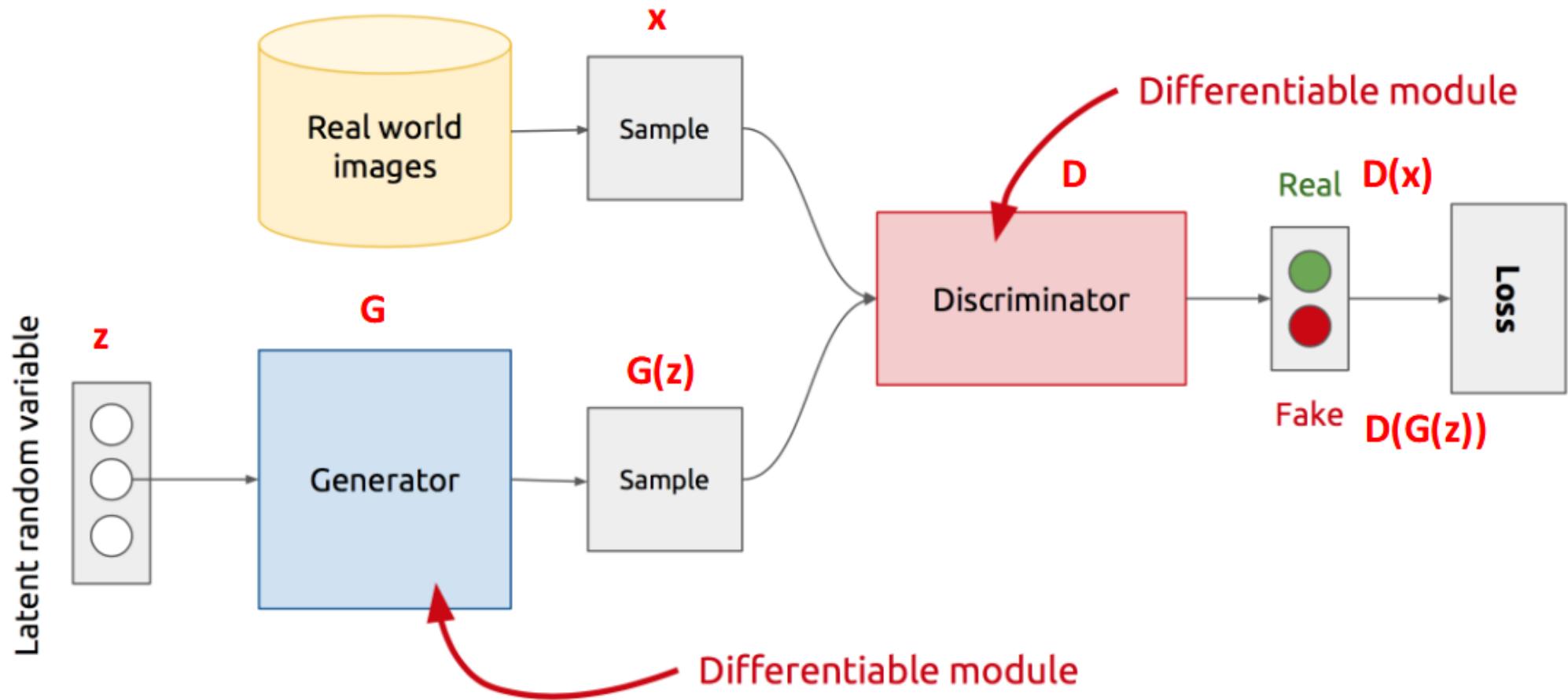


Generated images



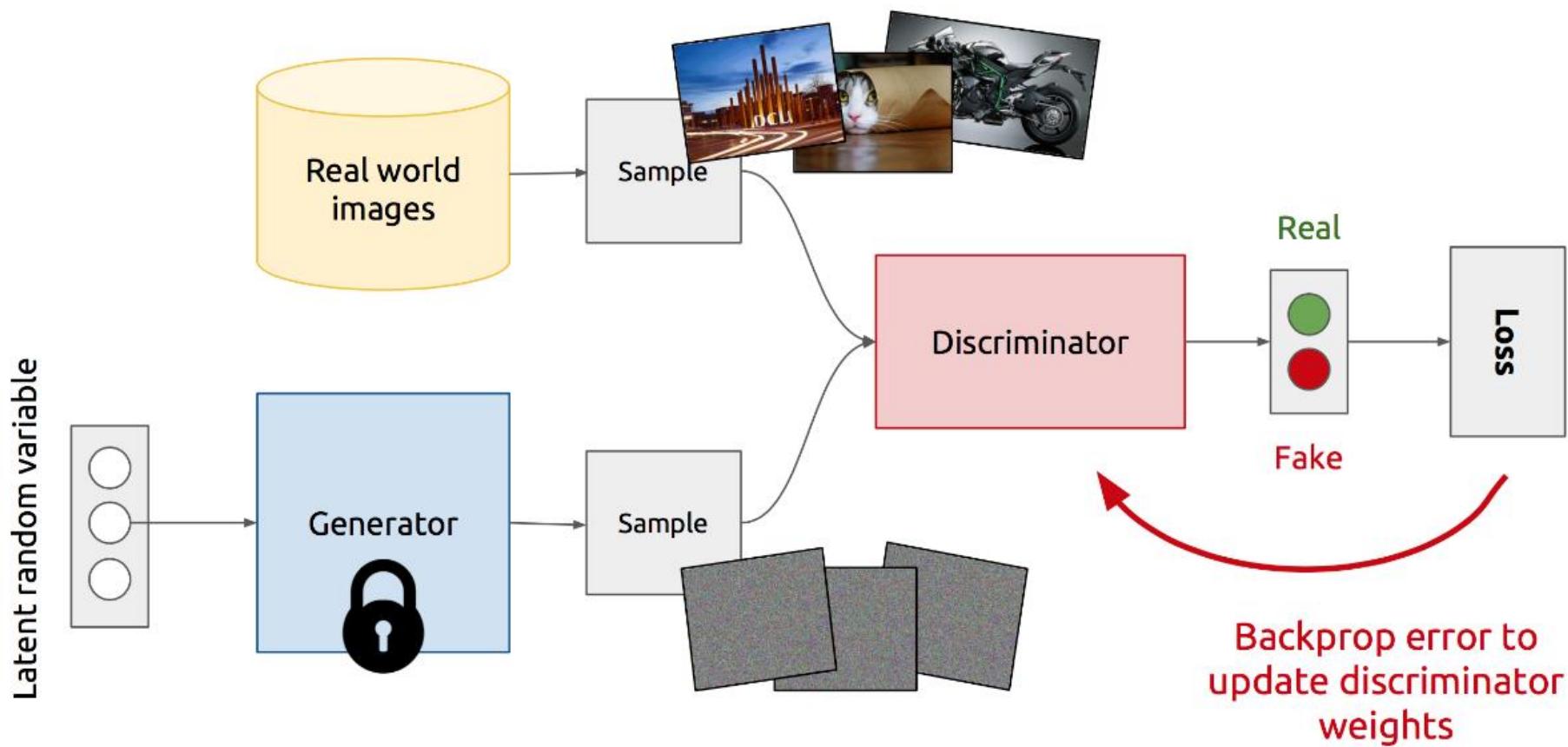
What about



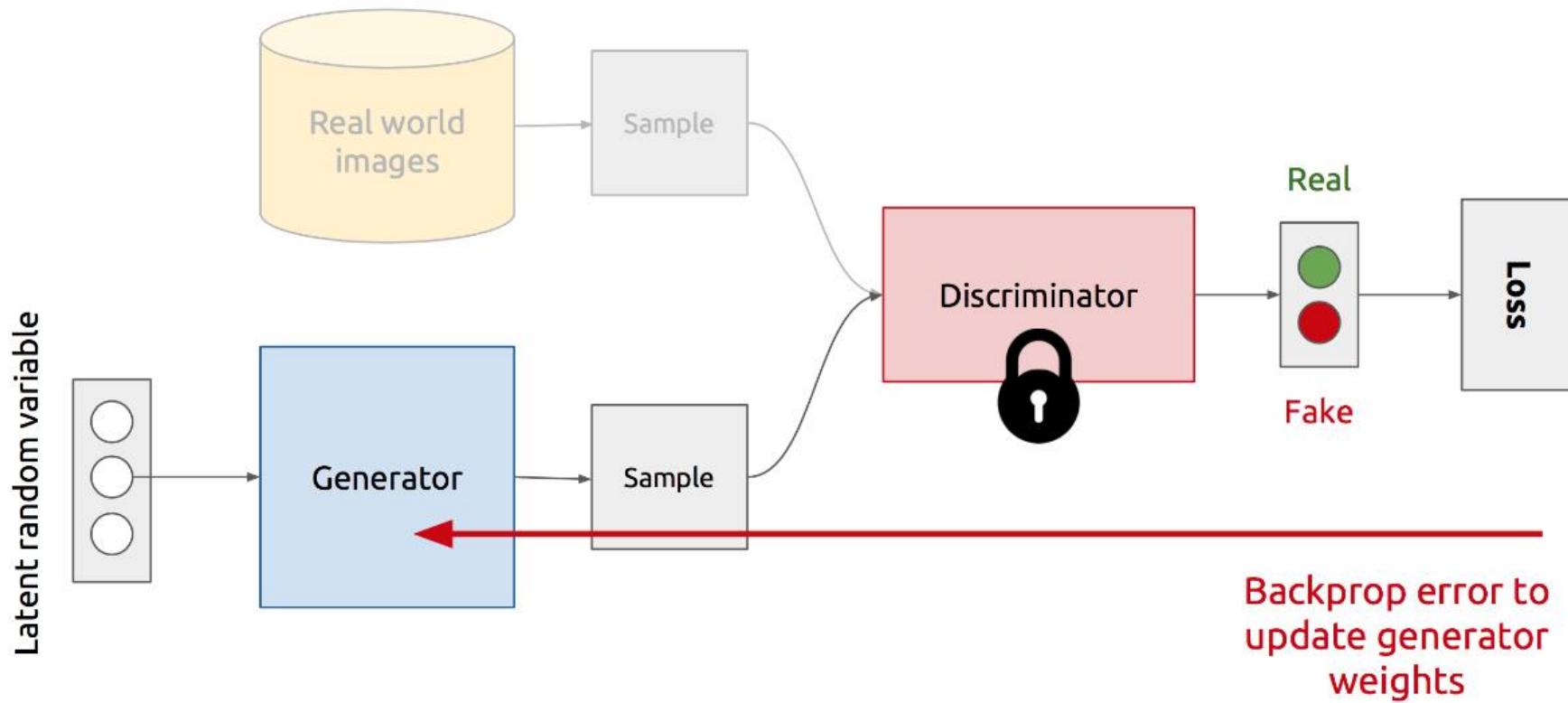


- Z is some random noise (Gaussian/Uniform).
- Z can be thought as the latent representation of the image.

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

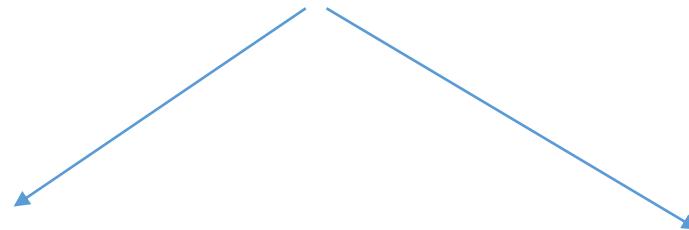
$$\min_{\phi} E_{z \sim p(z)} [\log(1 - D_{\theta}(G_{\phi}(z)))]$$

- 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

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



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

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

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

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

- 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)

- **Discriminator Loss:**

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

- **Generator Loss (Non-Saturating):**

$$\min_G -\mathbb{E}_{z \sim p_z} [\log(D(G(z)))]$$

- For instance, if $D(G(z)) = 0.01$, then $\log(0.01) \approx -4.605$.
- The generator's objective $-\log(D(G(z)))$ is therefore $\approx +4.605$, which is **high**; pushing it down yields a large gradient.
- In contrast, in the saturating version, $\log(1 - 0.01) = \log(0.99) \approx -0.01005$. That's nearly zero and doesn't push as hard.

Optimal D

For a fixed generator G , the distribution p_g is fixed. We want:

$$\max_D \int [p_r(x) \log D(x) + p_g(x) \log(1 - D(x))] dx.$$

Inside the integral, each x is independent in terms of $D(x)$. Define:

$$f(D(x)) = p_r(x) \log[D(x)] + p_g(x) \log[1 - D(x)].$$

Take the derivative w.r.t. $D(x)$ and set to zero:

$$\frac{\partial f}{\partial D(x)} = \frac{p_r(x)}{D(x)} - \frac{p_g(x)}{1 - D(x)} = 0 \implies p_r(x)[1 - D(x)] = p_g(x)[D(x)].$$

Solving for $D(x)$ gives the **optimal** (pointwise) discriminator:

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

Plug $D^*(x)$ back into $V(D, G)$:

$$\max_D V(D, G) = \int \left[p_r(x) \log\left(\frac{p_r(x)}{p_r(x) + p_g(x)}\right) + p_g(x) \log\left(\frac{p_g(x)}{p_r(x) + p_g(x)}\right) \right] dx.$$

Global optima

$$\text{JS}(P \parallel Q) = \frac{1}{2} \text{KL}\left(P \parallel \frac{P+Q}{2}\right) + \frac{1}{2} \text{KL}\left(Q \parallel \frac{P+Q}{2}\right),$$

where KL denotes the Kullback–Leibler divergence.

Let $P = p_r$ and $Q = p_g$. Then their midpoint is

$$M(x) = \frac{p_r(x) + p_g(x)}{2}.$$

$$\begin{aligned} \text{KL}(p_r \parallel M) &= \int p_r(x) \log\left[\frac{p_r(x)}{M(x)}\right] dx = \int p_r(x) \log\left[\frac{p_r(x)}{\frac{p_r(x) + p_g(x)}{2}}\right] dx. \\ &= \int p_r(x) \log\left[\frac{2p_r(x)}{p_r(x) + p_g(x)}\right] dx = \int p_r(x) \log\left[\frac{p_r(x)}{p_r(x) + p_g(x)}\right] dx + \int p_r(x) \log(2) dx. \end{aligned}$$

Since $\int p_r(x) dx = 1$,

$$\text{KL}(p_r \parallel M) = \int p_r(x) \log\left[\frac{p_r(x)}{p_r(x) + p_g(x)}\right] dx + \log(2). \quad \text{KL}(p_g \parallel M) = \int p_g(x) \log\left[\frac{p_g(x)}{p_r(x) + p_g(x)}\right] dx + \log(2).$$

$$\text{KL}(p_r \parallel M) + \text{KL}(p_g \parallel M) = \int p_r(x) \log\left[\frac{p_r(x)}{p_r(x) + p_g(x)}\right] dx + \int p_g(x) \log\left[\frac{p_g(x)}{p_r(x) + p_g(x)}\right] dx + 2 \log(2).$$

Global optima

$$\text{JS}(p_r \| p_g) = \frac{1}{2} \left[\int p_r(x) \log\left(\frac{p_r(x)}{p_r(x) + p_g(x)}\right) dx + \int p_g(x) \log\left(\frac{p_g(x)}{p_r(x) + p_g(x)}\right) dx + 2 \log(2) \right].$$

$$\int p_r(x) \log\left[\frac{p_r(x)}{p_r(x) + p_g(x)}\right] + \int p_g(x) \log\left[\frac{p_g(x)}{p_r(x) + p_g(x)}\right] = 2 [\text{JS}(p_r \| p_g) - \log(2)].$$

Recall:

$$\max_D V(D, G) = \int p_r(x) \log\left[\frac{p_r(x)}{p_r(x) + p_g(x)}\right] dx + \int p_g(x) \log\left[\frac{p_g(x)}{p_r(x) + p_g(x)}\right] dx.$$

$$\max_D V(D, G) = 2 [\text{JS}(p_r \| p_g) - \log(2)].$$

Interpretation

- When $p_g = p_r$:

$\text{JS}(p_r \| p_g) = 0$. Hence $\max_D V(D, G) = -2 \log(2)$. Equivalently, $-\log(4)$.

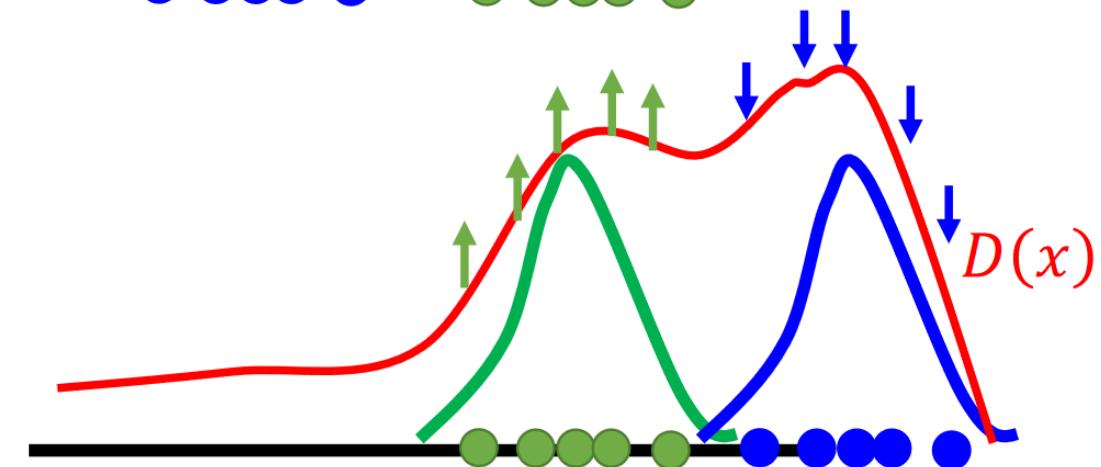
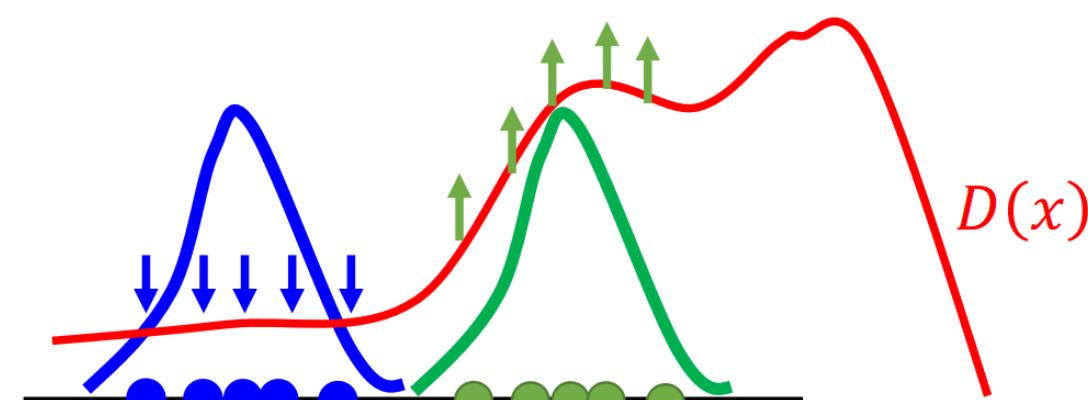
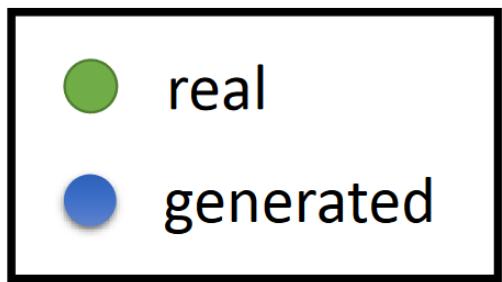
In this case, the discriminator's best guess is $D^*(x) = \frac{1}{2}$ everywhere (completely random guess).

- When $p_g \neq p_r$:

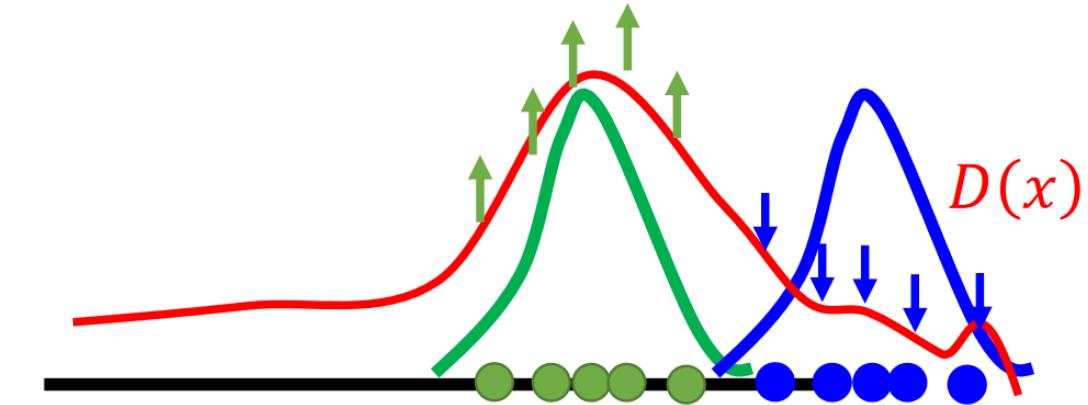
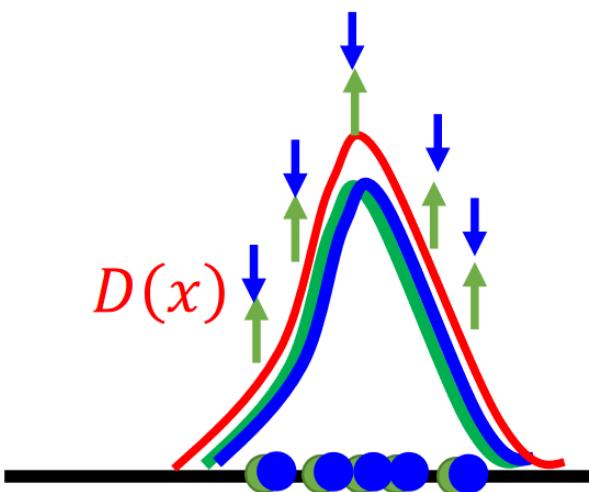
$\text{JS}(p_r \| p_g) > 0$. The discriminator can achieve a value higher than $-2 \log(2)$, up to 0 if the two distributions are perfectly separable.

Hence the **GAN game** ideally pushes p_g to match p_r (so $\text{JS} = 0$), achieving the global optimum.

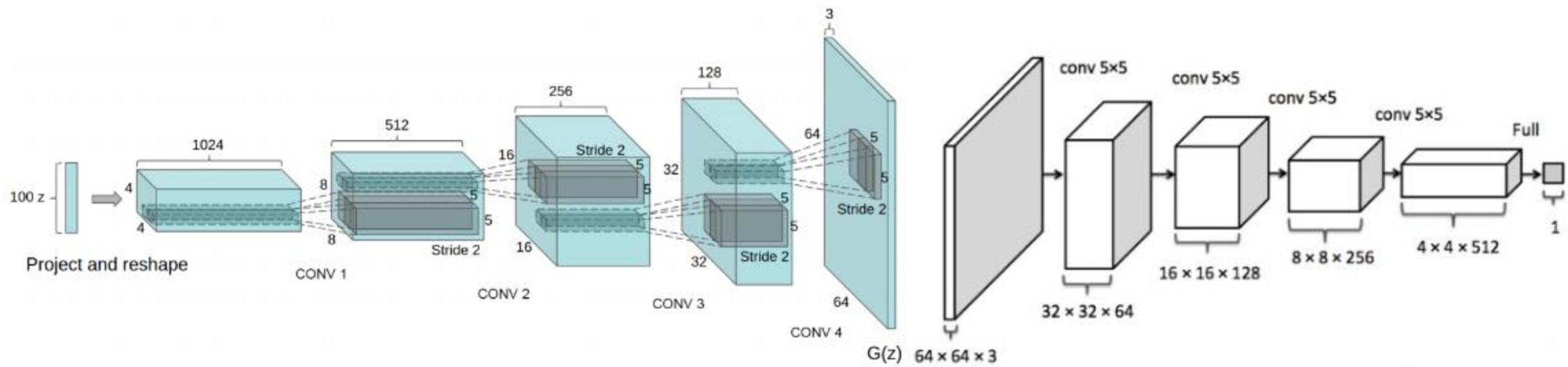
Discriminator - Training



In the end



DC GAN



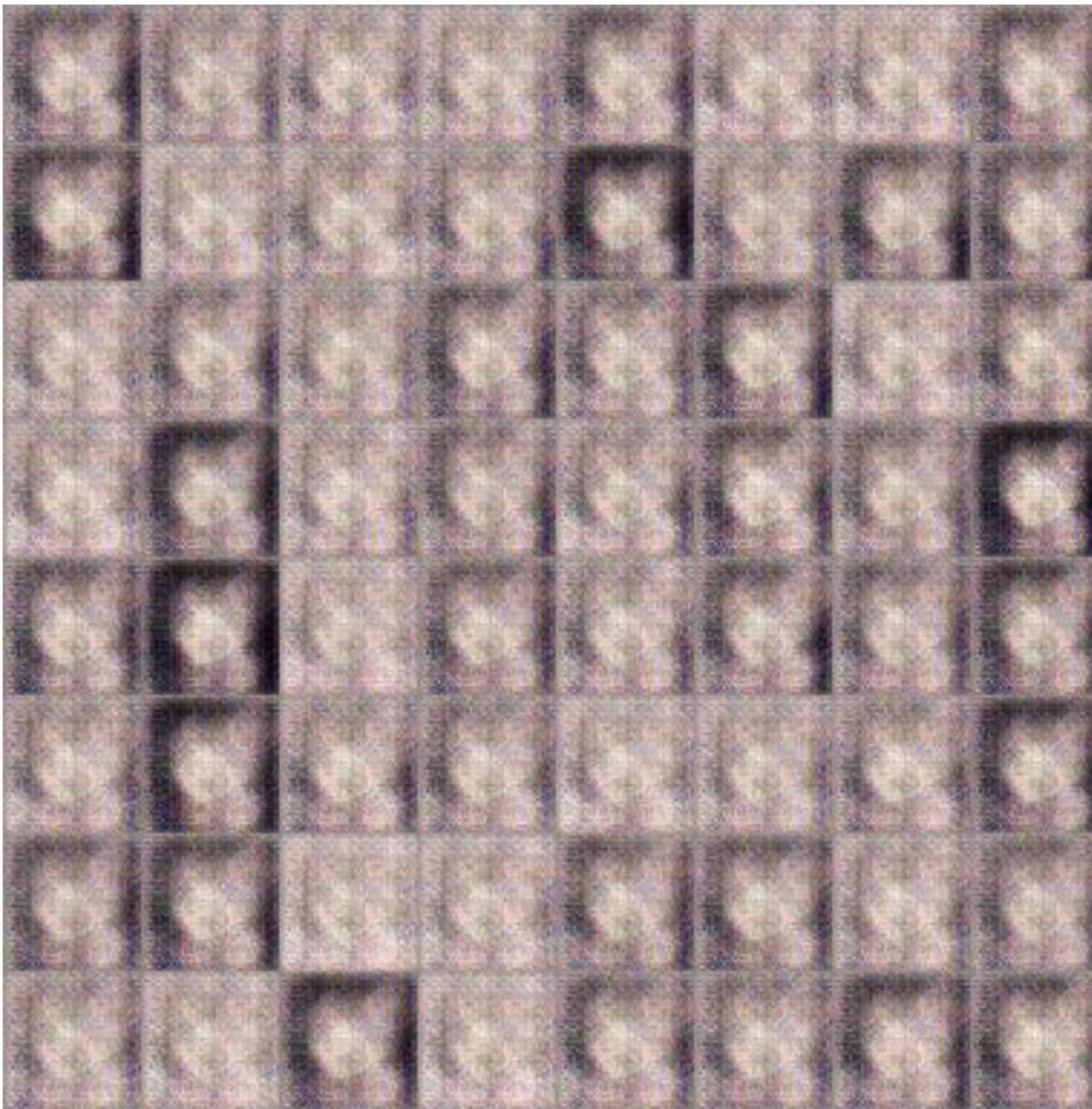


Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." arXiv:1511.06434 (2015).



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

100 updates



1000 updates



5000 updates



50,000 updates



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

- ## 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.

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.

Lets consider an example

$$\min_x \max_y xy$$

Gradient descent would update:

- $x \leftarrow x - \eta y$
- $y \leftarrow y + \eta x$

Even though the **true saddle point is at $(0, 0)$** , these updates keep rotating around the origin forever — they **never converge**.

$$\mathbf{z}_t = \begin{bmatrix} x_t \\ y_t \end{bmatrix}, \quad \mathbf{z}_{t+1} = \underbrace{\begin{bmatrix} 1 & -\eta \\ \eta & 1 \end{bmatrix}}_A \mathbf{z}_t$$

The update matrix A has **complex eigenvalues**:

$$\lambda = 1 \pm i\eta$$

- If $|\lambda| > 1$, the updates will **spiral outward**, causing **divergence**.
- If $|\lambda| \approx 1$, the updates **oscillate in circles**.

- $x_0 = 1$
- $y_0 = 0$

Step 1:

$$x_1 = 1 - \eta \cdot 0 = 1$$

$$y_1 = 0 + \eta \cdot 1 = \eta$$

Step 2:

$$x_2 = 1 - \eta \cdot \eta \approx 1 - \eta^2$$

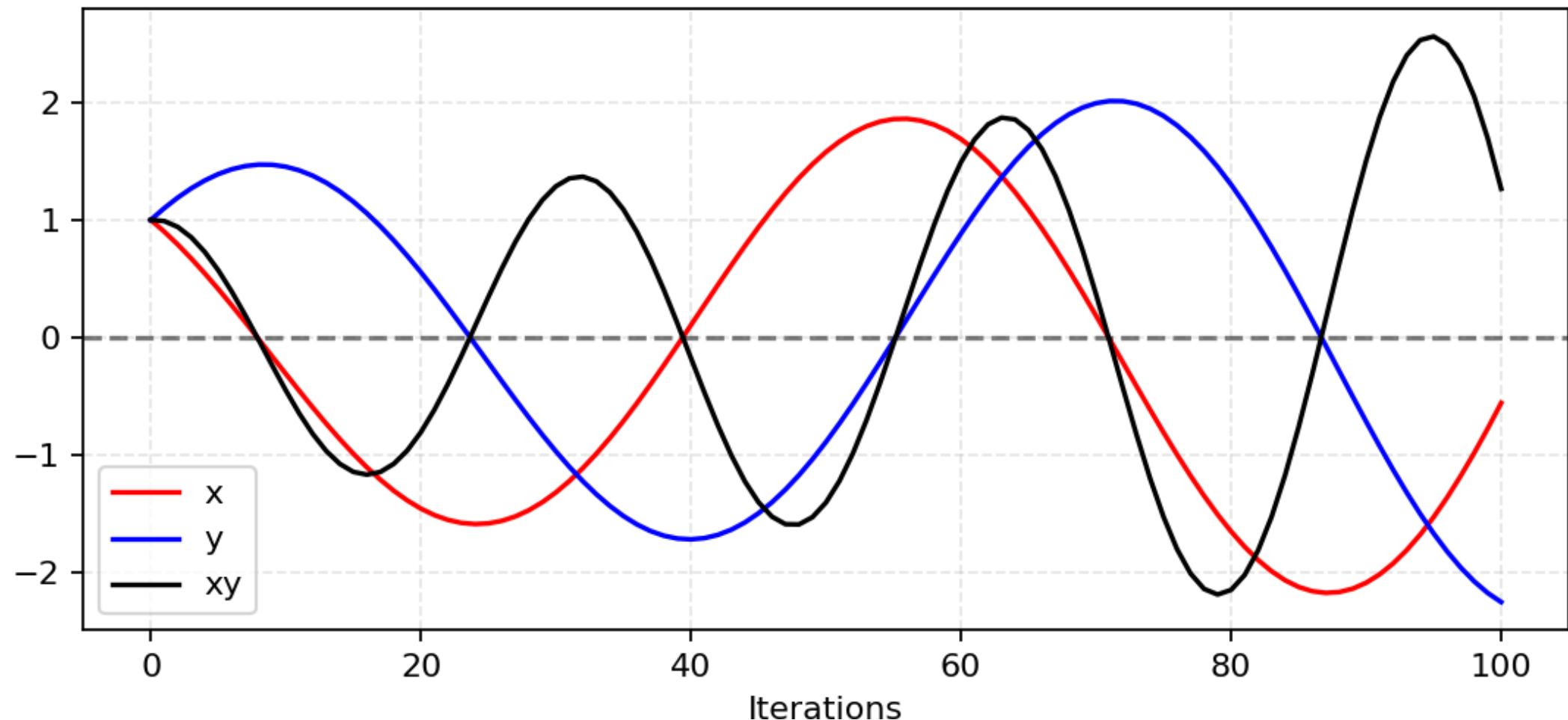
$$y_2 = \eta + \eta \cdot 1 = \eta(1 + \eta)$$

Step 3:

$$x_3 = 1 - \eta^2 - \eta(1 + \eta) \approx 1 - \eta^2 - \eta - \eta^2$$

$$y_3 = \eta(1 + \eta) + \eta(1 - \eta^2) \approx 2\eta + \eta^3$$

Problem in obtaining the equilibrium



What are the solutions

- Reduce learning rate
- Use different learning rates for the generator and discriminator
- Predict the next step's gradient and use it to stabilize updates.

1. Predictive Step (Lookahead):

- First, take a partial gradient step to predict where the opponent will move.
- Update to a temporary point \tilde{x}, \tilde{y} :

$$\tilde{x} = x_t - \eta \frac{\partial f}{\partial x}(x_t, y_t)$$

$$\tilde{y} = y_t + \eta \frac{\partial f}{\partial y}(x_t, y_t)$$

2. Corrective Step (Using Lookahead Gradient):

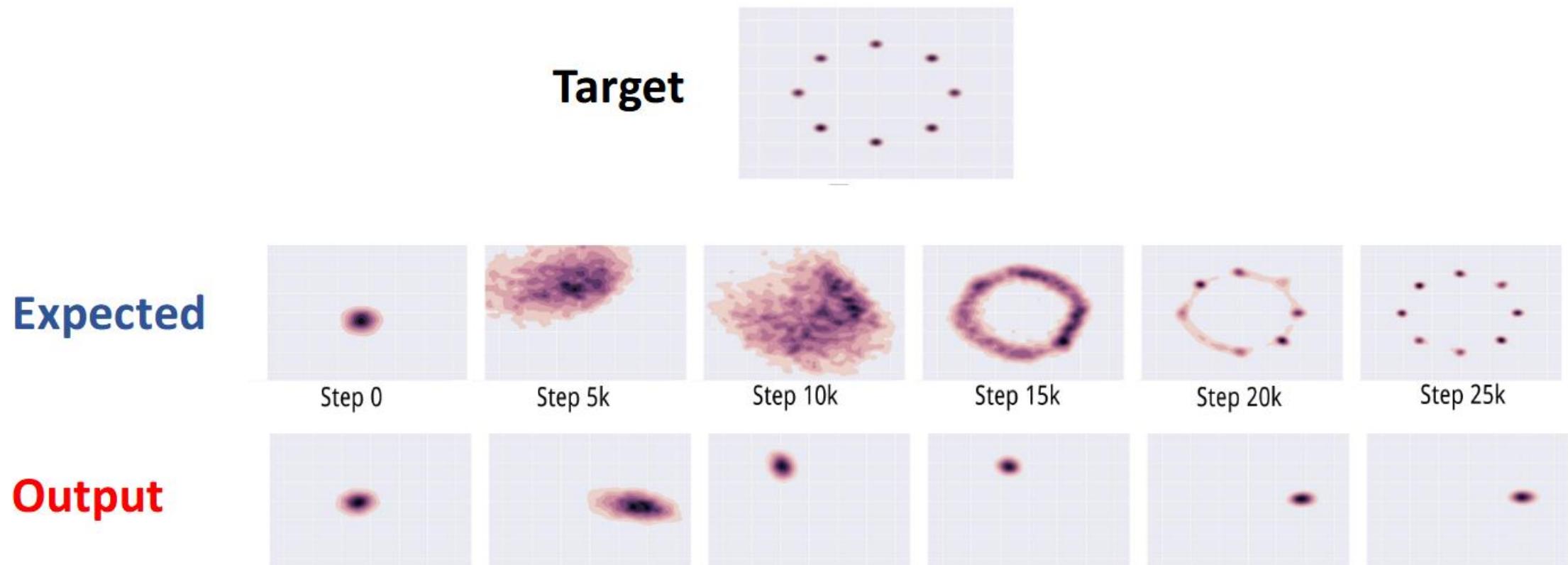
- Now, use the gradients at this new predicted point (\tilde{x}, \tilde{y}) to correct the original update:

$$x_{t+1} = x_t - \eta \frac{\partial f}{\partial x}(\tilde{x}, \tilde{y})$$

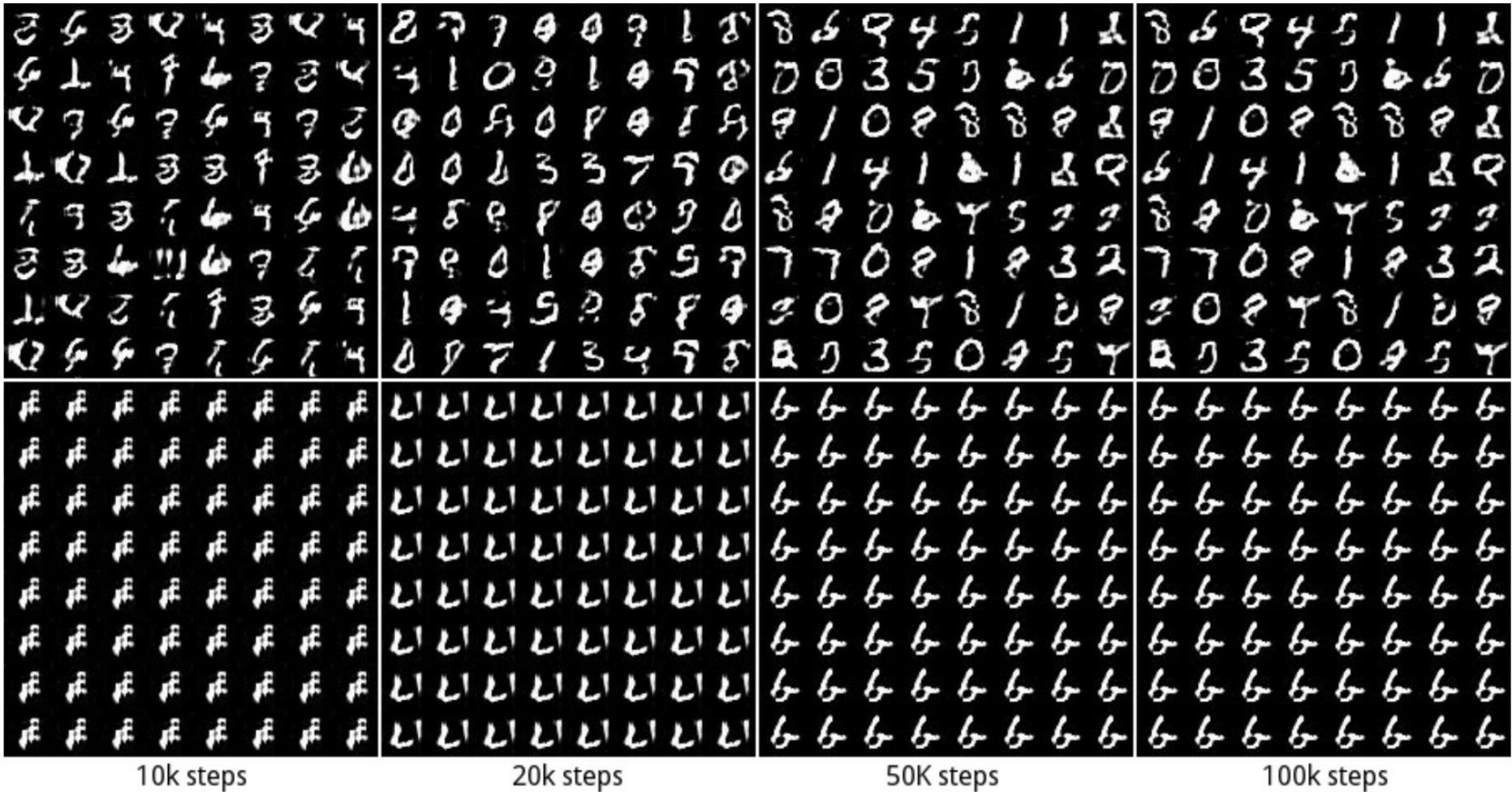
$$y_{t+1} = y_t + \eta \frac{\partial f}{\partial y}(\tilde{x}, \tilde{y})$$

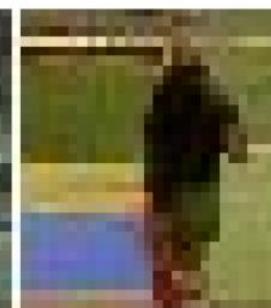
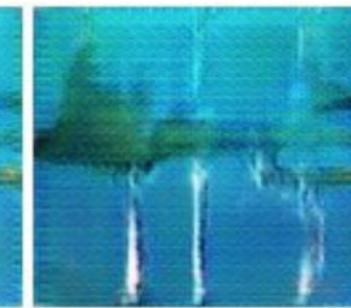
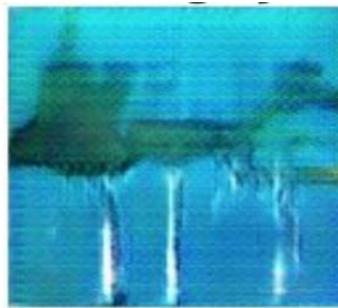
Mode collapse

- Generator fails to output diverse samples



Mode collapse in MNIST



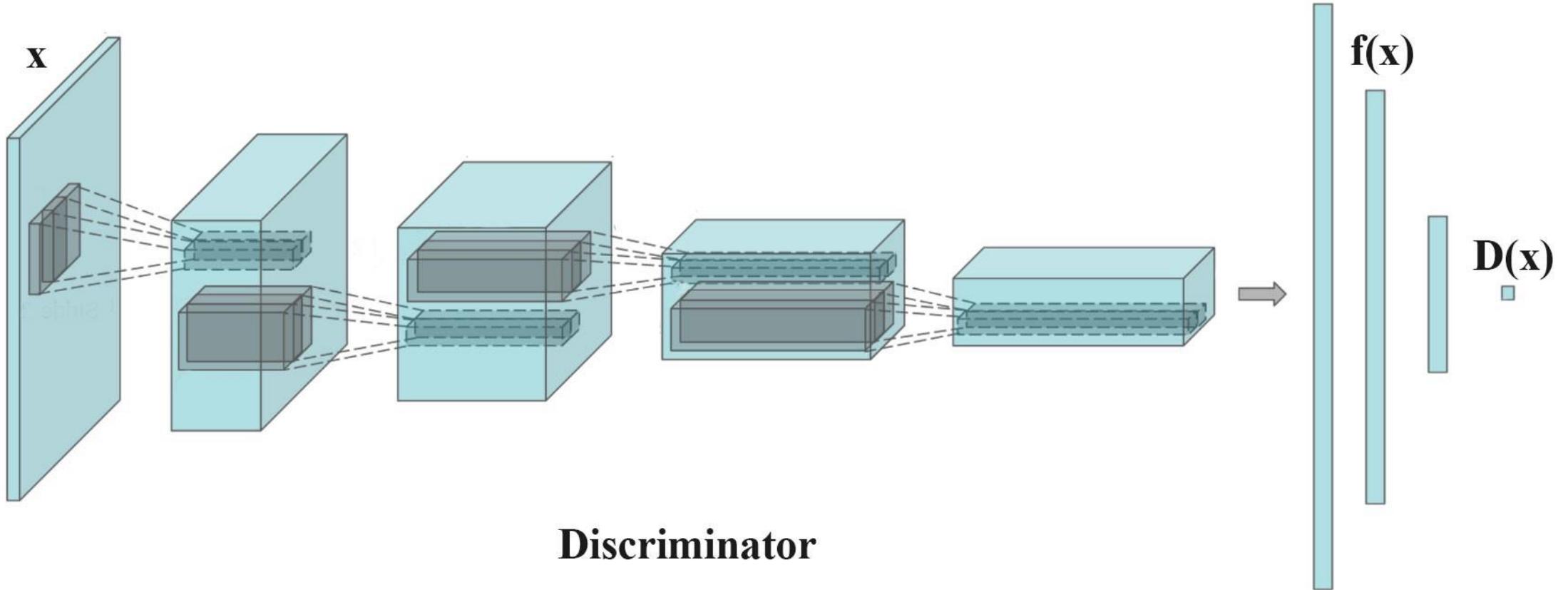


Feature matching

In Feature Matching:

- Instead of directly minimizing the divergence between real and generated distributions in pixel space, we:
 - **Extract feature representations** of real and generated samples using an intermediate layer of the discriminator.
 - **Match the statistics** (e.g., mean, covariance) of these feature representations to ensure that generated samples cover the same modes as the real samples.

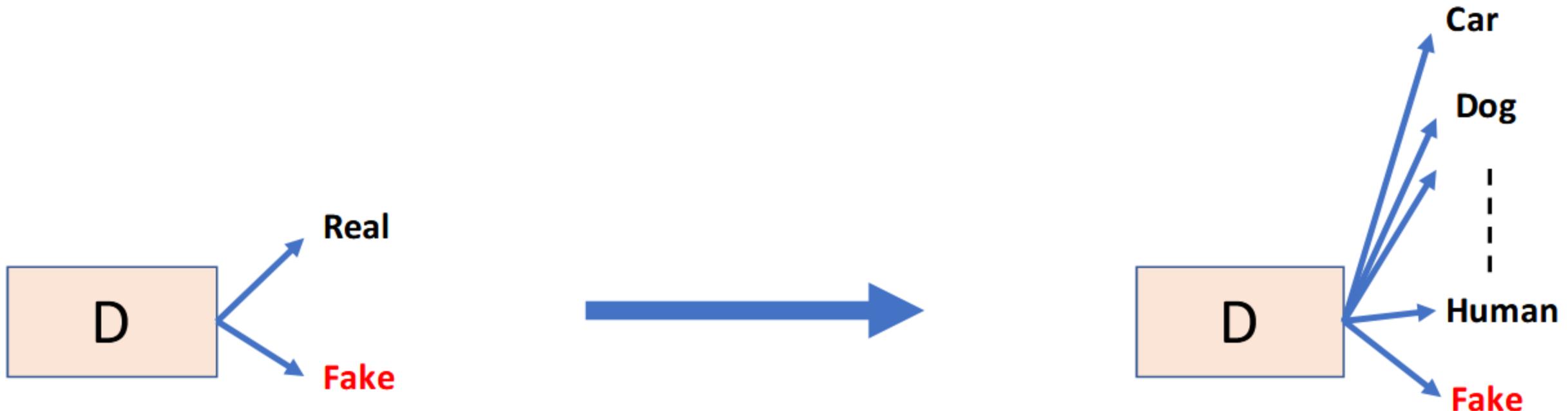
Feature matching

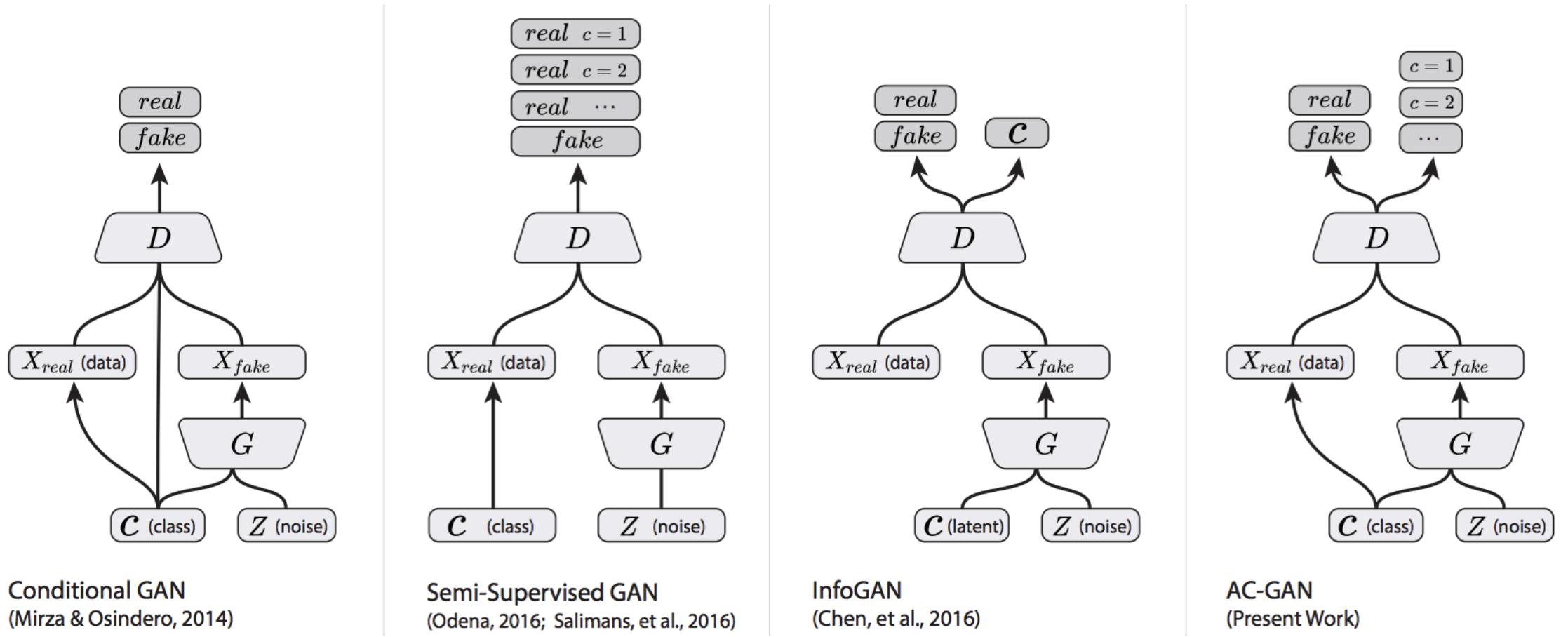


$$\|\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \mathbf{f}(\mathbf{x}) - \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} \mathbf{f}(G(\mathbf{z}))\|_2^2$$

- 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

Supervision with labels



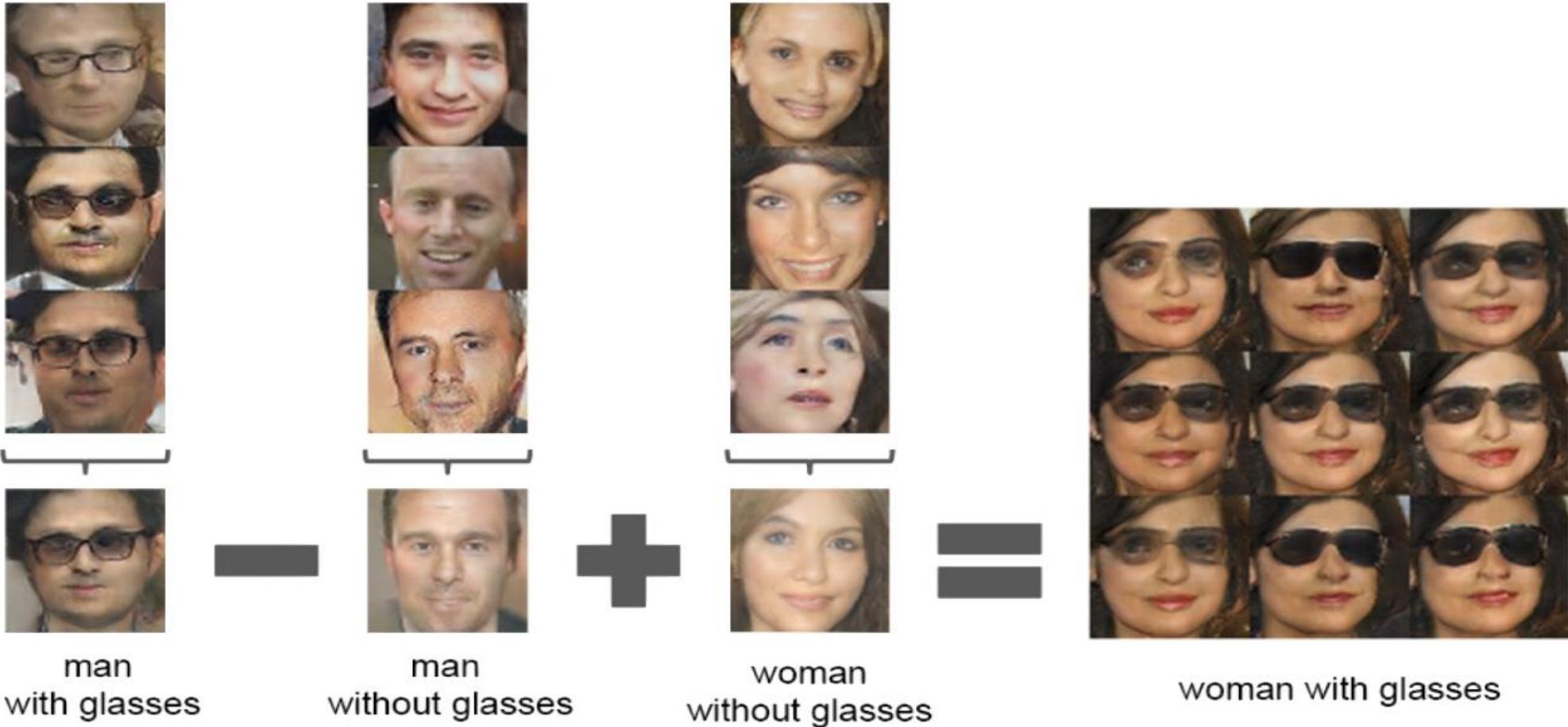


$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x}|\mathbf{y})] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})} [\log(1 - D(G(\mathbf{z}|\mathbf{y})))].$$

One issue of vanilla GAN

- The generator maps the noise vector $z \sim p_z$ to the data space.
- However, z lacks **structured information** about the factors of variation in the data.
- As a result:
 - Small changes in z can affect **multiple factors** at the same time.
 - There's **no control** over interpretable features such as rotation, thickness, shape, etc.
 - Latent dimensions are **entangled** — they don't correspond to independent factors of variation.

Latent vectors capture meaningful information





Entangled



Disentangled

Goal of InfoGAN

- Learn a structured and **disentangled latent space** where each latent variable controls a distinct, interpretable factor.
- Encourage the generator to **encode meaningful latent codes** that correspond to different aspects of the generated data.

- **Noise vector z :** Unstructured noise, as in standard GANs.
- **Latent code c :** Structured, interpretable variables that control specific factors.

$$G(z, c) \rightarrow x$$

- The generator $G(z, c)$ takes both z and c as input.
- c is explicitly encouraged to **control meaningful factors** by maximizing **mutual information** between c and the generated sample x .

InfoGAN

1. Generator $G(z, c)$:

- Takes **random noise** $z \sim p_z$ and **structured code** $c \sim p(c)$.
- Generates a sample $x = G(z, c)$.
- The goal is to embed meaningful information about c into the generated sample.

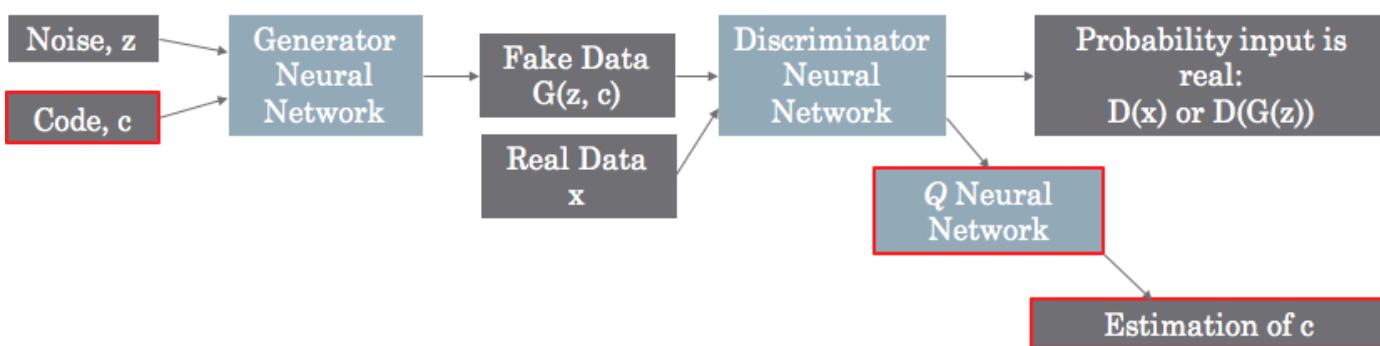
2. Discriminator $D(x)$:

- Tries to distinguish between real and fake samples.
- Trained using the **standard GAN loss**.

3. Auxiliary Network $Q(c | x)$:

- Learns to **recover** the latent code c from the generated sample.
- Ensures that the generator encodes interpretable information into the samples.

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



Idea of mutual information

Mutual information measures how much **knowing one variable** reduces uncertainty about another.

$$\mathbb{I}(c; x) = H(c) - H(c | x)$$

Where:

- $H(c)$ is the entropy of the latent code.
 - $H(c | x)$ is the conditional entropy of c given the generated sample.
-
- High mutual information means that knowing the generated sample x tells us a lot about c , ensuring that c is **recoverable** from x .
 - To maximize $\mathbb{I}(c; x)$, the generator must **embed the information in c into the generated samples**.

Calculating MI directly is difficult

Since $\mathbb{I}(c; x)$ is hard to compute directly, InfoGAN introduces a **variational lower bound**:

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

Where:

- $Q(c | x)$ is an **auxiliary network** that predicts c from the generated sample.
- **Training Objective:** Maximize $\log Q(c | x)$ to ensure that c is **recoverable**.



(a) Azimuth (pose)

(b) Presence or absence of glasses



(c) Hair style

(d) Emotion

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 $D(x) \rightarrow 0, D(G(z)) \rightarrow 1$

To reframe this as a **margin loss**, we introduce a decision function:

$$f(x) = 2D(x) - 1$$

- $f(x)$ maps $D(x)$ from $[0, 1]$ to $[-1, 1]$.

In this view:

- For real samples, the goal is to **push them below 0**.
- For fake samples, the goal is to **push them above 0**.

$$\mathcal{L}_{\text{margin}} = \mathbb{E}_{x \sim p_{\text{data}}} [\max(0, 1 + f(x))] + \mathbb{E}_{z \sim p_z} [\max(0, 1 - f(G(z)))]$$

Least-Square GAN

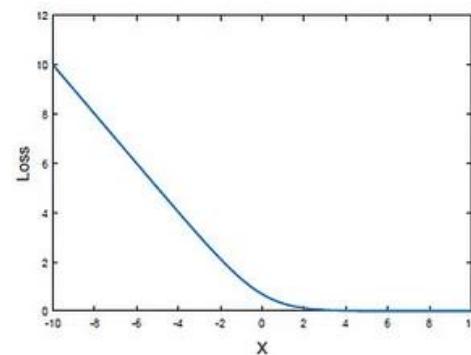
LSGAN minimizes the least squares error between the discriminator's output and the target values.

- For real samples:

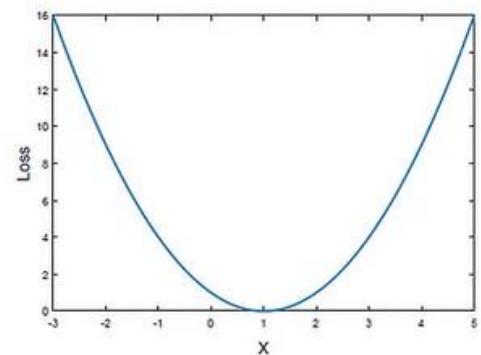
$$\mathcal{L}_D^{\text{real}} = \frac{1}{2} \mathbb{E}_{x \sim p_{\text{data}}} [(D(x) - 1)^2]$$

- For fake samples:

$$\mathcal{L}_D^{\text{fake}} = \frac{1}{2} \mathbb{E}_{z \sim p_z} [(D(G(z)))^2]$$



(a)



(b)

(a): The sigmoid cross entropy loss function. (b): The least squares loss function.

- The generator minimizes the least squares error to make fake samples look like real samples:

$$\mathcal{L}_G = \frac{1}{2} \mathbb{E}_{z \sim p_z} [(D(G(z)) - 1)^2]$$

Qualitative results



(a) Generated by LSGANs.



(b) Generated by DCGANs (Reported in [11]).



(c) Generated by EBGANs (Reported in [26]).



(a) Church outdoor.



(b) Dining room.

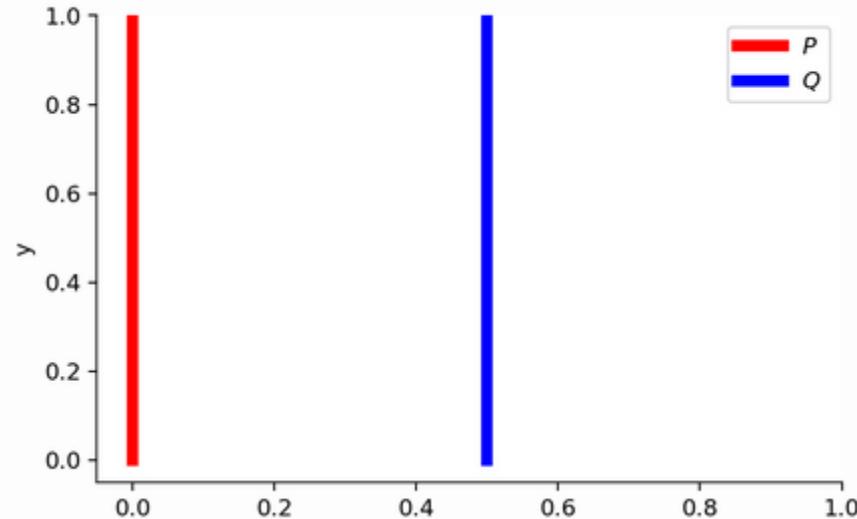


(c) Kitchen.



(d) Conference room.

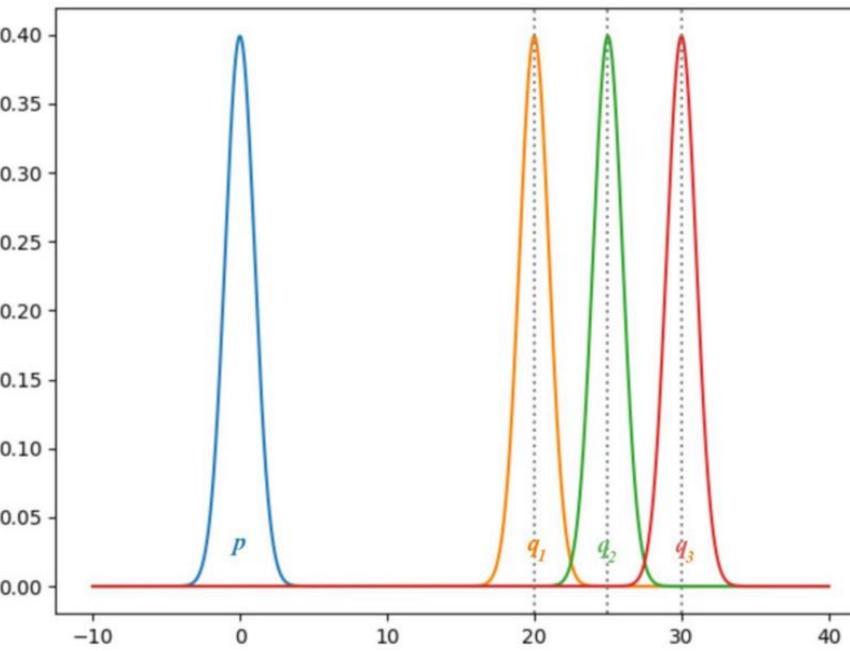
Calculate KL, JS



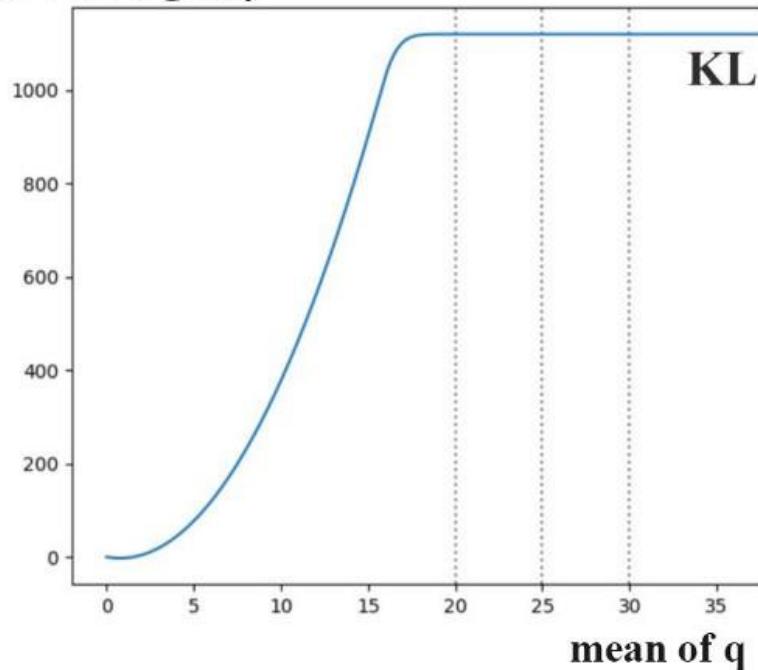
$$D_{KL}(P\|Q) = \sum_{x=0, y \sim U(0,1)} 1 \cdot \log \frac{1}{0} = +\infty$$

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

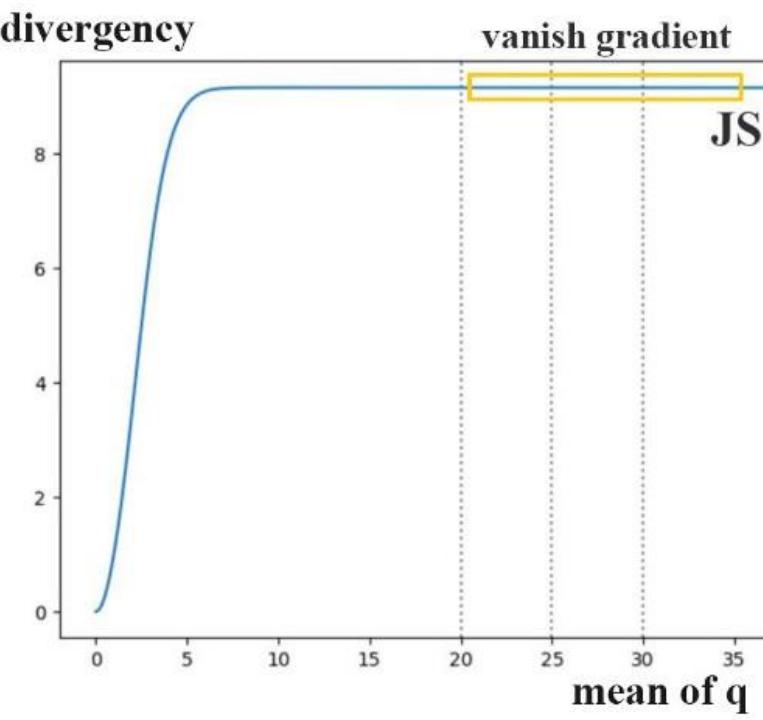
$$D_{JS}(P, Q) = \frac{1}{2} \left(\sum_{x=0, y \sim U(0,1)} 1 \cdot \log \frac{1}{1/2} + \sum_{x=0, y \sim U(0,1)} 1 \cdot \log \frac{1}{1/2} \right) = \log 2$$



KL-divergency

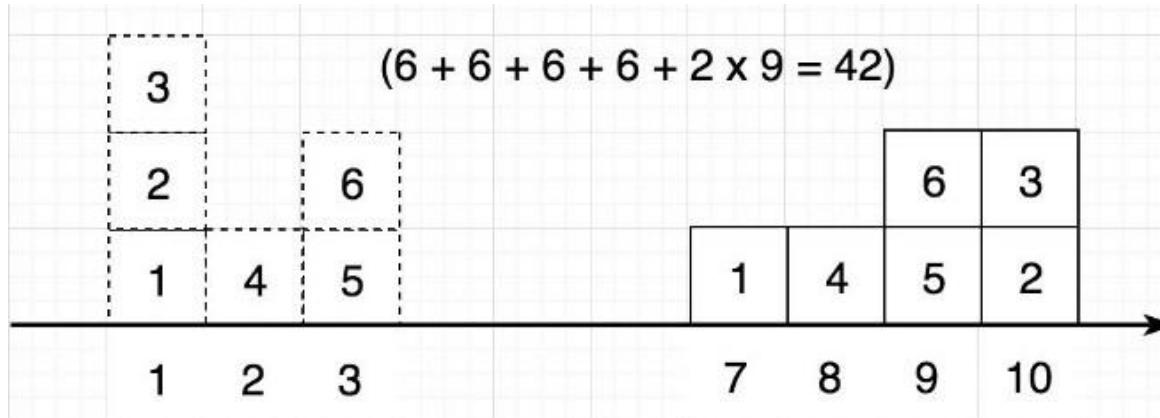


JS-divergency

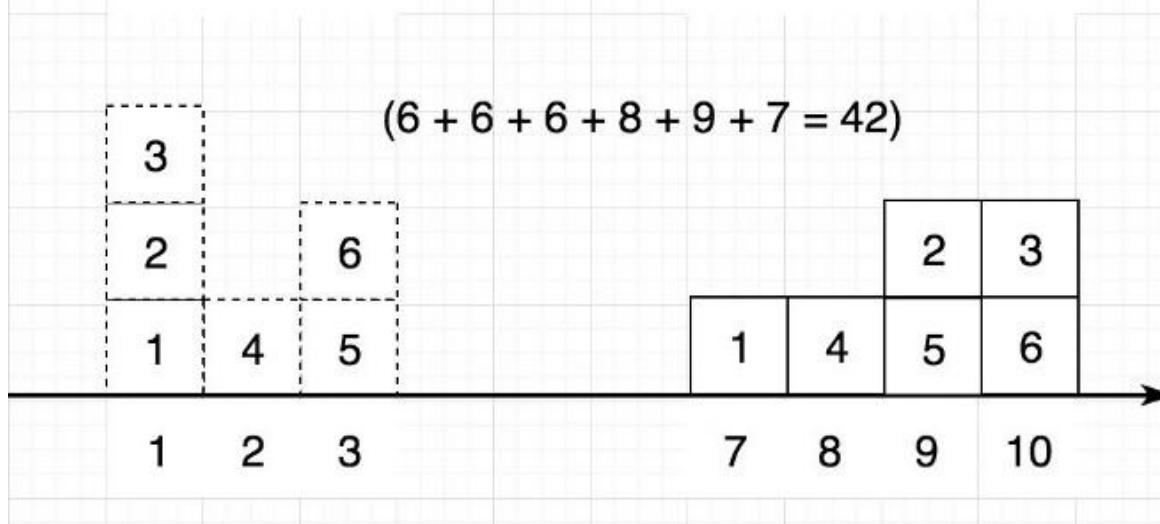


vanish gradient

EM distance

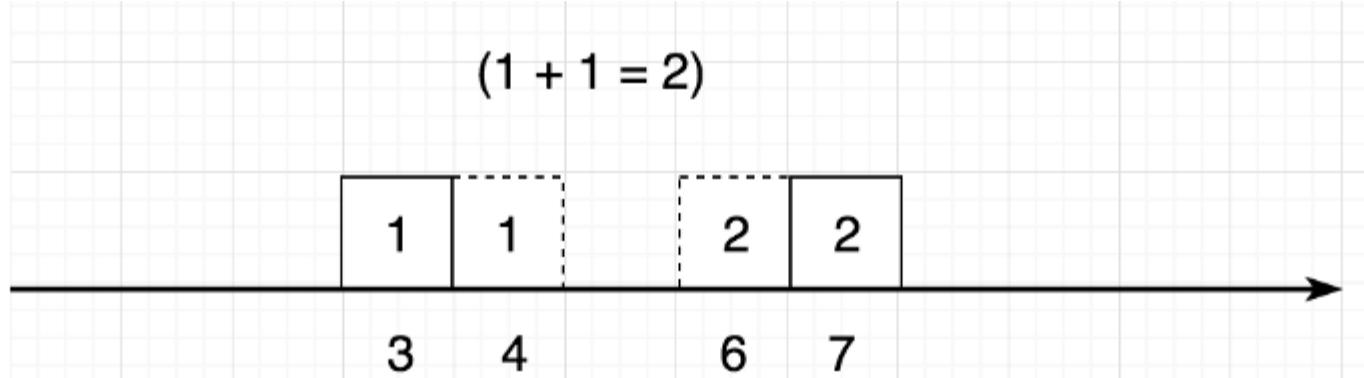


	7	8	9	10	
γ_1	1	1	0	0	2
	2	0	1	0	0
	3	0	0	2	0

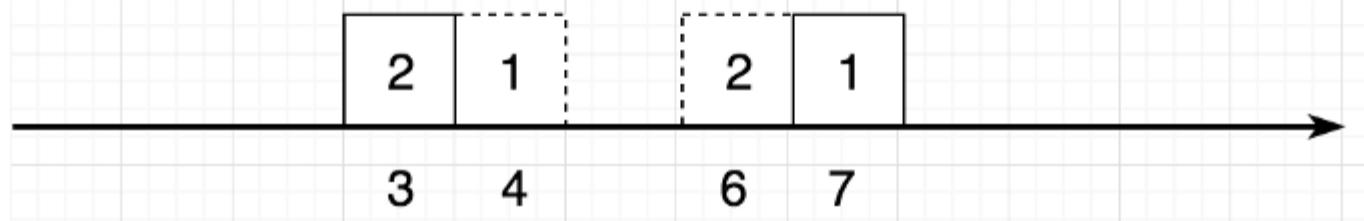


	7	8	9	10	
γ_2	1	1	0	1	1
	2	0	1	0	0
	3	0	0	1	1

$$(1 + 1 = 2)$$



$$(3 + 3 = 6)$$



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

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

$$|f(x_1) - f(x_2)| \leq |x_1 - x_2|.$$

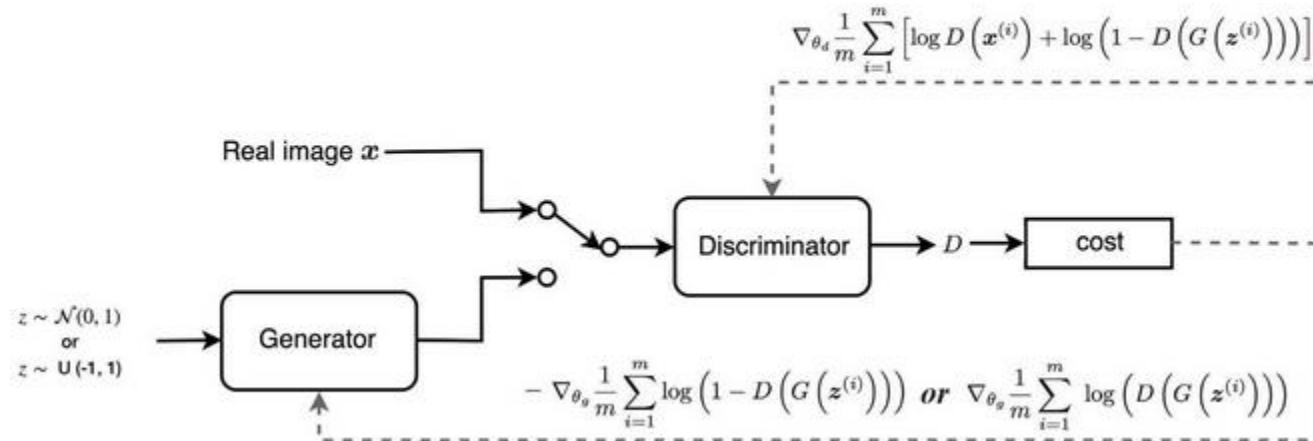
Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α , the learning rate. c , the clipping parameter. m , the batch size. n_{critic} , the number of iterations of the critic per generator iteration.

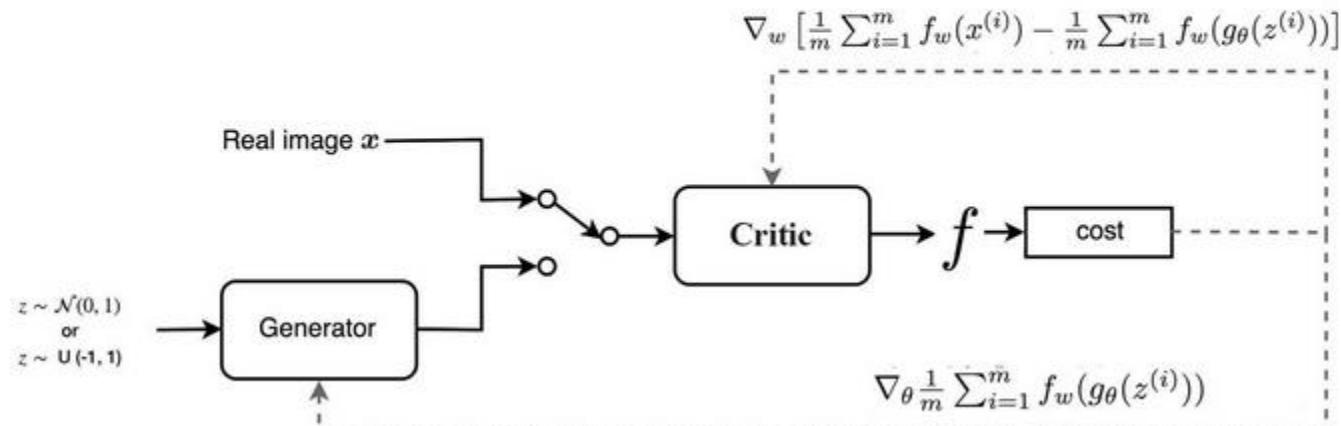
Require: : w_0 , initial critic parameters. θ_0 , initial generator's parameters.

- 1: **while** θ 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: $g_w \leftarrow \nabla_w [\frac{1}{m} \sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)}))]$
- 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:



WGAN



GAN

Discriminator/Critic

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[\log D\left(\mathbf{x}^{(i)}\right) + \log \left(1 - D\left(G\left(\mathbf{z}^{(i)}\right)\right)\right) \right]$$

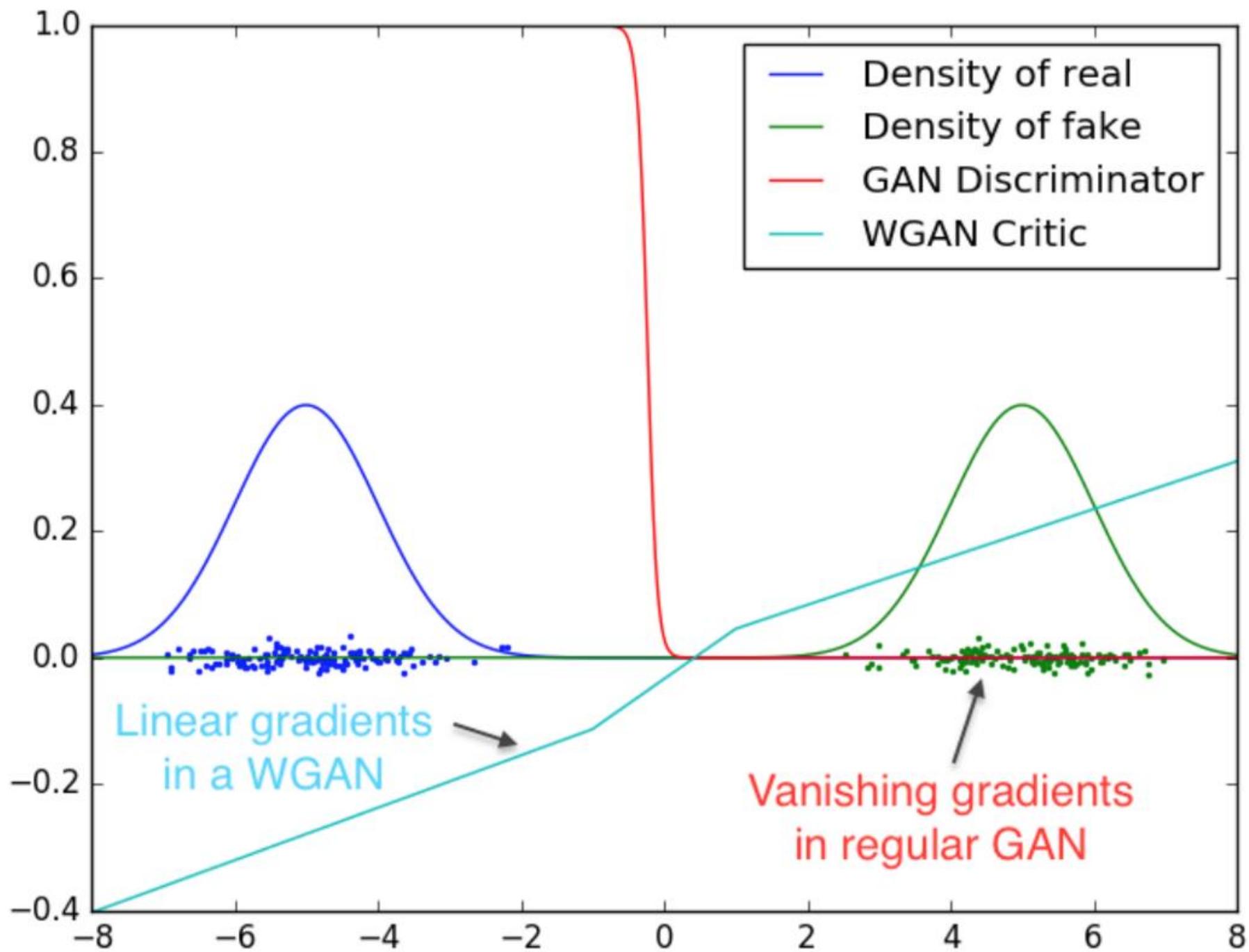
WGAN

$$\nabla_w \frac{1}{m} \sum_{i=1}^m \left[f(x^{(i)}) - f(G(z^{(i)})) \right]$$

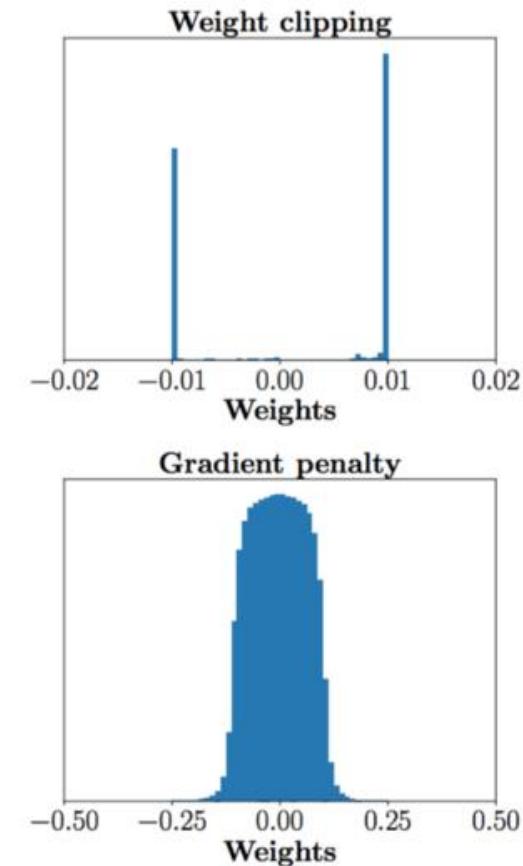
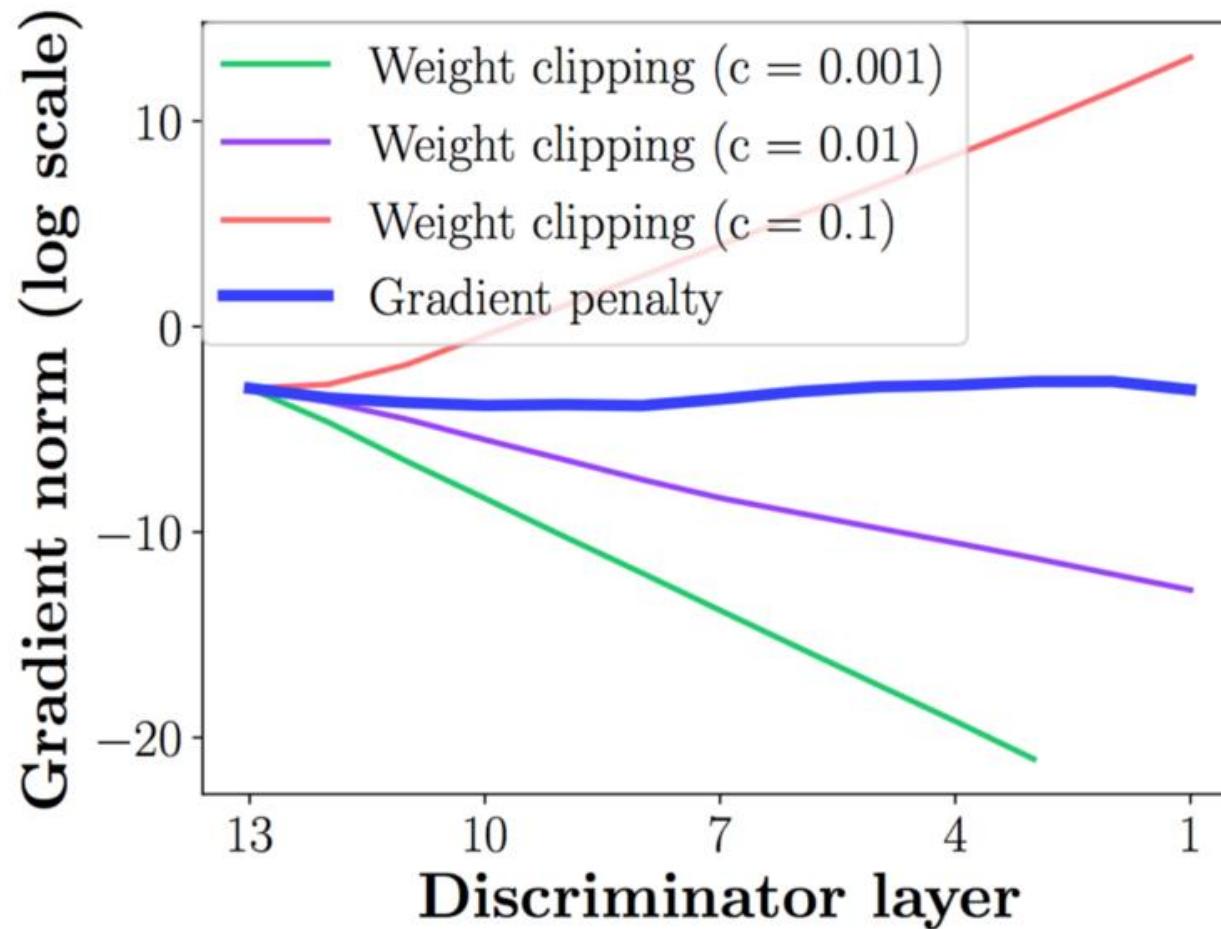
Generator

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log \left(D\left(G\left(\mathbf{z}^{(i)}\right)\right) \right)$$

$$\nabla_{\theta} \frac{1}{m} \sum_{i=1}^m f(G(z^{(i)}))$$



Problem with clipping parameter



W-GAN GP

if f^* is differentiable , $\pi(x = y) = 0$, and $x_t = tx + (1 - t)y$ with $0 \leq t \leq 1$,

it holds that $\mathbb{P}_{(x,y) \sim \pi} \left[\nabla f^*(x_t) = \frac{y-x_t}{\|y-x_t\|} \right] = 1$.

$$L = \underbrace{\mathbb{E}_{\tilde{\mathbf{x}} \sim \mathbb{P}_g} [D(\tilde{\mathbf{x}})] - \mathbb{E}_{\mathbf{x} \sim \mathbb{P}_r} [D(\mathbf{x})]}_{\text{Original critic loss}} + \lambda \underbrace{\mathbb{E}_{\hat{\mathbf{x}} \sim \mathbb{P}_{\hat{\mathbf{x}}}} \left[(\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})\|_2 - 1)^2 \right]}_{\text{Our gradient penalty}}.$$

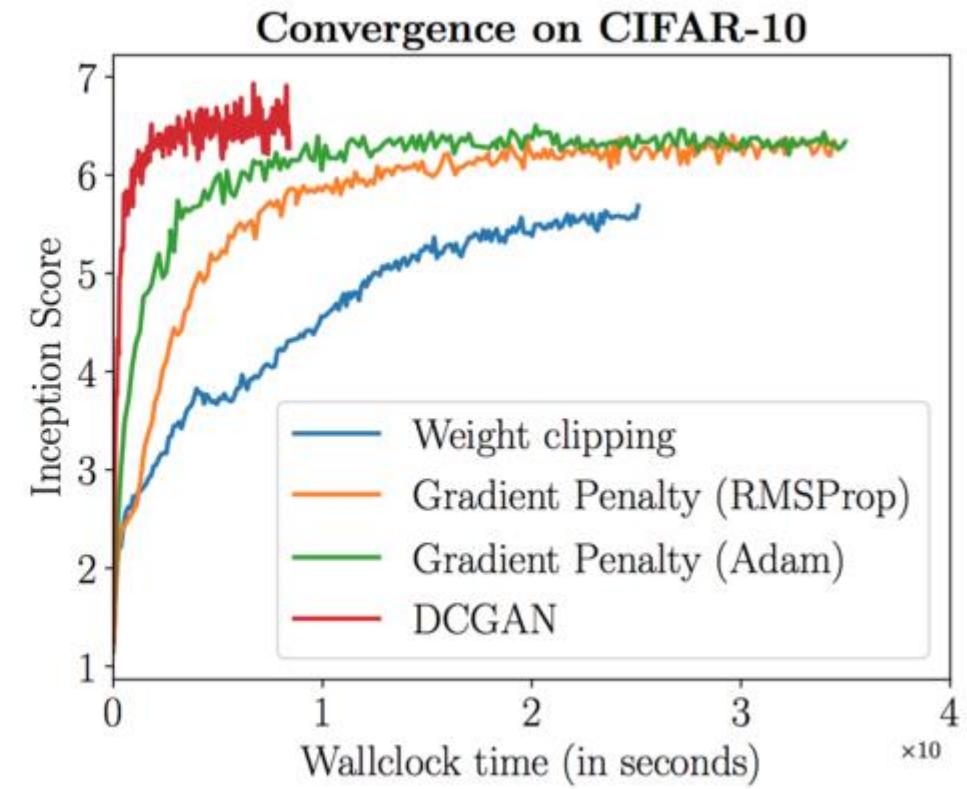
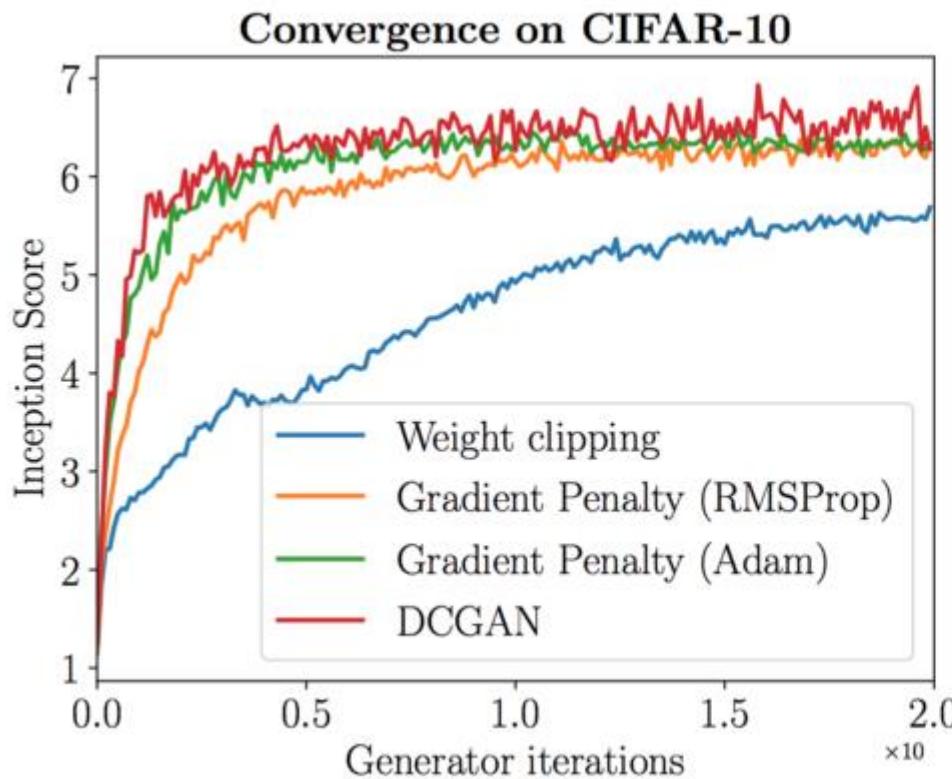
Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

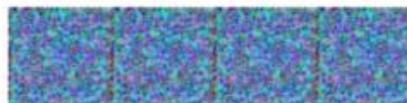
Require: The gradient penalty coefficient λ , the number of critic iterations per generator iteration n_{critic} , the batch size m , Adam hyperparameters α, β_1, β_2 .

Require: initial critic parameters w_0 , initial generator parameters θ_0 .

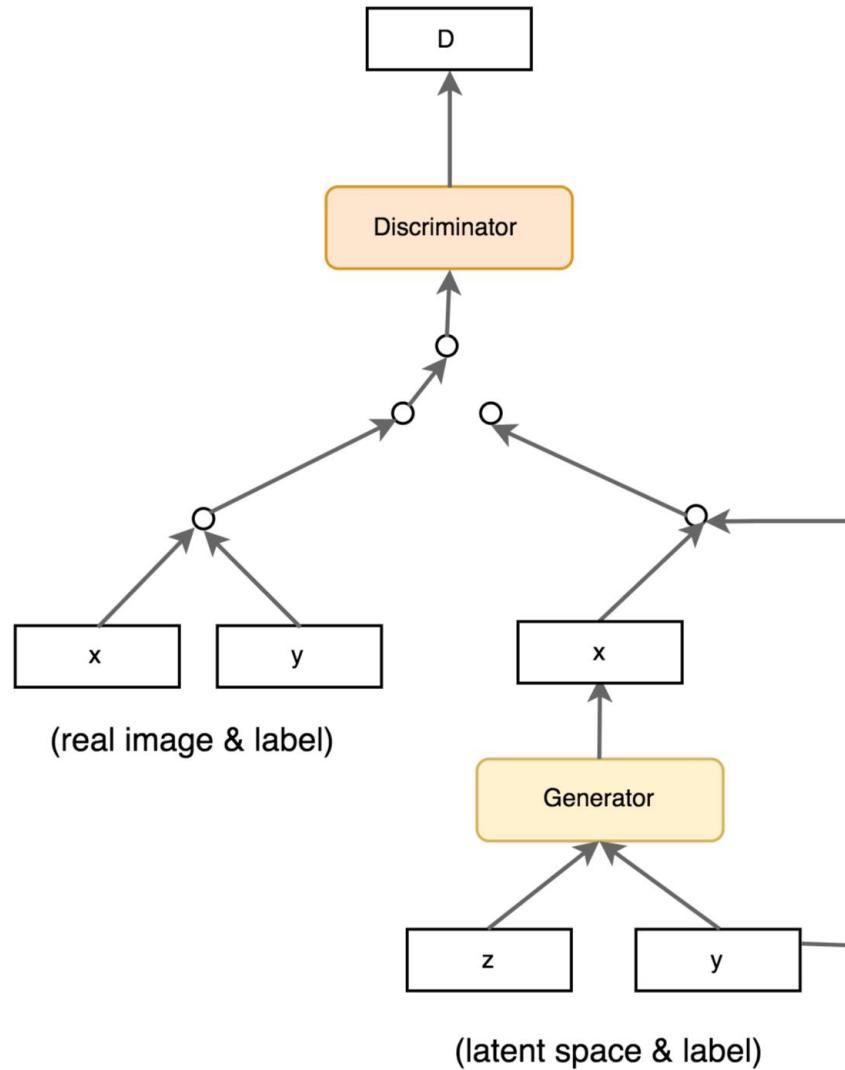
```
1: while  $\theta$  has not converged do
2:   for  $t = 1, \dots, n_{\text{critic}}$  do
3:     for  $i = 1, \dots, m$  do
4:       Sample real data  $\mathbf{x} \sim \mathbb{P}_r$ , latent variable  $\mathbf{z} \sim p(\mathbf{z})$ , a random number  $\epsilon \sim U[0, 1]$ .
5:        $\tilde{\mathbf{x}} \leftarrow G_\theta(\mathbf{z})$ 
6:        $\hat{\mathbf{x}} \leftarrow \epsilon \mathbf{x} + (1 - \epsilon) \tilde{\mathbf{x}}$ 
7:        $L^{(i)} \leftarrow D_w(\tilde{\mathbf{x}}) - D_w(\mathbf{x}) + \lambda(\|\nabla_{\hat{\mathbf{x}}} D_w(\hat{\mathbf{x}})\|_2 - 1)^2$ 
8:     end for
9:      $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^m L^{(i)}, w, \alpha, \beta_1, \beta_2)$ 
10:    end for
11:    Sample a batch of latent variables  $\{\mathbf{z}^{(i)}\}_{i=1}^m \sim p(\mathbf{z})$ .
12:     $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^m -D_w(G_\theta(\mathbf{z})), \theta, \alpha, \beta_1, \beta_2)$ 
13: end while
```

WGAN GP experiments



DCGAN**LSGAN****WGAN (clipping)****WGAN-GP (ours)**Baseline (G : DCGAN, D : DCGAN) G : No BN and a constant number of filters, D : DCGAN G : 4-layer 512-dim ReLU MLP, D : DCGANNo normalization in either G or D Gated multiplicative nonlinearities everywhere in G and D tanh nonlinearities everywhere in G and D 101-layer ResNet G and D 

CGAN – using the labels



InfoGAN – labels as latent variables

