Generative Adversarial Network (GAN)

DEEP LEARNING COURSE – 2023

E. FATEMIZADEH

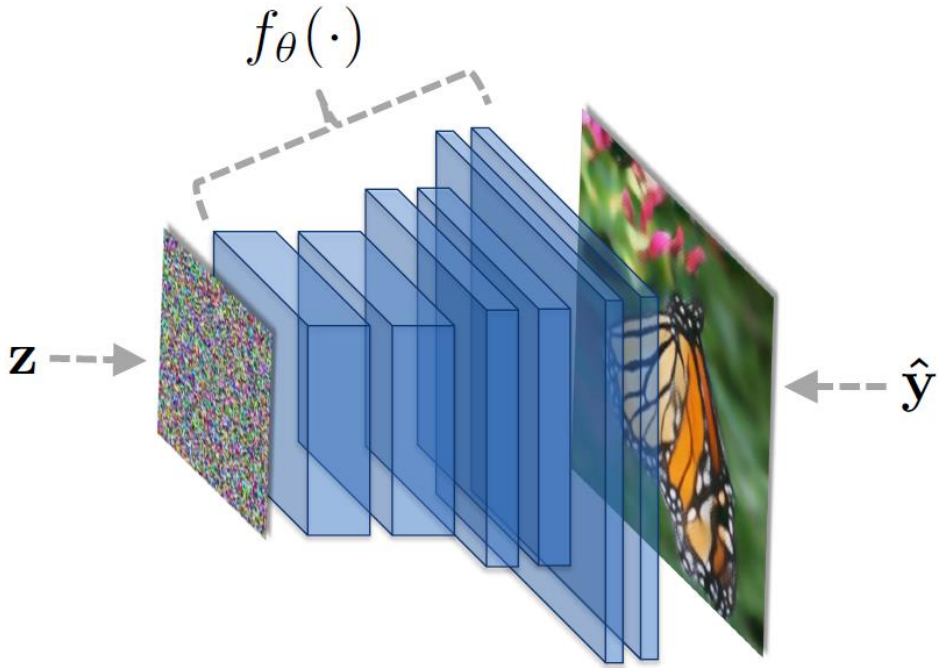
Sharif University of Technology, School of Electrical Engineering

GAN Final Goal

- Example: From a large collection of images of faces, can a network learn to generate new portrait.

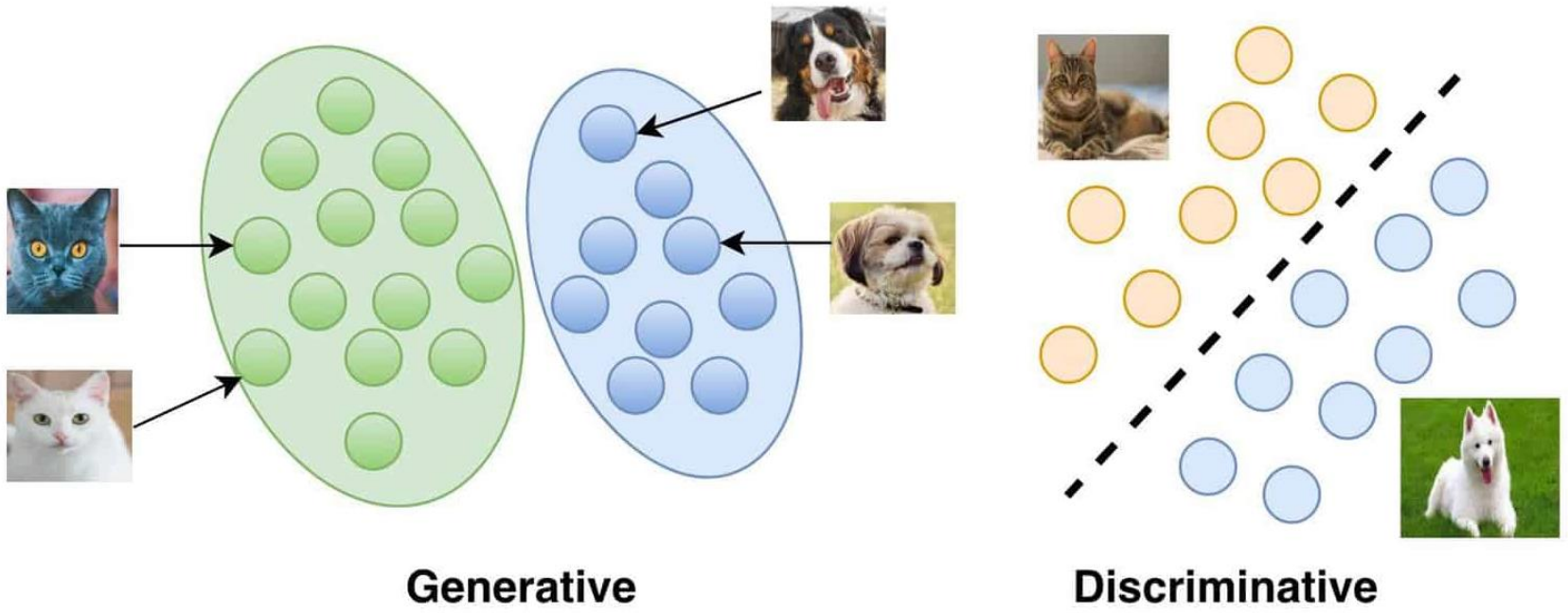
GAN Final Goal

- How to generate new images



Generative Adversarial Network - **GAN**

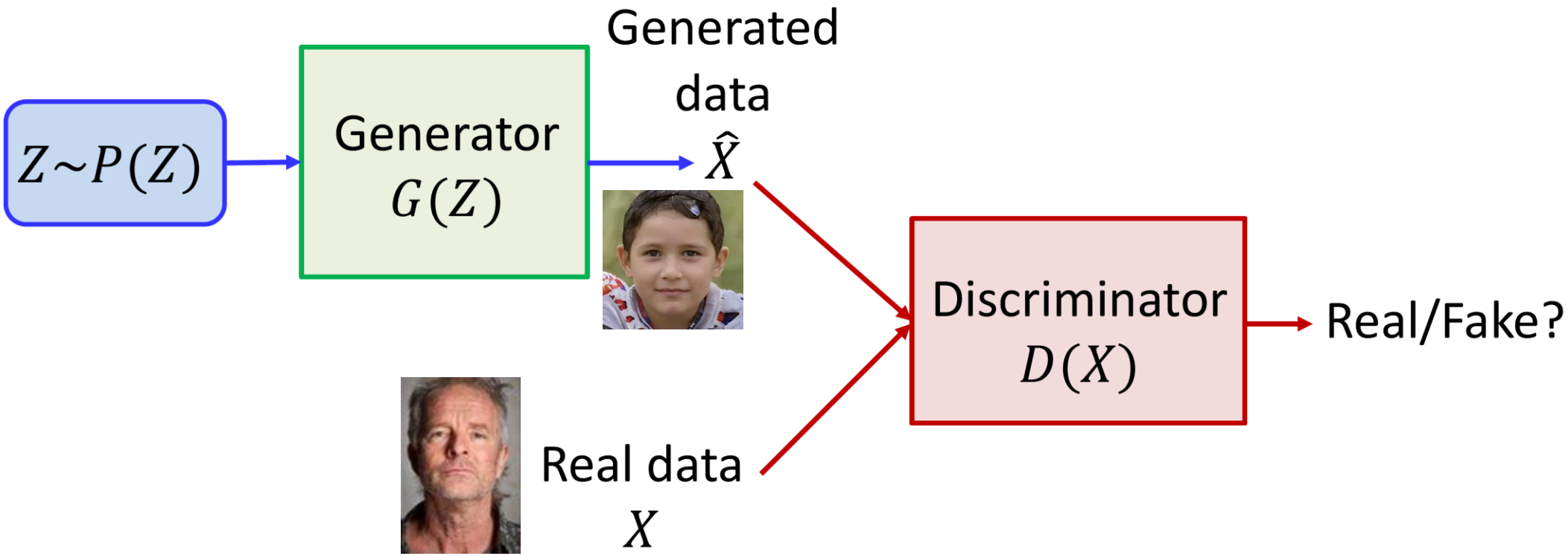
- Just to Remember:



Generative Adversarial Network - GAN

- Generative models: Learn a generative model, which generate data similar to the training data .
- Adversarial training: Trained in an adversarial setting, GANS are made up of two competing networks (adversaries) that are trying beat each other. A «game» is being played between the two networks.
- Network: Use Deep Neural Networks

GAN Block Diagram



Preliminary

- A Question:
  - Why Security System (Hardware/Software/policy) Evolve?
    - Banknote
    - Car Anti Theft
    - 2-step verification in Gmail/Telegram/Facebook/Whatsapp/….
    - Anti Theft Door and Lock
    - ….

Preliminary

- A Game Theory Related Question:
  - Why Security System (Hardware/Software/policy) Evolve?

    A Game between Good-Bad

  - Zero-Sum Game:

    Every amount that one player **wins** must be **lost** by another player

An Example from Game Theory

- Two manufacturers, producing the same kind of good in quantities of q_1 and q_2, respectively
- Market Clearing Price*: $p = A - q_1 - q_2$
- Cost of production is c_1 and c_2 respectively
- Profit for each one:

$$u_1(q_1, q_2) = q_1(A - q_1 - q_2) - q_1 c_1 = q_1(A - c_1 - q_1 - q_2)$$
$$u_2(q_1, q_2) = q_2(A - q_1 - q_2) - q_2 c_2 = q_2(A - c_2 - q_1 - q_2)$$

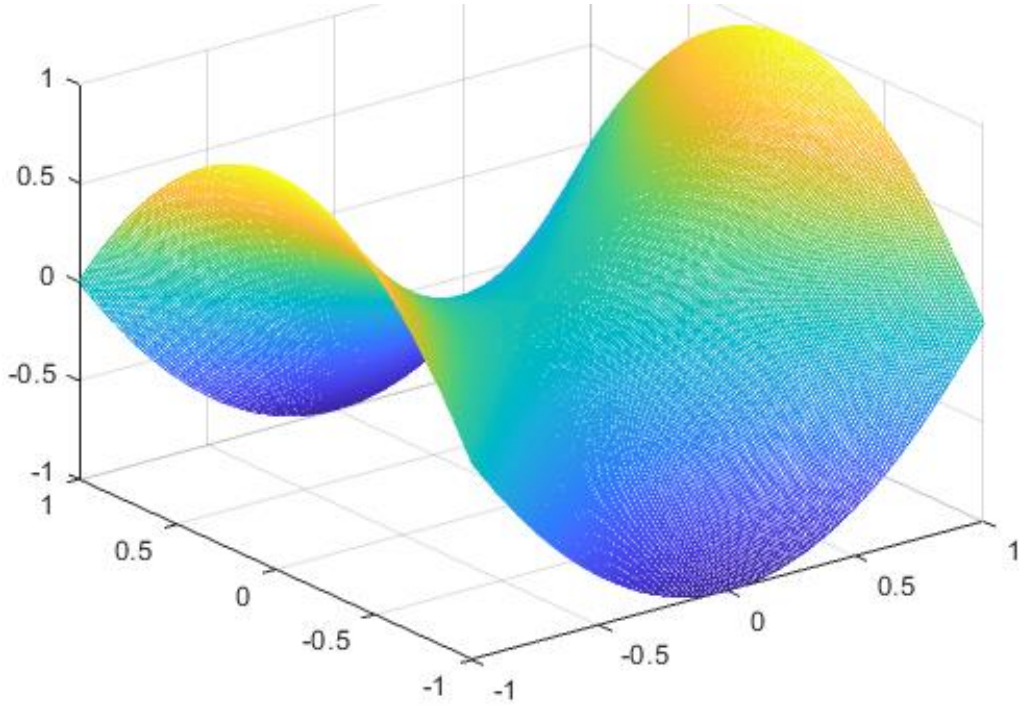
An Example from Game Theory

- Each one want to maximize its own profits
- $u_1(q_1, q_2) = q_1(A - q_1 - q_2) - q_1 c_1 = q_1(A - c_1 - q_1 - q_2)$
- $u_2(q_1, q_2) = q_2(A - q_1 - q_2) - q_2 c_2 = q_2(A - c_2 - q_1 - q_2)$
- $\frac{\partial u_1(q_1,q_2)}{\partial q_1} = 0 \rightarrow q_1 = \frac{A - c_1 - q_2}{2}$
- $\frac{\partial u_2(q_1,q_2)}{\partial q_2} = 0 \rightarrow q_2 = \frac{A - c_2 - q_1}{2}$
- $q_1^* = \frac{A + c_2 - 2c_1}{3}, \quad q_2^* = \frac{A + c_1 - 2c_2}{3}$
- This is Nash equilibrium

A miniMax problem

- Maximizing your own gain, and minimizing your opponent's gain
- Example:

$$\min_{x}\max_{y}(x^2 - y^2)$$



Again Divergence Measure

- KL (Kullback–Leibler) Divergence

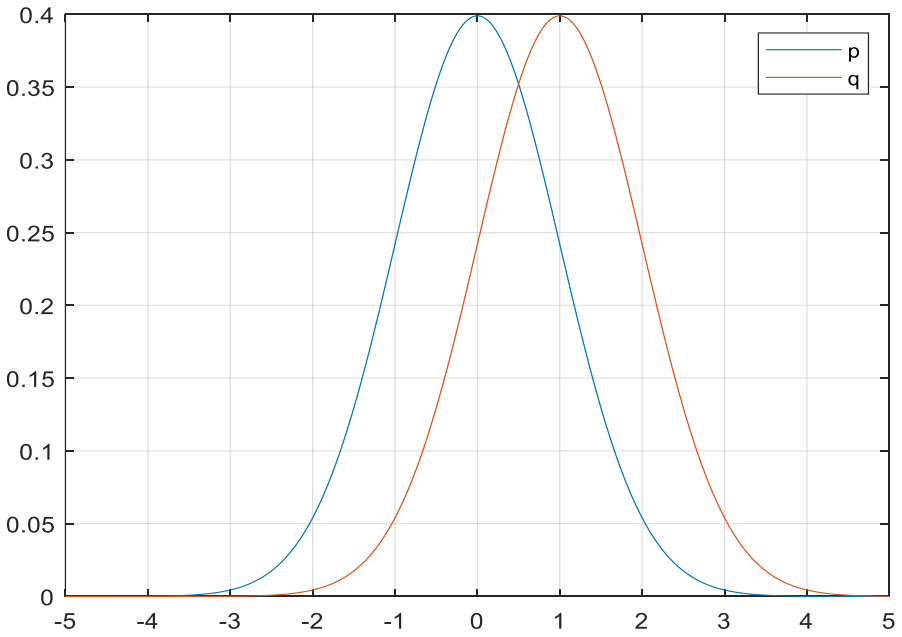
$$D_{KL}(p\|q) = \int_x p(x) log\left(\frac{p(x)}{q(x)}\right) dx$$

- Properties:
  - Asymmetric, $D_{KL}(p\|q) \neq D_{KL}(q\|P)$
  - For $p(x)$ close to zero and non-zero $q(x)$: $q(x)$ effect discarded!
  - Unbounded, $0 < D_{KL}(p\|q) \leq \infty$

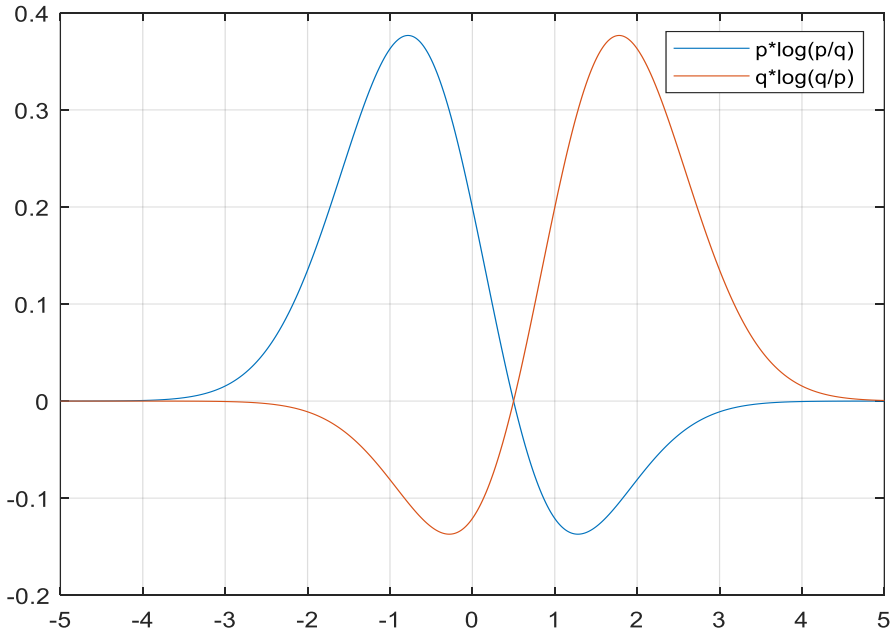
Again Divergence Measure

- Two Gaussian: $p \sim N(0,1)$ and $q \sim N(1,1)$

$p(x)$ *and* $q(x)$



$p(x)log\left(\frac{p(x)}{q(x)}\right)$ *and* $q(x)log\left(\frac{q(x)}{p(x)}\right)$



JS (Jensen–Shannon) Divergence

- JS (Jensen–Shannon) Divergence

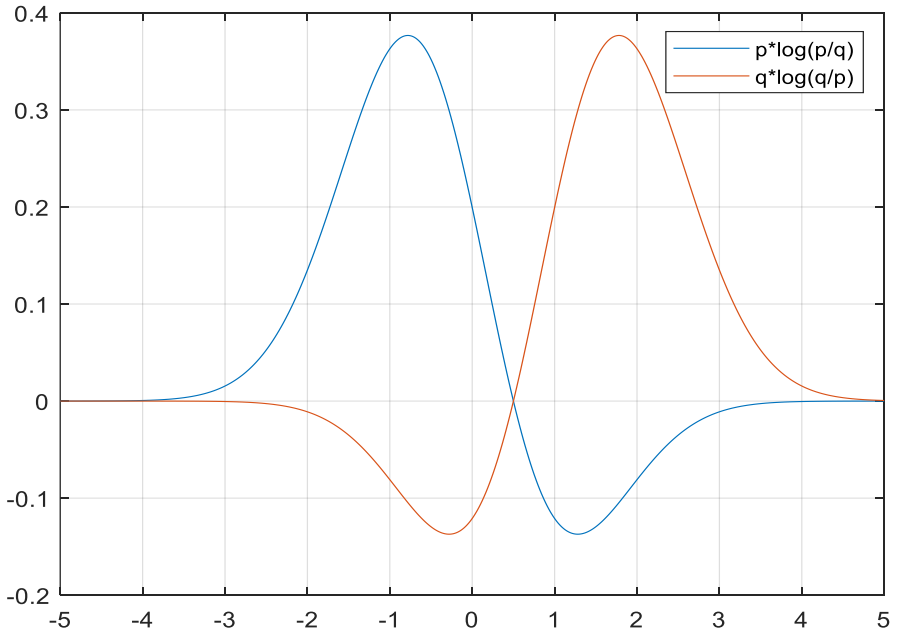
$$D_{JS}(p\|q) = \frac{1}{2} D_{KL}\left(p\|\frac{p+q}{2}\right) + \frac{1}{2} D_{KL}\left(q\|\frac{p+q}{2}\right)$$

- Symmetric and Bounded between 0 and log(2)

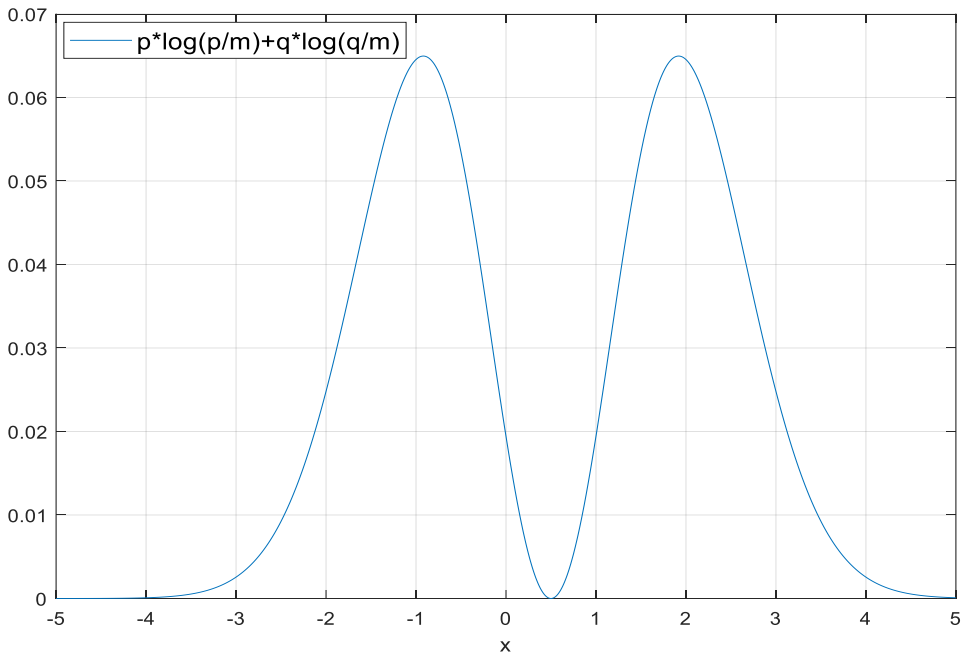
JS (Jensen–Shannon) Divergence

- JS (Jensen–Shannon) Divergence plot, $m(x) = \big(p(x) + q(x)\big)/2$

$$p(x)log\left(\frac{p(x)}{q(x)}\right) \text{ and } q(x)log\left(\frac{q(x)}{p(x)}\right)$$



$$p(x)log\left(\frac{p(x)}{m(x)}\right) + q(x)log\left(\frac{q(x)}{m(x)}\right)$$



GAN Application

- Generation of samples from some distribution
- Create Art
- Image-to-Image translation (Aerial images to Maps)
- Data Augmentation

GAN Application

- Same as VAE/CVAE but better performance!

$z = (0.3, 0.2, -0.6, ...) \xrightarrow{g(z)}$ 

$z \sim \mathcal{N}(0, 1)$

or

$z \sim U(-1, 1)$

$z = (-0.1, 0.1, 0.2, ...) \xrightarrow{g(z)}$ 

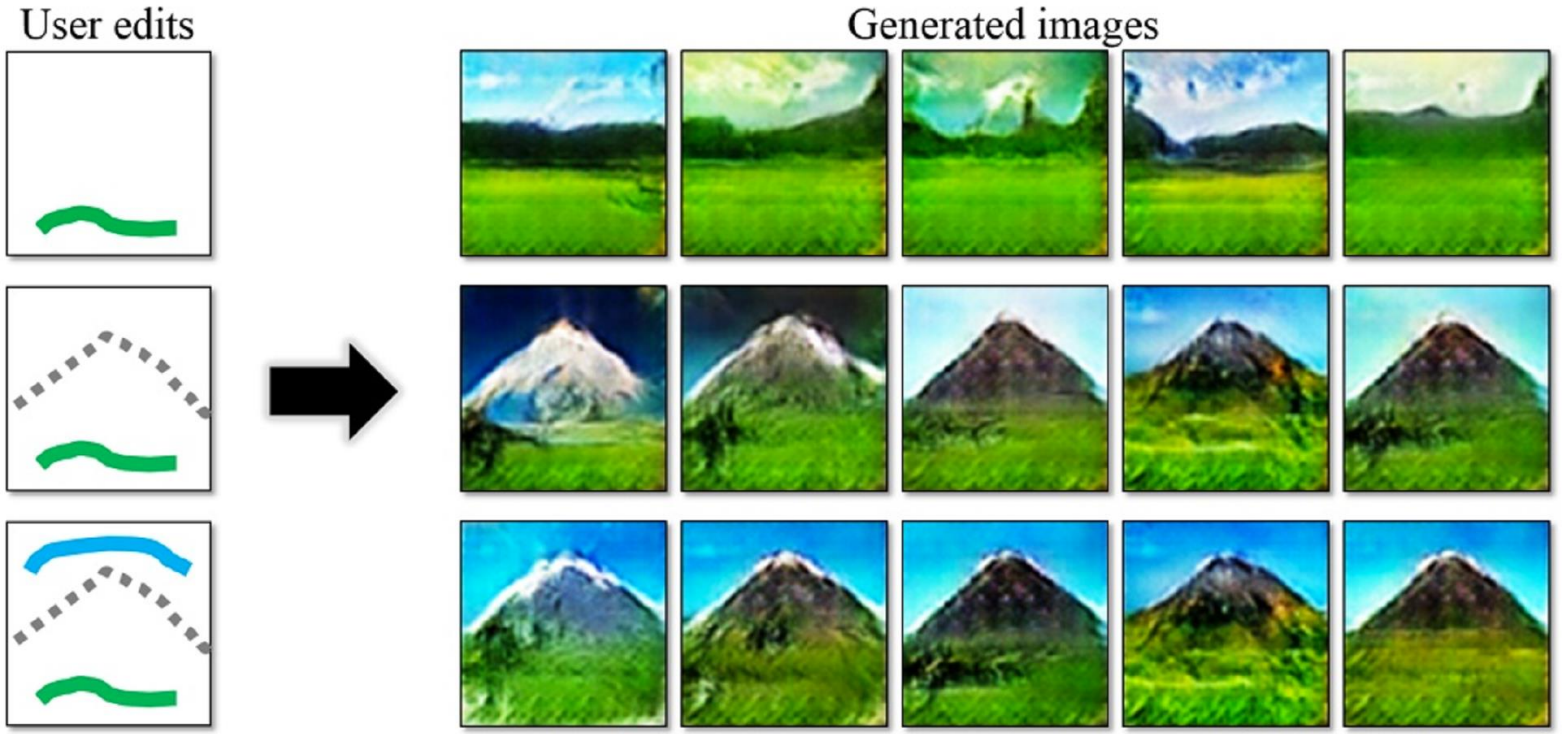
Examples

- Some results of GAN (Try https://thispersondoesnotexist.com)

Which one is Computer generated?

Examples

- Some results of GAN



Mathematical Preliminary

- Implicit Density Models
- Goal: Compare true probability $p(\boldsymbol{x})$ with approximate $q(\boldsymbol{x}|z)$
- n samples from true pdf and n' samples from model
- Ratio rest:

$$r(\boldsymbol{x}) = \frac{p(\boldsymbol{x})}{q(\boldsymbol{x}|z)}$$

Mathematical Preliminary

- Convert to classification problem (label 1/0 for true/model):

$$r(\boldsymbol{x}) = \frac{p(\boldsymbol{x})}{q(\boldsymbol{x}|z)} = \frac{p(\boldsymbol{x}|y=1)}{p(\boldsymbol{x}|y=0)} = \frac{p(y=1|\boldsymbol{x})p(y=0)}{p(y=0|\boldsymbol{x})p(y=1)}$$

- $p(y=1) = \pi$

$$r(x) = \frac{p(y=1|\boldsymbol{x})}{p(y=0|\boldsymbol{x})}\frac{1-\pi}{\pi}$$

- $\frac{1-\pi}{\pi} \approx \frac{n'}{n}$

Mathematical Preliminary

- Class weighted ratio for a binary classifier

$$r(\boldsymbol{x}) = \frac{p(y=1|\boldsymbol{x})}{p(y=0|\boldsymbol{x})}\frac{1-\pi}{\pi}$$

- Let define *Discrimination* function: $D(x;\omega_D) = p(y=1|\boldsymbol{x})$
- A *Bernoulli* scoring function:

$$V(\omega_D) = E_{p(\boldsymbol{x},y)}\left[ylogD(\boldsymbol{x};\omega_D) + (1-y)log\left(1-D(\boldsymbol{x};\omega_D)\right)\right]$$

Mathematical Preliminary

- $\pi = p(y = 1), 1 - \pi = p(y = 0)$

$$V(\omega_D) = E_{p(\boldsymbol{x},y)}\left[ylogD(\boldsymbol{x};\omega_D) + (1-y)log\left(1 - D(\boldsymbol{x};\omega_D)\right)\right]$$

$$V(\omega_D) = \pi E_{p(\boldsymbol{x}|y=1)}[logD(\boldsymbol{x};\omega_D)] + (1-\pi)E_{p(\boldsymbol{x}|y=0)}\left[log\left(1 - D(\boldsymbol{x};\omega_D)\right)\right]$$

- This is a discriminator that learn to indicate real or fake (model) data

Mathematical Preliminary

- Now let switch to a **Generator** network to create fake data/image!
- Like as VAE, using a latent variable z, Generator produce $G(z;\omega_G)$
- Loss function become:

$$V(\omega_D) = \pi E_{p(\boldsymbol{x}|y=1)}[logD(\boldsymbol{x};\omega_D)] + (1-\pi)E_{p(\boldsymbol{x}|y=0)}\left[log\left(1-D(\boldsymbol{x};\omega_D)\right)\right]$$

$$\mathcal{v}(\omega_D,\omega_G) = \pi E_{p(\boldsymbol{x})}[logD(\boldsymbol{x};\omega_D)] + (1-\pi)E_{p(\boldsymbol{z})}\left[log\left(1-D(G(z;\omega_G);\omega_D)\right)\right]$$

Mathematical Preliminary

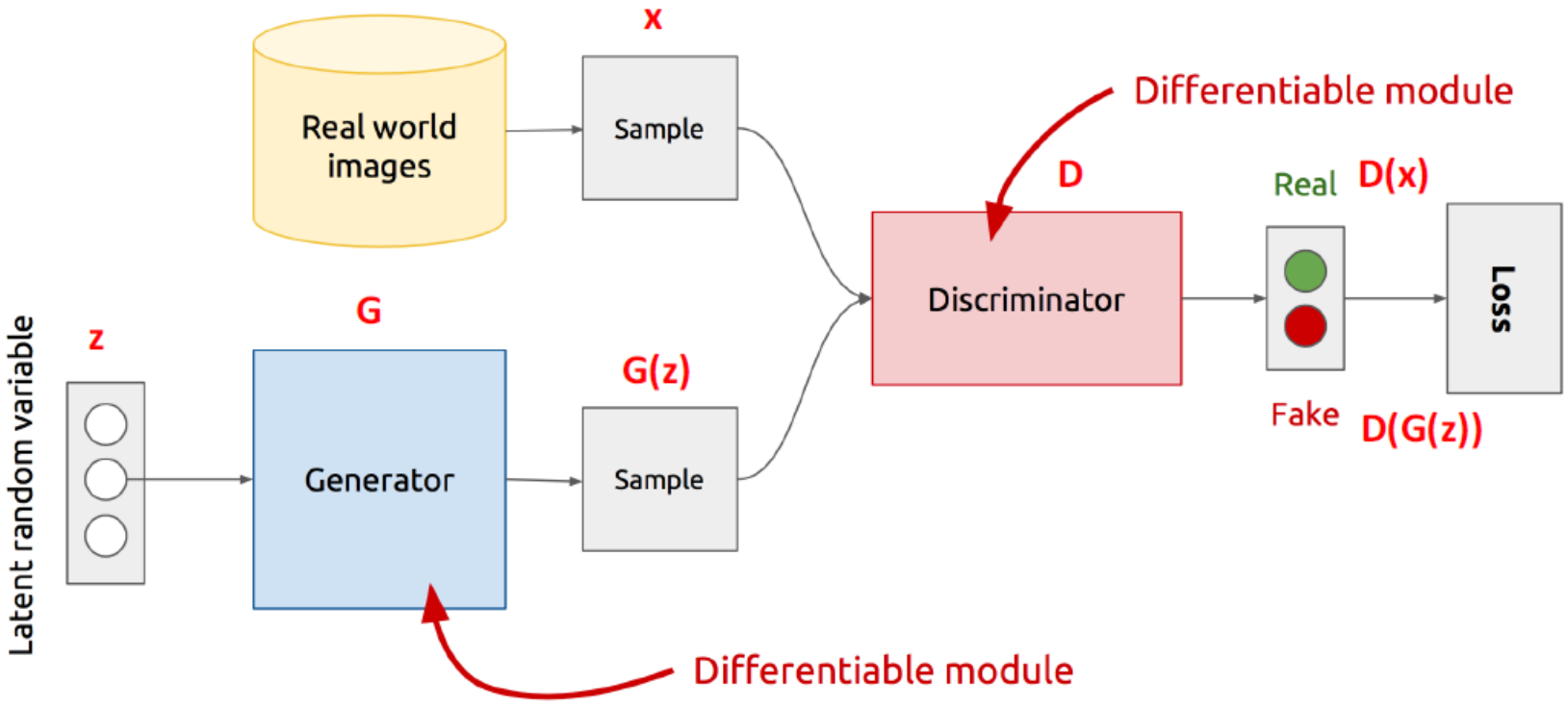
- Final Loss (in many cases π=0.5):

$$\mathcal{v}(\omega_D, \omega_G) = \pi E_{p(\boldsymbol{x})}[logD(\boldsymbol{x}; \omega_D)] + (1-\pi)E_{p(\boldsymbol{z})}\left[log\left(1 - D(G(z; \omega_G); \omega_D)\right)\right]$$

$$\mathcal{v}(\omega_D, \omega_G) = \pi E_{p(\boldsymbol{x})}[logD(\boldsymbol{x}; \omega_D)] + (1-\pi)E_{q(\boldsymbol{x}|z)}\left[log\left(1 - D(\boldsymbol{x}; \omega_D)\right)\right]$$

GAN Architecture

- A macro view



GAN as Game

- **G** and **D** play against each other during the training process:
  - **G** is trying to trick the discriminator
  - **D** is trying not to be cheated.
- Notation:
  - p_z: Data distribution over noise input z, usually just uniform
  - p_g: The generator's distribution over data $\boldsymbol{x}$
  - p_r: Data distribution over real sample $\boldsymbol{x}$

GAN as Game

- **G** is trying to trick the discriminator
- **D** is trying not to be cheated.
- Discriminator **D**'s try to maximize (real data) the following:

$$E_{\boldsymbol{x}\sim p_r(\boldsymbol{x})}[logD(\boldsymbol{x})]$$

- Discriminator **D**'s try to maximize (fake data) the following:

$$E_{\mathrm{z}\sim p_z(z)}\left[log\left(1-D\left(G(z)\right)\right)\right]$$

- Generator **G**'s try to minimize (fake data) the following:

$$E_{\mathrm{z}\sim p_z(z)}\left[log\left(1-D\left(G(z)\right)\right)\right]$$

GAN as Game

- **D** and **G** are playing a minimax game in which the following loss function should optimize:

$$\min_G \max_D L(D,G) = E_{\boldsymbol{x}\sim p_r(\boldsymbol{x})}[logD(\boldsymbol{x})] + E_{\mathrm{z}\sim p_z(z)}\left[log\left(1 - D\big(G(z)\big)\right)\right]$$

$$\min_G \max_D L(D,G) = E_{\boldsymbol{x}\sim p_r(\boldsymbol{x})}[logD(\boldsymbol{x})] + E_{\boldsymbol{x}\sim p_g(\boldsymbol{x})}\big[log\big(1 - D(\boldsymbol{x})\big)\big]$$

- First term has no impact on **G** during gradient descent updates

Optimal Discriminator

- Let's find optimal state:

$$\min_G \max_D L(D,G) = E_{x\sim p_r(x)}[logD(\boldsymbol{x})] + E_{z\sim p_z(z)}\left[log\left(1-D\big(G(z)\big)\right)\right]$$

$$\min_G \max_D L(D,G) = E_{\boldsymbol{x}\sim p_r(\boldsymbol{x})}[logD(\boldsymbol{x})] + E_{\boldsymbol{x}\sim p_g(\boldsymbol{x})}\big[log\big(1-D(\boldsymbol{x})\big)\big]$$

$$L(D,G) = \int_{\boldsymbol{x}} \Big(p_r(\boldsymbol{x})logD(\boldsymbol{x}) + p_g(\boldsymbol{x})log\big(1-D(\boldsymbol{x})\big)\Big)\,dx$$

- Using Calculus of variation:

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

Optimal Generator

- Optimal Discriminator is:

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

- For optimal Generator we should have: $p_r(x) = p_g(x)$, then optimal discriminator becomes 0.5 (Nash equilibrium)
- We will show optimal value of loss function: $-2\log(2)$

$$L(D, G) = \int_{\boldsymbol{x}} \Big(p_r(\boldsymbol{x}) log D(\boldsymbol{x}) + p_g(\boldsymbol{x}) log\big(1 - D(\boldsymbol{x})\big) \Big) dx$$

Optimal Generator

- Loss function for Optimal **Discriminator**:

$$L(G,D) = \int_x \left\{ p_r(\boldsymbol{x}) \log\left(\frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right) + p_g(\boldsymbol{x}) \log\left(1 - \frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right)\right\} d\boldsymbol{x}$$

$$L(G,D) = \int_x \left\{ p_r(\boldsymbol{x}) \log\left(\frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right) + p_g(\boldsymbol{x}) \log\left(\frac{p_g(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}\right)\right\} d\boldsymbol{x}$$

$$L(G,D) = \int_x \left\{ p_r(\boldsymbol{x}) \log\left(\frac{0.5p_r(\boldsymbol{x})}{0.5\left(p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})\right)}\right) + p_g(\boldsymbol{x}) \log\left(\frac{0.5p_g(\boldsymbol{x})}{0.5\left(p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})\right)}\right)\right\} d\boldsymbol{x}$$

$$L(G,D) = 2D_{JS}(p_r|p_g) - 2\log(2)$$

- Recall: $D_{JS}(p\|q) = \frac{1}{2}D_{KL}\left(p\|\frac{p+q}{2}\right) + \frac{1}{2}D_{KL}\left(q\|\frac{p+q}{2}\right)$

Optimal Discriminator and Ratio Test

- Back to ration test:

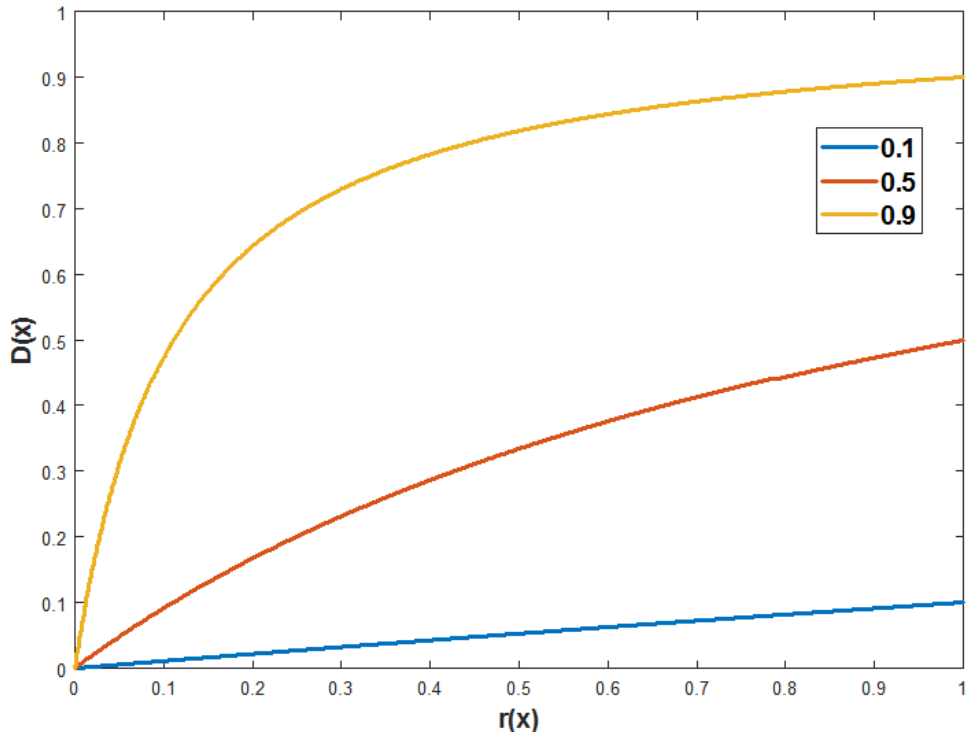
$$r(\boldsymbol{x}) = \frac{p(y=1|\boldsymbol{x})}{p(y=0|\boldsymbol{x})} \frac{1-\pi}{\pi}$$

$$D^*(x) = \frac{p_r(x)}{p_r(x)+p_g(x)} = \frac{p_r(x)/p_g(x)}{p_r(x)/p_g(x)+1} = \frac{\frac{\pi}{1-\pi} r(x)}{\frac{\pi}{1-\pi} r(x)+1}$$

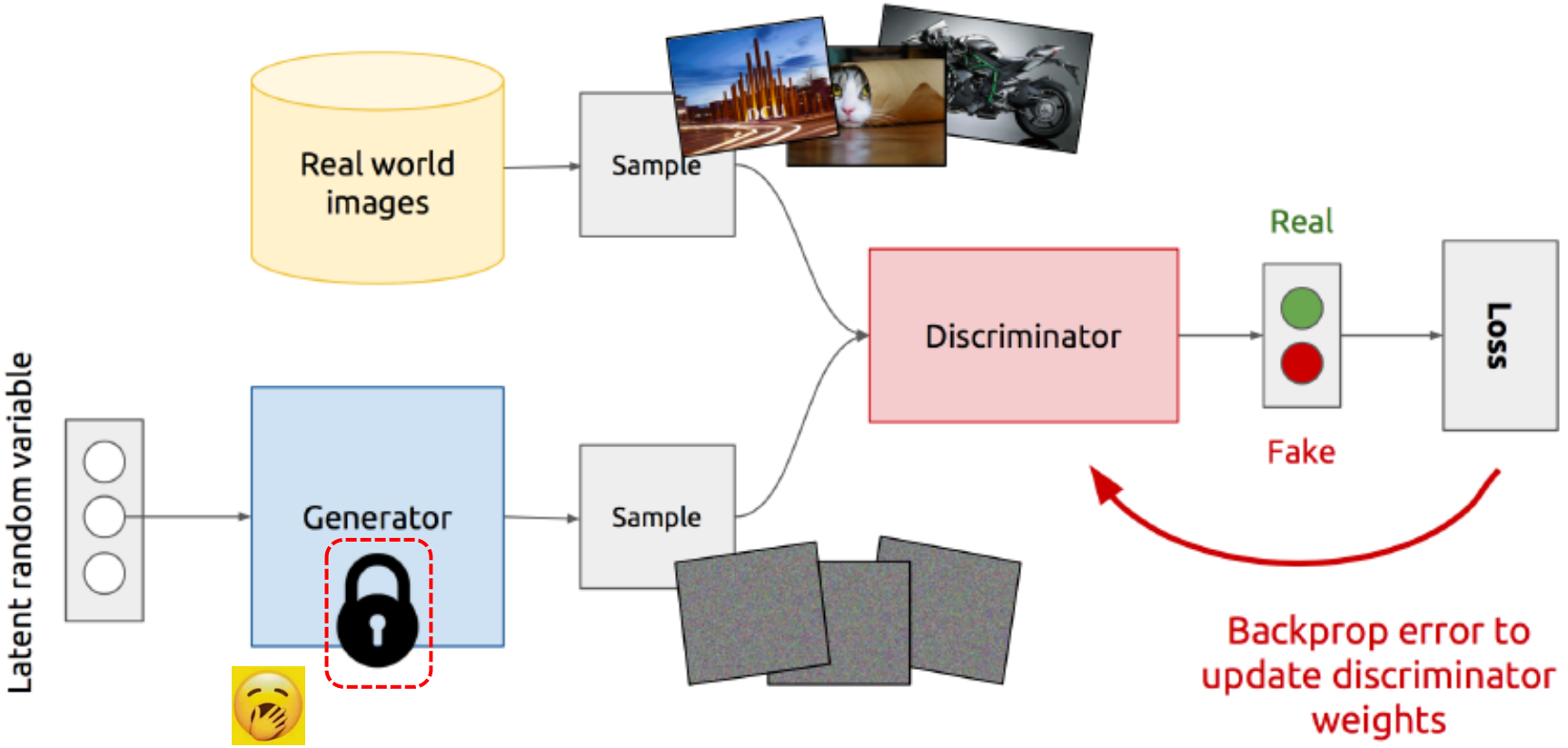
$$D^*(x) = \frac{\pi r(x)}{\pi r(x) + 1 - \pi}$$

Optimal Discriminator and Ratio Test

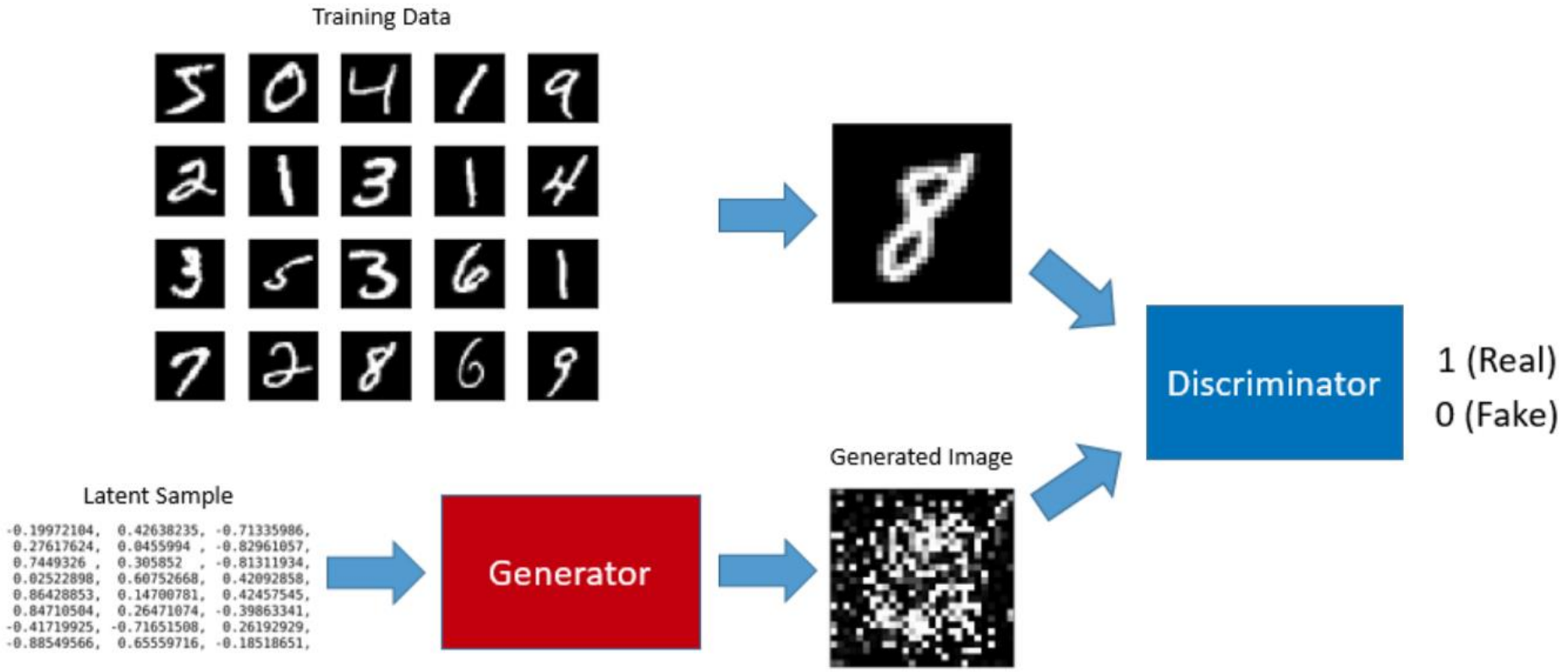
- π effect:



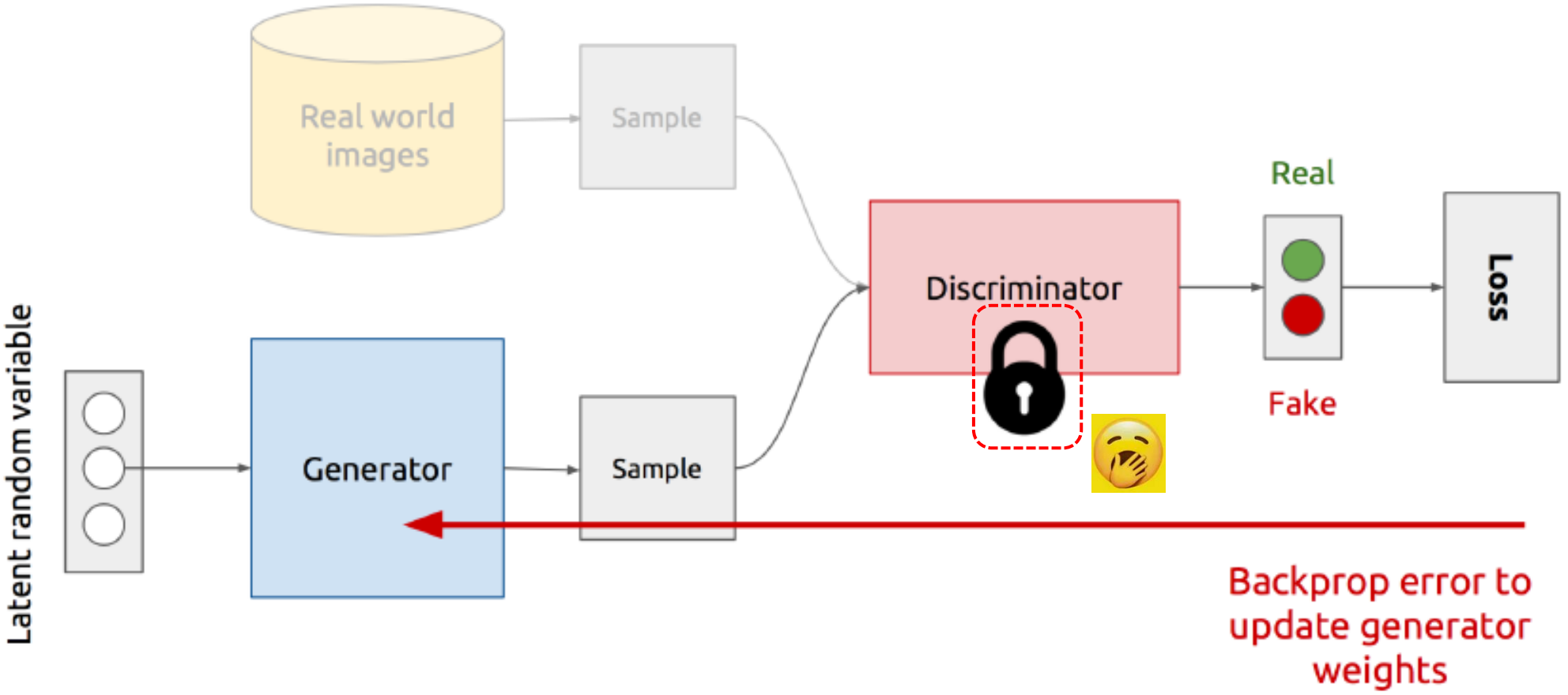
GAN Discriminator Training



GAN Discriminator Training



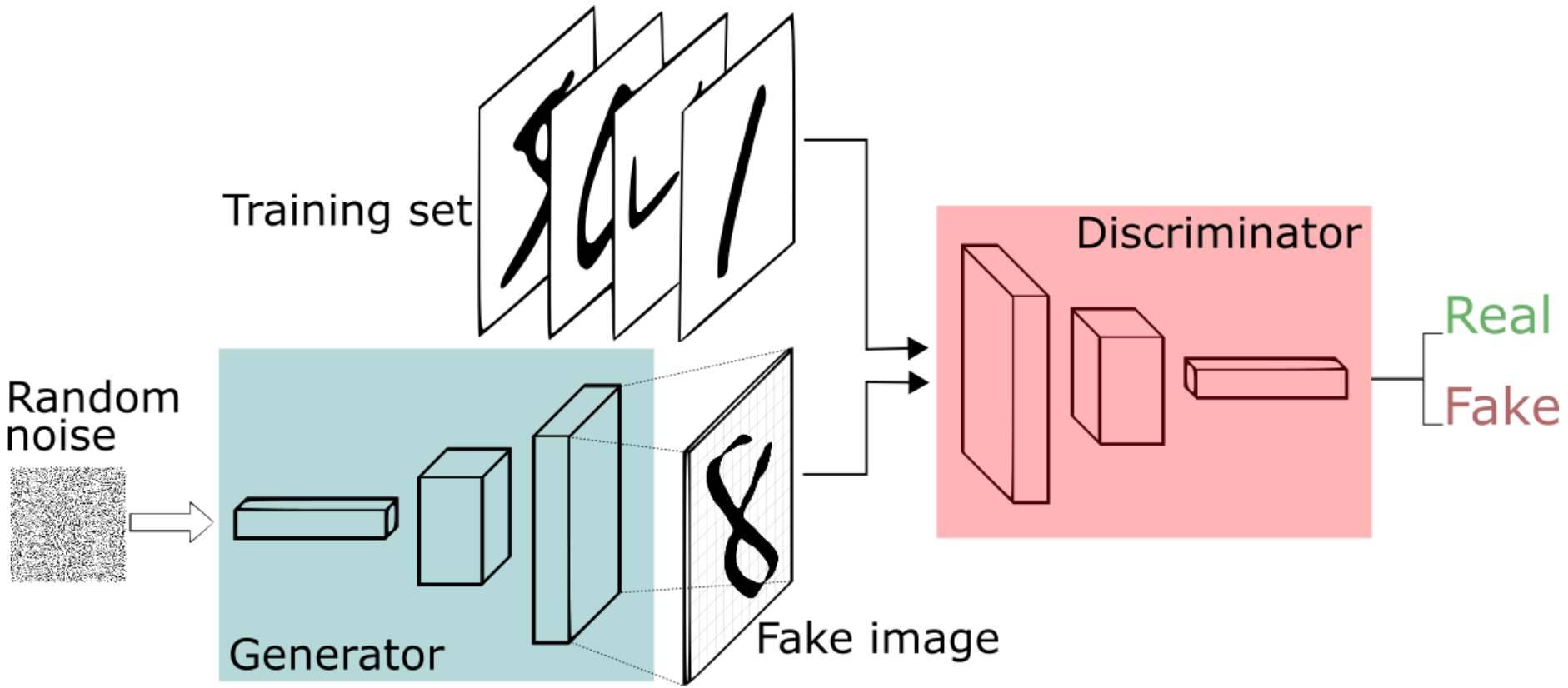
GAN Generator Training



Generative Adversarial Network - GAN

- GAN, **Generator** Training:

GAN in Application



GAN Algorithm

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

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

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

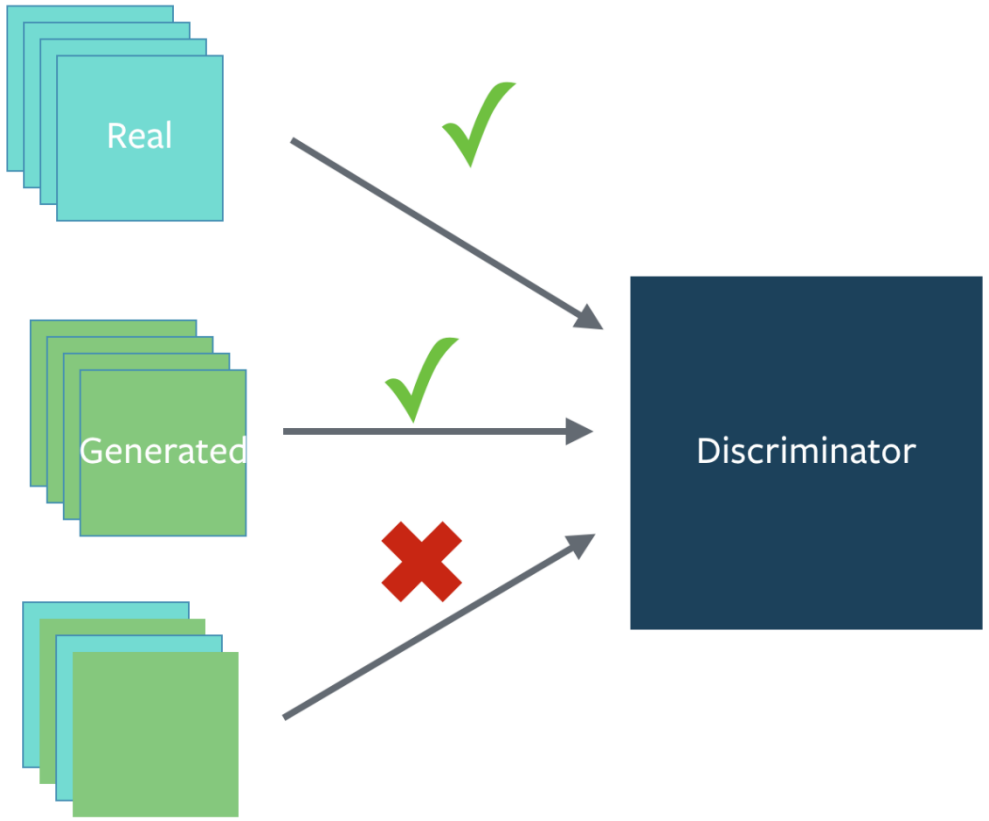
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Minibatch in GAN

- One – combined
- One – fake, one-real



GAN Hardness

- Hard to achieve Nash equilibrium?
- Vanishing gradient
- Mode collapse

Hard to achieve Nash equilibrium

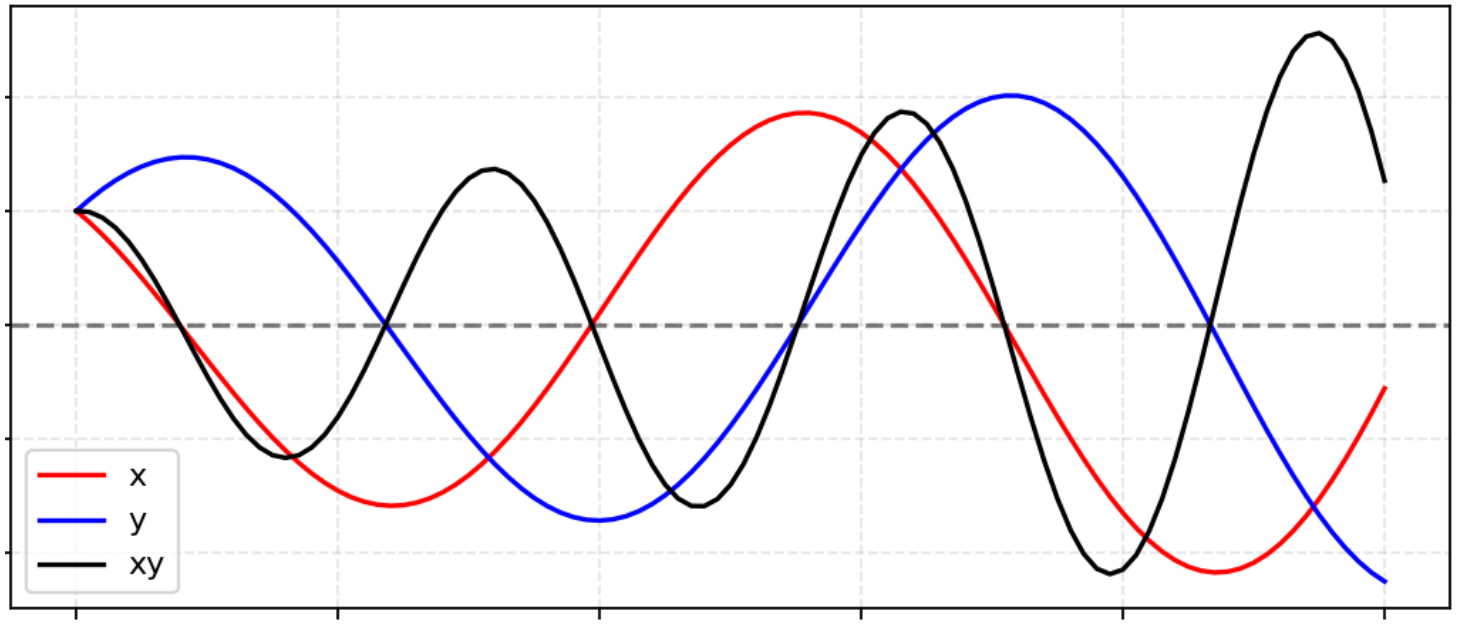
- Example: one player (control on x) want to minimize $f(x,y) = xy$, while other player (control on y) want maximize $f(x,y) = xy$ one by GD, one by GA

$$\begin{cases} \dfrac{\partial f}{\partial x} = y \rightarrow x^{new} = x^{old} - \varepsilon y^{old} \\ \dfrac{\partial f}{\partial y} = -x \rightarrow y^{new} = y^{old} + \varepsilon x^{old} \end{cases}$$

- Once x and y have different signs → oscillation!
- Nash equilibrium is $x = y = 0$

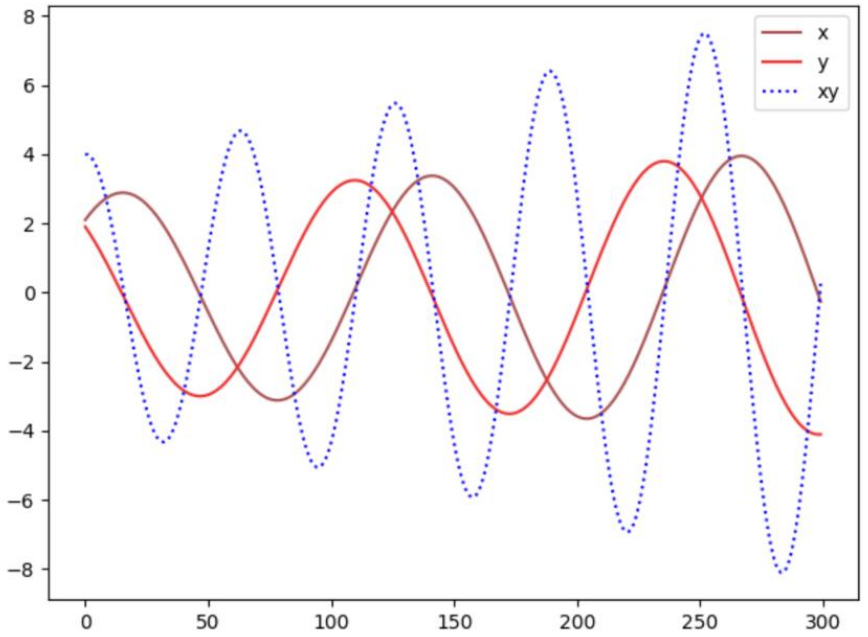
Hard to achieve Nash equilibrium

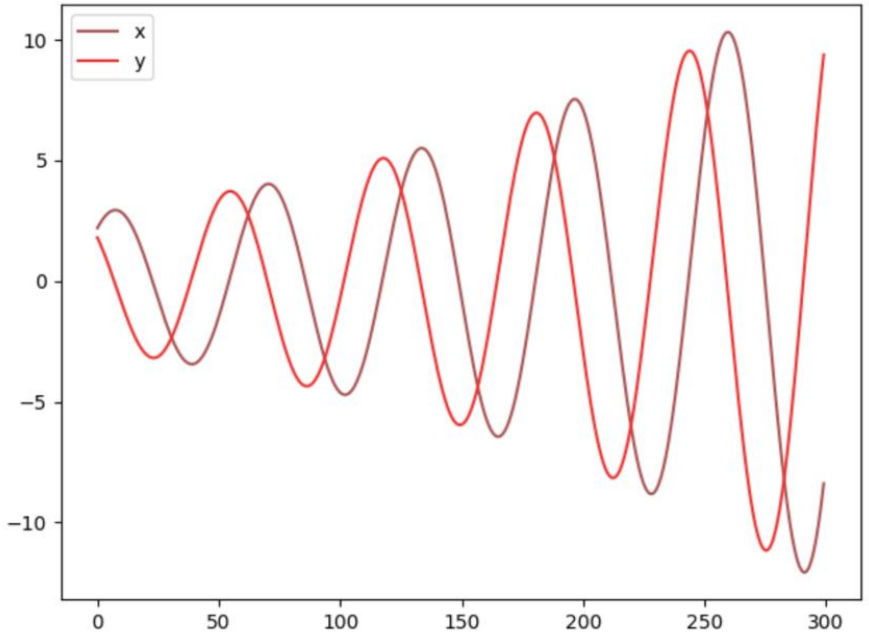
- Once x and y have different signs → oscillation!
- Cost functions may not converge using gradient descent in a minimax game.



Hard to achieve Nash equilibrium

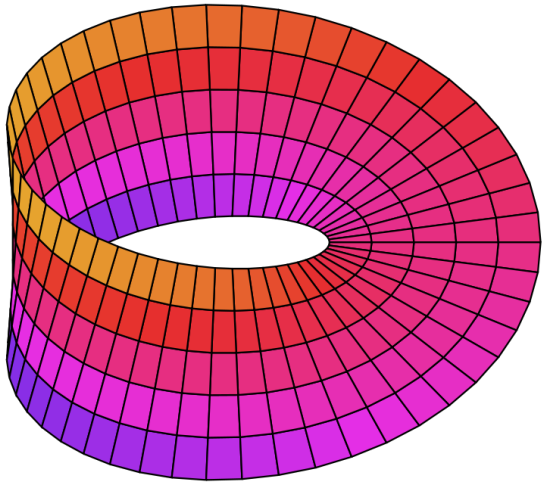
- Small (Left), Large (Right) Learning rate





Low Dimension Support

- Let's remember:
  - Manifold: A topological space that locally resembles a *nD* Euclidean space near each point.
  - Möbius Strip (n=2)



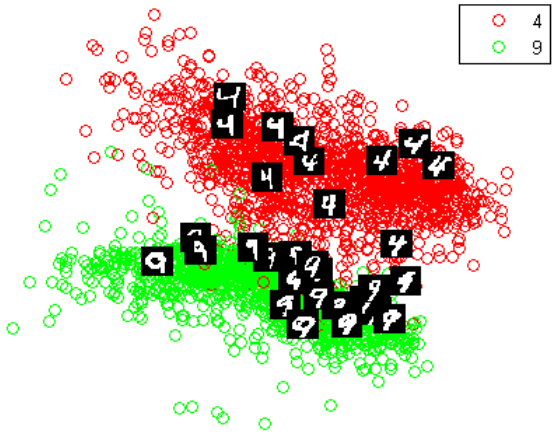
Low Dimension Support

- Support (of a real valued function): is the subset of the domain containing those elements which are not mapped to zero point.

$$supp(f) = \{x \in X | f(x) \neq 0\}$$

Low Dimension Support

- In a theoretical paper the problem of the supports of $p_r(x)$ and $p_g(x)$ lying on low dimensional manifolds has been discussed;
- For many real image (data) the dimension ($p_r(x)$) is redundant. They have been found to concentrate in a lower dimensional manifold.



Low Dimension Support

- If supports of $p_r(x)$ and $p_g(x)$ lying on low dimensional manifolds then:
  - High chance of perfect (100%) discrimination (disjoint manifold)

Vanishing Gradient in GAN

- For perfect discriminator, loss function drop to zero and no further update!
- GAN training Dilemma:
  - Bad discriminator: Non informative loss function → BAD feedback for generator
  - Perfect discriminator: Tiny! gradient→ Too Slow training

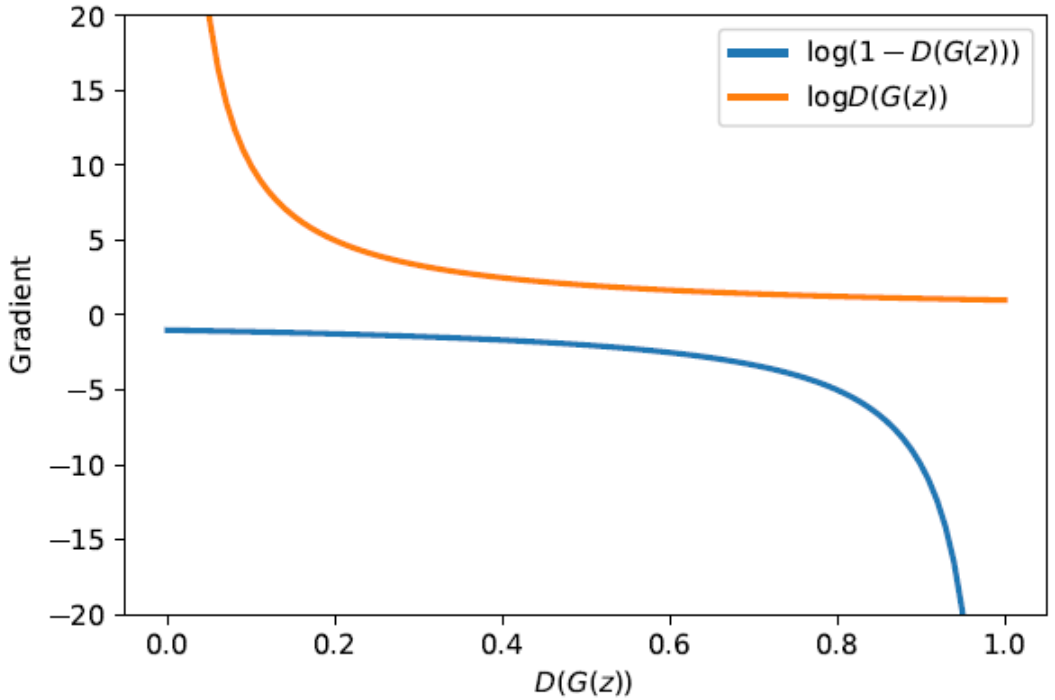
Vanishing Gradient in GAN

- Solution
  - High gradient for D~0

$$\nabla_{\theta_g} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \Rightarrow \nabla_{\theta_g} -\log\left(D\left(G\left(z^{(i)}\right)\right)\right)$$

Vanishing Gradient in GAN

- Solution
  - High gradient for D~0



Vanishing Gradient in GAN

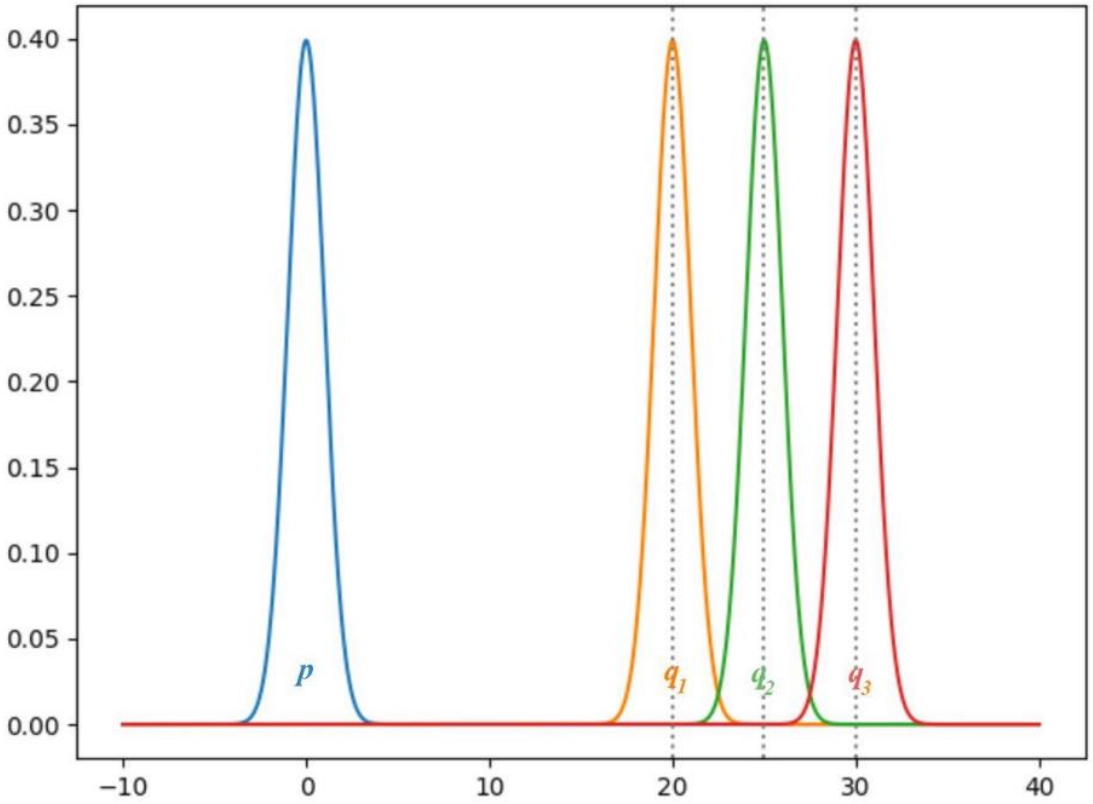
- Another Perspective

$$\min_G V(D^*, G) = 2D_{JS}\left(p_r \middle\| p_g\right) - 2log2$$

- What happened for JS divergence, if p and q does NOT match?

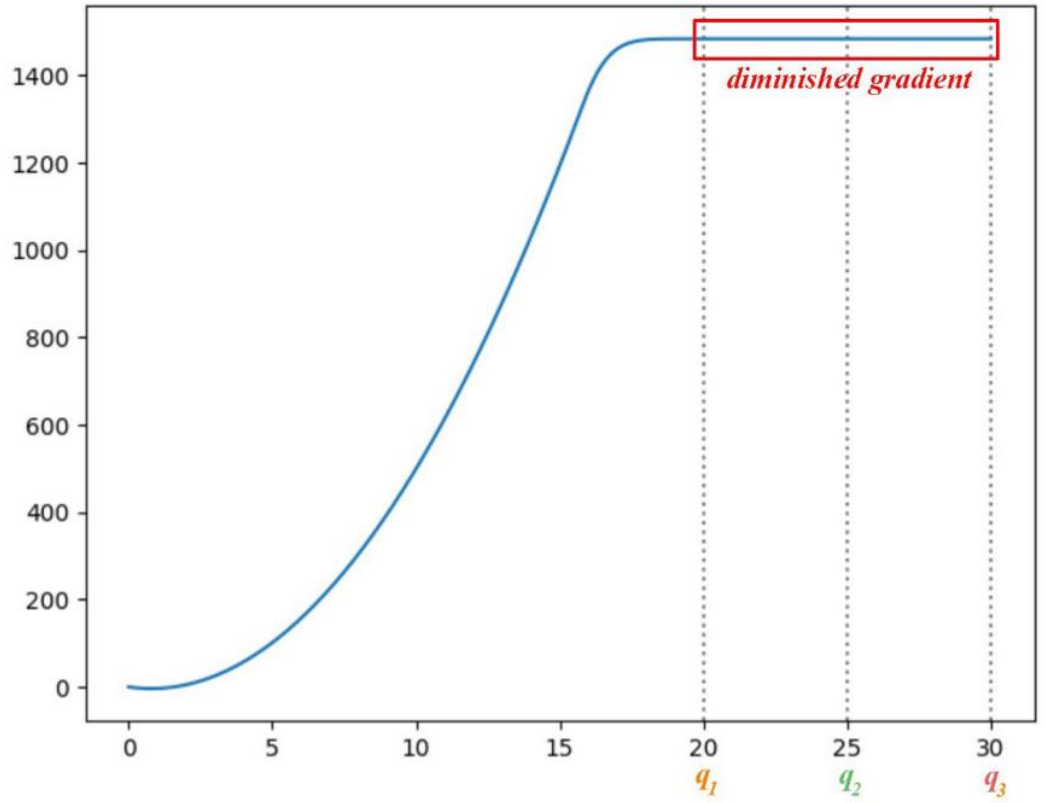
Vanishing Gradient in GAN

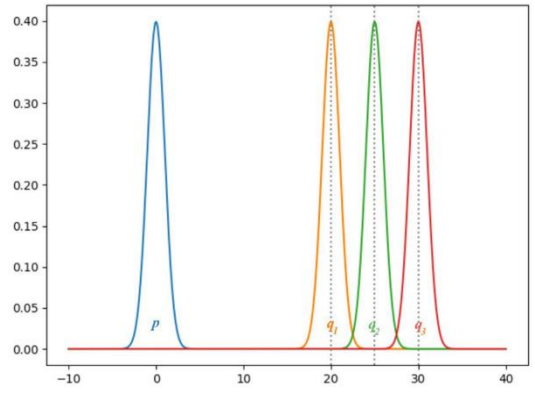
- Let plot the JS-divergence $JS(p,q), p \sim N(0,1), q \sim N(0 \cdots 30,1)$



Vanishing Gradient in GAN

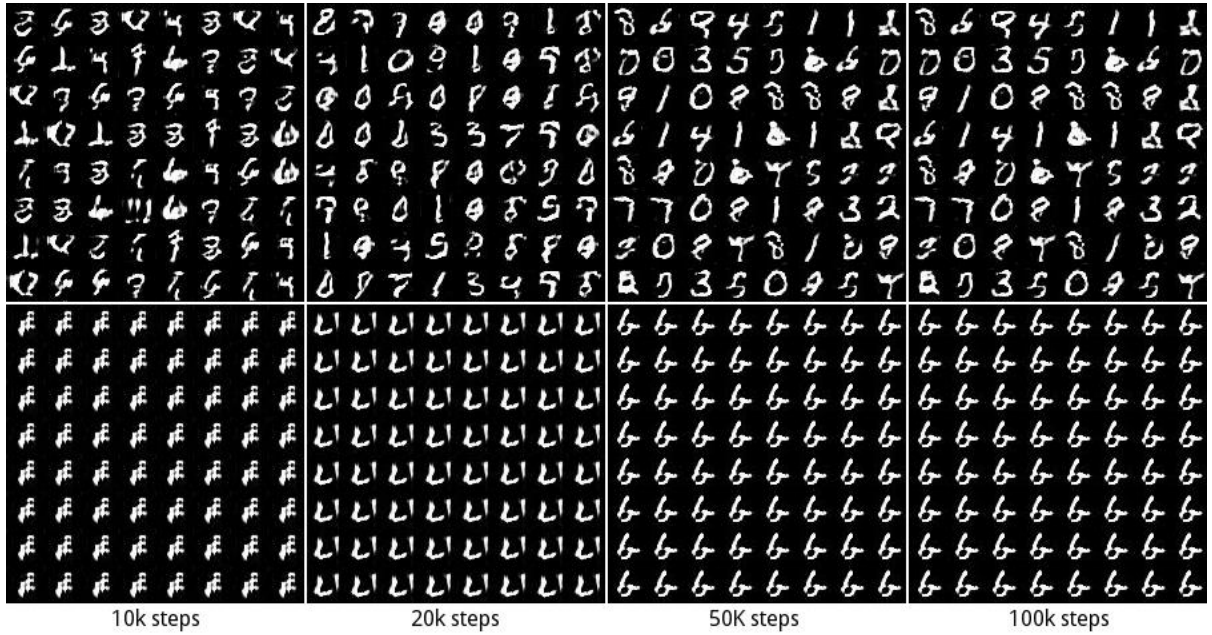
- Let plot the JS-divergence $JS(p,q), p \sim N(0,1), q \sim N(0 \cdots 30,1)$

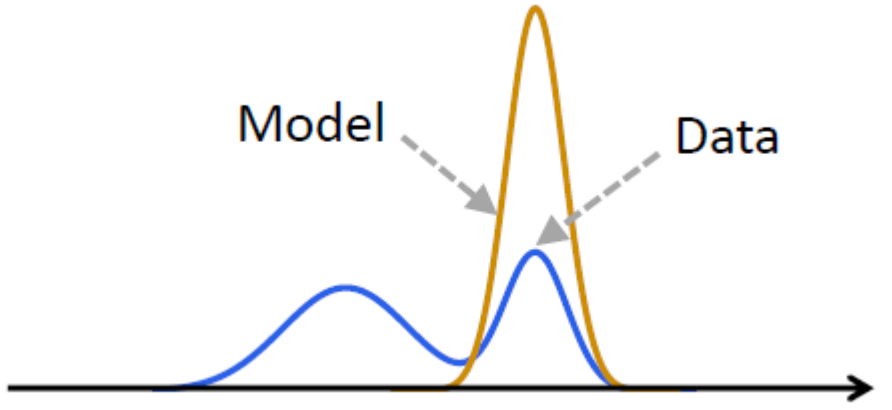




Mode Collapse

- Generator generates a limited diversity of samples (partial), or even the same sample (complete), regardless of the input.





Mode Collapse

- Generator generates a limited diversity of samples (partial), or even the same sample (complete), regardless of the input.
- The gradient of the discriminator may point in similar directions for many similar points.
- G is trained extensively without updates to D. The generated images will converge to find the optimal image $\boldsymbol{x}^*$ that fool D

Mode Collapse

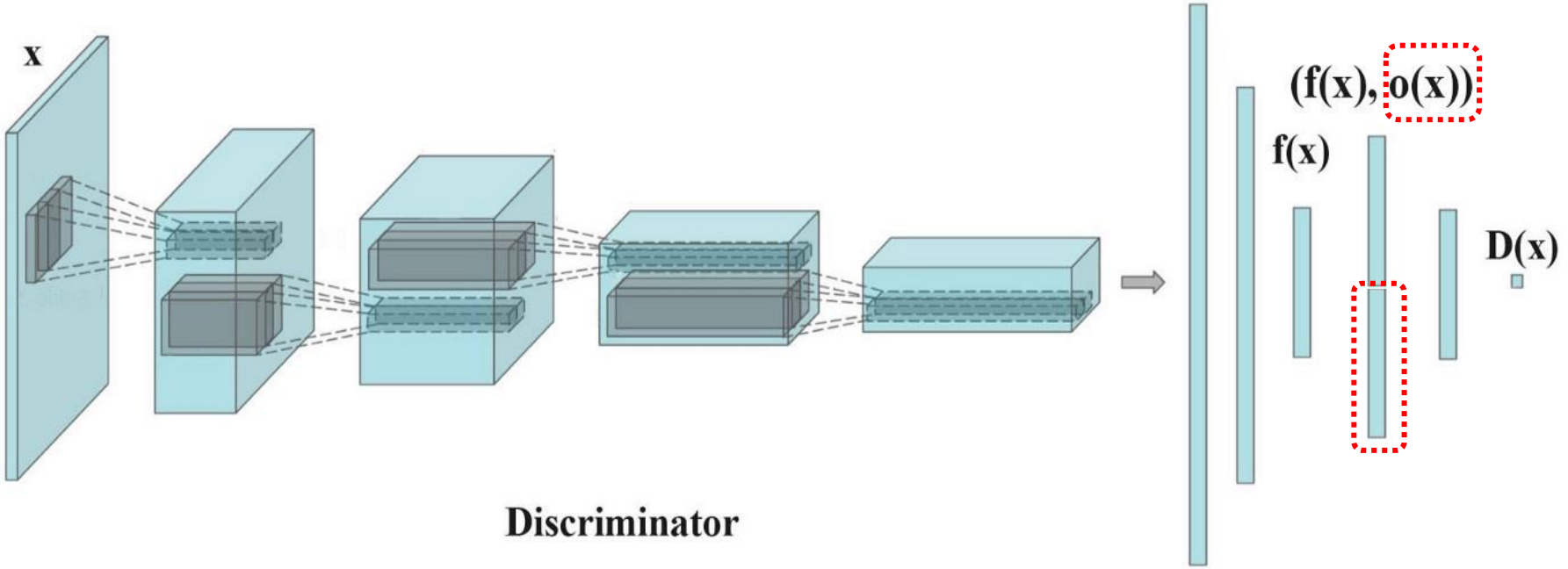
- No dissimilarity criteria.
- G may generate a single point $\boldsymbol{x}^* = G(z)$ such that $\boldsymbol{x}^* = \underset{x}{\operatorname{argmax}}\, D(x)$, will be independent of $\boldsymbol{z}$.
  - Minibatch
  - Newer version of GAN (Unrolled GAN)
  - Implicit Maximum Likelihood Estimation (IMLE)

Mode Collapse – Minibatch Discrimination

- Append the similarity between the image and other images in the same batch in one of the dense layers in the discriminator to classify whether this image is real or generated.
- If the mode starts to collapse, the similarity of generated images increases. The discriminator can use this score to detect generated images and penalize the generator if mode is collapsing.
- If not, the layer output small value (zero)!

Mode Collapse – Minibatch Discrimination

- Add a *minibatch discrimination layer* (similarity sensor)

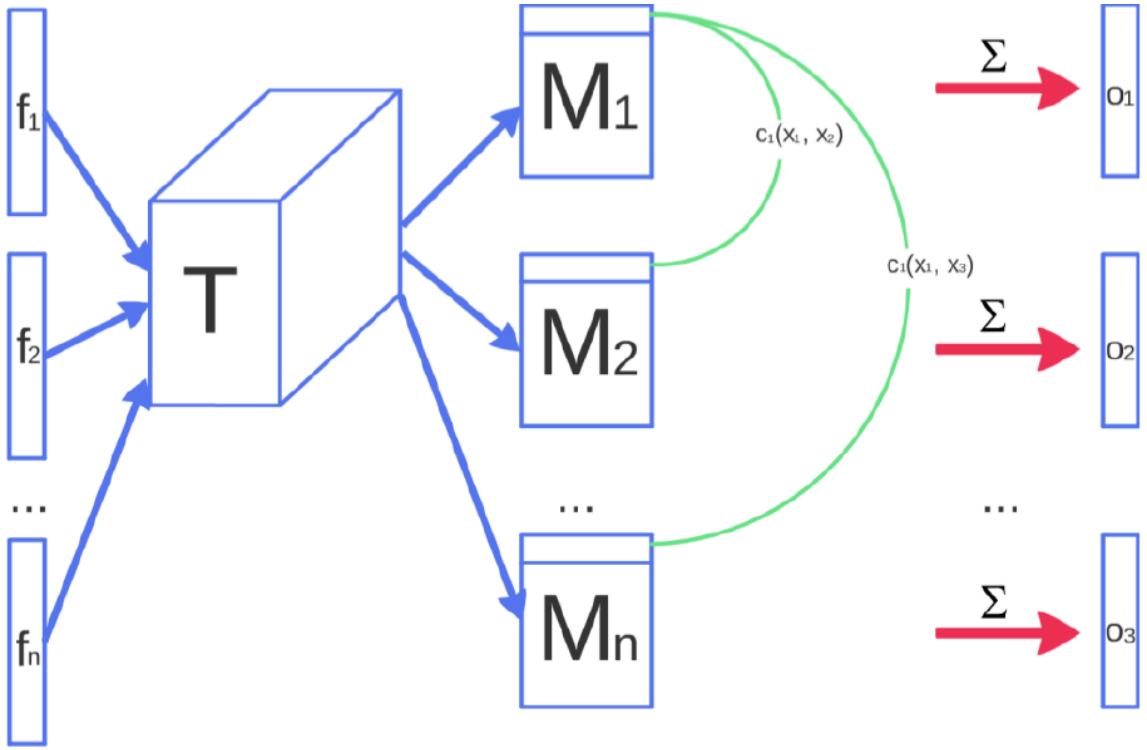


Mode Collapse – Minibatch Discrimination

- $f(\boldsymbol{x})$: vector of features for input $\boldsymbol{x}$, output of a intermediate layer in the discriminator, $i, j \in$ "$current\ MiniBatch$"
- $\mathbf{f}(x_i) \in \Re^A$
- $M_i = \mathbf{f}(x_i) \times T, \quad T \in \Re^{A \times B \times C} \Rightarrow M_i \in \Re^{B \times C}$
- $c_b(x_i, x_j) = exp\left(-\left\|M_{i,b} - M_{j,b}\right\|_{L_1}\right) = exp\left(-\left\|\mathbf{f}(x_i) \times T - \mathbf{f}(x_j) \times T\right\|_{L_1}\right) \in \Re$
- $o(x_i)_b = \sum_{j=1}^{n} c_b(x_i, x_j) \in \Re$
- $o(x_i) = [o(x_i)_1, o(x_i)_2, \cdots, o(x_i)_B] \in \Re^B$
- minibatch: $o(X) \in \Re^{n \times B}$

Mode Collapse – Minibatch Discrimination

- Architecture:



Mode Collapse – Feature Matching

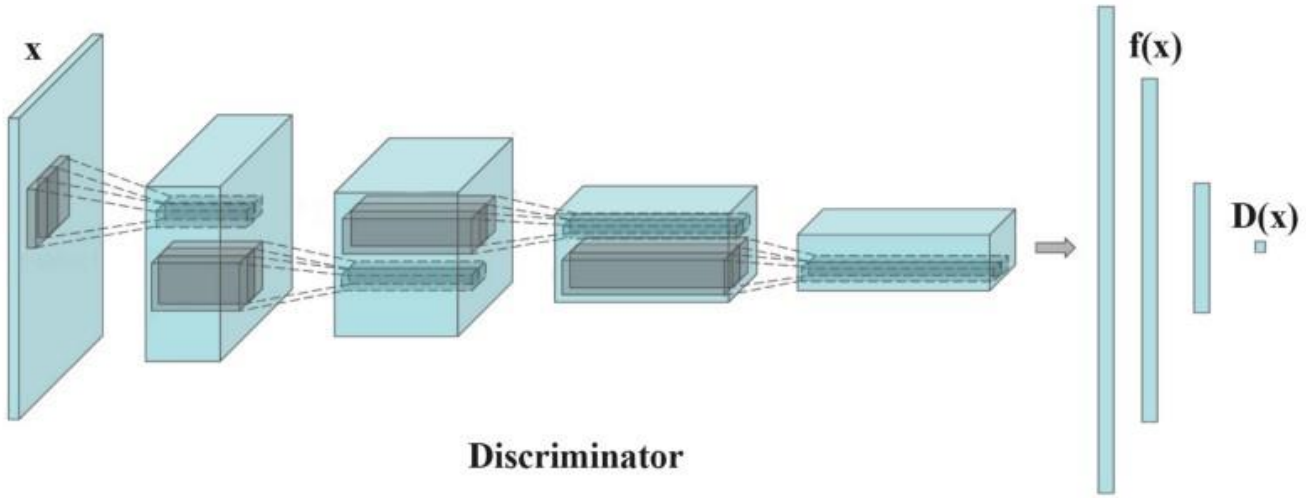
- Improve Convergence:
- New objective function for generator (instead discriminator output)
  - Push Generator to produce data that matches the statistics (features on an intermediate layer) of the real data.

$$\left\| \mathbb{E}_{\boldsymbol{x} \sim p_{data}} f(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{z} \sim p_z(z)} f\big(G(\boldsymbol{z})\big) \right\|_2^2$$

  - $f(x)$: activation of an intermediate layer of the discriminator or any meaningful features (median/mean/….)
  - $\mathbb{E}_{\boldsymbol{x} \sim p_{data}}$ is performed by sampling from real data in minibatch or whole data!
  - $\mathbb{E}_{\boldsymbol{z} \sim p_z(z)}$ is performed by sampling from noise in minibathch

Mode Collapse – Feature Matching

- It is effective when the GAN model is unstable during training.



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

Mode Collapse – Historical Averaging

- Penalize (add to) G and D cost to avoid fast changing (θ: net parameters)

$$\left\| W - \frac{1}{t}\sum_{i=1}^{t} W(i) \right\|_2^2$$

Mode Collapse – One side label smoothing

- One-Sided Label Smoothing:
  - Positive Label: 1→α $(0 \ll \alpha < 1)$
  - Negative Label: 0→β $(0 < \beta \ll 1)$
- But remember that (we show for α=1, β=0):

$$D(\boldsymbol{x}) = \frac{\alpha p_{data}(\boldsymbol{x}) + \beta p_{model}(\boldsymbol{x})}{p_{data}(\boldsymbol{x}) + p_{model}(\boldsymbol{x})}$$

- For small p_{data} and large p_{model} →☹
  - Only smooth 1 to α
  - Negative label set at zero

Tips and Tricks to Train GAN

- Large kernels and more filters: 3×3 → 5×5 (More Flexible)
- Soft and Noisy labels {0,1} →{[0,0.1],[0.9,1]}
- One class at a time (CGAN)
- Feature Matching✓
- Minibatch Discrimination ✓
- Use a spherical Z (not uniform in cube),
- Avoide Sparse Gradients: ReLU, MaxPool (Use Leaky ReLu, Avgerage Pooling)
- Use the ADAM Optimizer

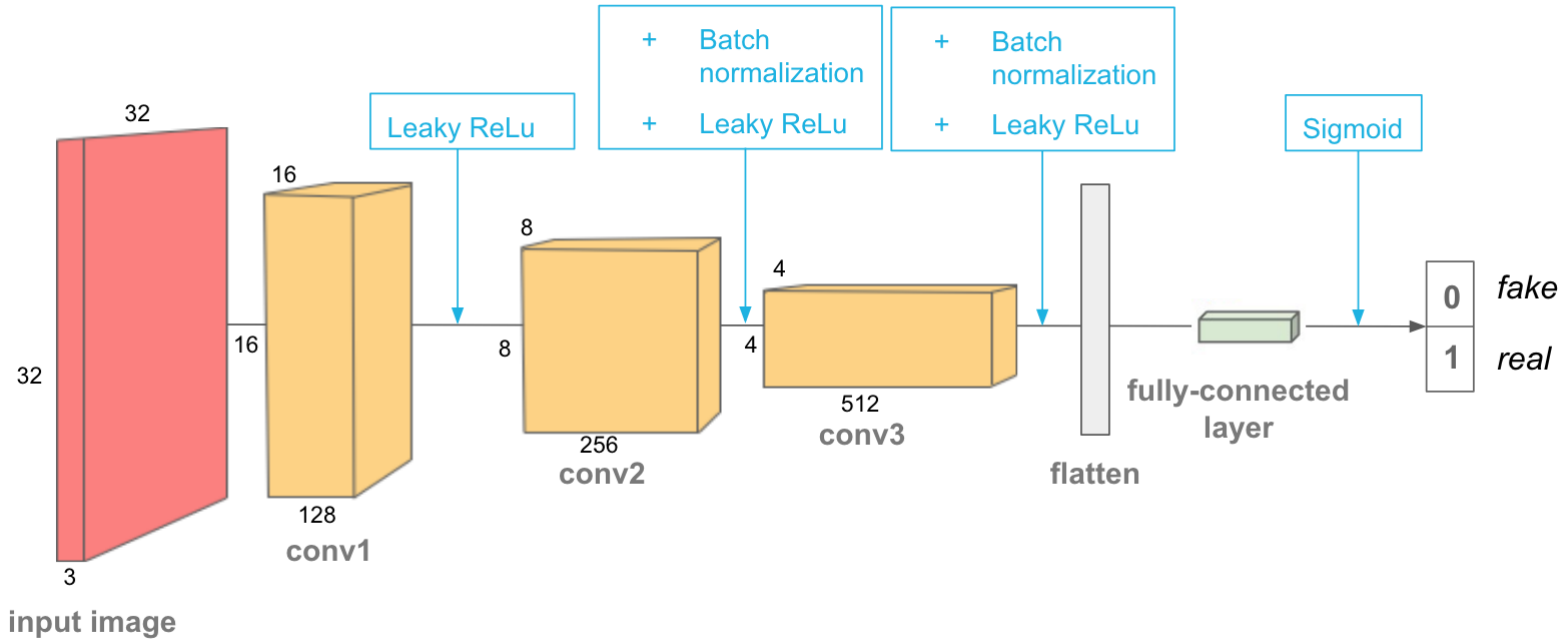
Some Famous GAN

DCGAN (Deep Convolutional GANs)

- DCGAN: A popular and successful network design for GAN
- Replace any pooling layers with stride convolutions (discriminator) and fractional-stride convolutions (generator).
- Use BN layer in both the generator and the discriminator.
- Remove FC hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses tanh.
- Use LeakyReLU activation in the discriminator for all layers.

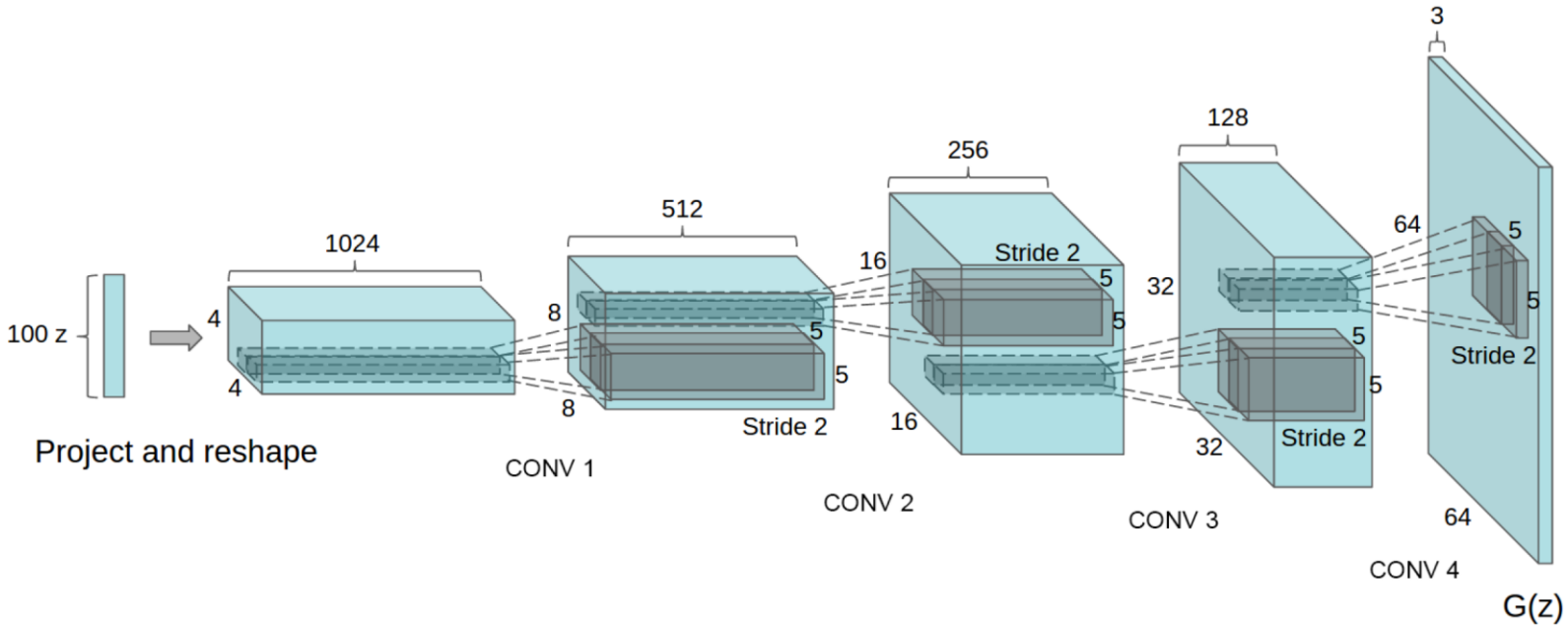
DCGAN (Deep Convolutional GANs)

- Discriminator:



DCGAN (Deep Convolutional GANs)

- Generator:



DCGAN (Deep Convolutional GANs)

- Fake bedroom: (feed rand vector to generator)

DCGAN (Deep Convolutional GANs)

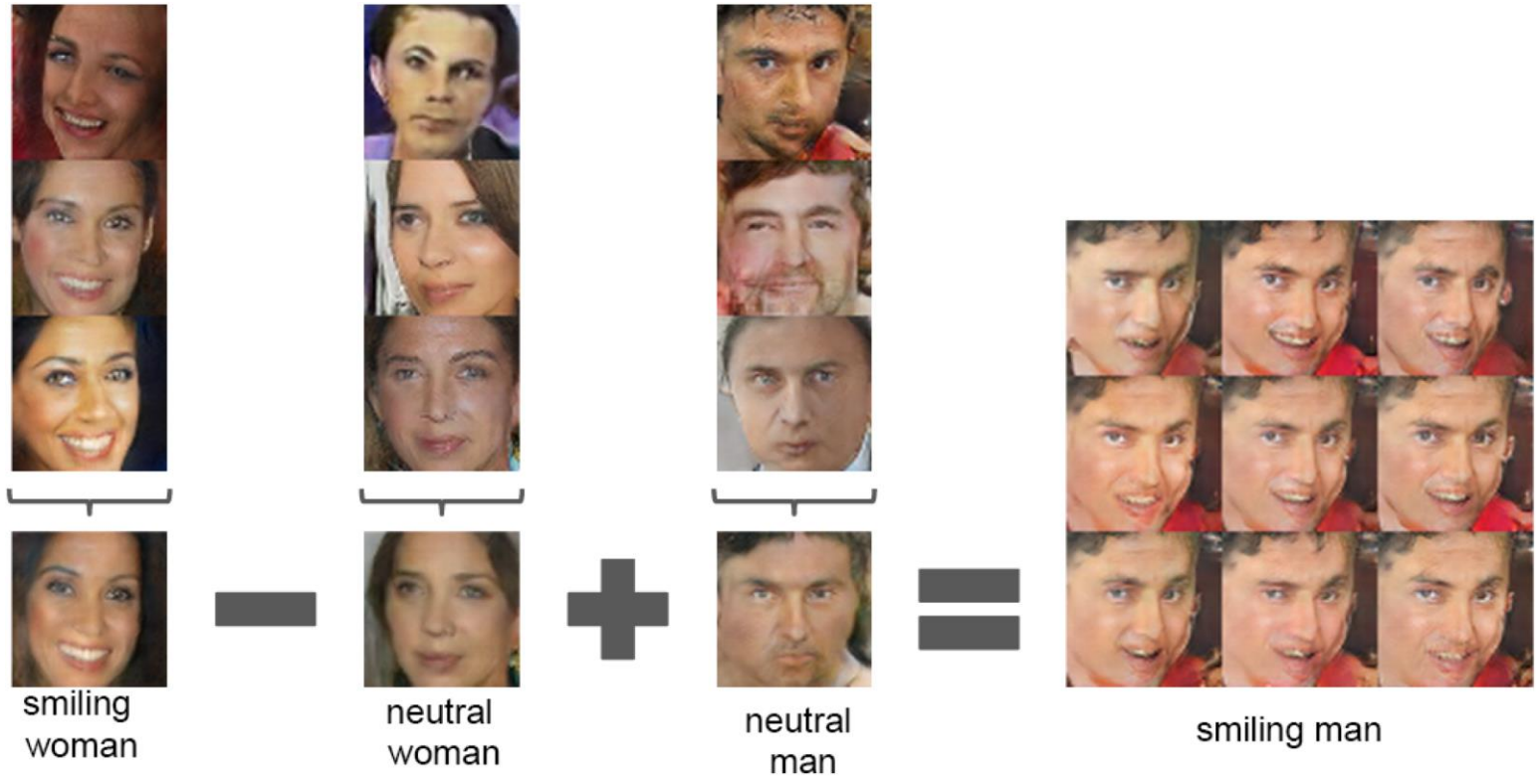
- Vector Arithmetic:
  - Z: random vector

$$Y = Z_{avrage}(smiling\ woman) - Z_{average}\ (neutral\ woman) + Z_{average}\ (neutral\ man);$$

- Feed Y to the generator → Smiling man

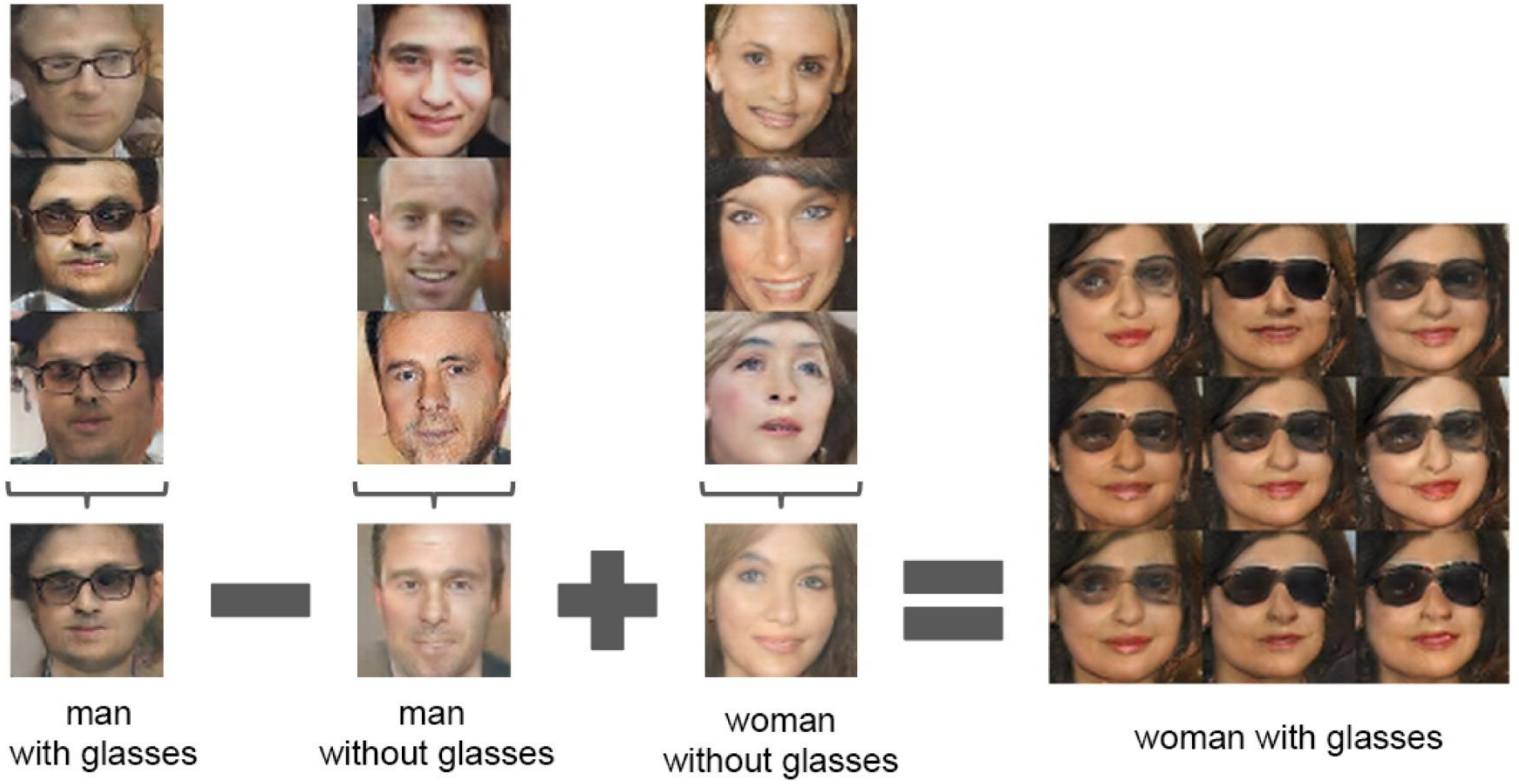
DCGAN (Deep Convolutional GANs)

- Vector Arithmetic:



DCGAN (Deep Convolutional GANs)

- Vector Arithmetic:



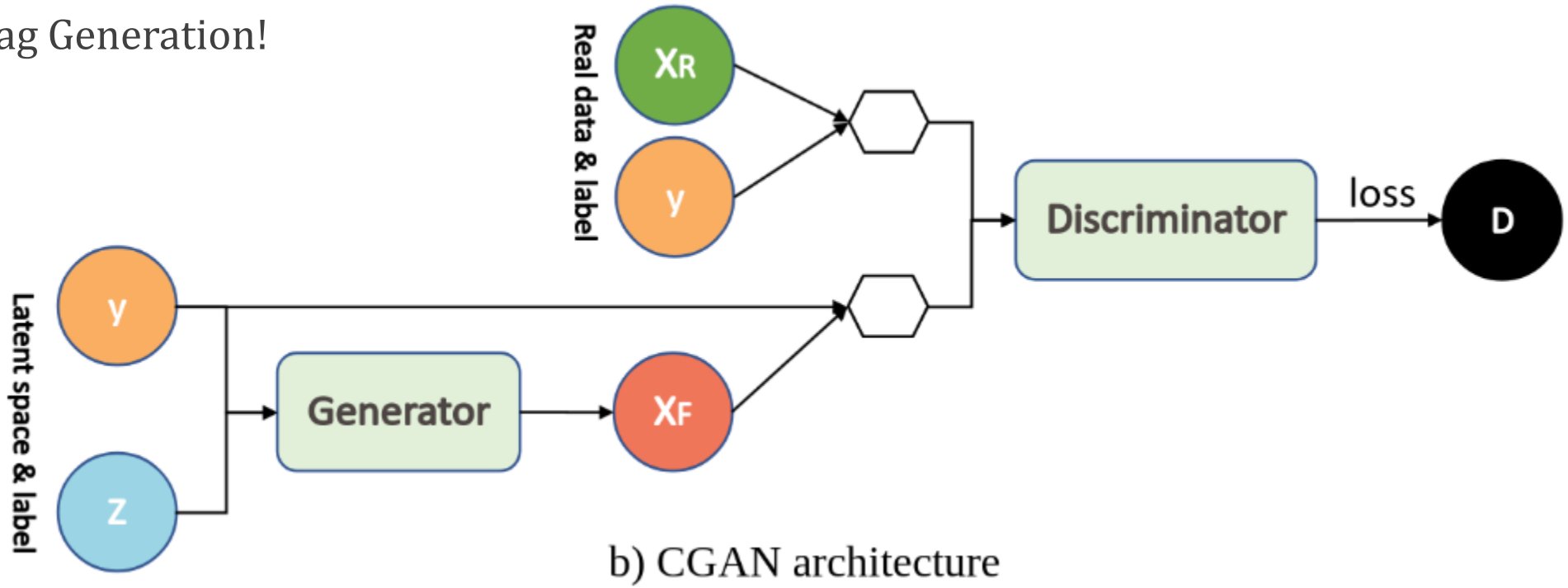
Conditional GAN – CGAN

- Same idea as CVAE!
- y: Label/Attribute/…

$$\min_G \max_D V(D,G) = E_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[logD(\boldsymbol{x}|\boldsymbol{y})] + E_{\text{z}\sim p_z(z)}\left[log\left(1 - D\big(G(\boldsymbol{z}|\boldsymbol{y})\big)\right)\right]$$

Conditional GAN – CGAN

- Concatenate x and y
  - Concatenate at **G**?
    - Pre-train a DNN for image and tags features
  - Tag Generation!



b) CGAN architecture

Conditional GAN – CGAN

- Sample: y (Left) and *Generated* (Right)

Conditional GAN – CGAN

- Tag Generation (Multimodal Configuration)
  - A CNN Pre-trained for best label prediction, use its 4096D feature layer (freeze)
  - A Skip-Gram (size of 200) trained from concatenation of user-tags/titles/description (freeze)
  - Images with **multiple tags** were **repeated** inside the training set, once for each associated tag.
  - Discriminator input(s): {word vector code, image feature}
  - Discriminator output: {1D sigmoid}
  - Generator input(s): {noise$\in \mathbb{R}^D$, image feature}
  - Generator output: Tags{$\in \mathbb{R}^Q$ as word vector code}

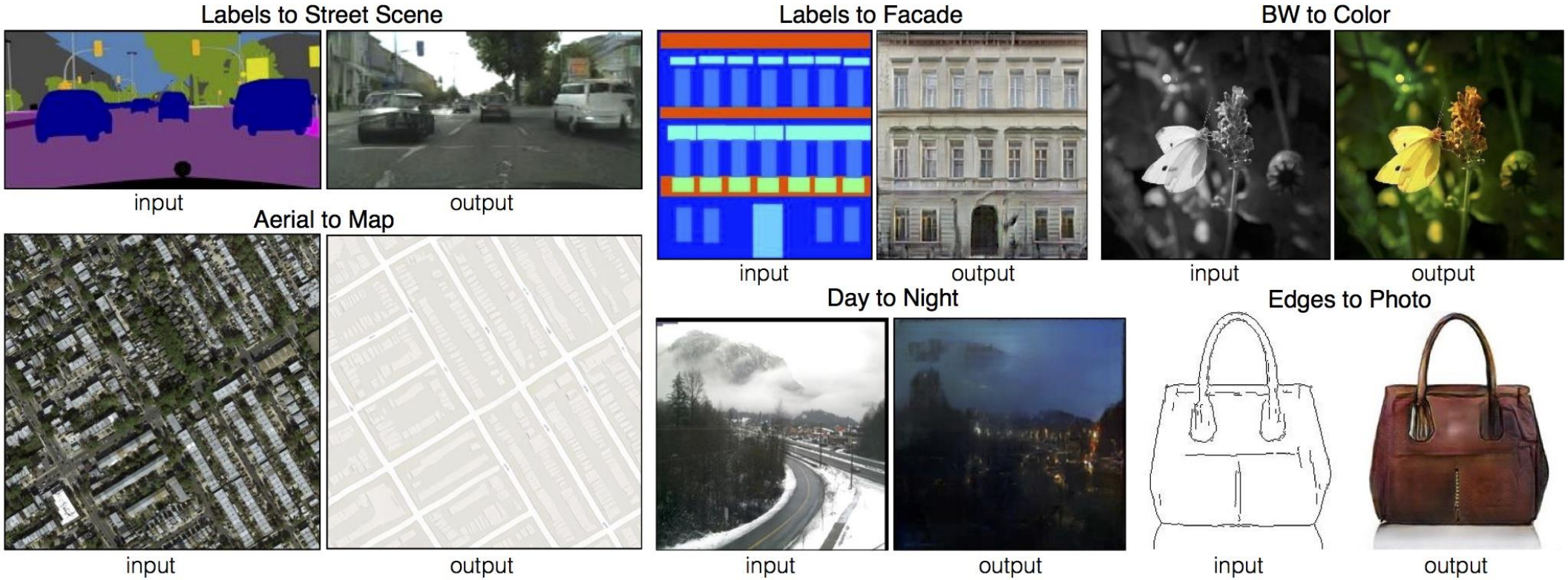
Conditional GAN – CGAN

- Results:

	User tags + annotations	Generated tags
	montanha, trem, inverno, frio, people, male, plant life, tree, structures, transport, car	taxi, passenger, line, transportation, railway station, passengers, railways, signals, rail, rails
	food, raspberry, delicious, homemade	chicken, fattening, cooked, peanut, cream, cookie, house made, bread, biscuit, bakes
	water, river	creek, lake, along, near, river, rocky, treeline, valley, woods, waters

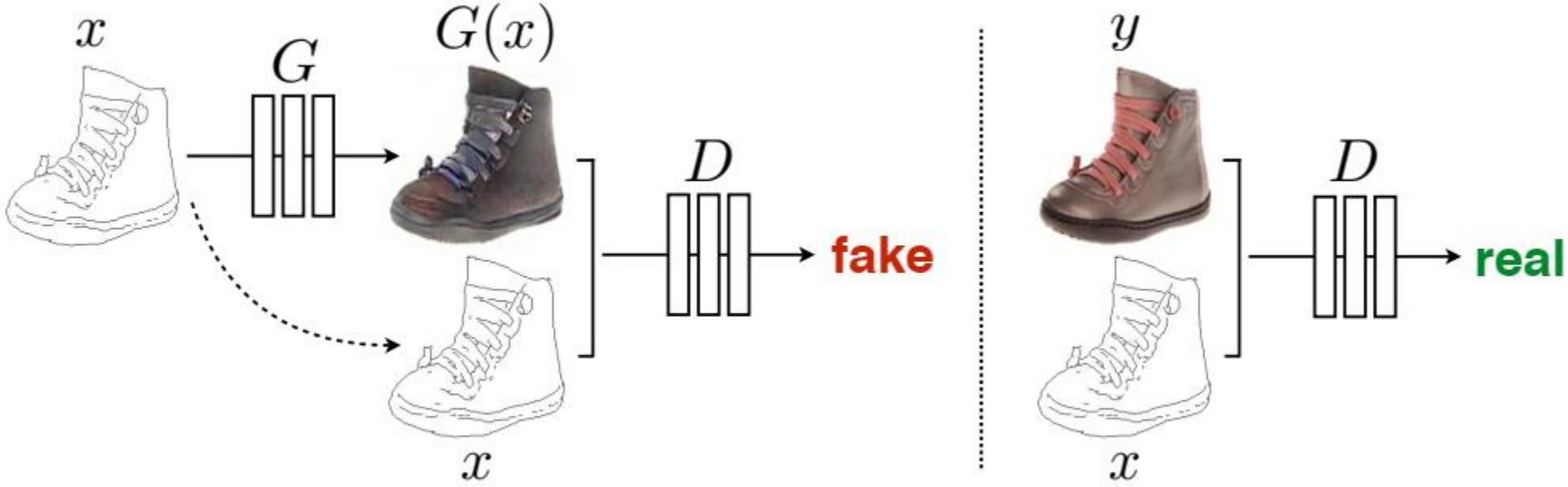
CGAN– Application pix2pix

- Image-to-Image Translation (pix2pix):



CGAN– Application pix2pix

- Pix2Pix training (High Level Scheme):
  - Application: Edge→ Photo
  - The conditioning image, x is applied as the input to the generator and as input to the discriminator.



CGAN– Application pix2pix

- Pix2Pix Loss Function (x is conditioning):

$$\mathcal{L}_{cGAN}(G,D) = \mathbb{E}_{x,y}[logD(x,y)] + \mathbb{E}_{x,z}\left[log\left(1 - D\big(x, G(x,z)\big)\right)\right]$$

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[\|y - G(x,z)\|_1]$$

$$G^* = arg\min_G \max_D \{\mathcal{L}_{cGAN}(G,D) + \lambda\mathcal{L}_{L1}(G)\}$$

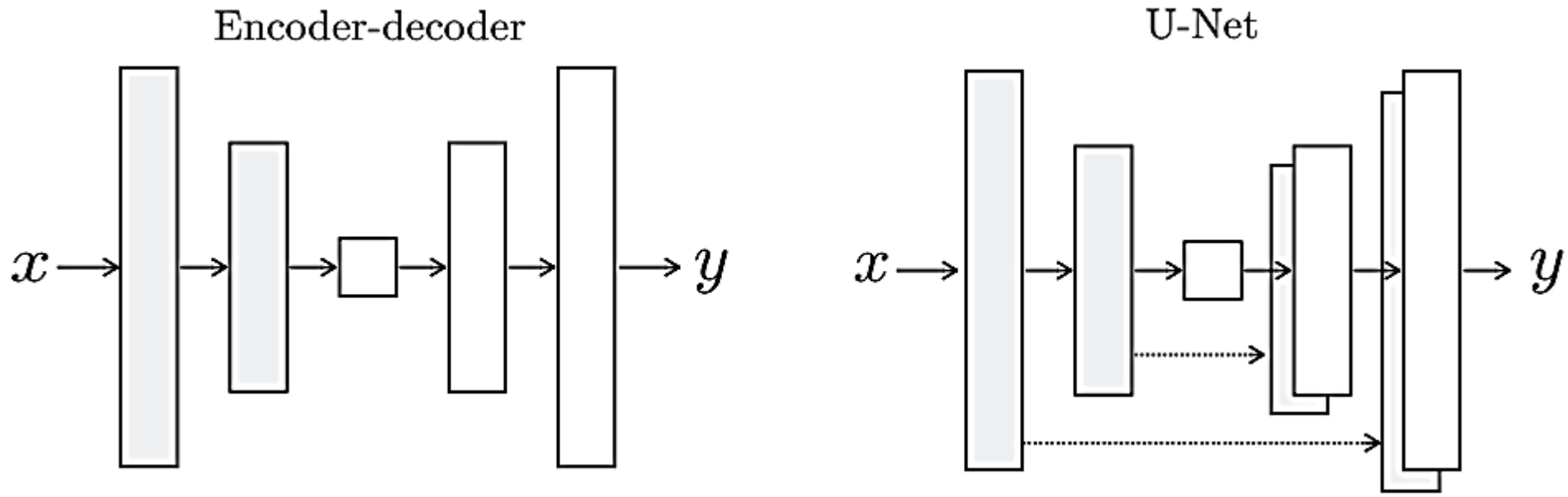
CGAN– Application pix2pix

- Pix2Pix noise input, z:
  - There is no explicit stochastic noise input, $z \in \mathbb{R}^D$, input noise performed by dropout in train and test phase, see the experiment from paper:

> Without z, the net could still learn a mapping from x to y, but would produce deterministic outputs, and therefore fail to match any distribution other than a delta function. Past conditional GANs have acknowledged this and provided Gaussian noise z as an input to the generator, in addition to x (e.g., [55]). In initial experiments, we did not find this strategy effective – the generator simply learned to ignore the noise – which is consistent with Mathieu et al. [40]. Instead, for our final models, we provide noise only in the form of dropout, applied on several layers of our generator at both training and test time. Despite the dropout

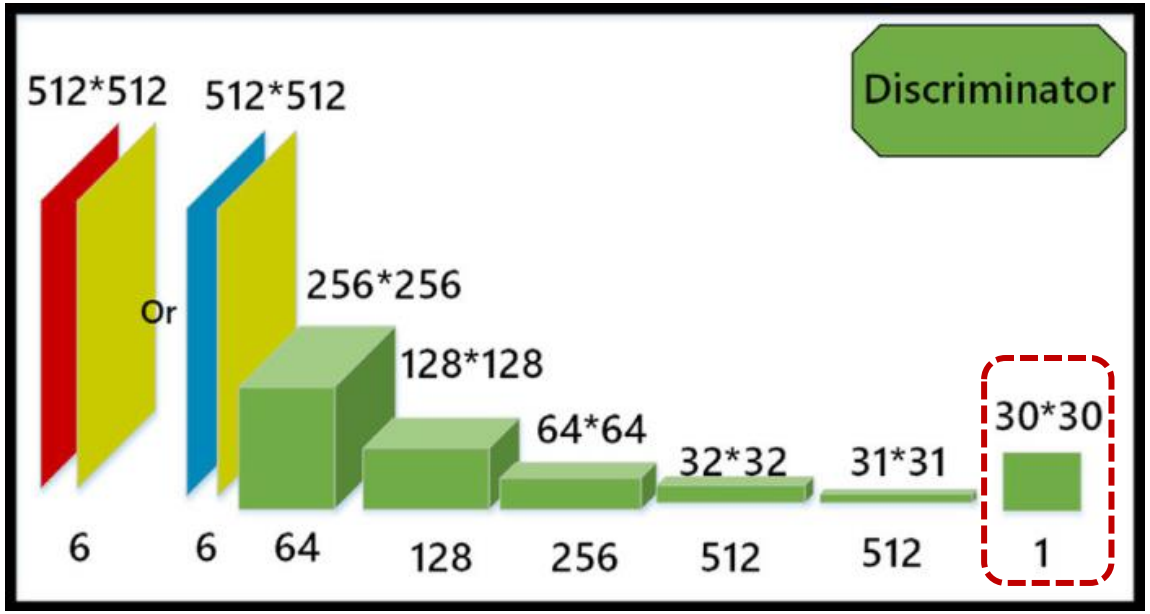
CGAN– Application pix2pix

- Pix2Pix – Two alternatives for generator



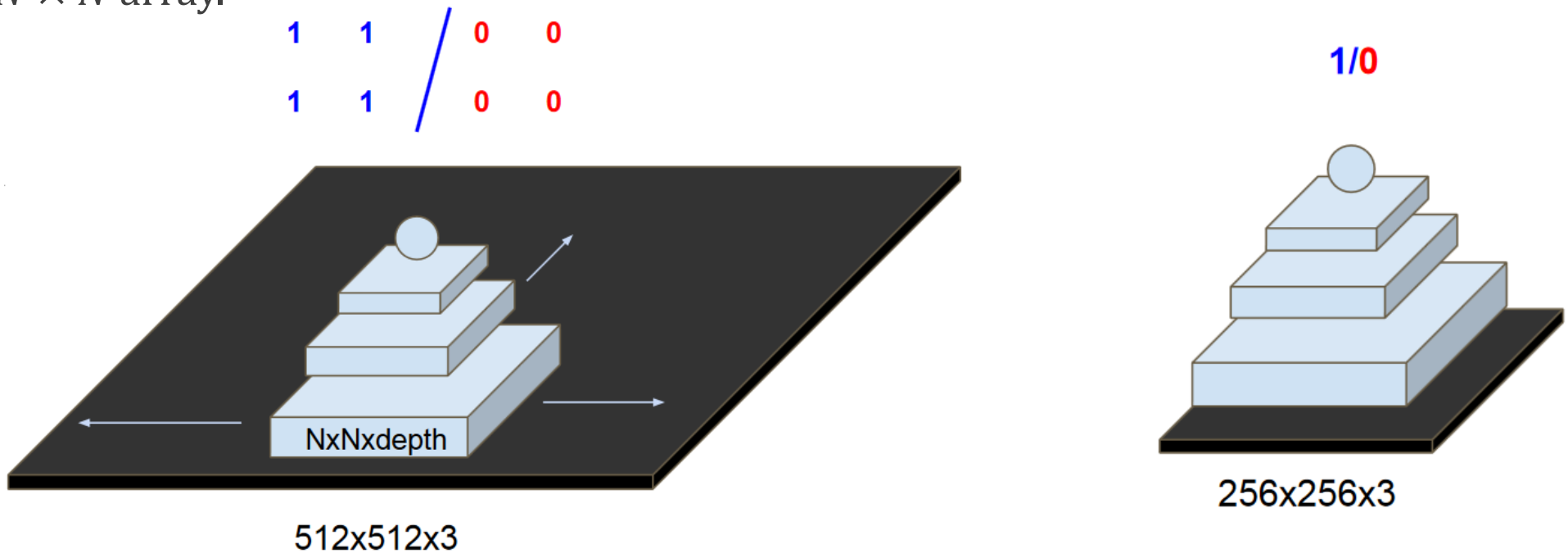
CGAN– Application pix2pix

- PatchGAN idea:
  - Instead of producing decision output as single scalar vector (real vs. fake) it generates an NxN array.



CGAN– Application pix2pix

- PatchGAN idea:
  - Instead of producing decision output as single scalar vector (real vs. fake) it generates an $N \times N$ array.



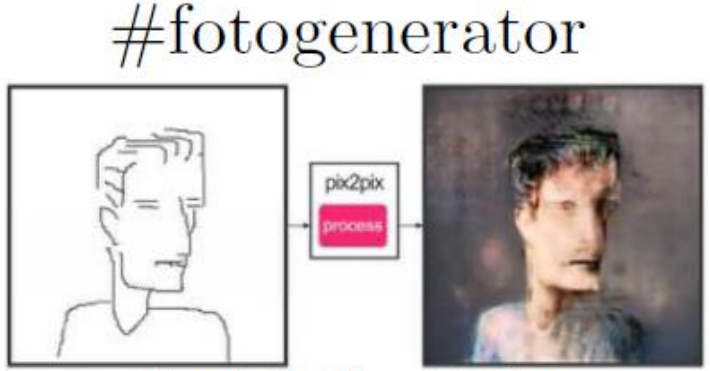
CGAN– Application pix2pix

- Pix2Pix (Edge2Photo):

CGAN– Application pix2pix

- Pix2Pix:

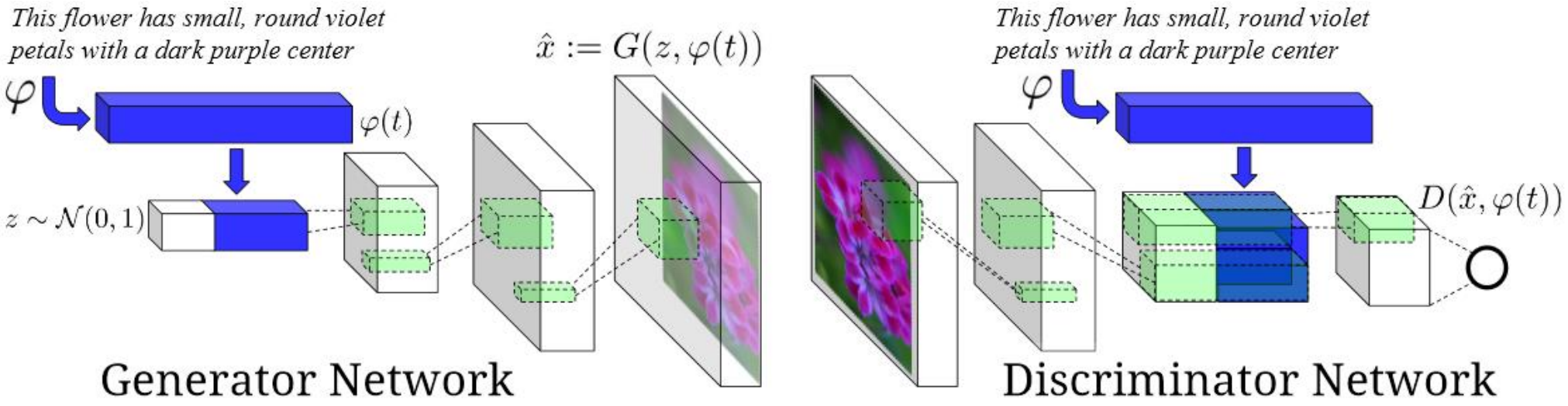
by Mario Klingemann



sketch by Yann LeCun

CGAN – Text to Image

- Text-to-Image-Synthesis (G/D)



CGAN – Text to Image

- Text-to-Image-Synthesis
- Training

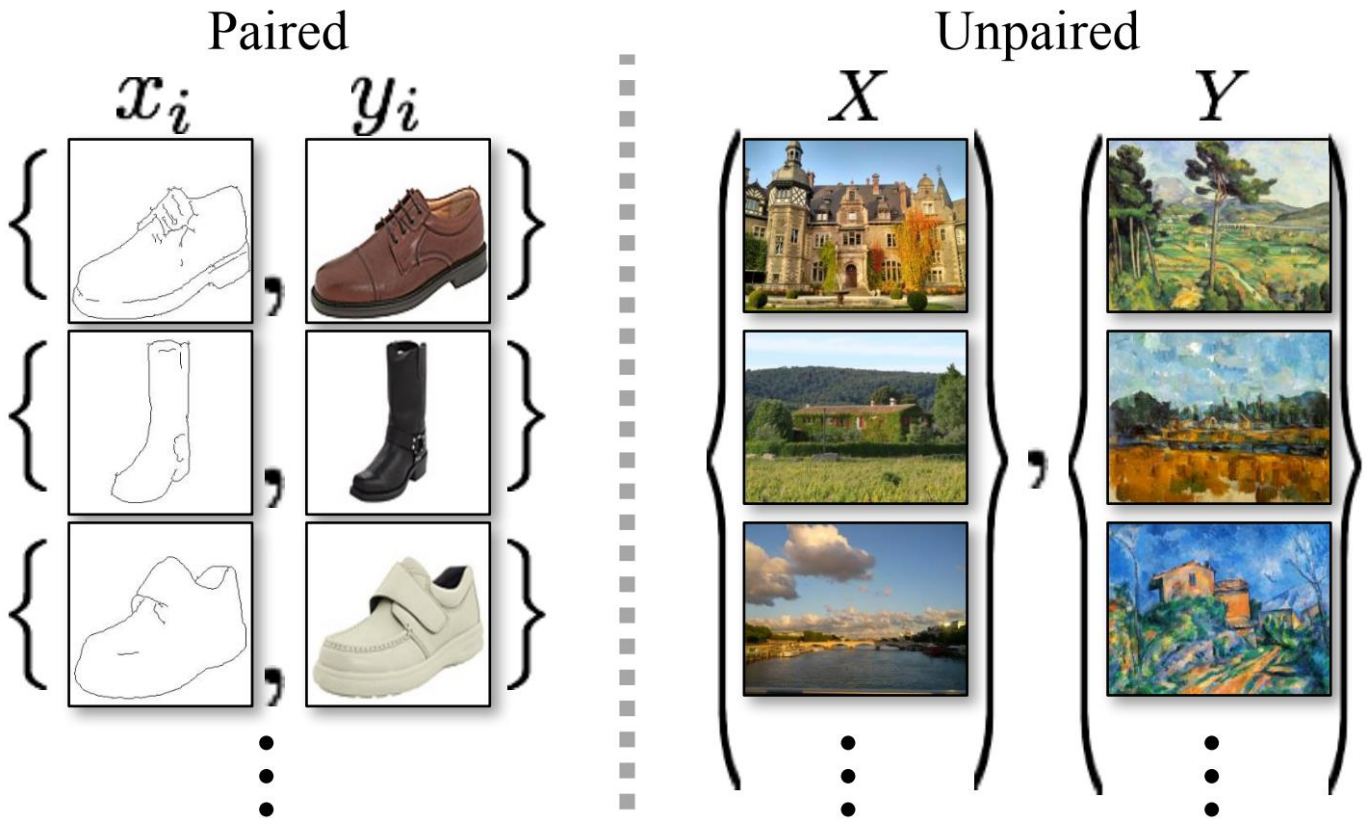
Algorithm 1 GAN-CLS training algorithm with step size α, using minibatch SGD for simplicity.

Input: minibatch images x, matching text t, mismatching $\hat{t}$, number of training batch steps S
for $n = 1$ **to** S **do**
  $h \leftarrow \varphi(t)$ {Encode matching text description}
  $\hat{h} \leftarrow \varphi(\hat{t})$ {Encode mis-matching text description}
  $z \sim \mathcal{N}(0, 1)^Z$ {Draw sample of random noise}
  $\hat{x} \leftarrow G(z, h)$ {Forward through generator}
  $s_r \leftarrow D(x, h)$ {real image, right text}
  $s_w \leftarrow D(x, \hat{h})$ {real image, wrong text}
  $s_f \leftarrow D(\hat{x}, h)$ {fake image, right text}
  $\mathcal{L}_D \leftarrow \log(s_r) + (\log(1 - s_w) + \log(1 - s_f))/2$
  $D \leftarrow D - \alpha \partial \mathcal{L}_D / \partial D$ {Update discriminator}
  $\mathcal{L}_G \leftarrow \log(s_f)$
  $G \leftarrow G - \alpha \partial \mathcal{L}_G / \partial G$ {Update generator}
end for

CycleGAN – Unpaired Image Translation

- Image Translation (CycleGan):

CycleGAN – Paired-Unpaired Data



CycleGAN – Main Goal

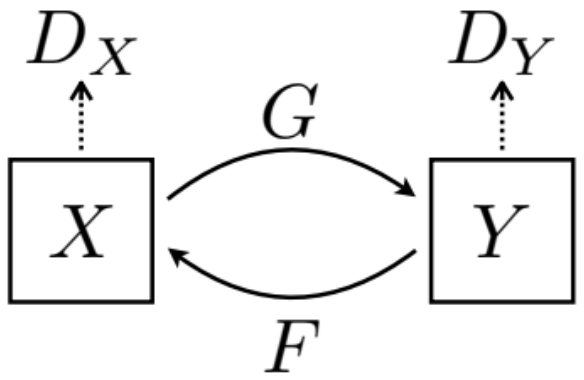
- An algorithm that can learn to translate between two domains without paired input-output examples.
- Assumption: There is some relationship between the domains

Horse ⇄ Zebra

- Data: we are given one set of images in domain $\{x_i\}_{i=1}^{N} \in X$ and a different set in domain $\{y_j\}_{j=1}^{M} \in Y$.

CycleGAN – Basic Structure

- Two Mapping (two *Generators and two Discriminator*):
  - $G\colon X \longrightarrow Y, \hat{y} = G(x), x \in X$
  - $F\colon Y \longrightarrow X, \hat{x} = F(y), y \in Y$
  - Ideally: $p_{fake}(\hat{y}) = p_{data}(y)$ and $p_{fake}(\hat{x}) = p_{data}(x)$
  - Translate pdf2pdf not object2object (remember paired/unpaired data)



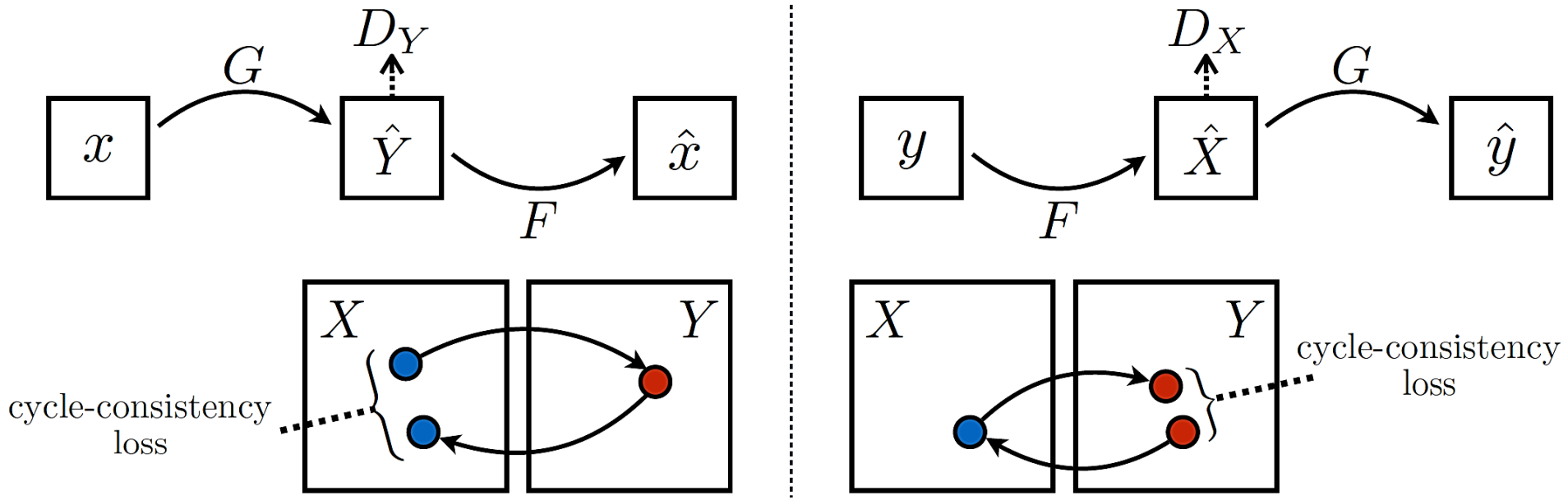
CycleGAN – Cycle Consistency Criteria

- G and F should be inverses of each other.
- We need a loss term for simultaneously:

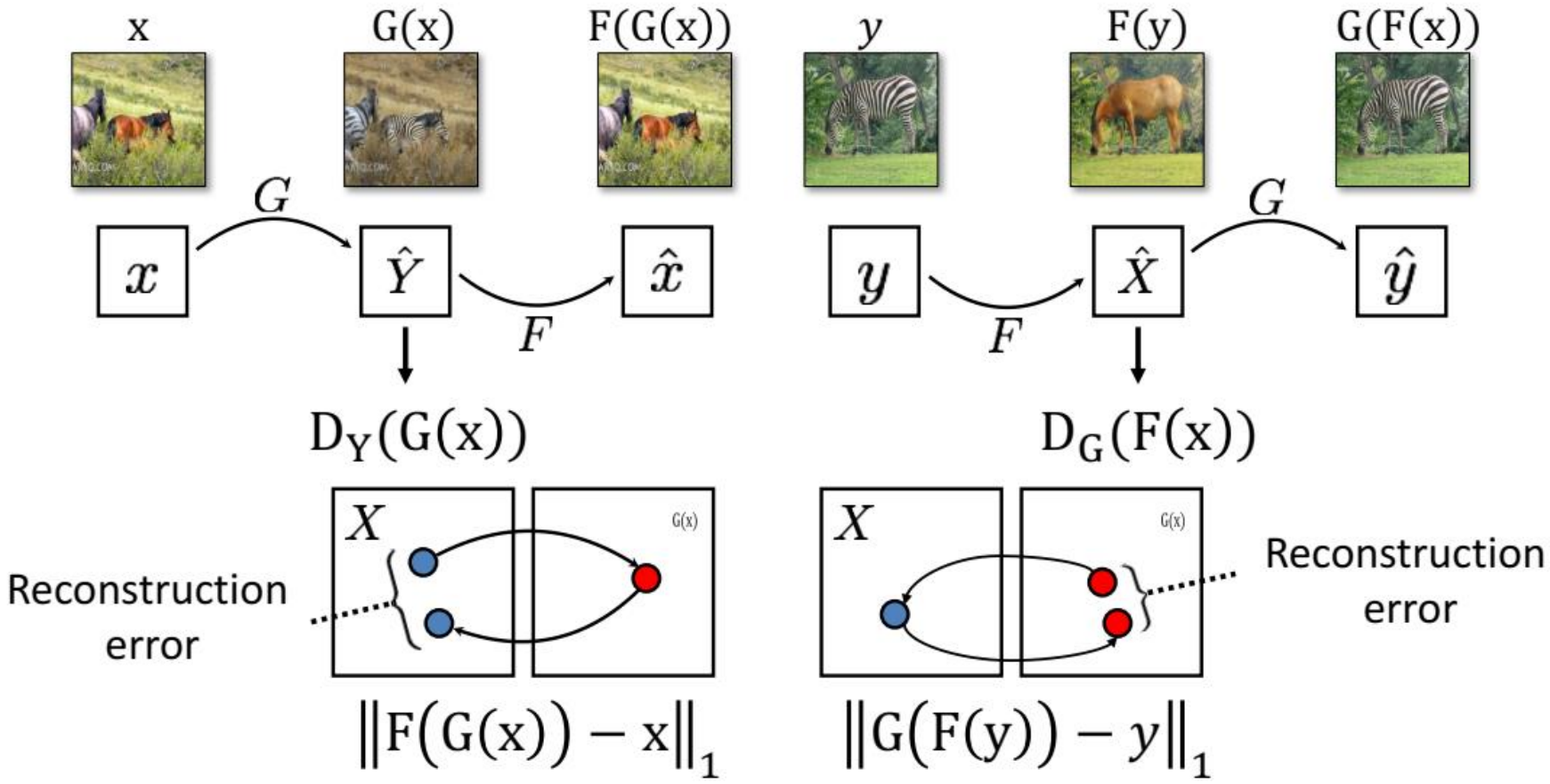
$$F\big(G(x)\big) \approx x$$
$$G\big(F(y)\big) \approx y$$

CycleGAN – Cycle Consistency Criteria

- Cycle Consistency Loss:



CycleGAN – Cycle Consistency Criteria



CycleGAN – Loss

- Adversary Loss (Vanilla GAN or others):

$$\mathcal{L}_{GAN}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{data}(y)}[log D_Y(y)] + \mathbb{E}_{x \sim p_{data}(x)}\left[log\left(1 - D_Y\left(G(x)\right)\right)\right]$$

$$\mathcal{L}_{GAN}(F, D_X, Y, X) = \mathbb{E}_{x \sim p_{data}(x)}[log D_X(x)] + \mathbb{E}_{y \sim p_{data}(y)}\left[log\left(1 - D_X\left(F(y)\right)\right)\right]$$

CycleGAN – Loss

- Cycle Consistency Loss:

$$\mathcal{L}_{CYC}(G,F) = \mathbb{E}_{x\sim p_{data}(x)}\left\{\left\|F\left(G(x)\right)-x\right\|_1\right\} + \mathbb{E}_{y\sim p_{data}(y)}\left\{\left\|G\left(F(y)\right)-y\right\|_1\right\}$$

CycleGAN – Loss

- Total Loss:

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(F, D_X, Y, X) + \lambda \mathcal{L}_{CYC}(G, F)$$

- Total Loss Optimization:

$$\{G^*, F^*\} = arg \underbrace{min}_{G,F} \underbrace{max}_{D_X, D_Y} \mathcal{L}(G, F, D_X, D_Y)$$

- Equivalent Problem: Design two auto-encoder:

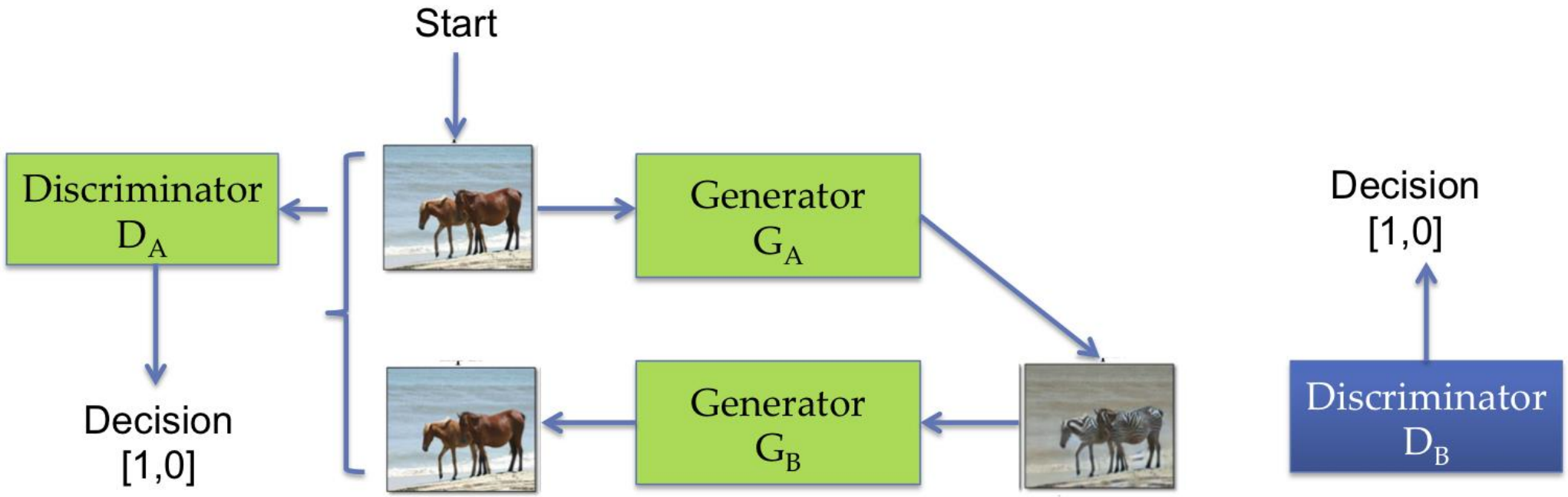
$$FoG: X \to X \; and \; GoF: Y \to Y$$

CycleGAN – Training

- Generate image from generator X using image from domain X, Similarly generate an image from generator Y using image from domain Y.
- Train discriminator X on batch using images from domain X and images generated from generator Y as real and fake image respectively.
- Train discriminator Y on batch using images from domain Y and images generated from generator X as real and fake image respectively.
- Train generator on batch using the combined model.
- Repeat steps from 1 to 4 for every image in the training dataset and then repeat this process for 200 epochs.

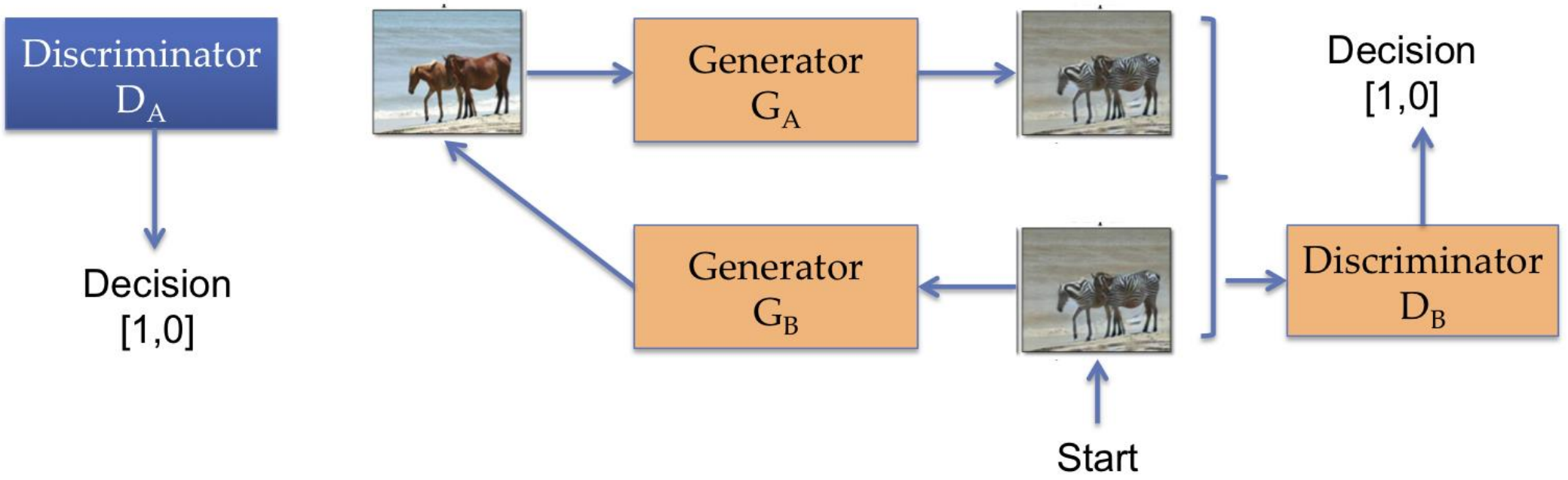
CycleGAN – Schematic

- Signal Flow



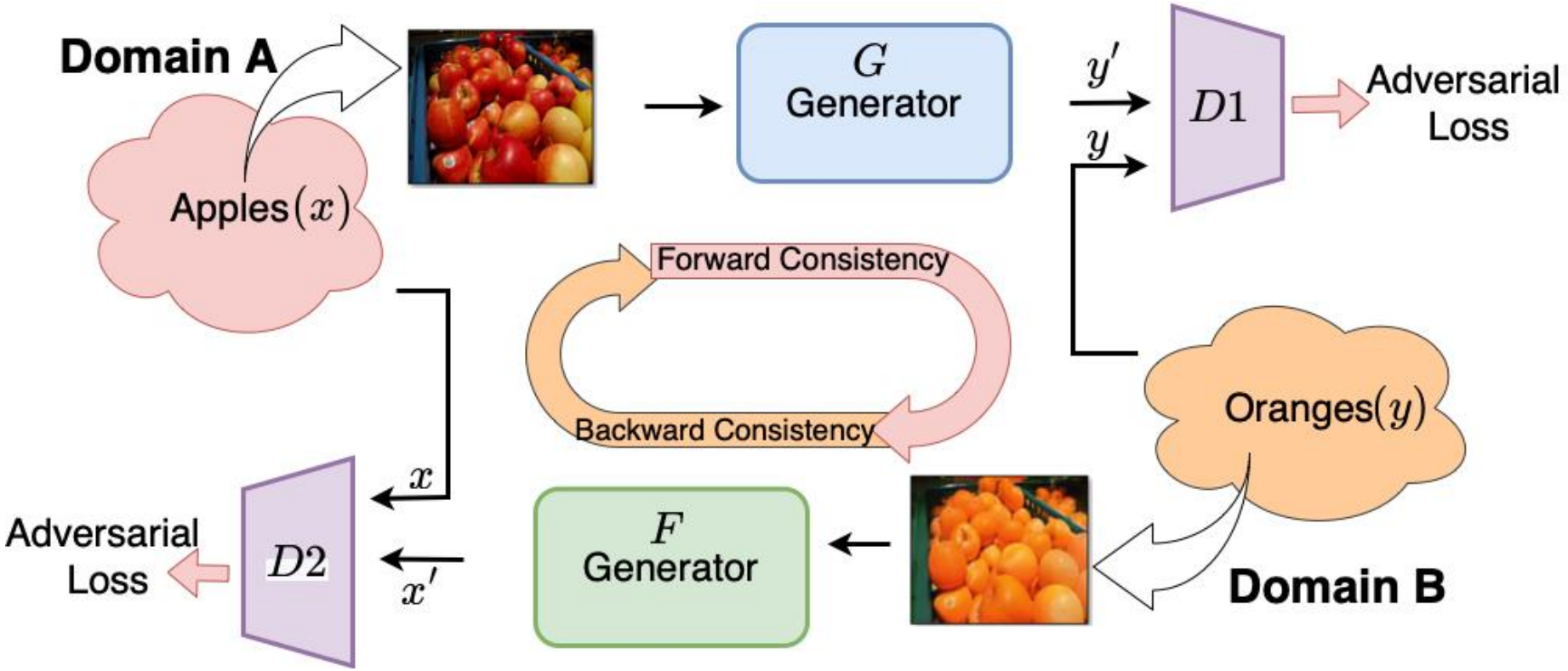
CycleGAN – Schematic

- Signal Flow

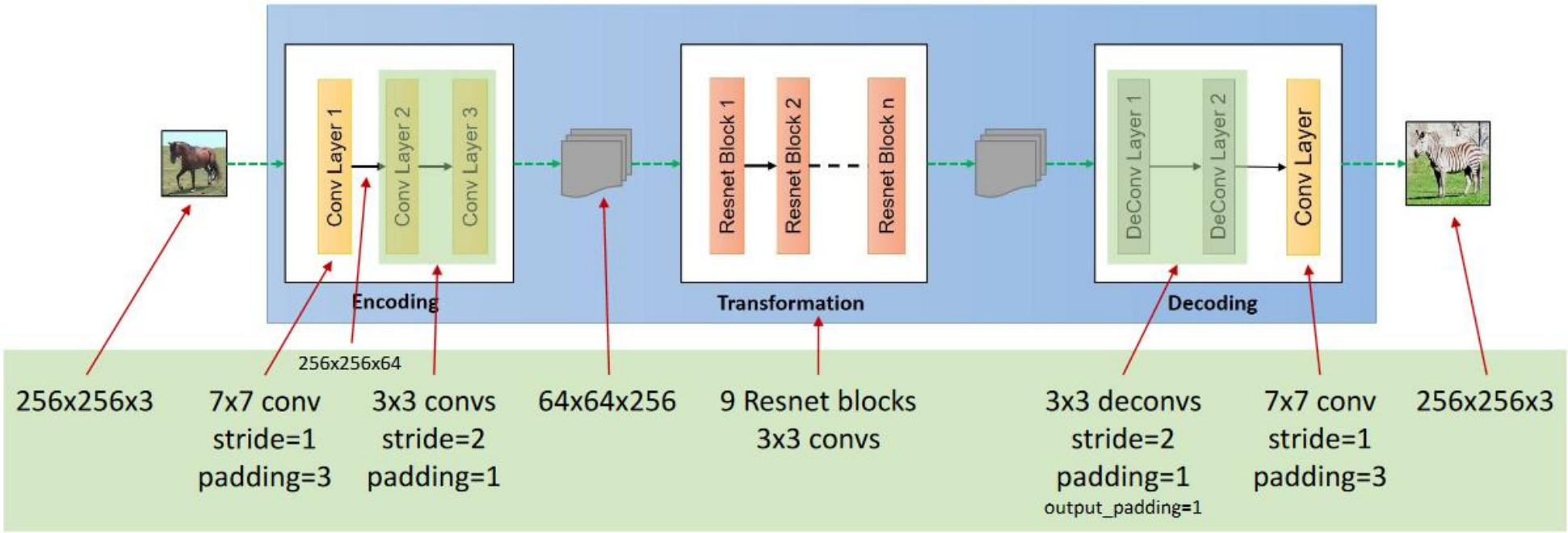


CycleGAN – Schematic

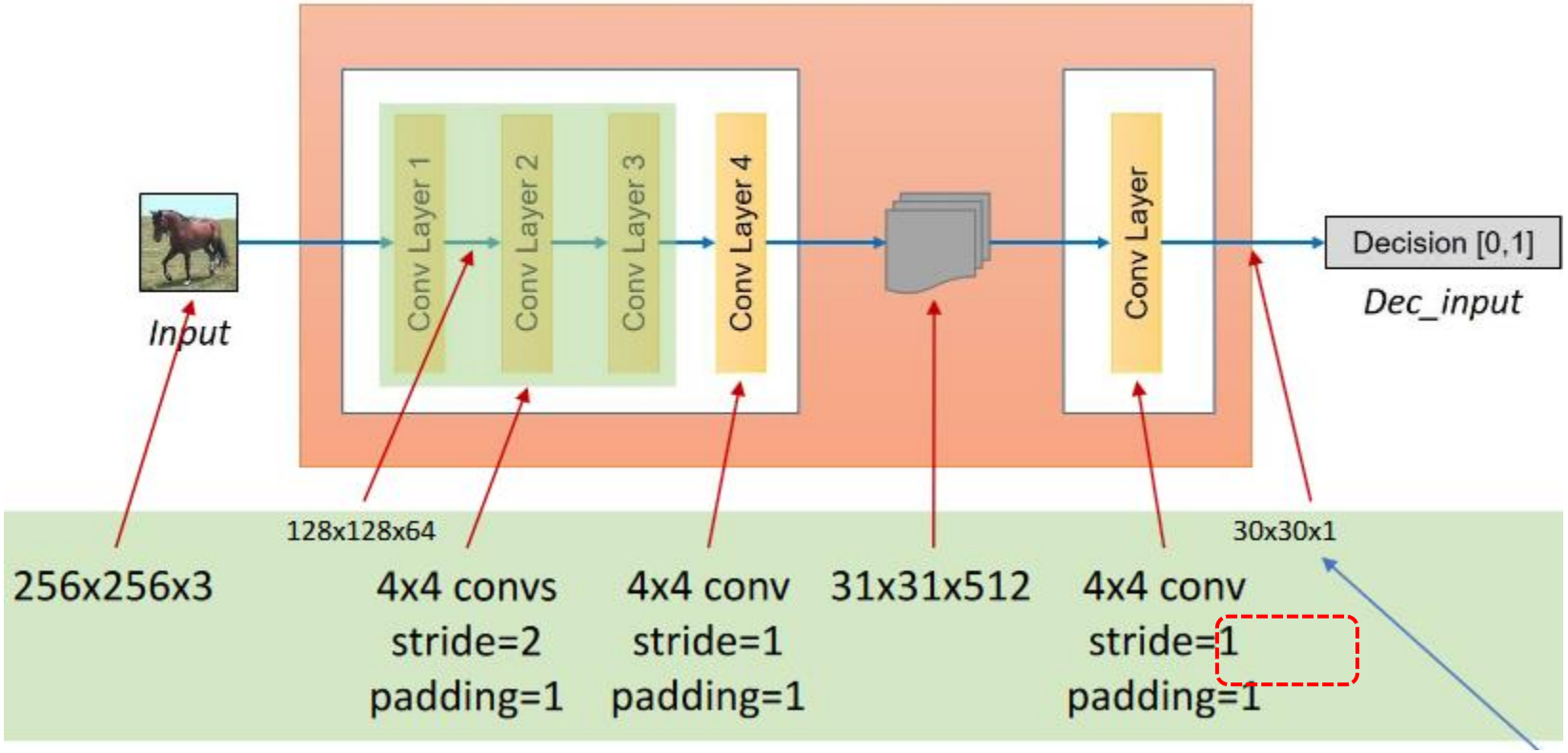
- Signal Flow



CycleGAN – Generator

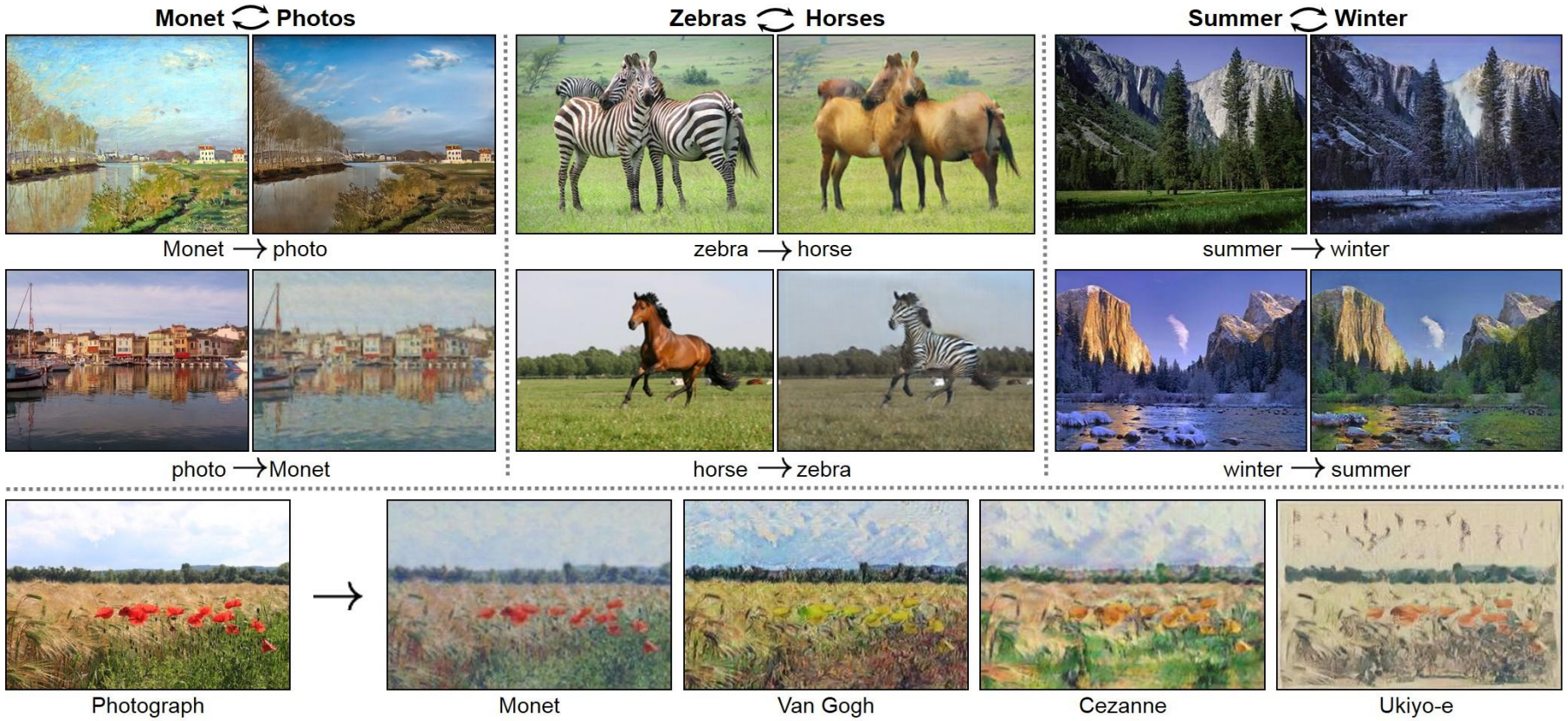


CycleGAN – Discriminator



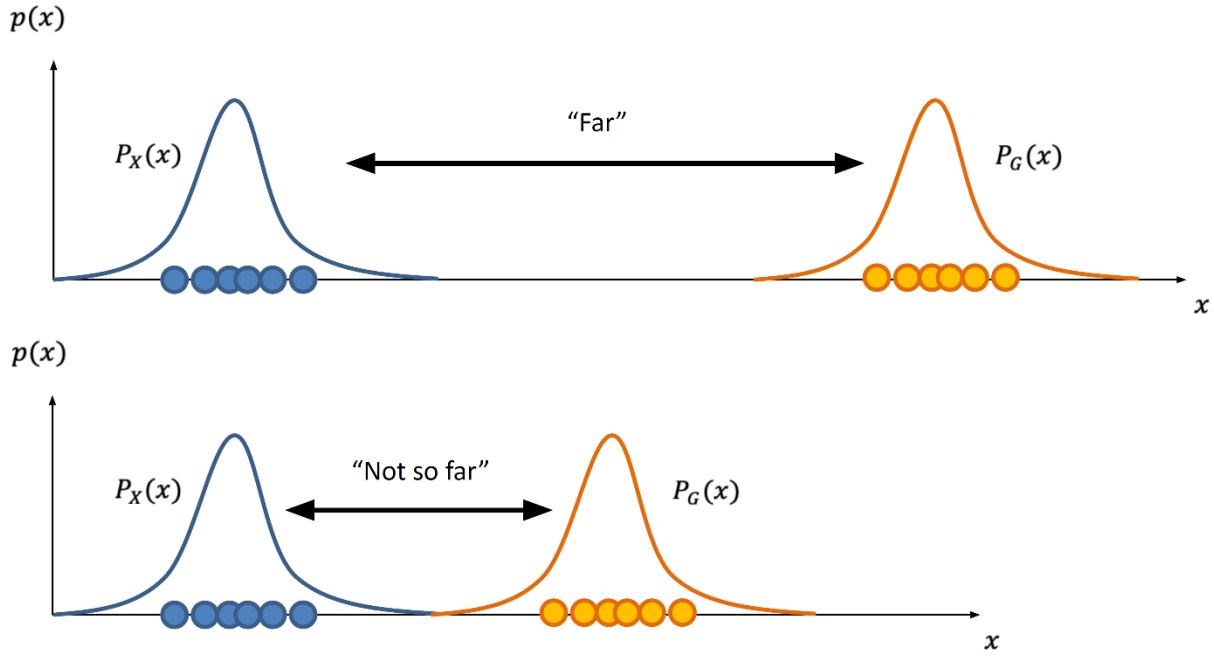
CycleGAN – Example

- https://junyanz.github.io/CycleGAN/



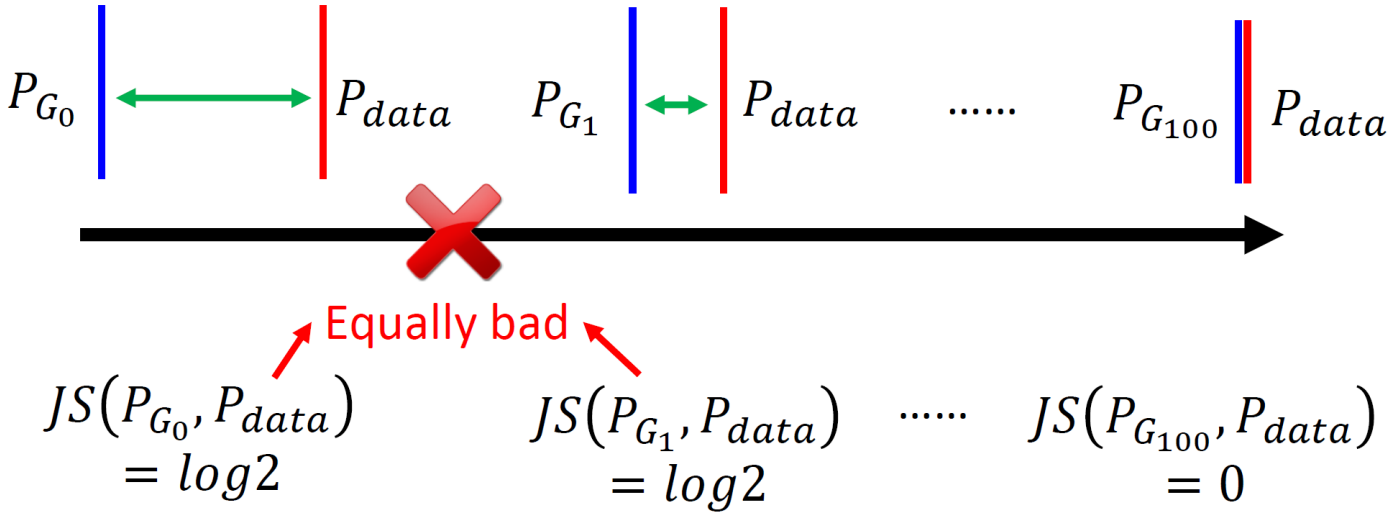
Wasserstein GAN (WGAN)

- Problem with JS or KL distance:
- In most cases, P_{data} and P_G are not overlapped (in early stage of learning)
- Even though P_{data} and P_G have overlap we have NOT sufficient samples



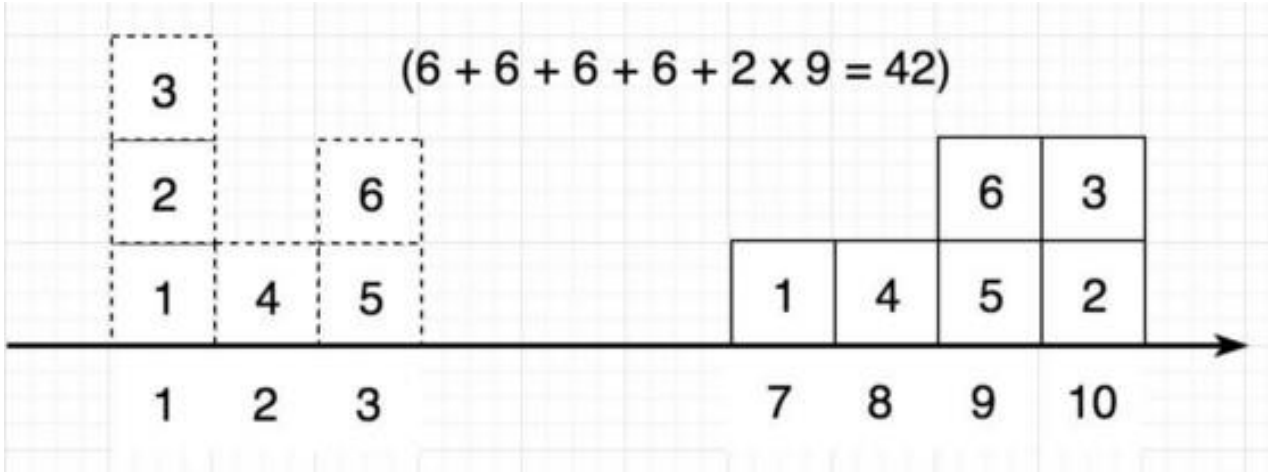
Wasserstein GAN (WGAN)

- Problem with JS or KL distance:
- In most cases, P_{data} and P_G are not overlapped (in early stage of learning)
- Even though P_{data} and P_G have overlap we have NOT sufficient samples



WGAN - Preliminary

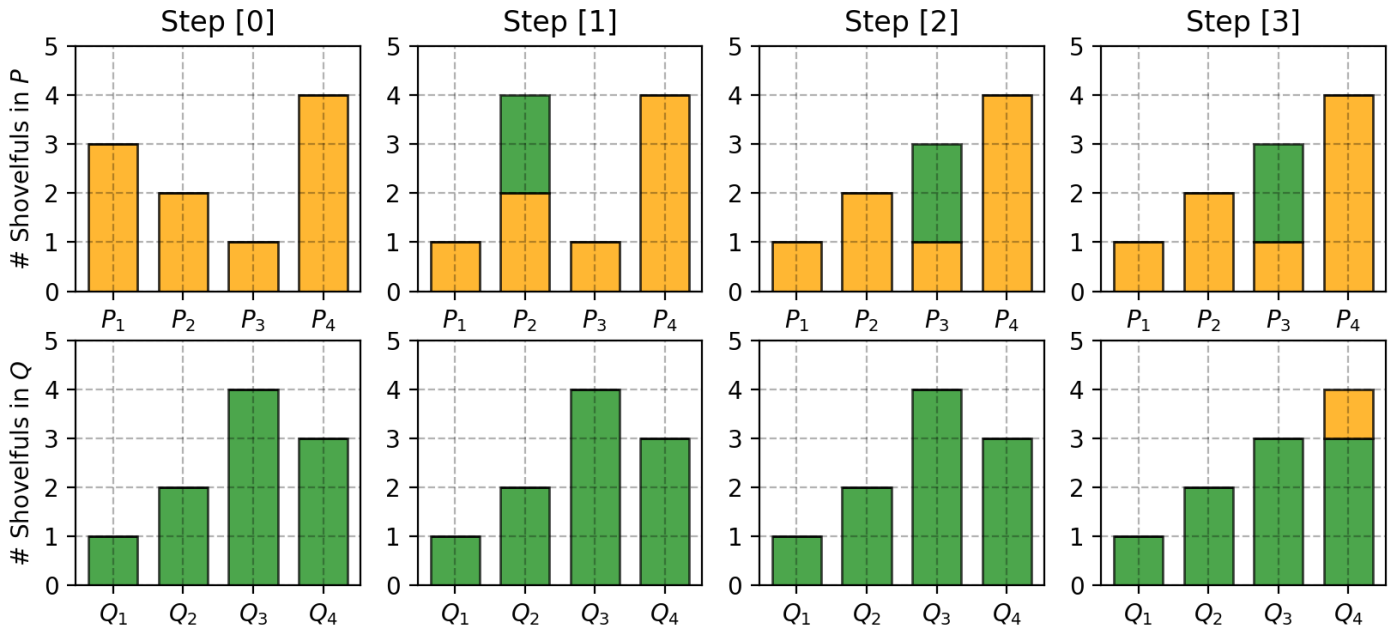
- Earth-Mover (EM) distance: Move-Box game
  - Cost of box movement: *Weight of each box* AND *Distance*
    - 1: 1→7 (6)
    - 2: 1→10 (9)
    - 3:1→10 (9)
    - 4: 2→8 (6)
    - 5: 3→9 (6)
    - 6: 3→9 (6)
    - Total Cost: 42



- Question: What is minimum cost, if we know source and target distributions

WGAN - Preliminary

- Earth Mover Distance:
  - Suppose we want match distribution P={3,2,1,4} to Q={1,2,4,3}
- Move 2 block from P1 to P2
- Move 2 block from P2 to P3
- Move 1 block from Q3 to Q4



WGAN

- Wasserstein Distance between two distribution:

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

- *inf* (infimum): Greatest Lower Bound (Best Moving strategy)
- $\Pi(p_r, p_g)$: set of all joint distributions $\gamma(x, y)$ whose marginal distributions are respectively p_r and p_g.
  - $\sum_x \gamma(x, y) = p_g(y)$
  - $\sum_y \gamma(x, y) = p_r(x)$

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

WGAN - Comparison

- The Total Variation (TV) distance:

$$\delta(P_r, P_g) = sup|P_r(A) - P_g(A)|$$

  - Sup (supremum): Least Upper Bound

- The Kullback-Leibler (KL) distance:

$$KL(P_r \parallel P_g) = \int_x \log\left(\frac{P_r(x)}{P_g(x)}\right) P_r(x)dx$$

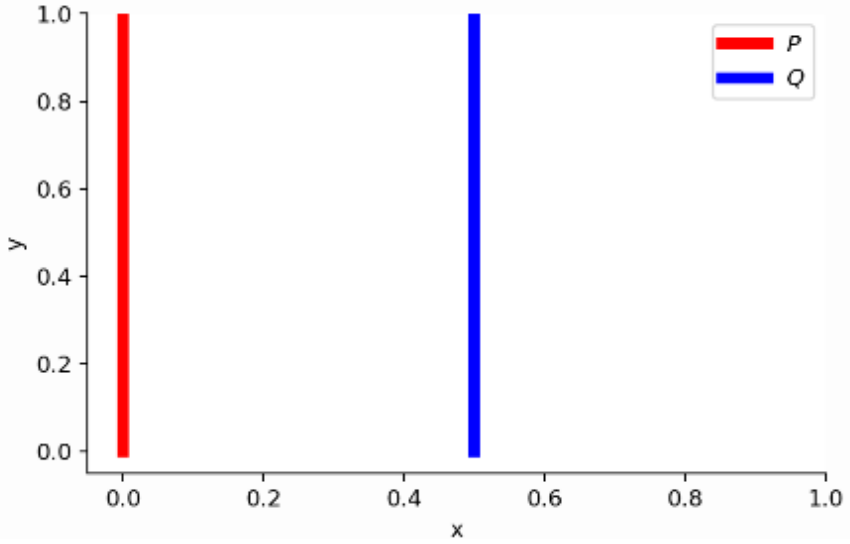
- The Jenson-Shannon (JS) distance:

$$JS(P_r, P_g) = \frac{1}{2}KL(P_r \parallel P_m) + \frac{1}{2}KL(P_g \parallel P_m), P_m = \frac{P_g + P_r}{2}$$

WGAN - Comparison

- $W(P_0, P_\theta) = |\theta|$
- $JS(P_0, P_\theta) = \begin{cases} log2, & \theta \neq 0 \\ 0, & \theta = 0 \end{cases}$
- $KL(P_0, P_\theta) = KL(P_\theta, P_0) = \begin{cases} \infty, & \theta \neq 0 \\ 0, & \theta = 0 \end{cases}$
- $\delta(P_0, P_\theta) = \begin{cases} 1, & \theta \neq 0 \\ 0, & \theta = 0 \end{cases}$
- Vanishing Gradient!
- W is Continuous (for continuous P_θ)!

$\forall (x, y) \in P, x = 0 \text{ and } y \sim U(0, 1)$
$\forall (x, y) \in Q, x = \theta, 0 \leq \theta \leq 1 \text{ and } y \sim U(0, 1)$



WGAN - Mathematics

- Kantorovich-Rubinstein duality:

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

- where $f(\cdot)$ is K-**L**ipschitz continuous:

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

- Where is deep structure? $f(\cdot)$ may be a deep (discriminator)

WGAN - Mathematics

- From mathematics:

$$\max_{w \in \mathcal{W}} \left\{ \mathbb{E}_{x \sim P_r}[f_w(x)] - \mathbb{E}_{x \sim P_\theta}[f_w(x)] \right\} \leq \sup_{\|f\|_L \leq K} \left\{ \mathbb{E}_{x \sim P_r}[f(x)] - \mathbb{E}_{x \sim P_\theta}[f(x)] \right\}$$
$$= K \cdot W(P_r, P_\theta)$$

- A maximum of a set must be an element of the set. A supremum need not be.

$$L(p_r, p_g) = W(p_r, p_g) = \max_{w \in W} \left\{ \mathbb{E}_{x \sim p_r}[f_w(x)] - \mathbb{E}_{z \sim p_r(z)}[f_w(g_\theta(z))] \right\}$$

WGAN - Mathematics

- WGAN Gradient:

$$L(p_r, p_g) = W(p_r, p_g) = \max_{w \in W} \{\mathbb{E}_{x \sim p_r}[f_w(x)] - \mathbb{E}_{z \sim p_r(z)}[f_w(g_\theta(z))]\}$$

$$\Rightarrow \nabla_\theta W(P_r, P_\theta) = \nabla_\theta \left(\mathbb{E}_{x \sim P_r}[f_w(x)] - \mathbb{E}_{z \sim Z}[f_w(g_\theta(z))]\right)$$
$$= -\mathbb{E}_{z \sim Z}[\nabla_\theta f_w(g_\theta(z))]$$

- What for «*K-Lipschitz continuous*» condition:
  - Clipping $[-c, c], c \sim 0.01$

WGAN - Training

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.

while θ has not converged **do**

  for $t = 0, \ldots, n_{\text{critic}}$ **do**

    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.

    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.

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

    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$

    $w \leftarrow \text{clip}(w, -c, c)$

  end for

  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.

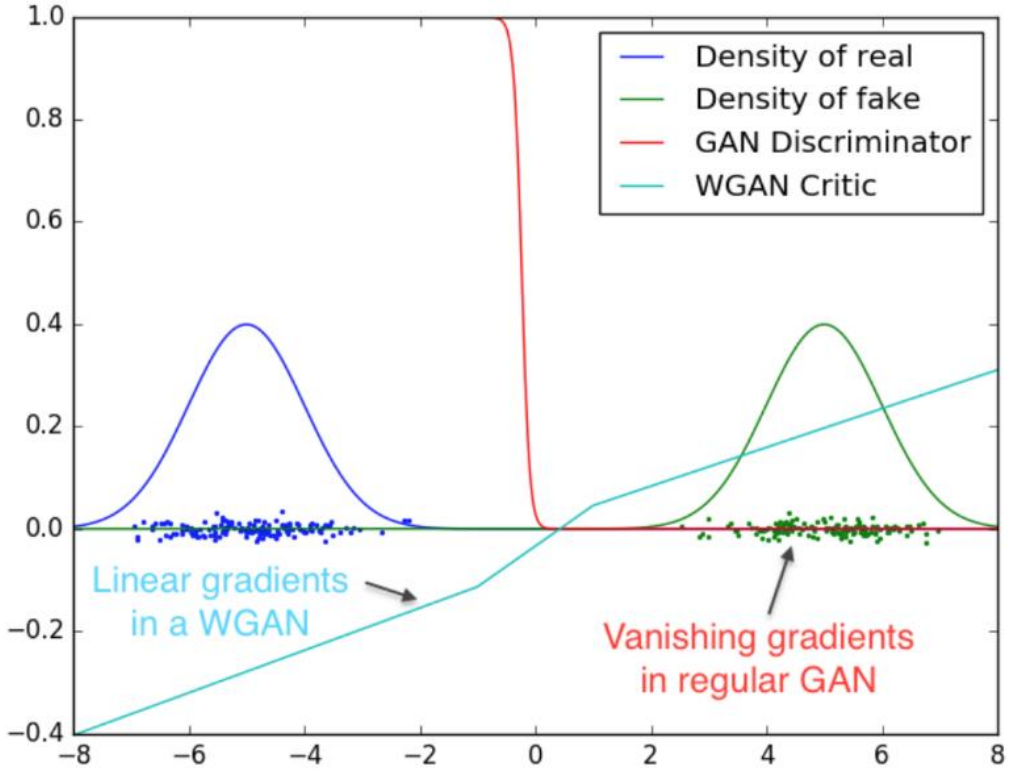
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$

  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$

end while

WGAN

- $D(x)$ plot for two Gaussian (fake/real)



WGAN

- Experiment:
  - Improved stability of learning
  - Get rid of mode collapse
  - Meaningful learning curves

Figure 5: Algorithms trained with a DCGAN generator. Left: WGAN algorithm. Right: standard GAN formulation. Both algorithms produce high quality samples.

WGAN-GP

- Alternative way to enforce the Lipschitz constraint (Gradient-Penalty), WGAN-GP

$$Loss = \underbrace{\mathbb{E}_{\hat{x}\sim\mathbb{P}_g}[D(\hat{x})] - \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)]}_{Original\ Loss} + \underbrace{\lambda\mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}[(\|\nabla_{\hat{x}}D(\hat{x})\|_2 - 1)^2]}_{Gradient\ Penalty}$$

- Sampling distribution: It implicitly define $\mathbb{P}_{\hat{x}}$ sampling uniformly along straight lines between pairs of points sampled from the data distribution $\mathbb{P}_r$ and the generator distribution $\mathbb{P}_g$.

WGAN-GP

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.

while θ has not converged **do**
  for $t = 1, ..., n_{\text{critic}}$ **do**
    for $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

GAN Tips and Tricks

- Improve GAN training Stability:
  - Change $\log(1-D) \Longrightarrow -\log(D)$ (Gradient Vanishing)
  - DCGAN (Gradient Vanishing)
  - f-divergence (Training instability)

GAN – Roots of Problems

- GD/SGD is not well solver for “two player game”
- Simultaneous updates needs $G \Leftrightarrow D$ balance
- No guarantee to reach Nash equilibrium
- Mode Collapse
- Adversarial optimization (2 player game) is harder than 1 player game.

GAN – Roots of Problems

- G/D balance:
  - Weak trained discriminator $\Longrightarrow$ Worse situation for generator
  - Over–trained discriminator $\Longrightarrow$ Weak gradient to update generator
- How to balance:
  - Depend on **G/D** network complexity
  - Adjust two different learning rates
  - Make **D** is ahead of **G** for training: *k minibatches* (**D**) against 1 (**G**)
  - Train **D** to reach a pre-defined loss then switch to **G** (Better):
    - if $D_{loss} > DT_{HR}$ then train **D**
    - if $G_{loss} > GT_{HR}$ then train **G**

What to do with GAN!

- GAN is tricky to train
- There is no General trick.
- Most papers: Claim to solve problems!
- Then: A combination of methods and tricks

GAN Progress with Time

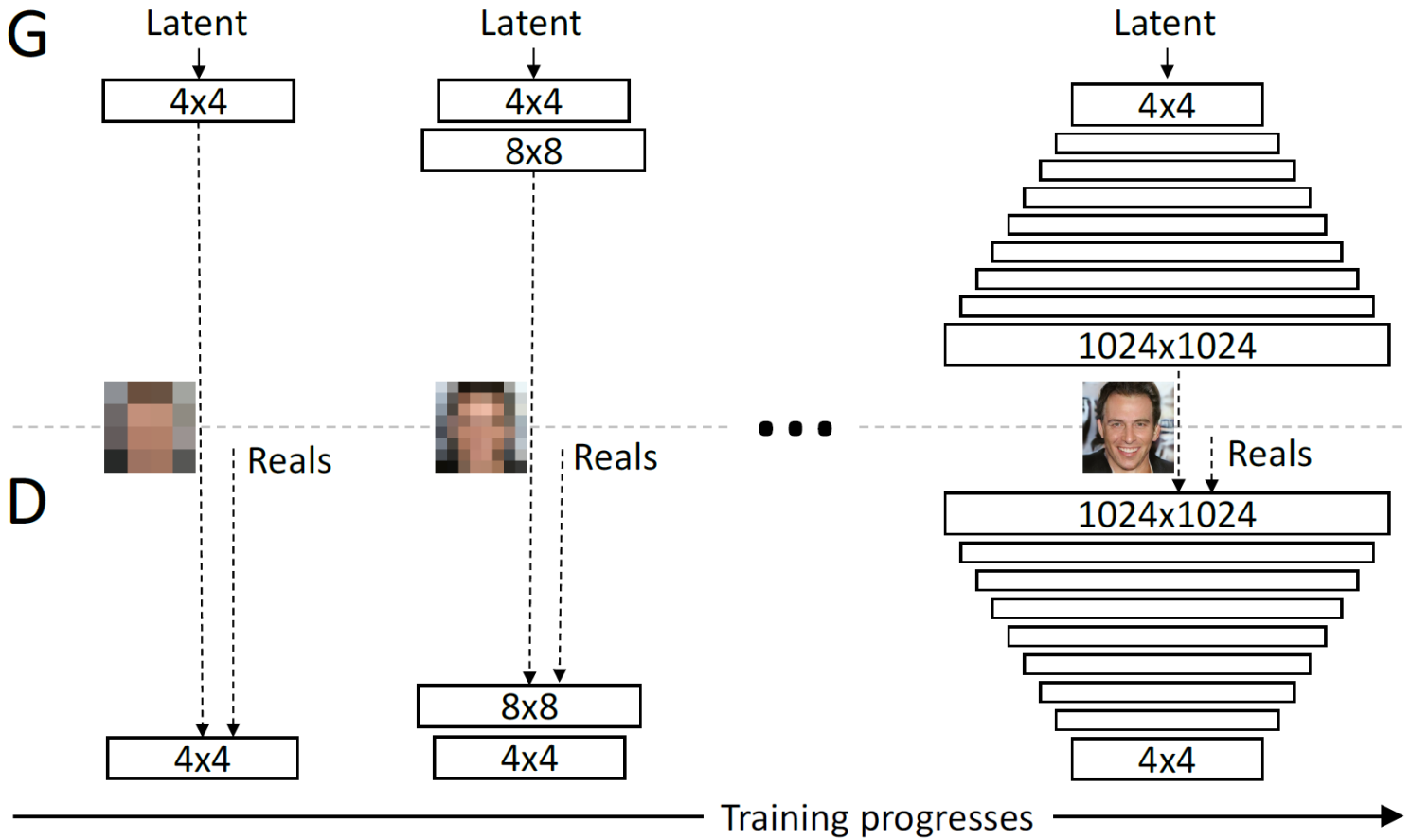
Deep Learning Last Words

- Suggested works:
  - StyleGAN (1-2-3) (NVIDIA Lab)
  - StyleGAN-T (Text to Image)
  - StarGAN
  - LAPGAN
  - ProGAN
  - BiGAN
  - ….

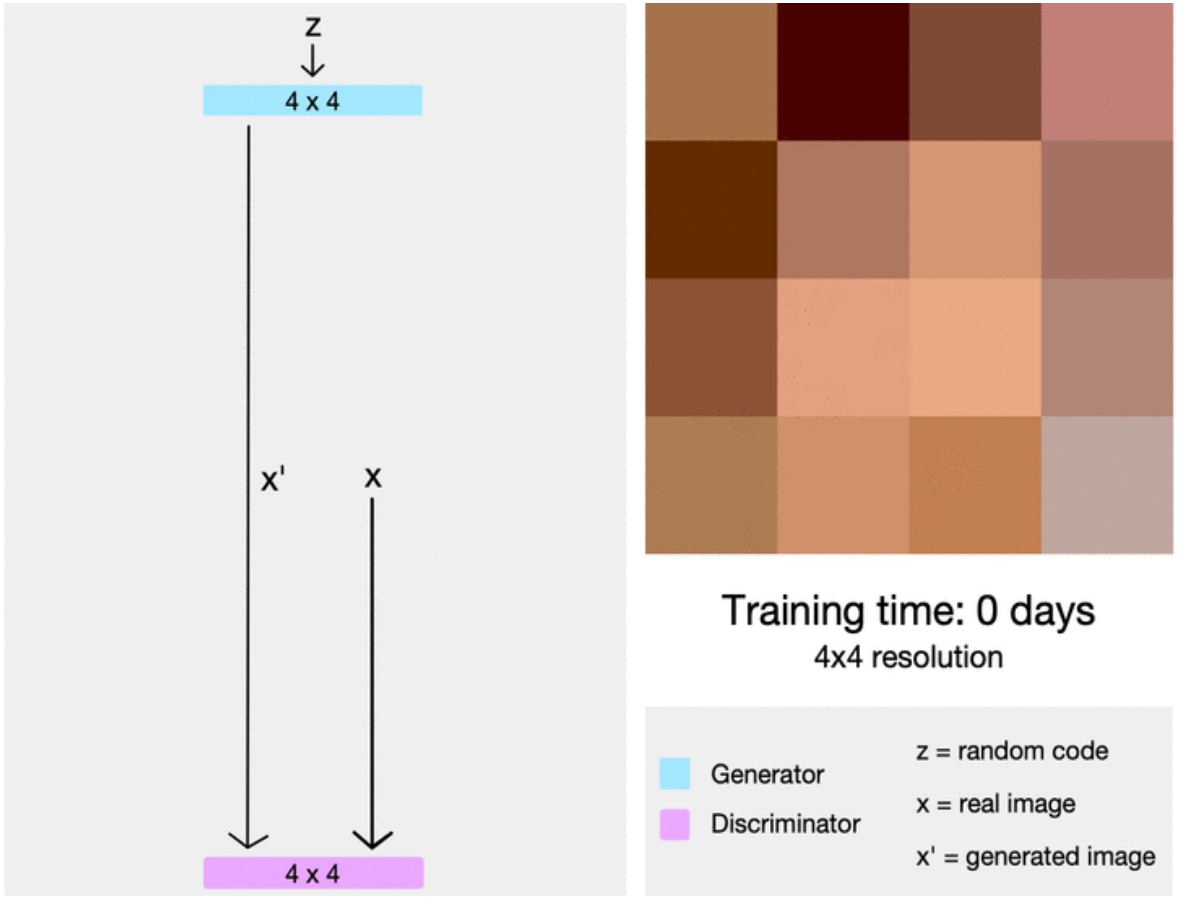
Progressive GAN

- Solve low quality generated image issues:
- It use generator and discriminator networks that are mirror images of each other and always grow in synchrony. Training starts with both the generator (G) and discriminator (D) having a low spatial resolution of 4×4 pixels.
- As the training advances, it incrementally add layers to G and D, thus increasing the spatial resolution of the generated images. All existing layers remain *trainable* throughout the process.
- $\boxed{N \times N}$: refers to convolutional layers operating on $N \times N$ spatial resolution.

Progressive GAN

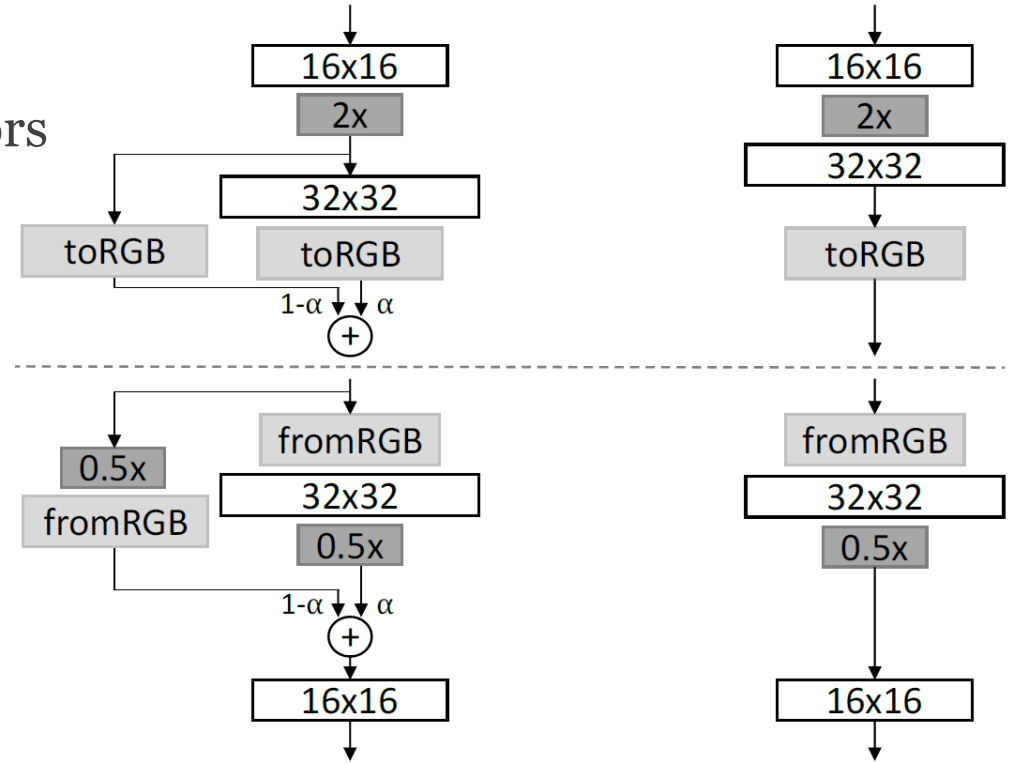


Progressive GAN



Progressive GAN

- When new layers are added to the networks, we fade them in smoothly, as illustrated here:
- toRGB : Projects feature vectors to RGB colors
- fromRGB : Reverse of toRGB

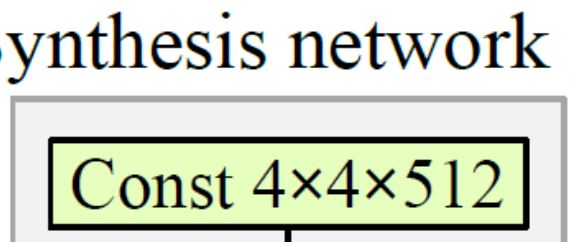


StyleGAN - Nvidia

- Most Powerful GAN in the market
- There are three changes in the StyleGAN Generator:
  - Backbone is Progressive GAN (training loss: WGAN-GP)
  - A starting learnable constant
  - Mapping network
  - Adaptive Instance Normalization (AdaIN)
  - Noise Injection for Stochastic Variation

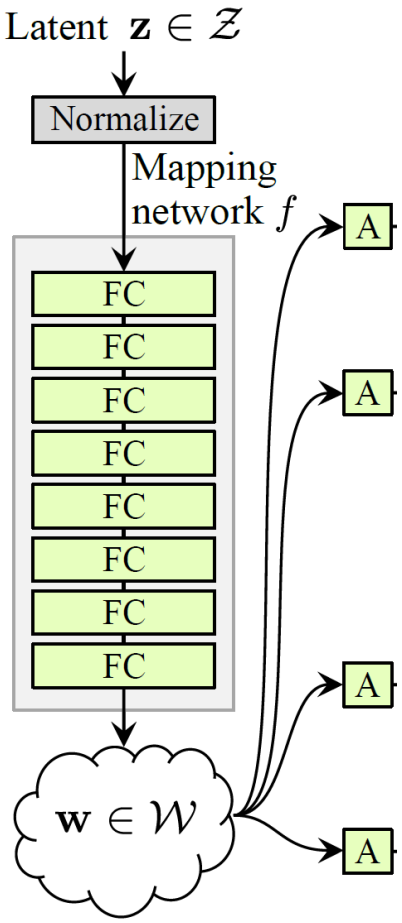
StyleGAN - Starting Constant

- Vanilla *GANs Generator* generates images from a latent code (random samples). Instead, StyleGAN starts with a constant learned image of size 4x4. The progressive growth of the generator will add new content and upscale this image by applying style and noise at each block.



StyleGAN -Mapping Network

- The latent code z, which we often referred in GAN as the mapping of the image at latent space, is now used to produce the *style* of the image.
- The vector z is first sampled from a predefined distribution (uniform or Gaussian) at latent space Z. Then it is mapped into an intermediate latent space W to produce w. The mapping network is implemented using an 8-layer MLP ($\mathbb{R}^{512} \Rightarrow \mathbb{R}^{512}$)
- An affine transformation, $\boxed{\text{A}}$, on the intermediate latent code w will produce *style* $y = (y_s, y_b)$ and feed into the AdaIN layer, following up with a convolution to draw new contents to the image.
- It is shown this network help to entangled problem.



StyleGAN - Adaptive Instance Normalization (AdaIN)

- The latent code w produced by the mapping network is then fed into a learned affine transform and AdaIN layer. The affine transform is implemented using two linear layers to create a style with *scale* = y_s and *bias* =y_b. The AdaIN operation formula, after each convolution layer of the synthesis network G :

$$AdaIN(x_i, y) = y_{s,i}\frac{x_i - \mu(x_i)}{\sigma(x_i)} + y_{b,i}$$

- where each feature map x_i is normalized separately, and then scaled and biased using the corresponding scalar components from *style* y. Thus the dimensionality of y is twice the number of feature maps on that layer.

StyleGAN - Adaptive Instance Normalization (AdaIN)

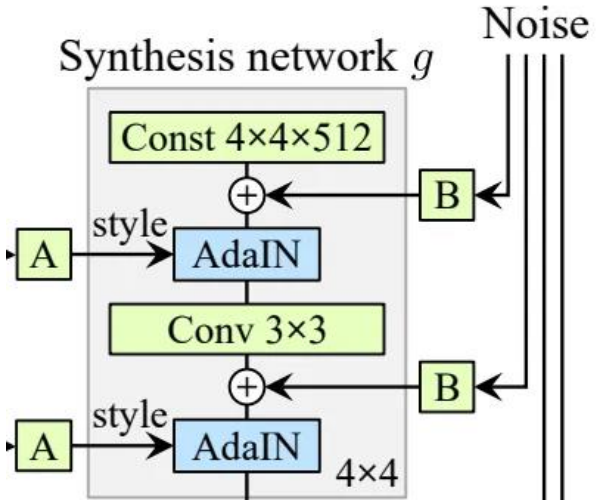
- Style transfer using normalization:

$$AdaIN(x_i, y) = y_{s,i}\frac{x_i - \mu(x_i)}{\sigma(x_i)} + y_{b,i}$$

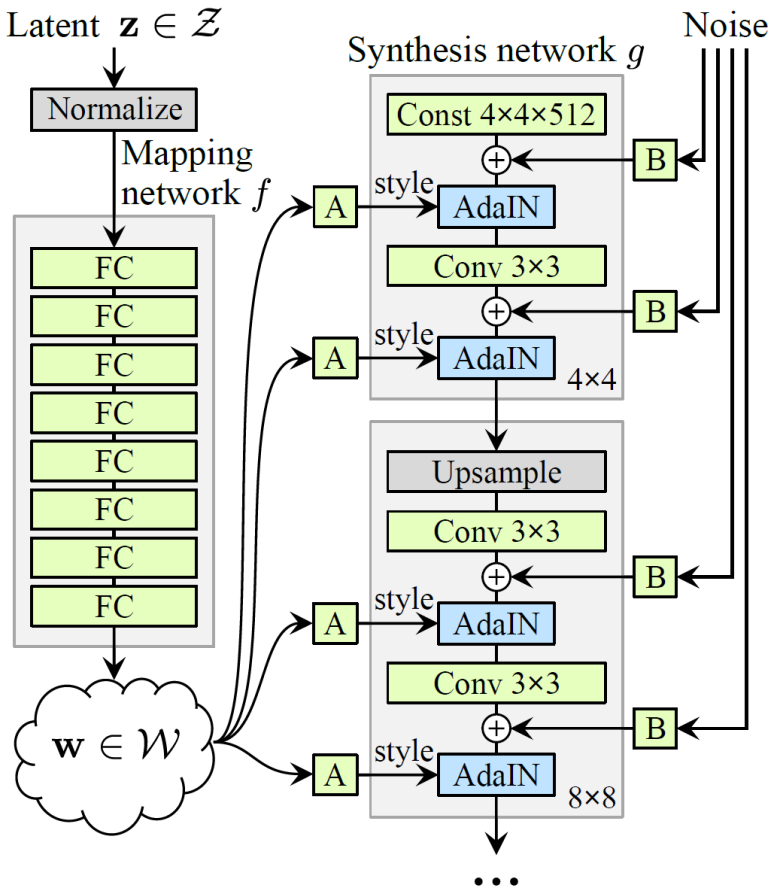
- This means the *style* y will control the statistic of the feature map for the next convolutional layer. The style decides which channels will have more contribution in the next convolution.

StyleGAN - Noise Injection

- Stochastic variations in an image are small details that do not change the overall context of the image. For example, hair placement, smile angle, stubble, freckles, etc. They do not change the overall image but are there, and the Generator will learn to generate them. The Noise Injection to the StyleGAN generator before AdaIN layers help generate such variations.



StyleGAN – Overall System



StyleGAN – Overall System

- Generations:
  - ProGAN (2017)
  - StyleGAN-1 (2019)
  - StyleGAN-2 (2020)
  - StyleGAN-3 (2021)
  - StyleGAN-T (2023), Text Prompt

Any Question