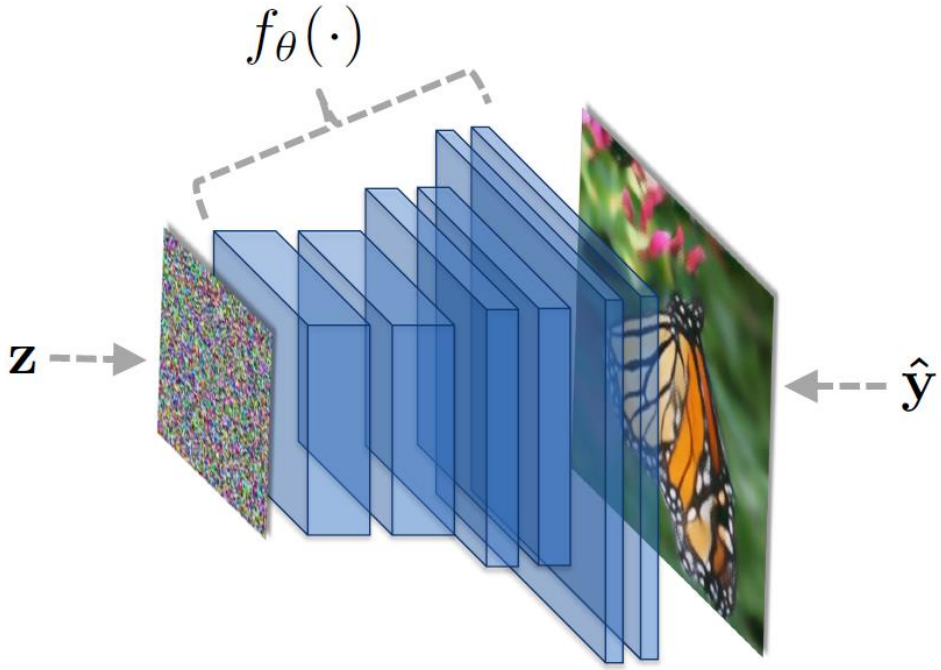
GAN Farm

Deep Learning Course
Sharif University of technology
Fall 2021
E. Fatemizadeh

Generative Adversarial Network - **GAN**

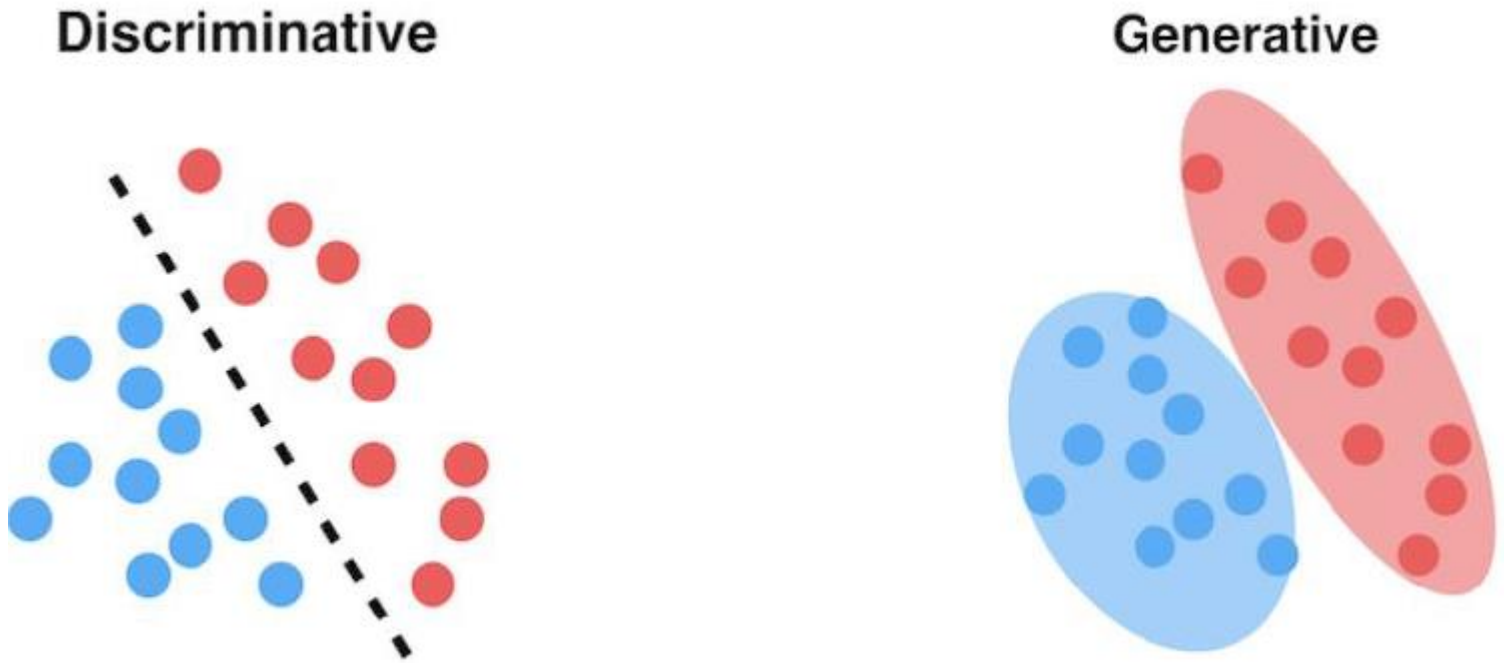
› GAN – Final Goal!



› GAN Power: https://research.nvidia.com/publication/2017-10_Progressive-Growing-of

Generative Adversarial Network - **GAN**

› Just to Remember:



Generative Adversarial Network - GAN

- GAN:
- Generative: Learn a Generative Model
- Adversarial: Trained in an Adversarial Setting
- Network: Use Deep Neural Networks

Generative Adversarial Network - GAN

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

Generative Adversarial Network - GAN

- 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

Generative Adversarial Network - GAN

- 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:

$$u_1(q_1,q_2)=q_1(A-q_1-q_2)-q_1c_1=q_1(A-c_1-q_1-q_2)$$
$$u_2(q_1,q_2)=q_2(A-q_1-q_2)-q_2c_2=q_2(A-c_2-q_1-q_2)$$

Generative Adversarial Network - GAN

- An Example from Game Theory:
  - Each one want to maximize its own profits

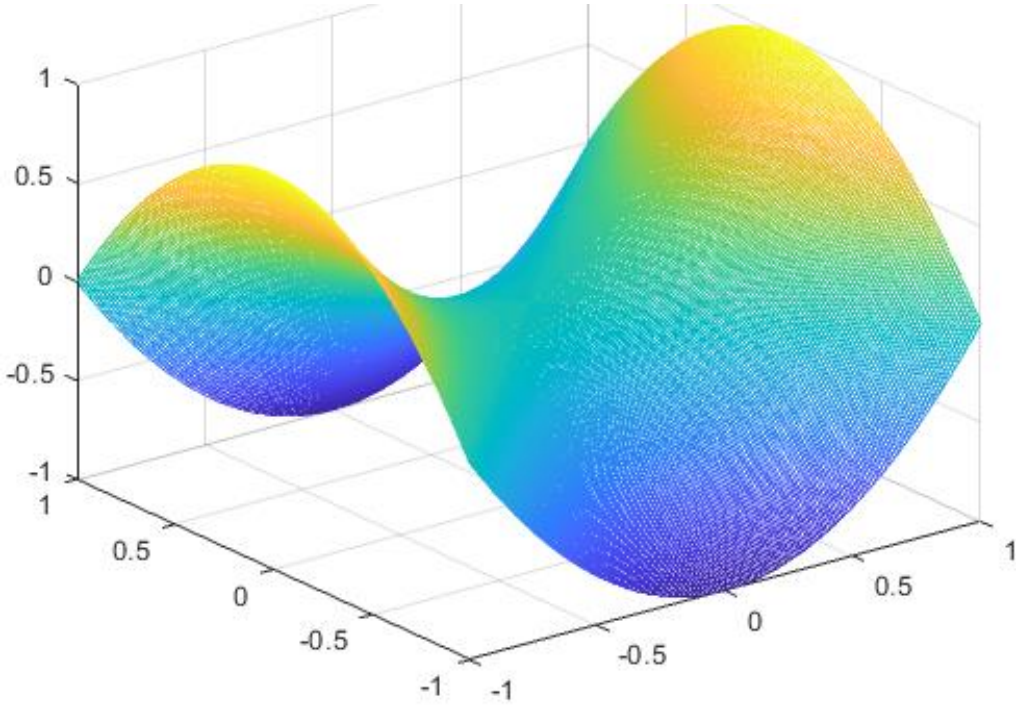
$$u_1(q_1,q_2)=q_1(A-q_1-q_2)-q_1c_1=q_1(A-c_1-q_1-q_2)$$
$$u_2(q_1,q_2)=q_2(A-q_1-q_2)-q_2c_2=q_2(A-c_2-q_1-q_2)$$
$$\frac{\partial u_1(q_1,q_2)}{\partial q_1}=0\rightarrow q_1=\frac{A-c_1-q_2}{2}$$
$$\frac{\partial u_2(q_1,q_2)}{\partial q_2}=0\rightarrow q_2=\frac{A-c_2-q_1}{2}$$
$$q_1^*=\frac{A+c_2-2c_1}{3},\quad q_2^*=\frac{A+c_1-2c_2}{3}$$

  - Nash Equilibrium!

Generative Adversarial Network - GAN

› A miniMax problem:
  – Maximizing your own gain, and minimizing your opponent's gain
  – Example:

$$\min_{x} \operatorname{Max}_{y} \left(x^2 - y^2 \right)$$



Generative Adversarial Network - GAN

- Again Divergence Measure:
  - KL (Kullback–Leibler) Divergence

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

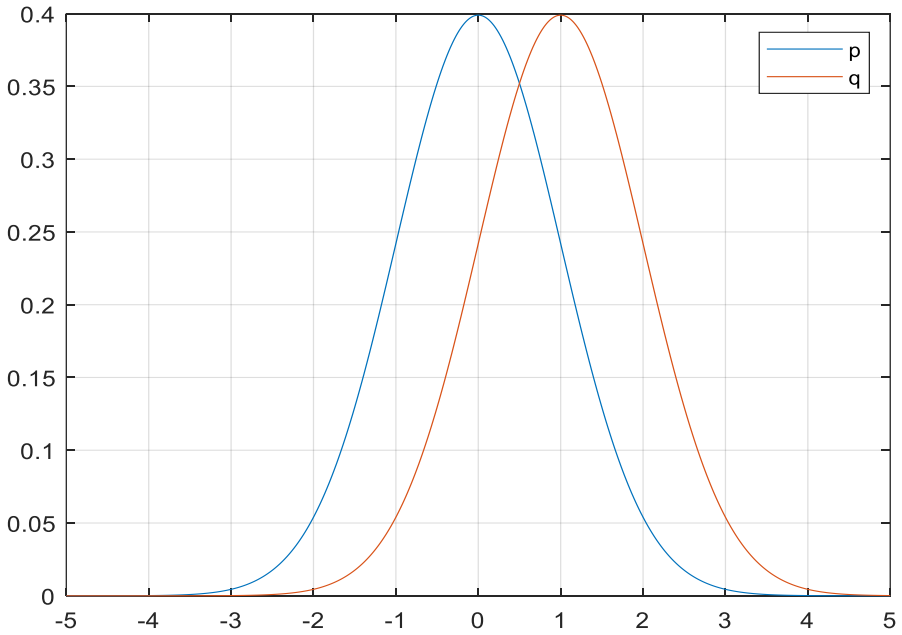
  - Asymmetric:
    - For $p(x)$ close to zero and non-zero $q(x)$: $q(x)$ effect discarded! ☹
    - Unbounded!

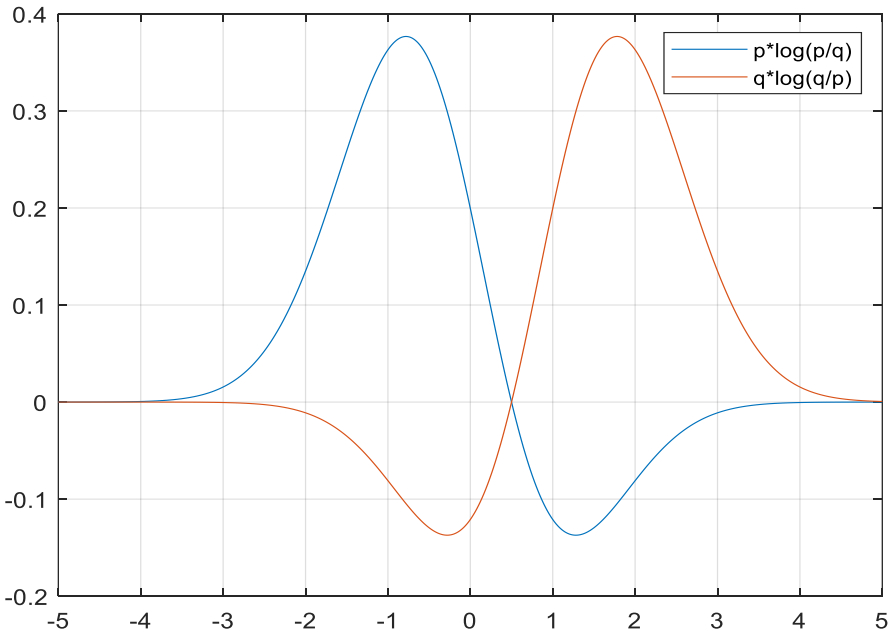
Generative Adversarial Network - GAN

› Again Divergence Measure:

– Two Gaussian: p=GaussianPdf(0,1), p=GaussianPdf(1,1)

$$p(x)\log\frac{p(x)}{q(x)}$$





Generative Adversarial Network - GAN

› JS (Jensen–Shannon) Divergence

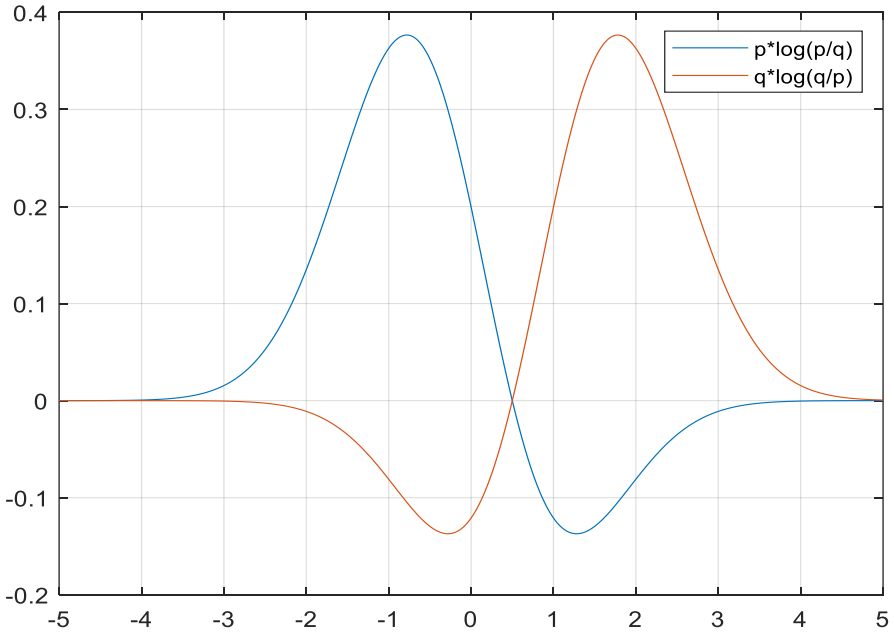
$$D_{JS}(p\|q) = \frac{1}{2}D_{KL}(p\|\frac{p+q}{2}) + \frac{1}{2}D_{KL}(q\|\frac{p+q}{2})$$

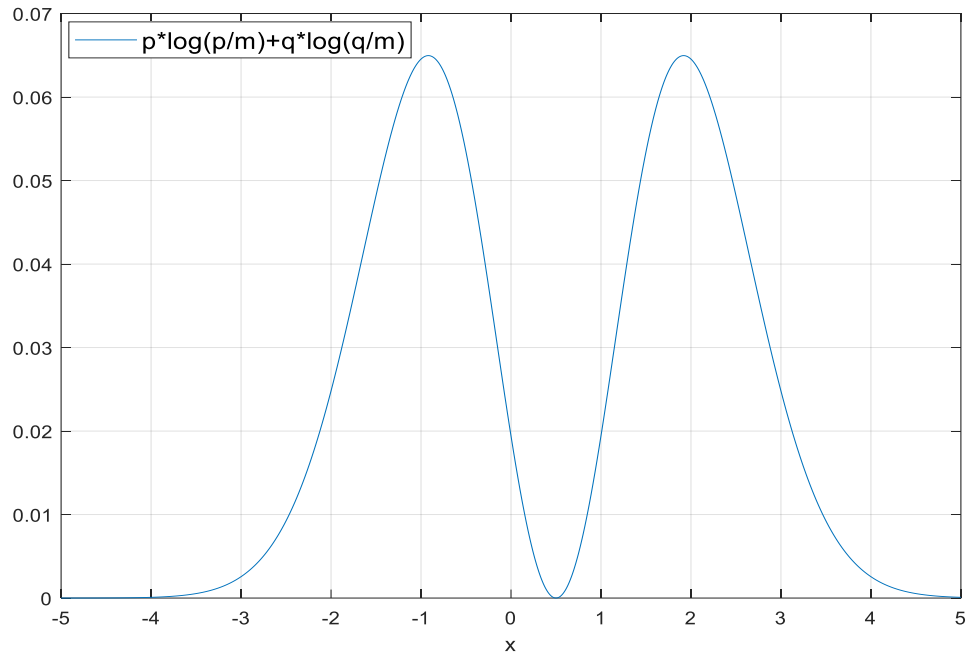
– Symmetric and Bounded (0 – $ln(2)$

– $m=(p+q)/2$

Generative Adversarial Network - GAN

› JS (Jensen–Shannon) Divergence
  – $m=(p+q)/2$





Generative Adversarial Network - GAN

- GAN final goal:
  - Generation of samples from some distribution
  - Create Art
  - Image-to-Image translation (Aerial images to Maps)

Generative Adversarial Network - GAN

› Same as VAE/CVAE but better performance!

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

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

or

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

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

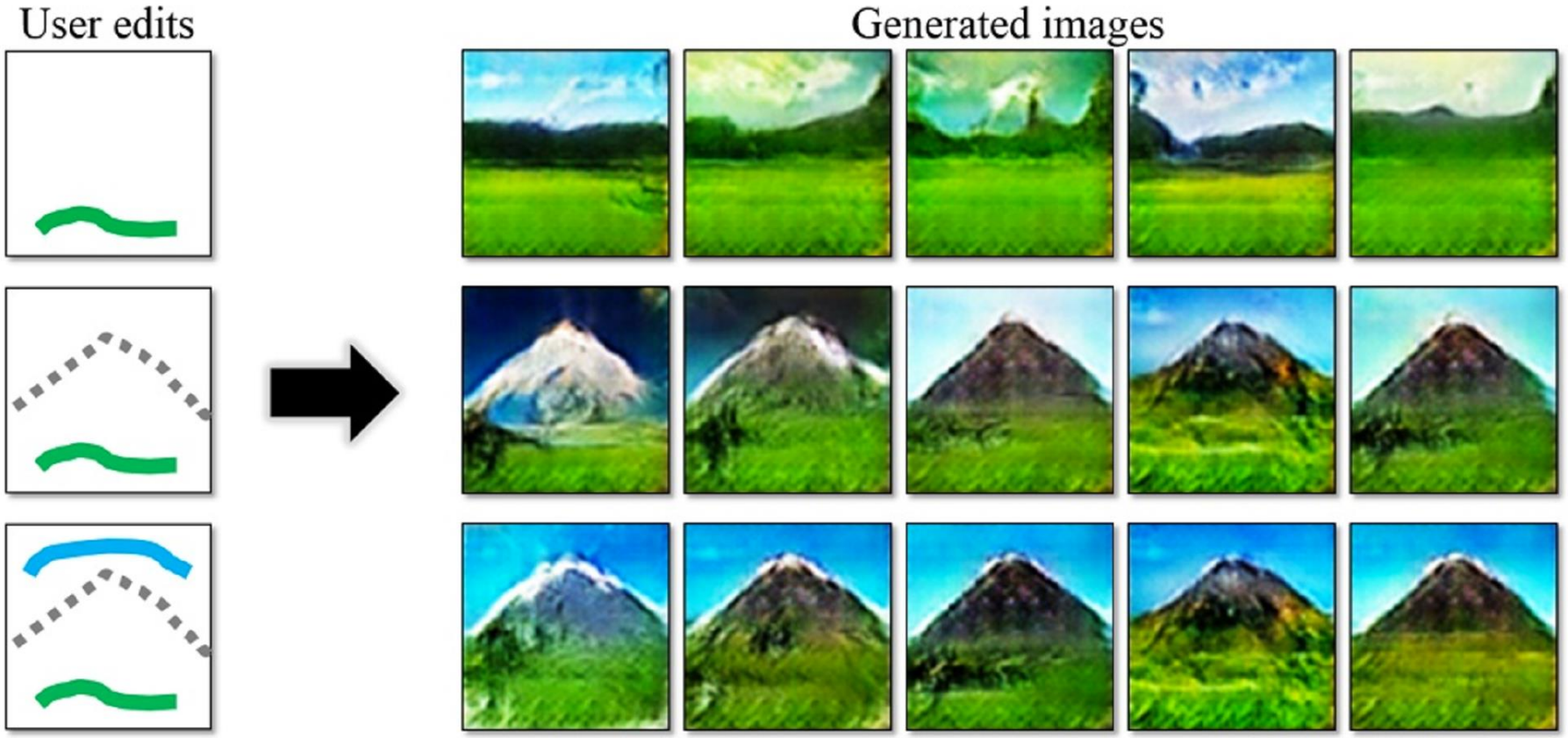
Generative Adversarial Network - GAN

› Some Magic of GAN

Which one is Computer generated?

Generative Adversarial Network - GAN

› Some Magic of GAN



Generative Adversarial Network - GAN

- Mathematical Preliminary:
  - Implicit Density Models
- Goal: Compare true probability $p(x)$ with approximate $q(x|z)$
- n samples from true pdf and n' samples from model
- Ratio Test:

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

Generative Adversarial Network - GAN

› Mathematical Preliminary:

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

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

› $p(y=1) = \pi$

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

› $\frac{1-\pi}{\pi} \approx \frac{n\prime}{n}$

Generative Adversarial Network - GAN

› Mathematical Preliminary:

› Class weighted ration for a binary classifier

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

› Let define **Discrimination** function: $D(x;\omega_D)=p(y=1|x)$

› A *Bernoulli* scoring function:

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

Generative Adversarial Network - GAN

› Mathematical Preliminary:

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

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

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

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

Generative Adversarial Network - GAN

- 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:

$$V(w_D) = \pi E_{p(\mathbf{x}|y=1)} [\log D(\mathbf{x}; w_D)] + (1-\pi) E_{p(\mathbf{x}|y=0)} [\log(1 - D(\mathbf{x}; w_D))]$$

$$\mathcal{V}(w_D, w_G) = \pi E_{p(\mathbf{x})} [\log D(\mathbf{x}; w_D)] + (1-\pi) E_{p(z)} [\log(1 - D(G(z; w_G); w_D))]$$

Generative Adversarial Network - GAN

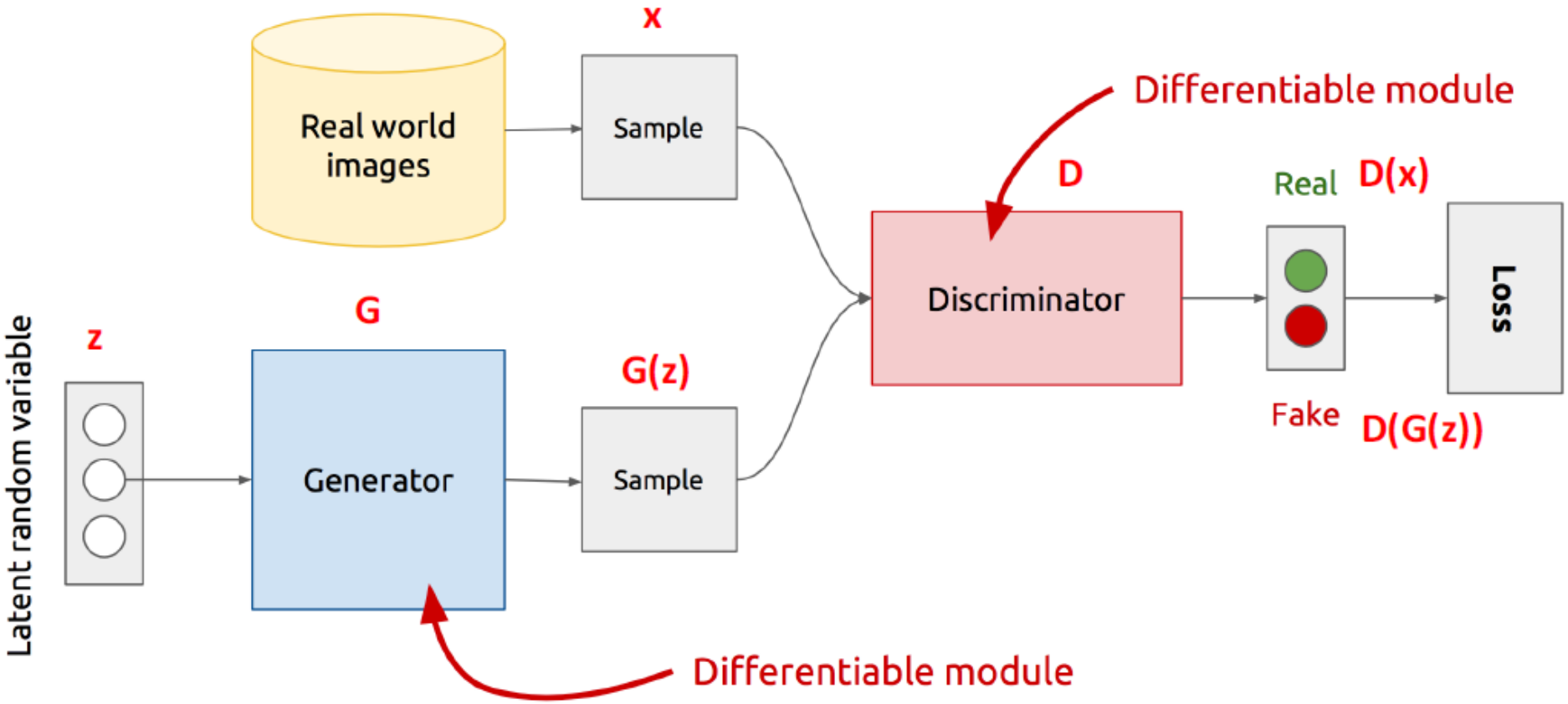
› Mathematical Preliminary:

› Final Loss (in many cases π=0.5):

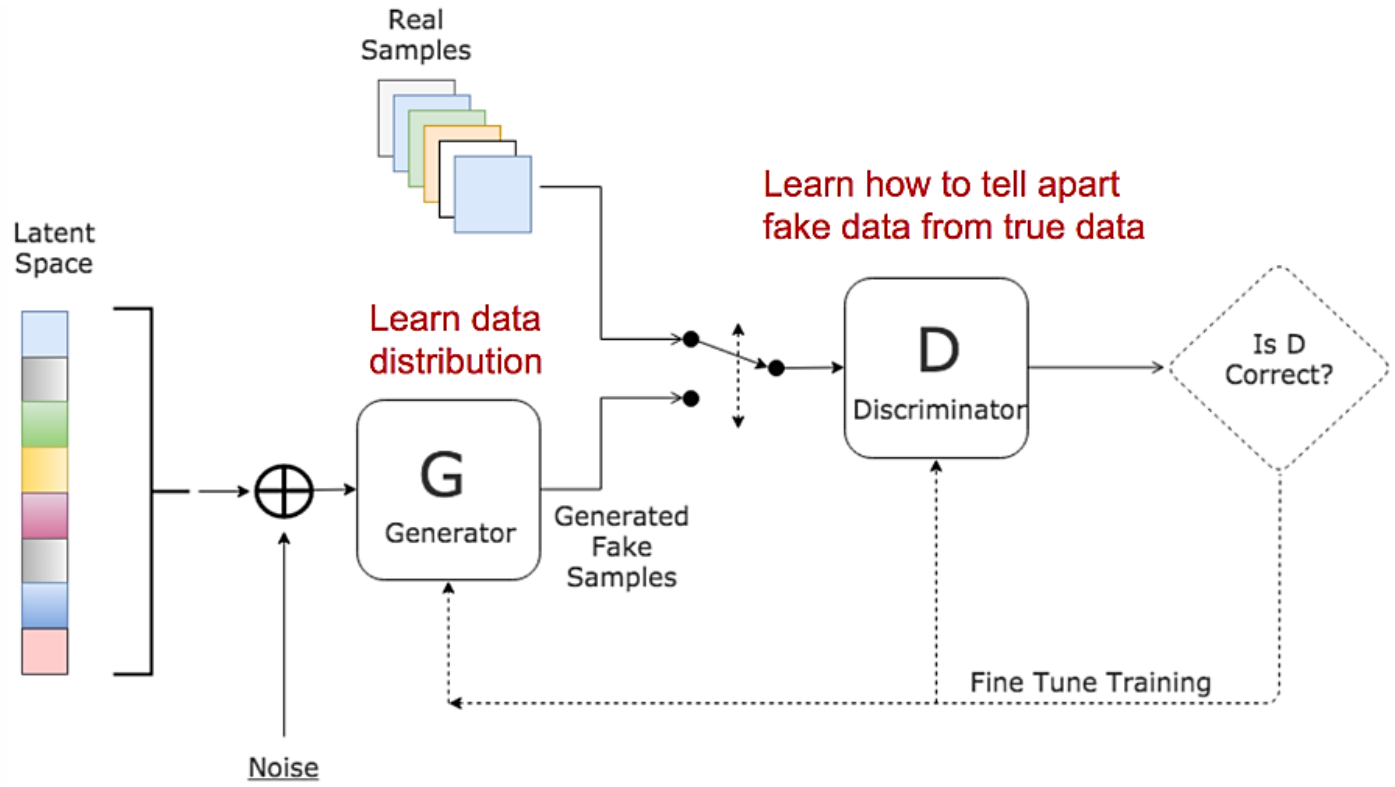
$$\mathcal{V}(w_D, w_G) = \pi E_{p(\mathbf{x})}[\log D(\mathbf{x}; w_D)] + (1-\pi) E_{q(\mathbf{x}|z)}[\log(1 - D(\mathbf{x}; w_D))]$$

Generative Adversarial Network - GAN

› GAN Architecture!



Generative Adversarial Network - GAN



Generative Adversarial Network - GAN

- **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.

Symbol	Meaning	Notes
p_z	Data distribution over noise input z	Usually, just uniform.
p_g	The generator's distribution over data x	
p_r	Data distribution over real sample x	

Generative Adversarial Network - GAN

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

$$\mathbb{E}_{x \sim p_r(x)} [\log D(x)]$$

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

$$\mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

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

$$\mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

Generative Adversarial Network - GAN

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

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

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

Generative Adversarial Network - GAN

› Optimal Discriminator:

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

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

› Results of Calculus of variation:

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

Generative Adversarial Network - GAN

› Optimal Discriminator:

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

› For optimal Generator ($pg = p_r$), then optimal discriminator becomes **0.5** (Nash equilibrium)

› And optimal value of loss function: -2log(2)

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

Generative Adversarial Network - GAN

› Loss function for Optimal **Discriminator**:

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

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

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

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

$$m=\frac{p_r+pg}{2}$$

Generative Adversarial Network - GAN

› Optimal **Discriminator** and Ratio Test:

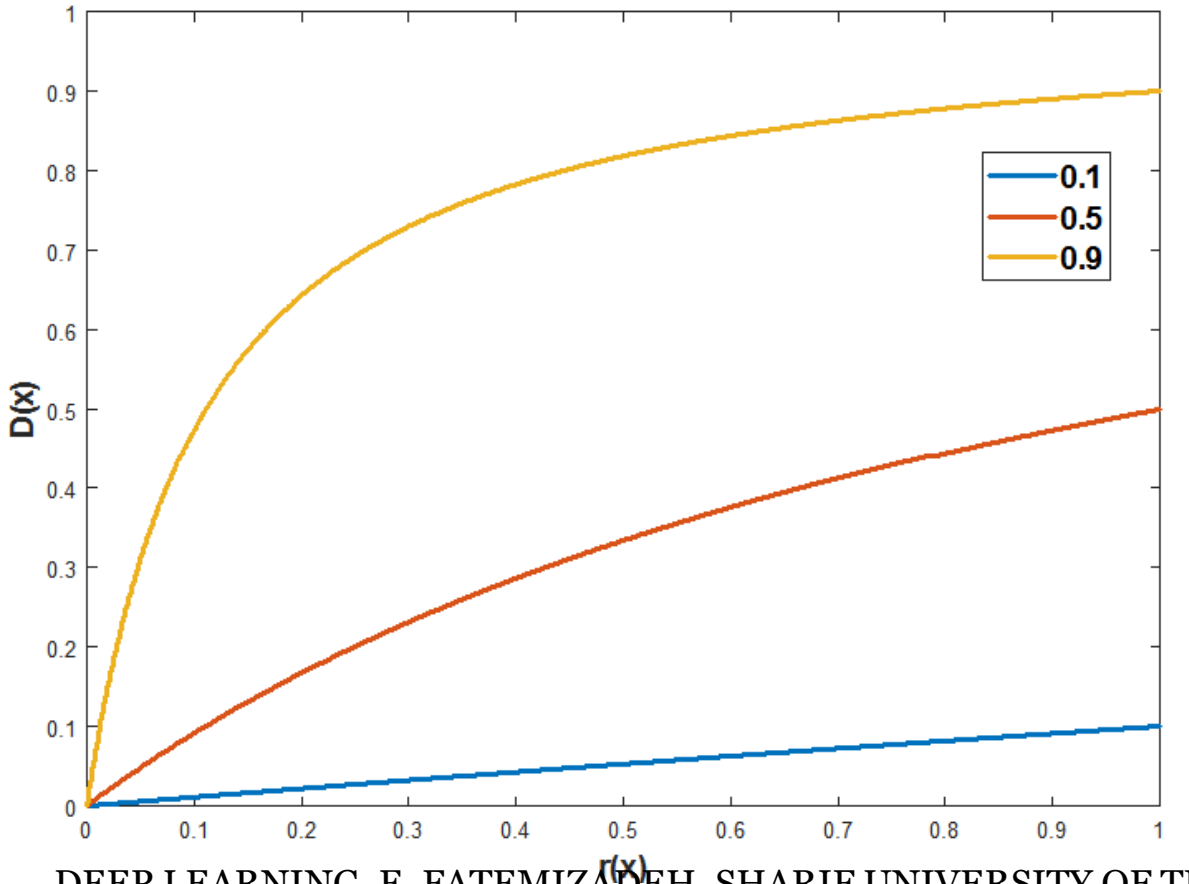
$$r(\mathbf{x}) = \frac{p(y=1|\mathbf{x})}{p(y=0|\mathbf{x})} \boxed{\frac{1-\pi}{\pi}}$$

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

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

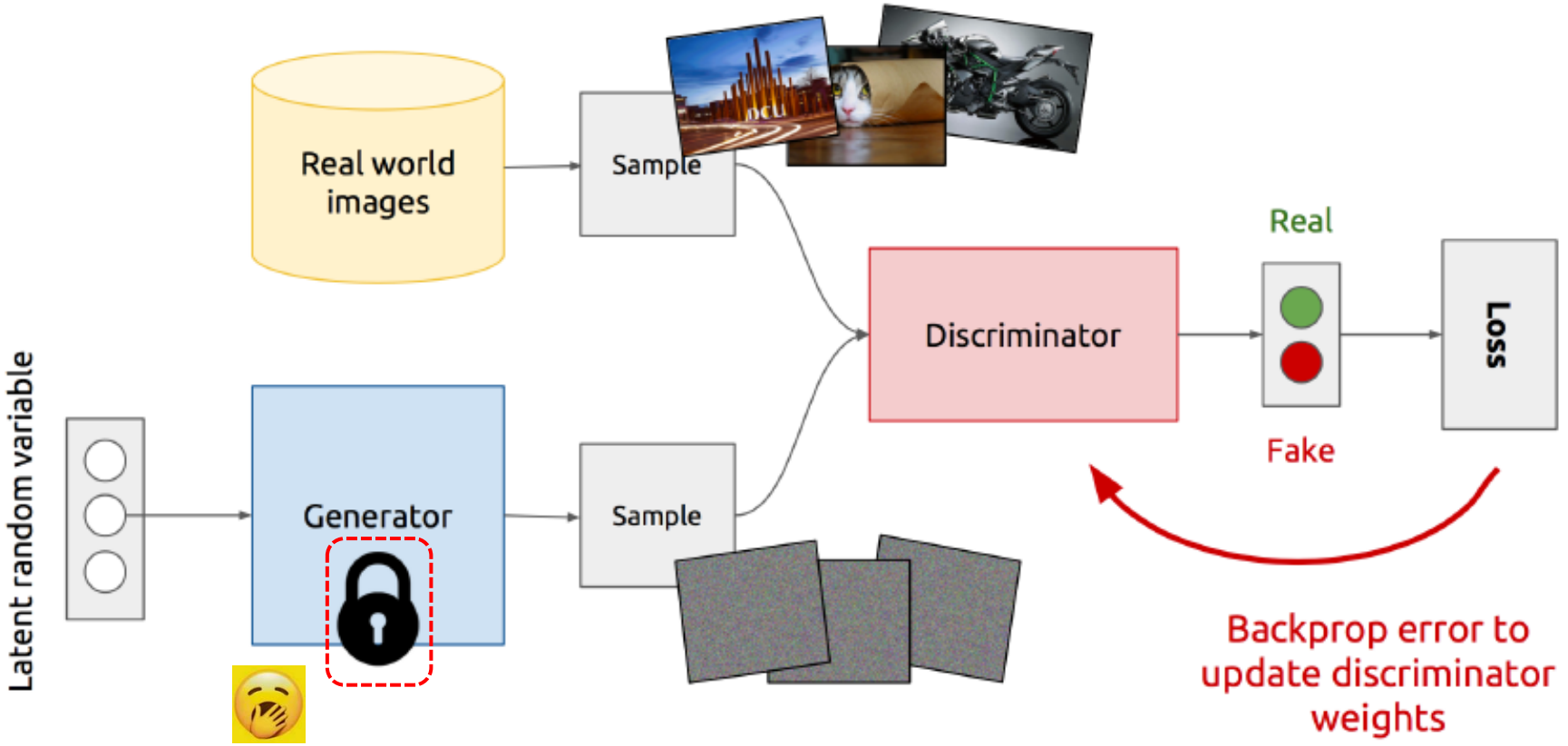
Generative Adversarial Network - GAN

› Optimal **Discriminator** and Ratio Test:



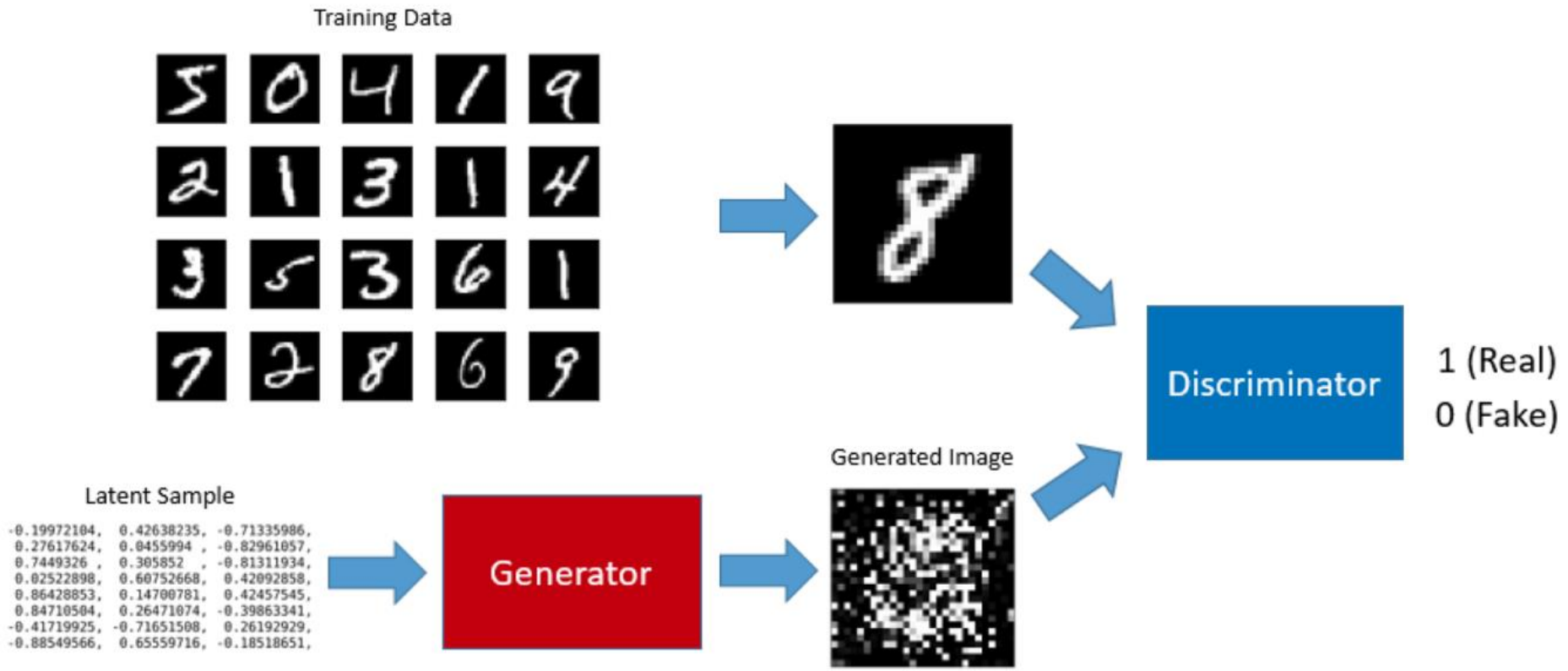
Generative Adversarial Network - GAN

› GAN, **Discriminator** Training:



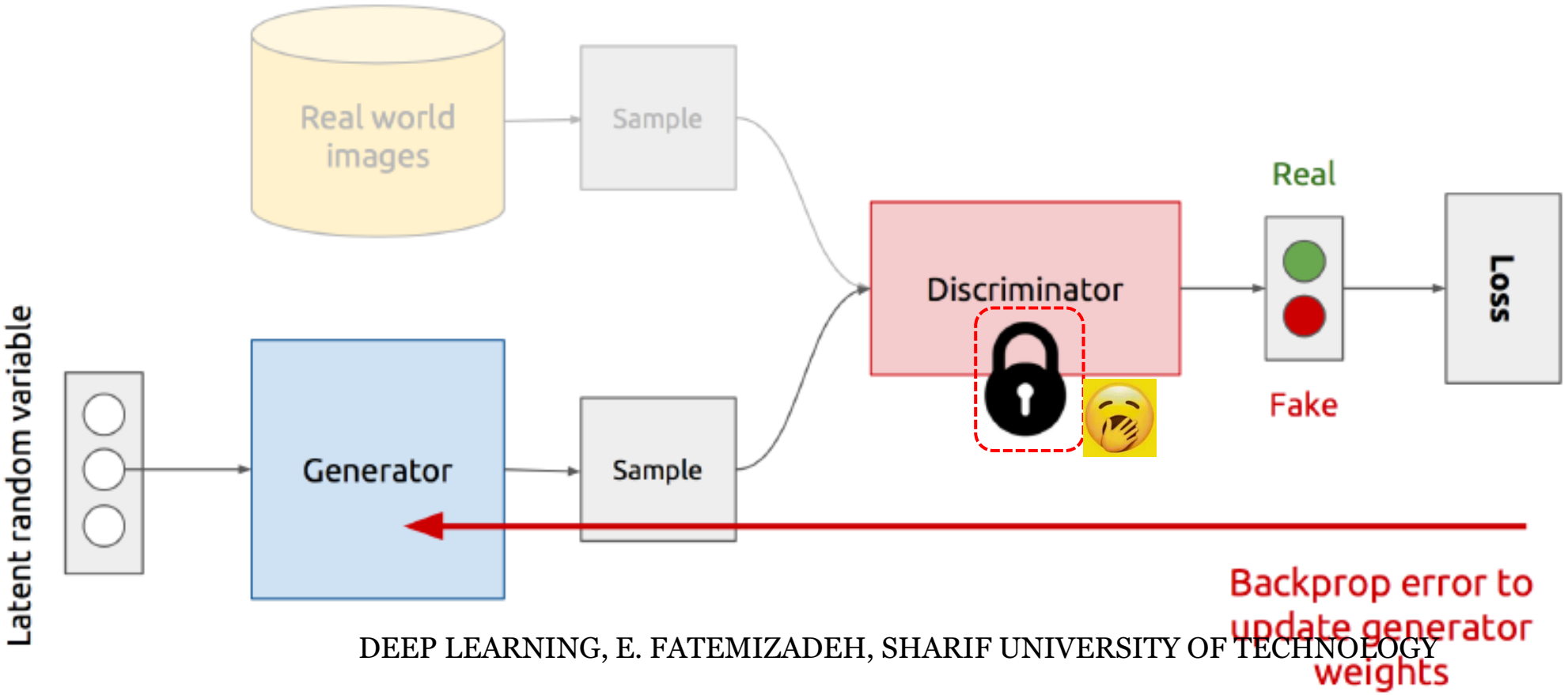
Generative Adversarial Network - GAN

› GAN, **Discriminator** Training:



Generative Adversarial Network - GAN

› GAN, **Generator** Training:

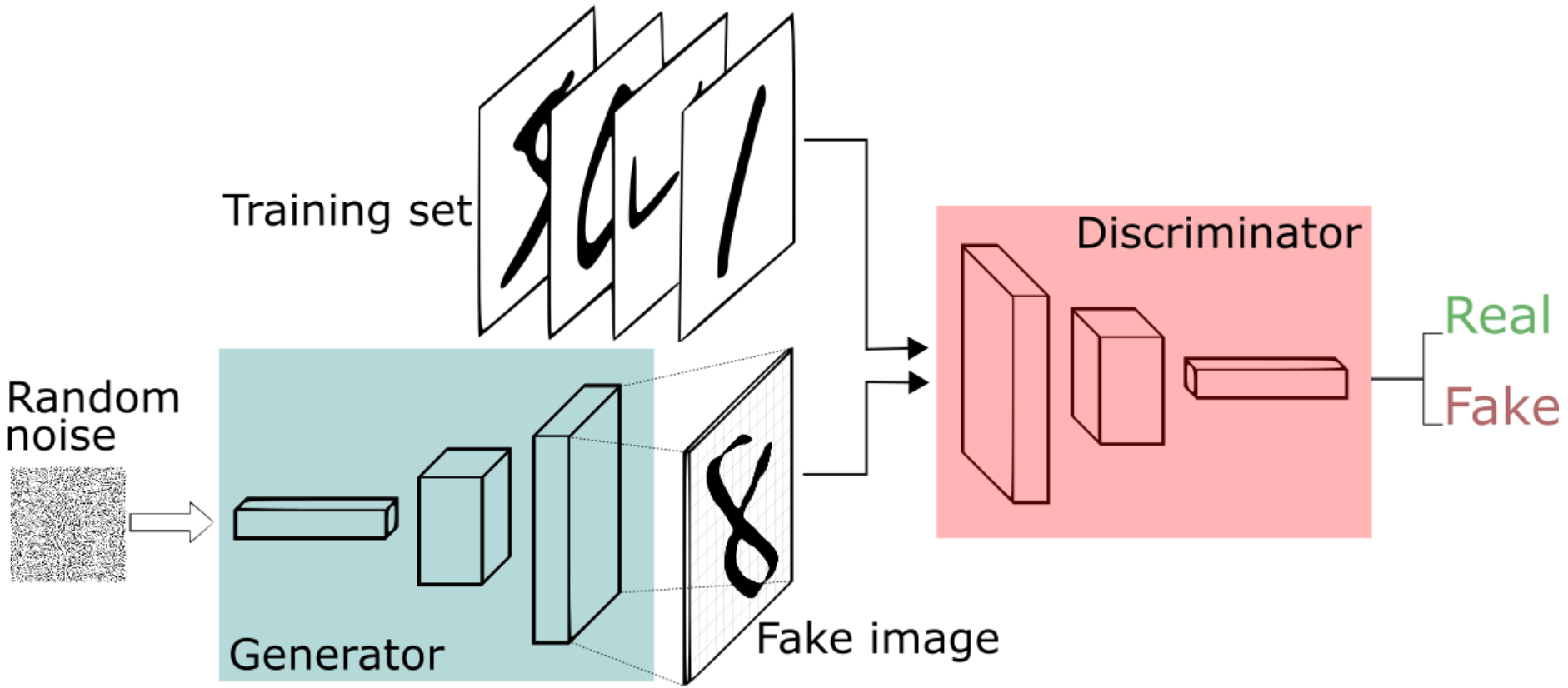


Generative Adversarial Network - GAN

› GAN, **Generator** Training:

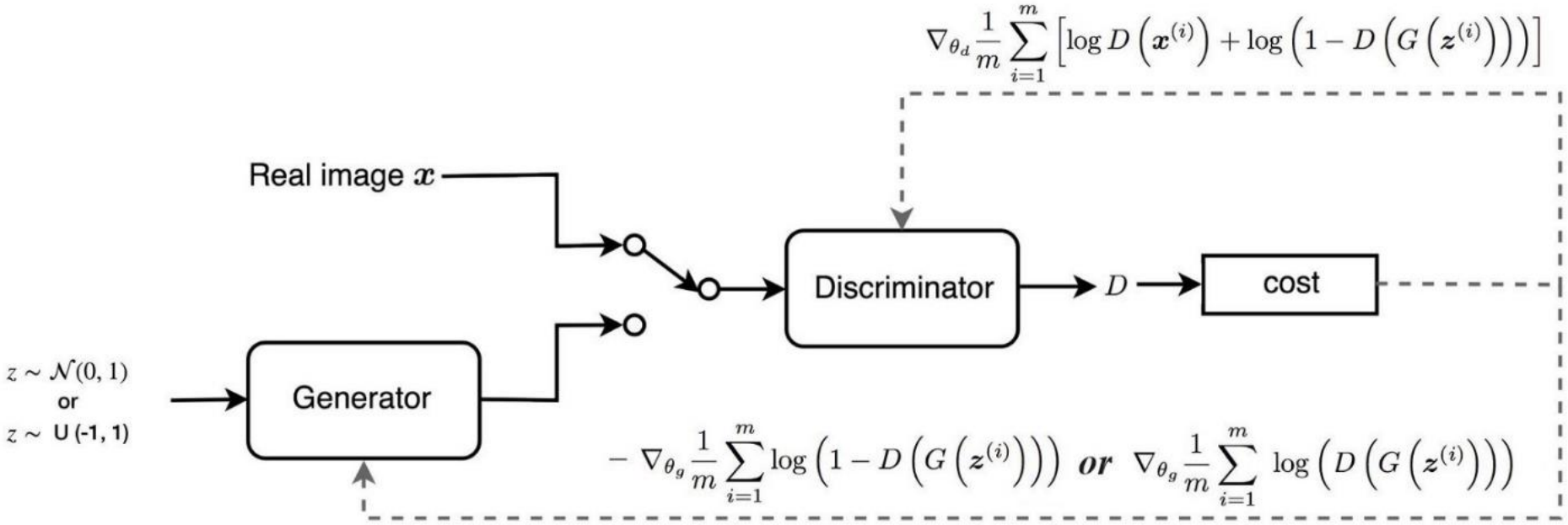
Generative Adversarial Network - GAN

› Structure of Discriminator/Generator in GAN



Generative Adversarial Network - GAN

› GAN Learning – in Brief



Generative Adversarial Network - GAN

› 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:

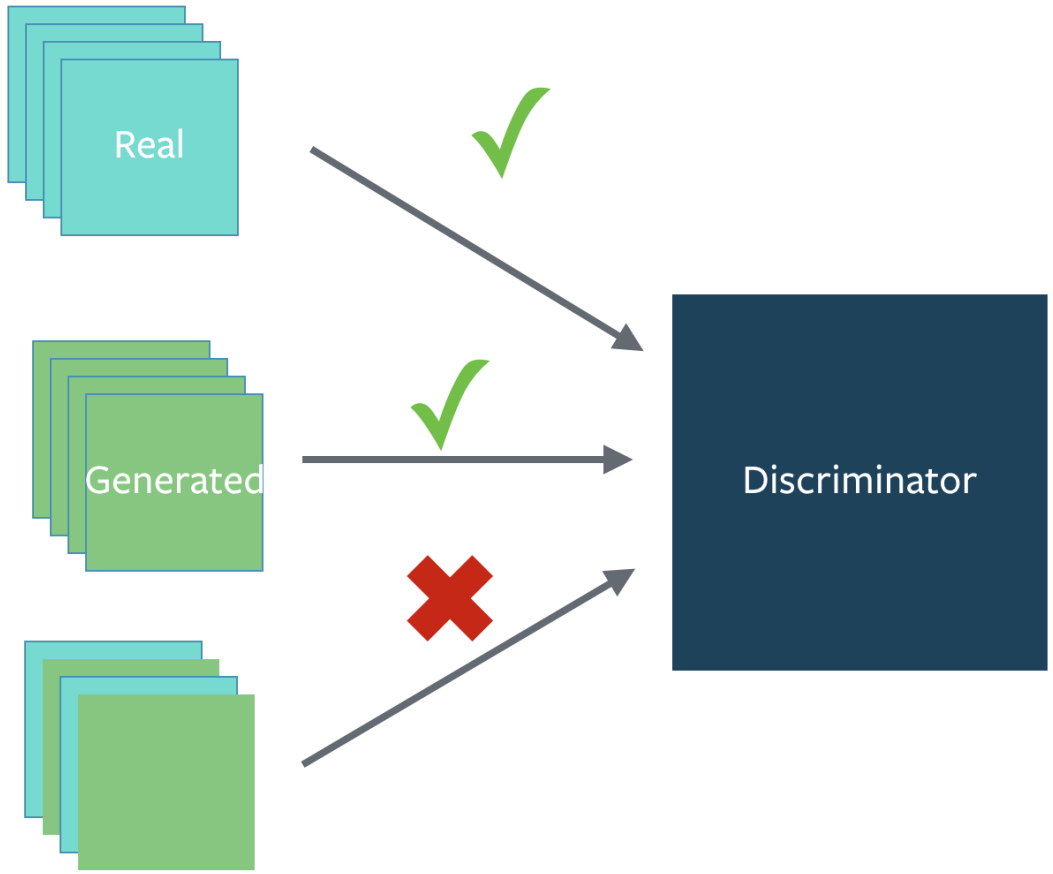
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

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

Generative Adversarial Network - GAN

- Two type of minibatch:
  - One – combined
  - One – fake, one-real



Generative Adversarial Network - GAN

- GAN Hardness!
  - Hard to achieve Nash equilibrium?
  - Vanishing gradient
  - Mode collapse

Generative Adversarial Network - GAN

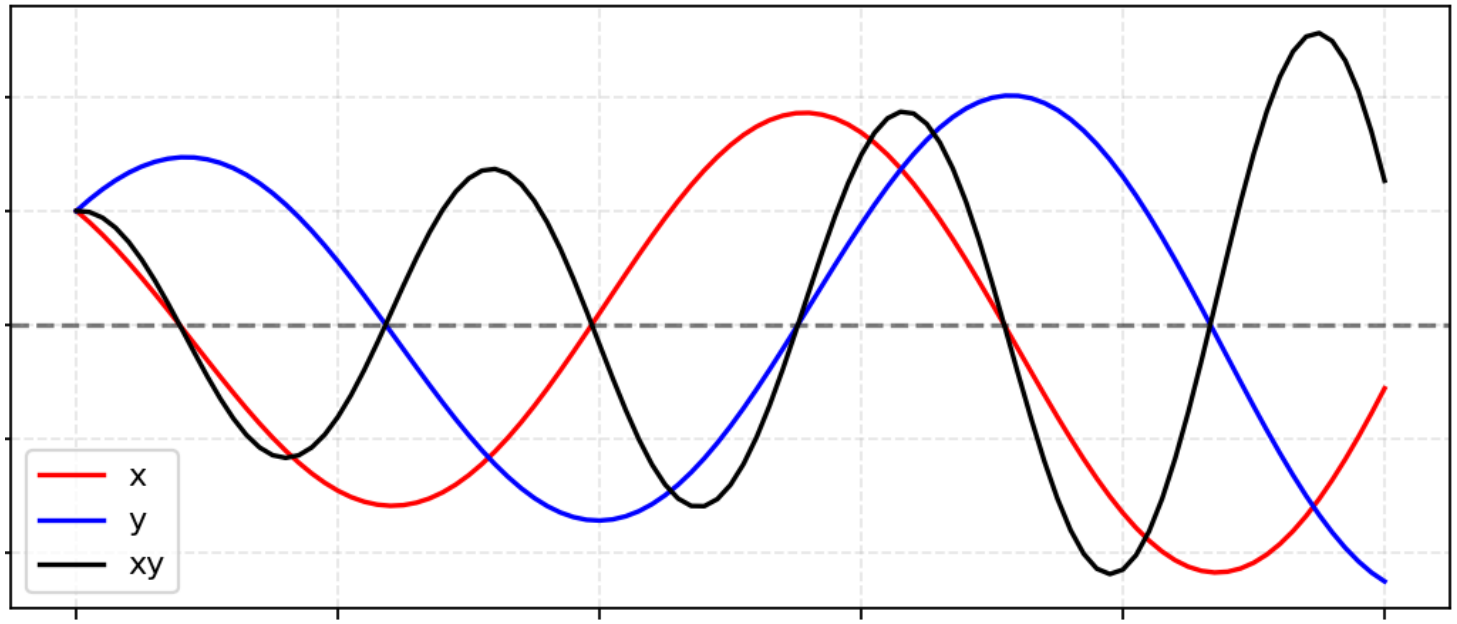
- Hard to achieve Nash equilibrium!
  - Example: one player (control on x) want to minimize $f(x)=xy$, while other player (control on y) want maximize $f(x)=xy$

$$\begin{cases} \frac{\partial f_1}{\partial x} = y \rightarrow x^{new} = x^{old} - \varepsilon y^{old} \\ \frac{\partial f_2}{\partial y} = -x \rightarrow y^{new} = y^{old} + \varepsilon x^{old} \end{cases}$$

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

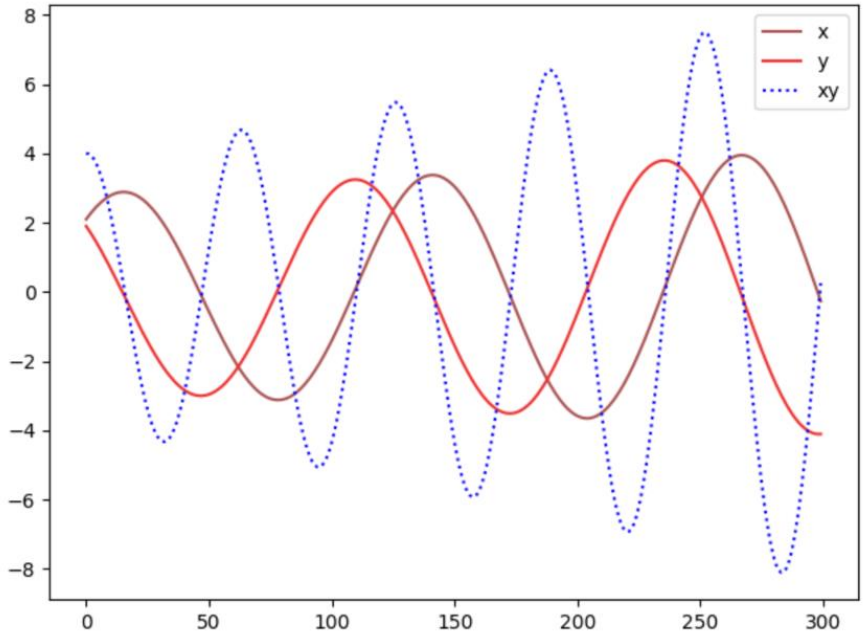
Generative Adversarial Network - GAN

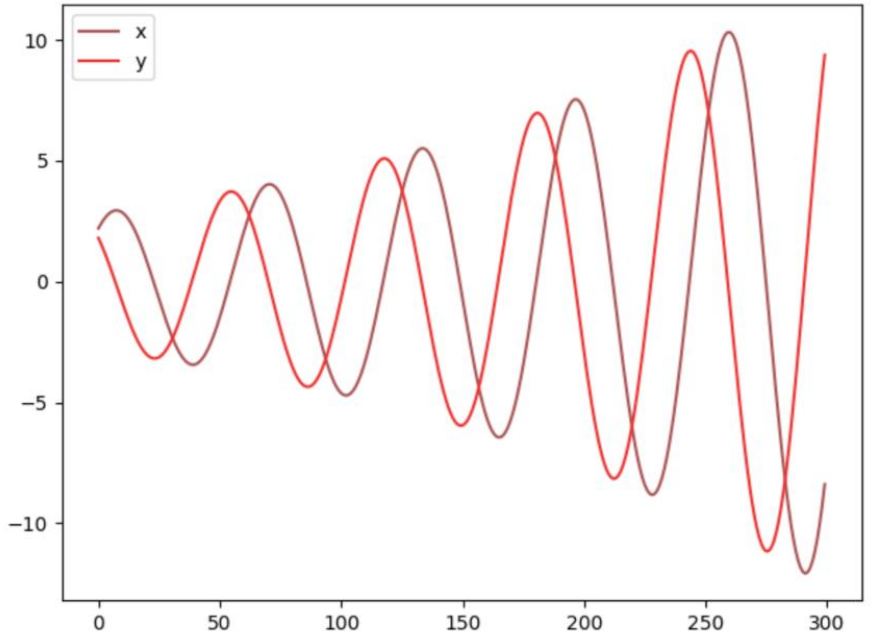
- Hard to achieve Nash equilibrium!
  - Once x and y have different signs → oscillation!
  - Cost functions may not converge using gradient descent in a minimax game.



Generative Adversarial Network - GAN

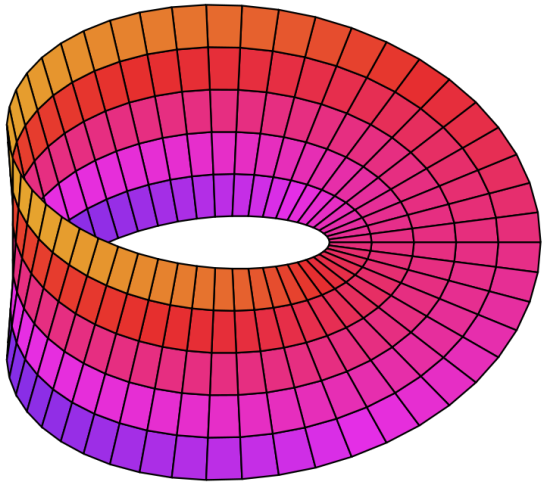
› Small (Left), Large (Right) Learning rate





Generative Adversarial Network - GAN

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



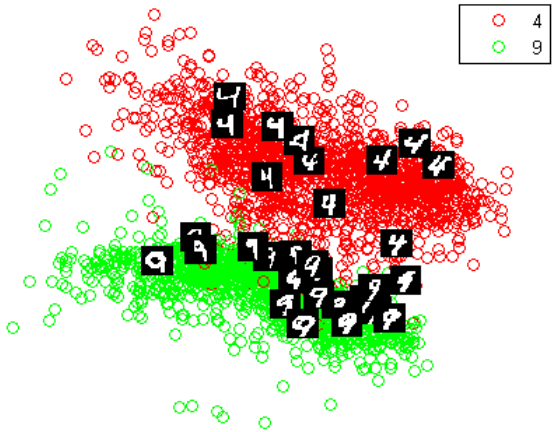
Generative Adversarial Network - GAN

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

$$\operatorname{supp}(f) = \{x \in X \mid f(x) \neq 0\}.$$

Generative Adversarial Network - GAN

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



Generative Adversarial Network - GAN

- Low Dimension Support:
- If (proof in paper*) supports of p_r and p_g lying on low dimensional manifolds then:
  - High chance of perfect (100%) discrimination (disjoint manifold)

Generative Adversarial Network - GAN

- Vanishing Gradient:
  - 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

Generative Adversarial Network - GAN

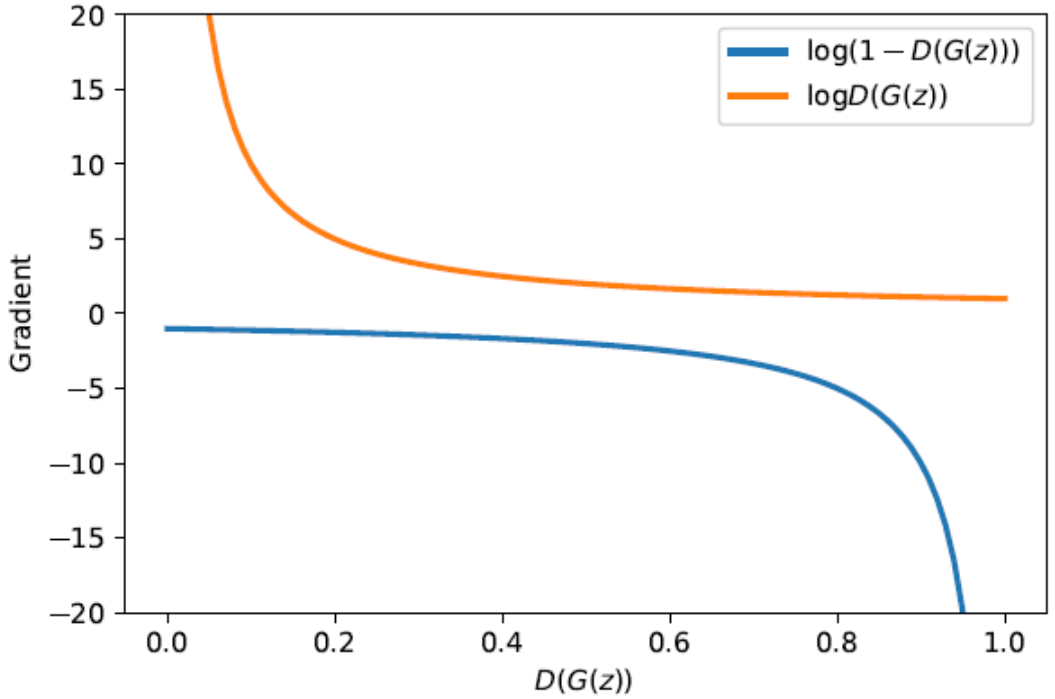
› Vanishing Gradient Solution:
  – High gradient for D~0

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

$$\mathbb{E}_{z\sim p(z)}\left[-\nabla_\theta \log D^*(g_\theta(z))|_{\theta=\theta_0}\right] = \nabla_\theta\left[KL(\mathbb{P}_{g_\theta}\|\mathbb{P}_r) - 2JSD(\mathbb{P}_{g_\theta}\|\mathbb{P}_r)\right]|_{\theta=\theta_0}$$

Generative Adversarial Network - GAN

› Vanishing Gradient Solution:
  – High gradient for D~0



Generative Adversarial Network - GAN

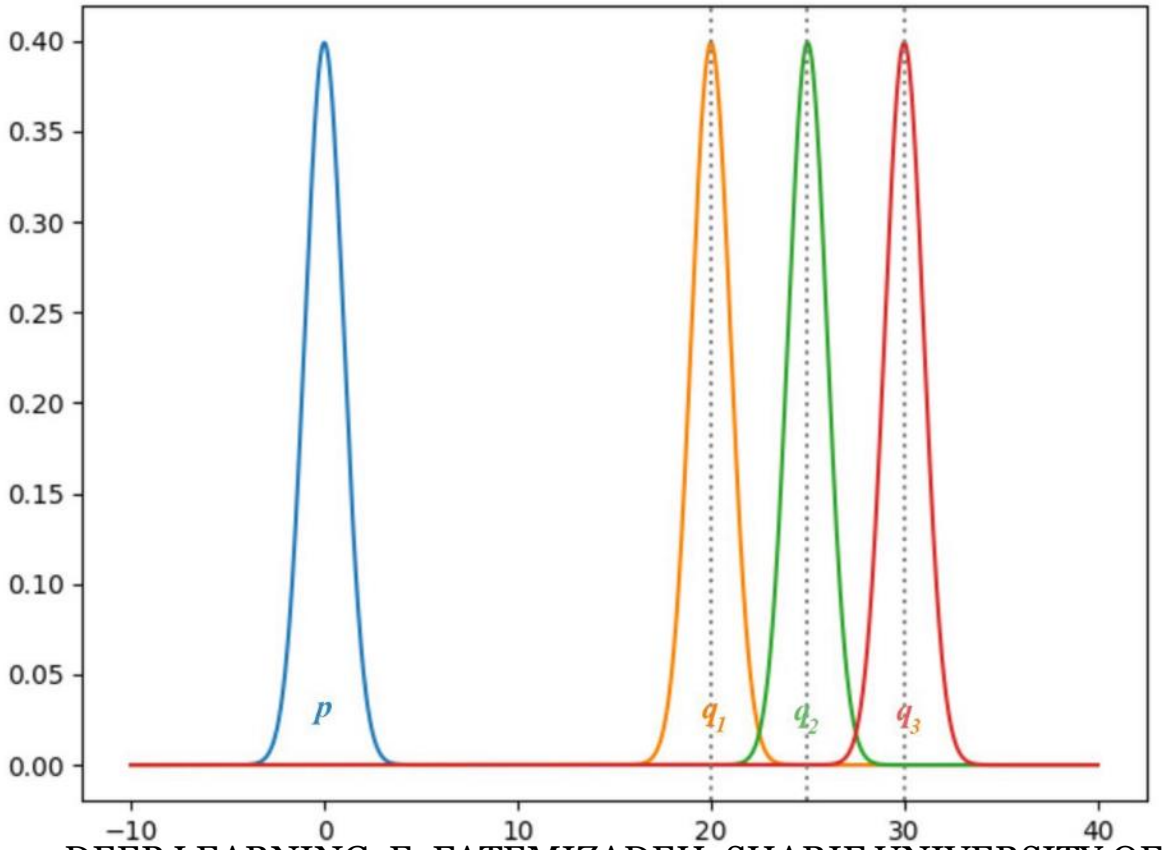
› Vanishing Gradient Solution – Another Perspective

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

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

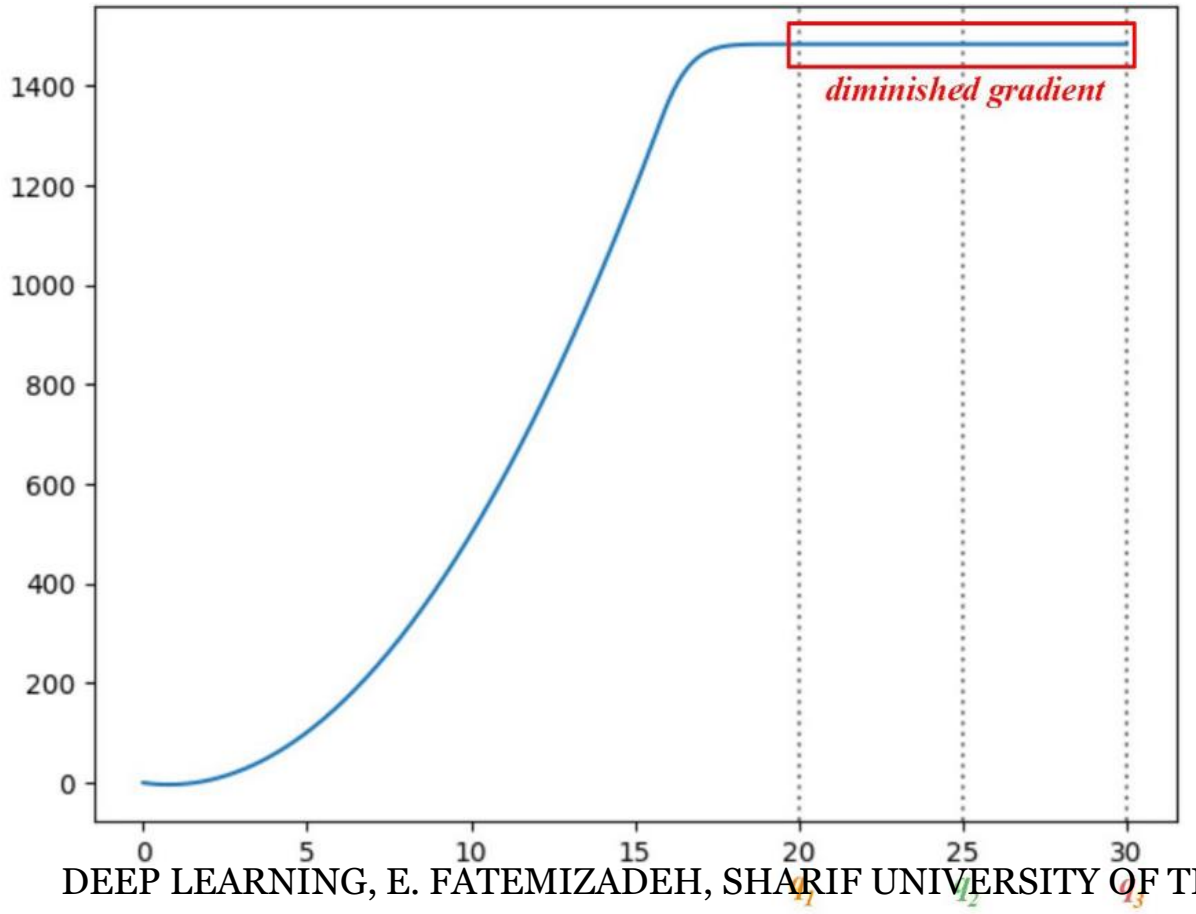
Generative Adversarial Network - GAN

› Let plot the JS-divergence JS(p, q), p ~ N(0,1), q ~ N(0…30,1)

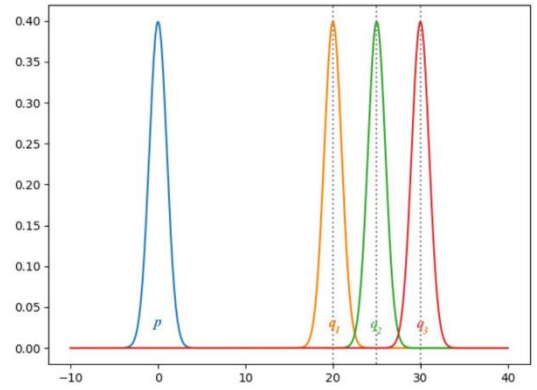


Generative Adversarial Network - GAN

› Let plot the JS-divergence JS(p, q), p ~ N(0,1), q ~ N(0…30,1)



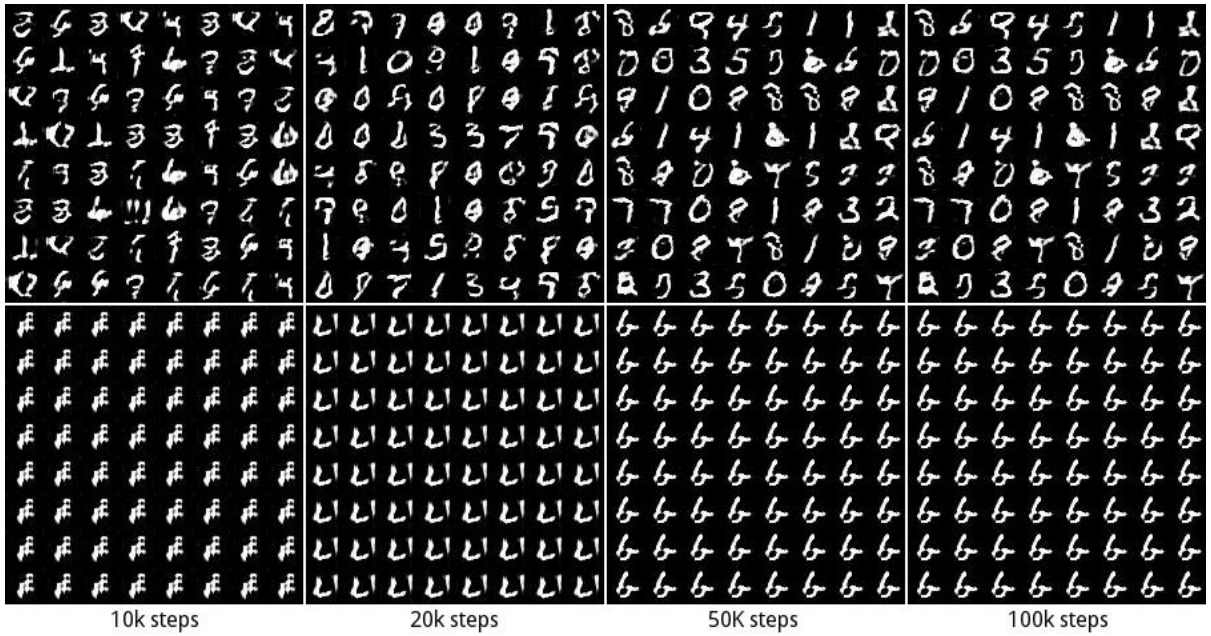
JS Distance

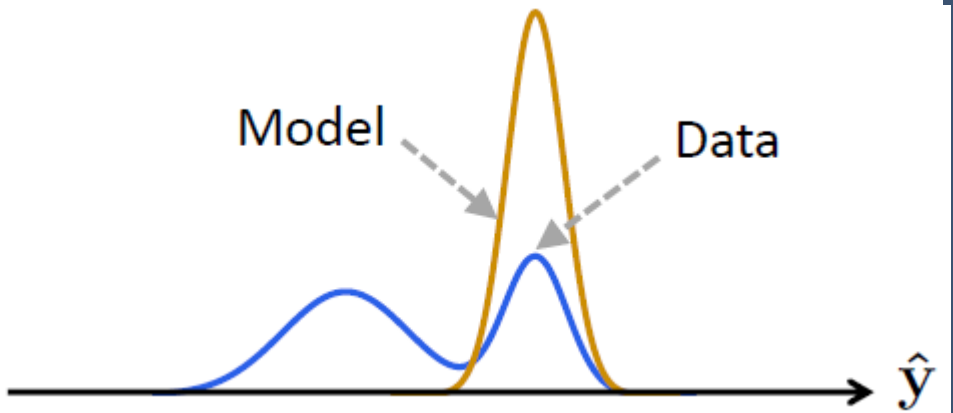


Generative Adversarial Network - GAN

› Mode Collapse:

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





Generative Adversarial Network - GAN

- 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 x* that fool D

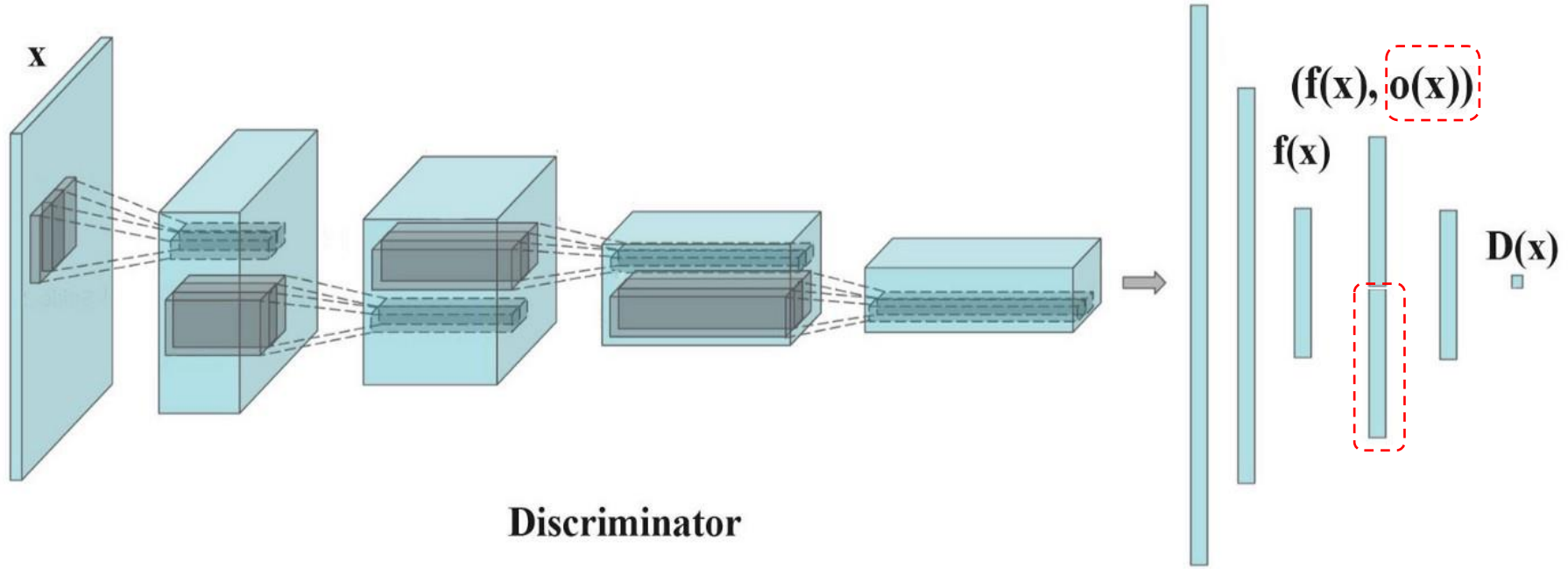
Generative Adversarial Network - GAN

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

Generative Adversarial Network - GAN

› Minibatch Discrimination to avoid Mode Collapse:

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



Generative Adversarial Network - GAN

- 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)!

Generative Adversarial Network - GAN

› Minibatch Discrimination:

– $f(x)$: vector of features for input x_i, output of a intermediate layer in the discriminator, $i, j \in$ *"current MiniBatch"*

$$\mathbf{f}(x_i) \in \Re^A$$

$$M_i = \mathbf{f}(x_i) \times T, \quad T \in \Re^{A \times B \times C} \Rightarrow M_i \in \Re^{B \times C}$$

$$c_b(x_i, x_j) = \exp\left(-\left|M_{i,b} - M_{i,b}\right|_{L_1}\right) = \exp\left(-\left|\mathbf{f}(x_i) \times T - \mathbf{f}(x_j) \times T\right|_{L_1}\right) \in \Re$$

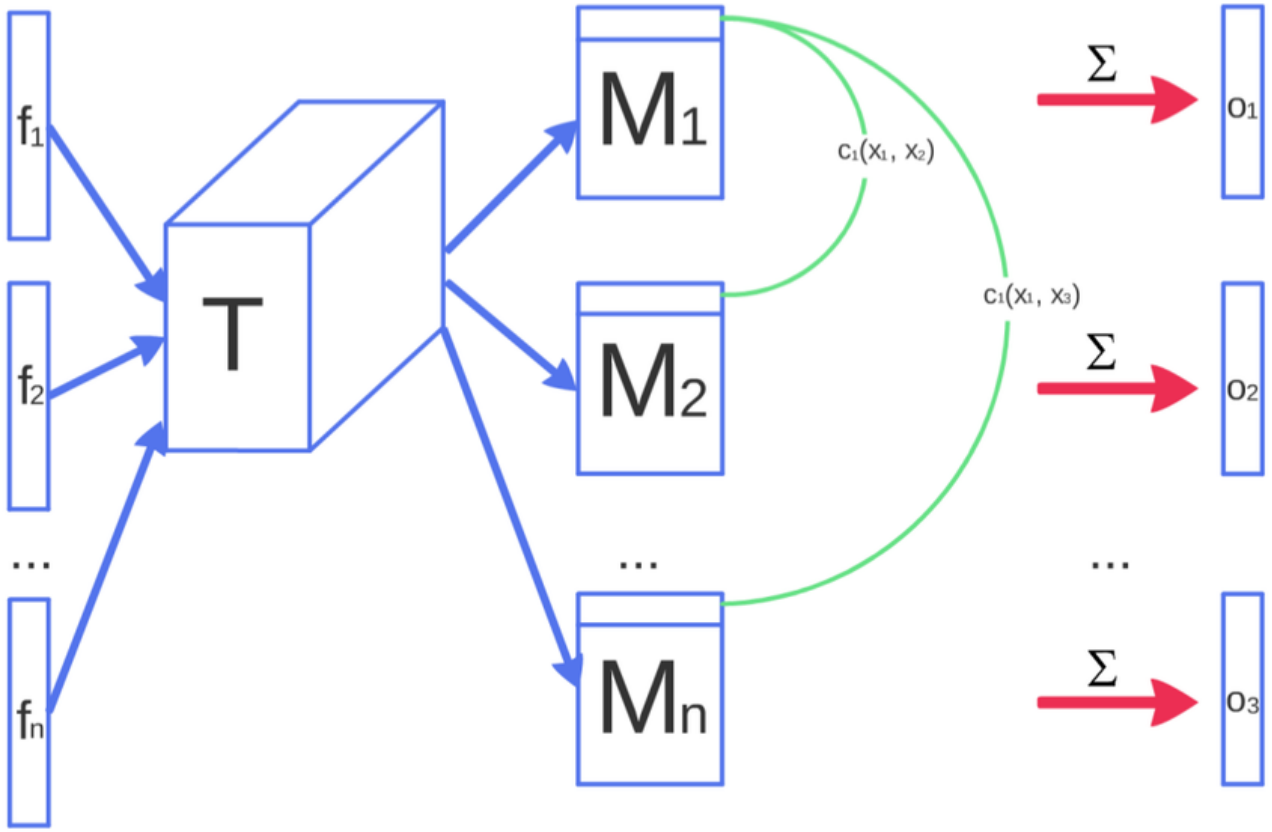
$$o(x_i)_b = \sum_{j=1}^{n} c_b(x_i, x_j) \in \Re$$

$$o(x_i) = \left[o(x_i)_1, o(x_i)_2, \ldots, o(x_i)_B\right] \in \Re^B$$

$$\text{minibatch: } o(X) \in \Re^{n \times B}$$

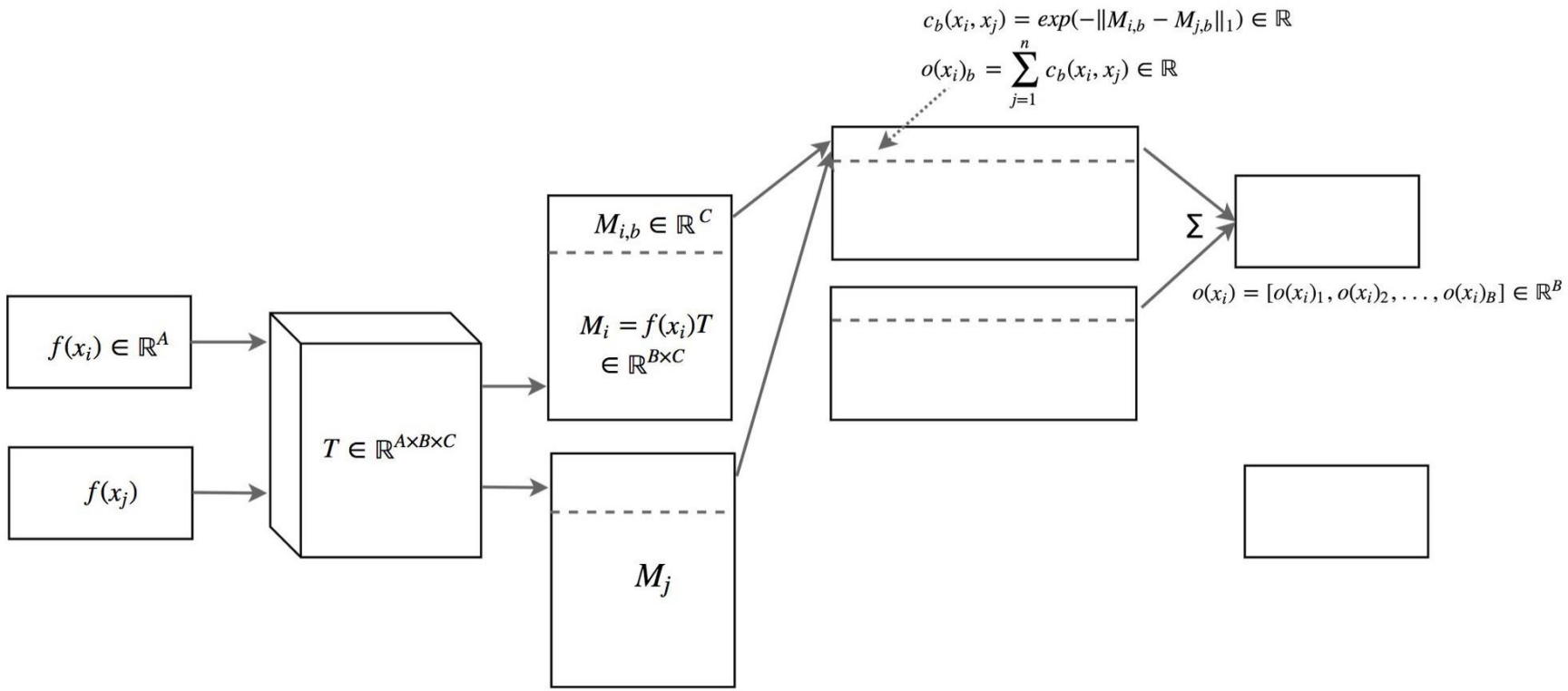
Generative Adversarial Network - GAN

› Minibatch Discrimination:



Generative Adversarial Network - GAN

› Minibatch Discrimination:



Generative Adversarial Network - GAN

- 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.

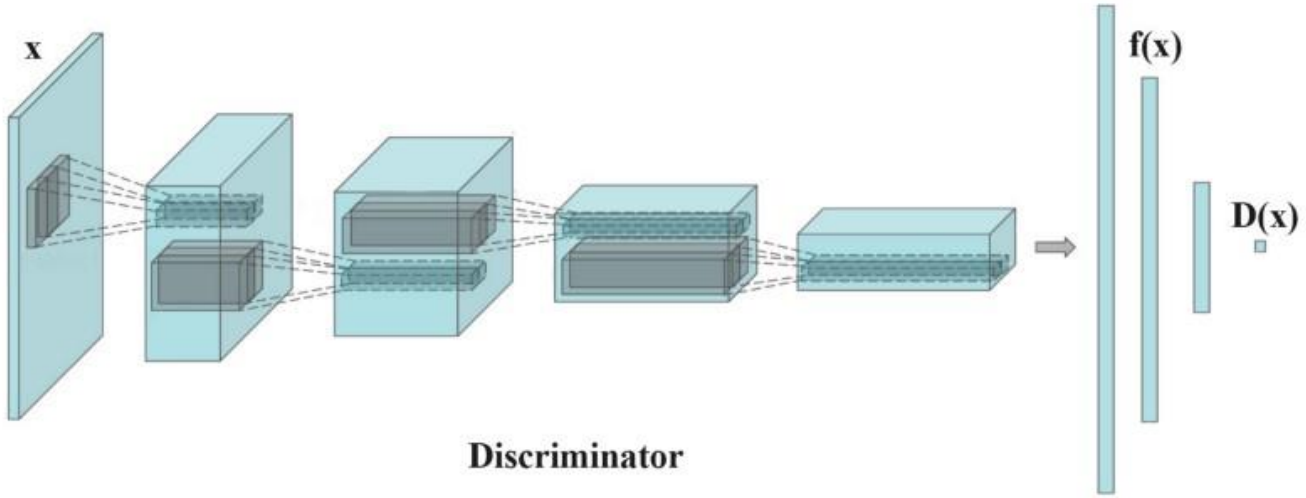
$$||\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}\mathbf{f}(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}\mathbf{f}(G(\boldsymbol{z}))||_2^2$$

  - $f(x)$: activation of an intermediate layer of the discriminator or any meaningful features (median/mean/….)
  - E_{data} is performed by sampling from real data in minibatch or whole data!
  - E_z is performed by sampling from noise in minibathch

Generative Adversarial Network - GAN

› Feature Matching (Improve Convergence/No Collapse):

› Is effective when the GAN model is unstable during training.



$$\|\mathbb{E}_{x\sim p_r} f(x) - \mathbb{E}_{z\sim p_z(z)} f(G(z))\|_2^2$$

Tips and Tricks to Train GAN! - Improved Techniques for Training GANs,

› Historical Averaging/Smoothing: Penalize (add to) G and D cost to avoid fast changing (θ: net parameters)

$$||\boldsymbol{\theta} - \frac{1}{t}\sum_{i=1}^{t}\boldsymbol{\theta}[i]||^2,$$

Tips and Tricks to Train GAN! - Improved Techniques for Training GANs,

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

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

- For small p_{data} and large p_{model} →☹
  - Only smooth 1 to α
  - Negative label set at 0!

Tips and Tricks to Train GAN! - Improved Techniques for Training GANs,

- 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!), Sampling Generative Networks,2017
- Avoide Sparse Gradients: ReLU, MaxPool (Use Leaky ReLu, Avg. Pool)
- Use the ADAM Optimizer

GAN Implementation

› Two Layer net for Discriminator (784-128-128-1)

```
# Discriminator Net
X = tf.placeholder(tf.float32, shape=[None, 784], name='X')

D_W1 = tf.Variable(xavier_init([784, 128]), name='D_W1')
D_b1 = tf.Variable(tf.zeros(shape=[128]), name='D_b1')

D_W2 = tf.Variable(xavier_init([128, 1]), name='D_W2')
D_b2 = tf.Variable(tf.zeros(shape=[1]), name='D_b2')

theta_D = [D_W1, D_W2, D_b1, D_b2]
```

GAN Implementation

› Two Layer net for Generator (100-128-128-784)

```
# Generator Net
Z = tf.placeholder(tf.float32, shape=[None, 100], name='Z')

G_W1 = tf.Variable(xavier_init([100, 128]), name='G_W1')
G_b1 = tf.Variable(tf.zeros(shape=[128]), name='G_b1')

G_W2 = tf.Variable(xavier_init([128, 784]), name='G_W2')
G_b2 = tf.Variable(tf.zeros(shape=[784]), name='G_b2')

theta_G = [G_W1, G_W2, G_b1, G_b2]
```

GAN Implementation

› Generator/Discriminator function/net:

```
def generator(z):
    G_h1 = tf.nn.relu(tf.matmul(z, G_W1) + G_b1)
    G_log_prob = tf.matmul(G_h1, G_W2) + G_b2
    G_prob = tf.nn.sigmoid(G_log_prob)

    return G_prob


def discriminator(x):
    D_h1 = tf.nn.relu(tf.matmul(x, D_W1) + D_b1)
    D_logit = tf.matmul(D_h1, D_W2) + D_b2
    D_prob = tf.nn.sigmoid(D_logit)

    return D_prob, D_logit
```

GAN Implementation

› Run

```
G_sample = generator(Z)
D_real, D_logit_real = discriminator(X)
D_fake, D_logit_fake = discriminator(G_sample)

D_loss = -tf.reduce_mean(tf.log(D_real) + tf.log(1. - D_fake))
G_loss = -tf.reduce_mean(tf.log(D_fake))
```

GAN Implementation

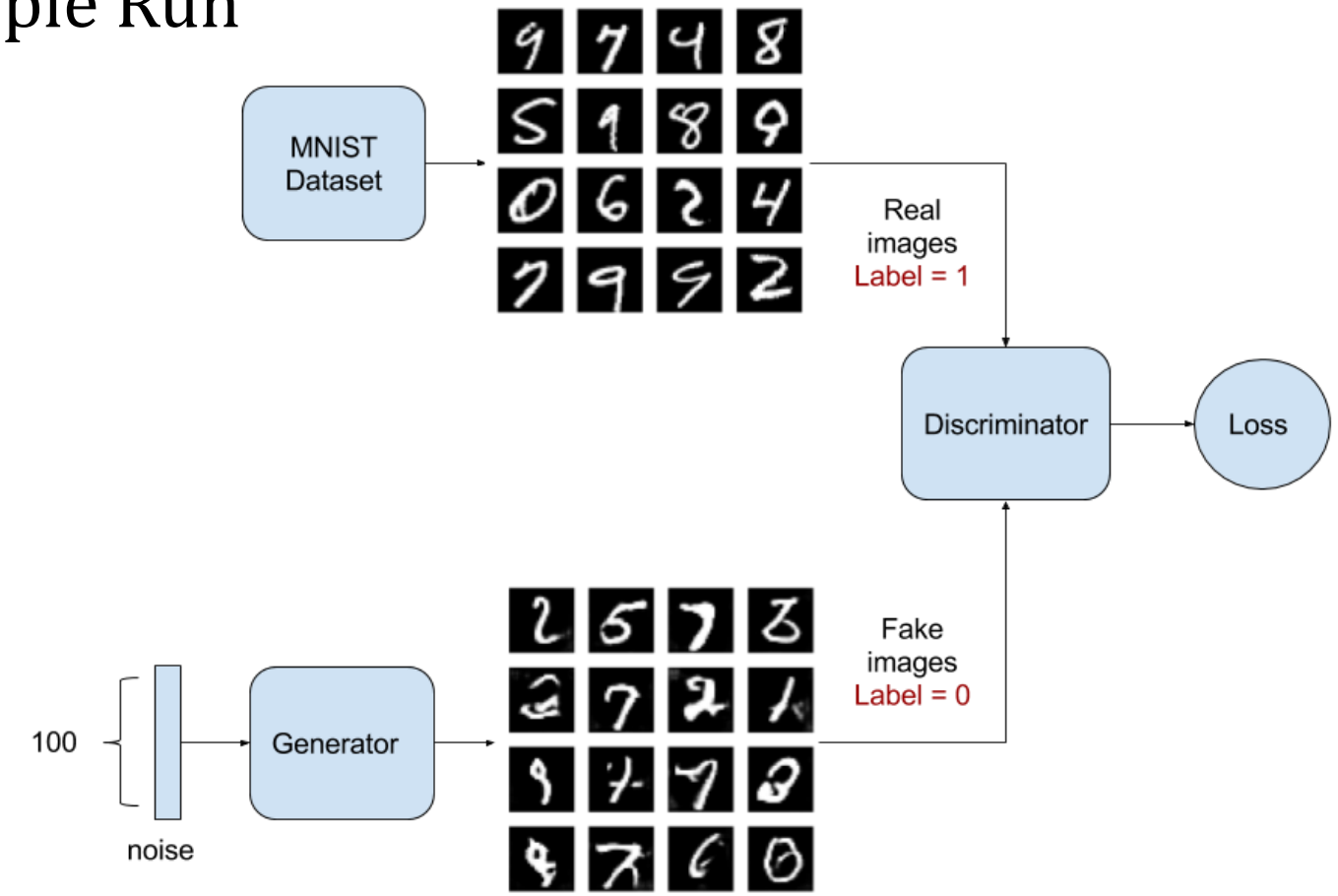
› Run

```
# Only update D(X)'s parameters, so var_list = theta_D
D_solver = tf.train.AdamOptimizer().minimize(D_loss, var_list=theta_D)
# Only update G(X)'s parameters, so var_list = theta_G
G_solver = tf.train.AdamOptimizer().minimize(G_loss, var_list=theta_G)

def sample_Z(m, n):
    '''Uniform prior for G(Z)'''
    return np.random.uniform(-1., 1., size=[m, n])
```

GAN Implementation

› A Sample Run

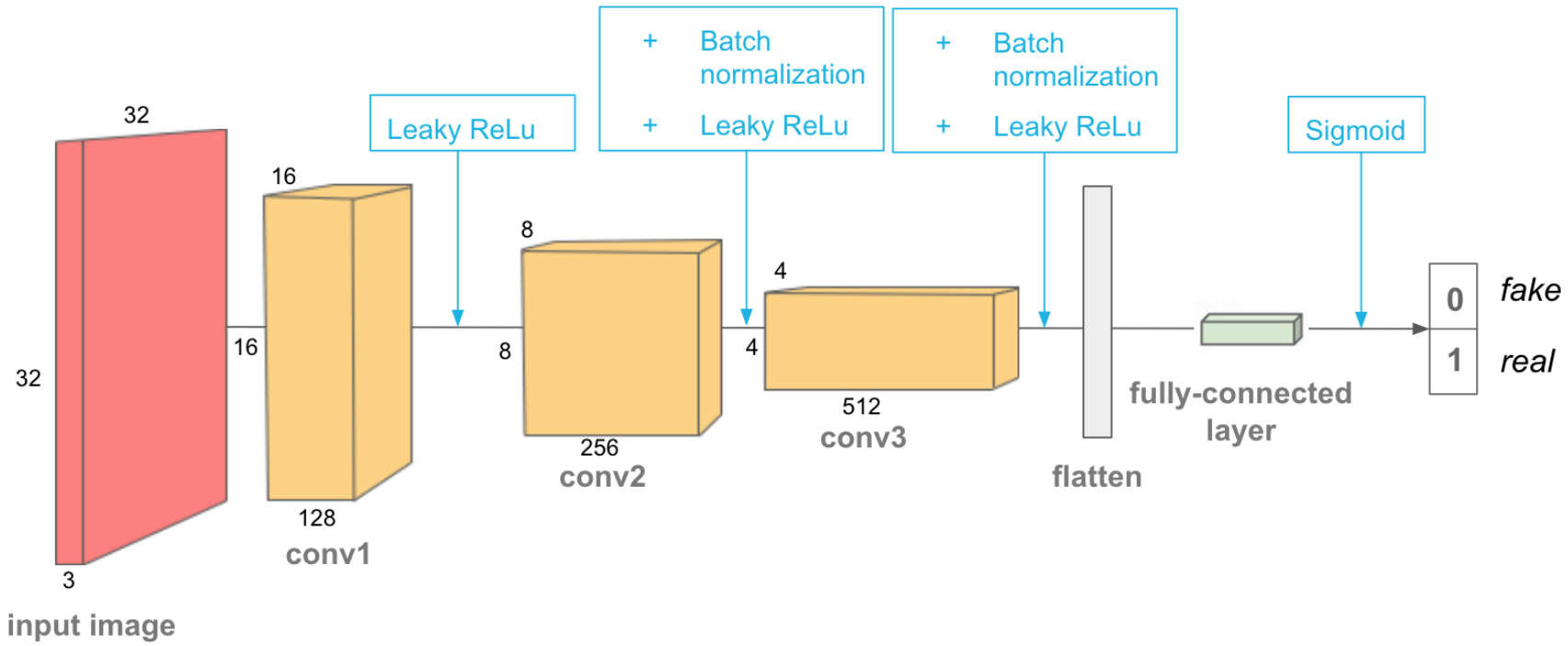


DCGAN (Deep Convolutional GANs) - Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks, 2016

- DCGAN: A popular and successful network design for GAN
- Replace any pooling layers with stride convolutions (discriminator) and fractional-stride convolutions (generator).
- Use batchnorm 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.

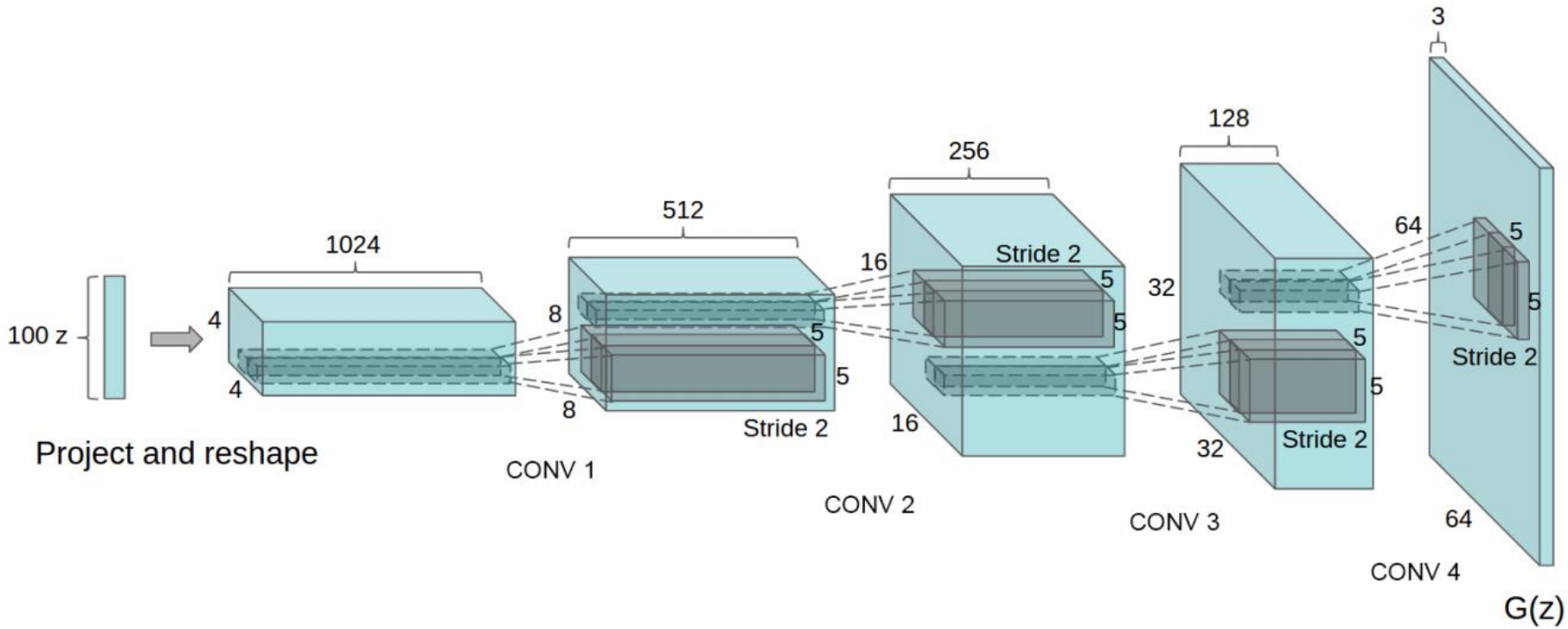
DCGAN (Deep Convolutional GANs) - Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks, 2016

› Discriminator:



DCGAN (Deep Convolutional GANs) - Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks, 2016

› Generator:



DCGAN (Deep Convolutional GANs) - Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks, 2016

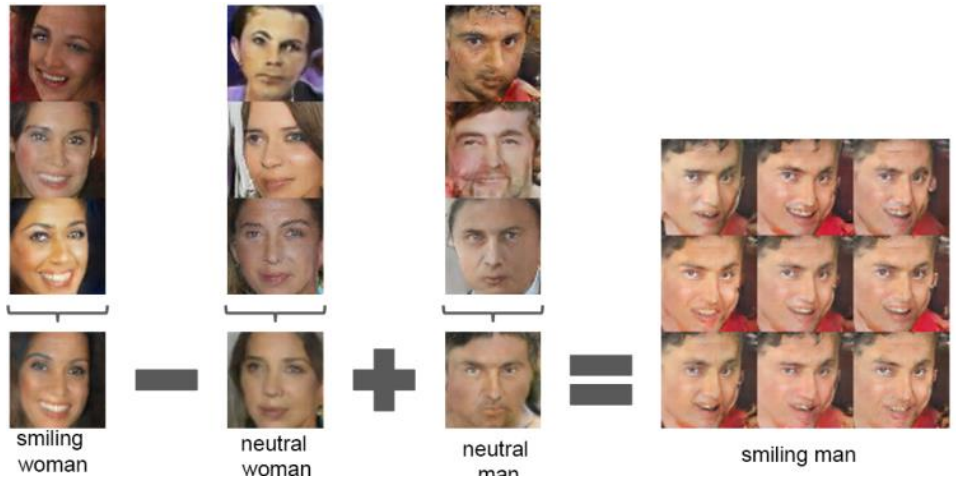
› Fake bedroom: (feed rand vector to generator)

DCGAN (Deep Convolutional GANs) - Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks, 2016

- Vector Arithmetic:
- Z: random vector

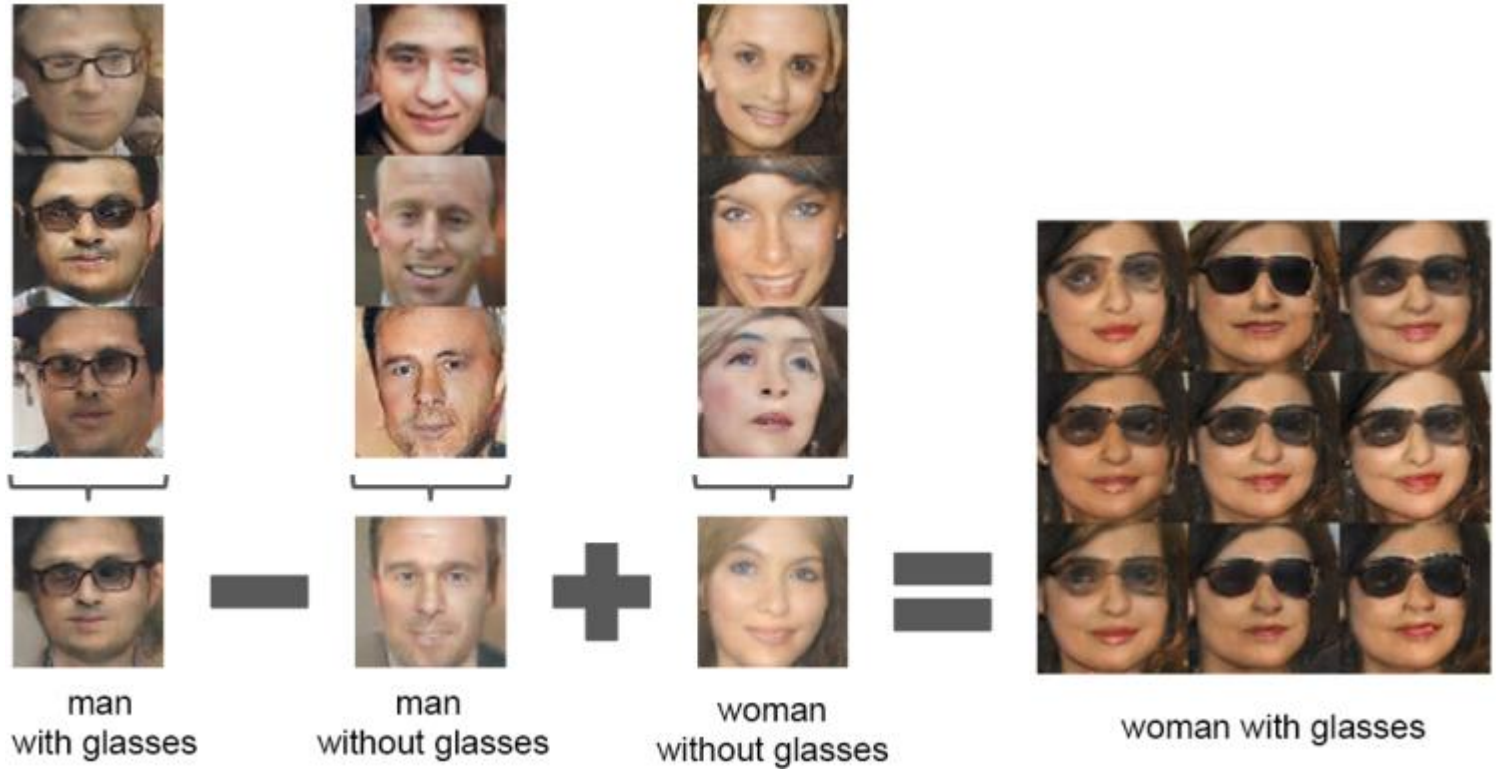
$Y = Z_{avrage}(\text{smiling woman}) - Z_{average}(\text{neutral woman}) + Z_{average}(\text{neutral man});$

- Feed Y to Generator → Smiling man !!!



DCGAN (Deep Convolutional GANs) - Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks, 2016

› Vector Arithmetic:



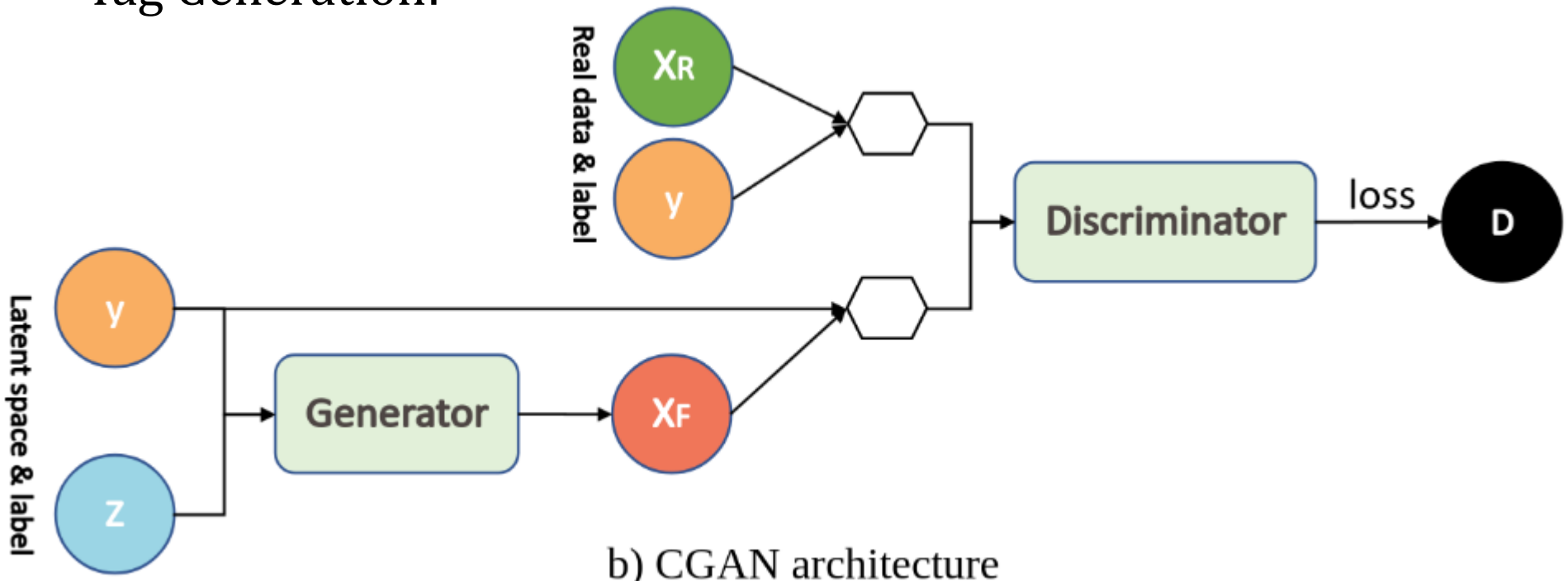
Conditional GAN – CGAN, M. Mehdi and S. Osindero, "Conditional generative adversarial nets.", 2014

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

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

Conditional GAN – CGAN, M. Mehdi and S. Osindero, "Conditional generative adversarial nets.", 2014

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



b) CGAN architecture

Conditional GAN – CGAN, M. Mehdi and S. Osindero, "Conditional generative adversarial nets."

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

Conditional GAN – CGAN, M. Mehdi and S. Osindero, "Conditional generative adversarial nets."

- 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}

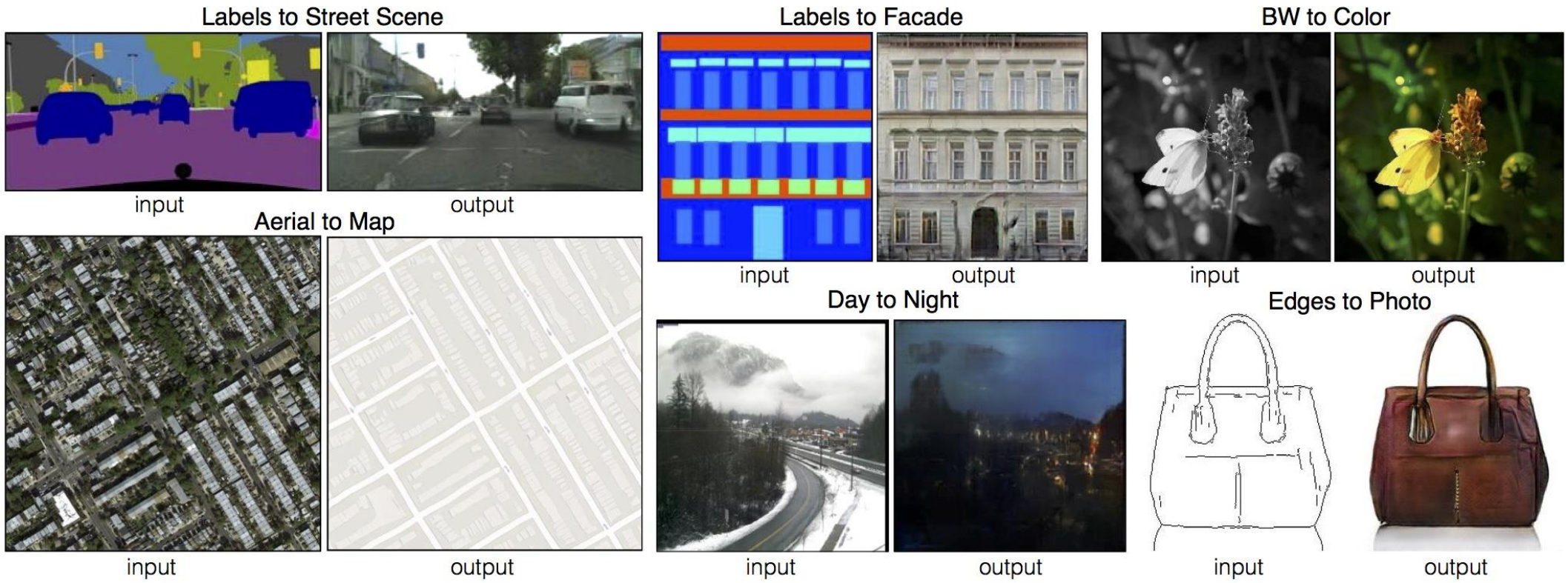
Conditional GAN – CGAN, M. Mehdi and S. Osindero, "Conditional generative adversarial nets."

› 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

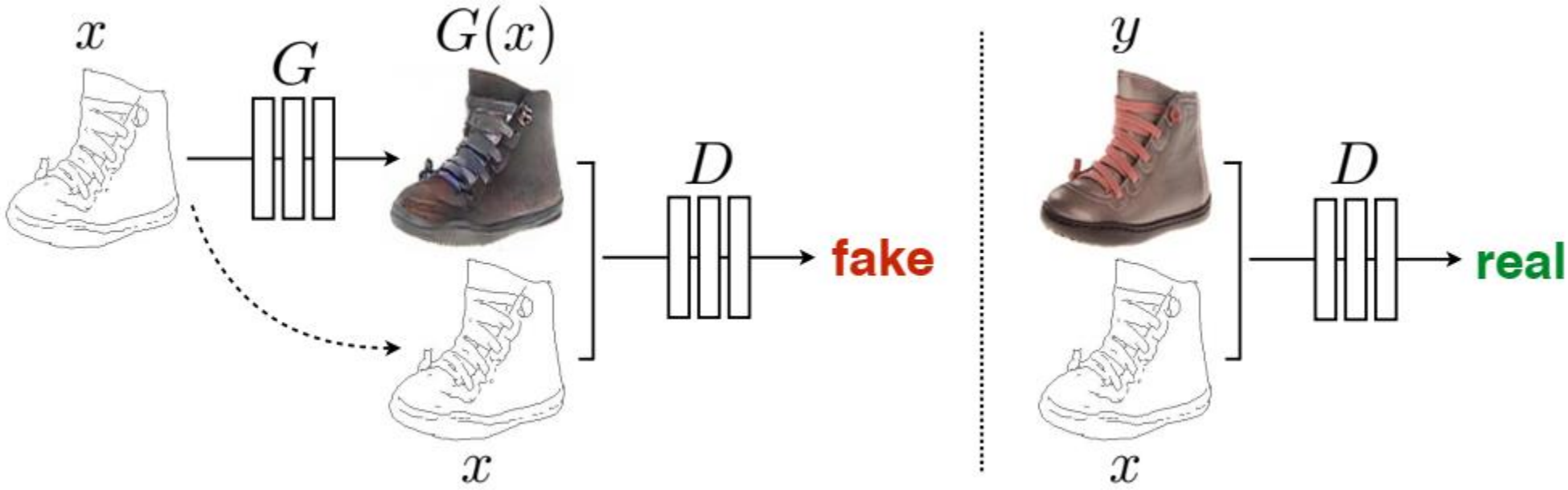
CGAN– Application pix2pix - Image-to-Image Translation with Conditional Adversarial Networks, CVPR 2017

› Image-to-Image Translation (pix2pix):



CGAN– Application pix2pix - Image-to-Image Translation with Conditional Adversarial Networks, CVPR 2017

› 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 - Image-to-Image Translation with Conditional Adversarial Networks, CVPR 2017

› Pix2Pix Loss Function (x is conditioning):

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

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

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

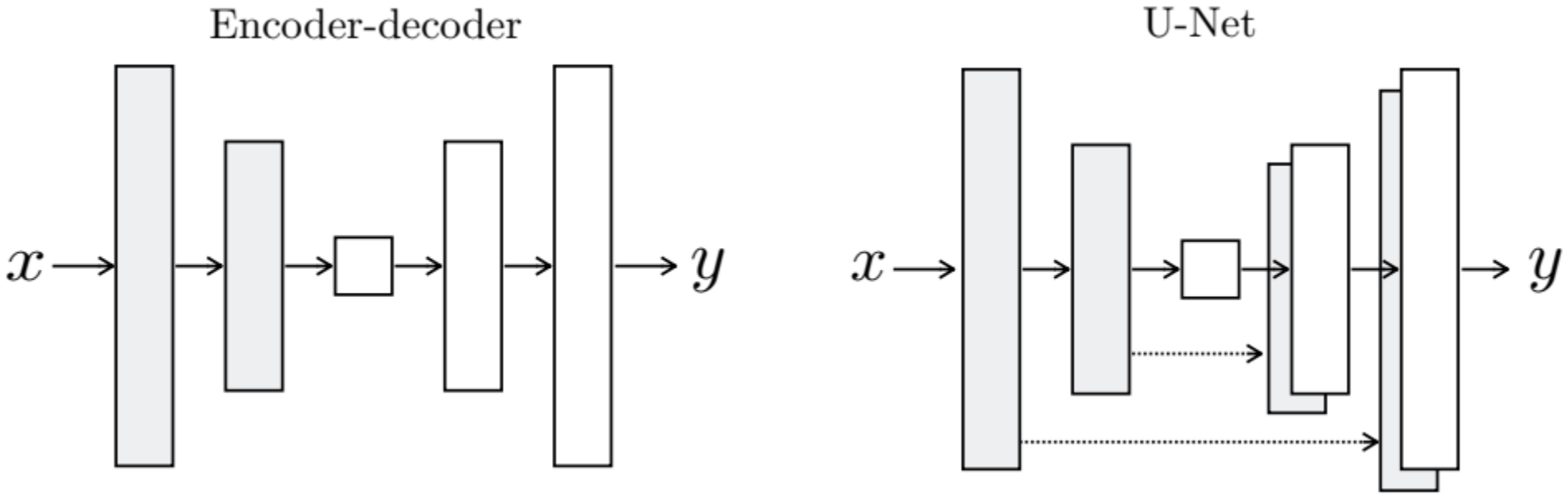
CGAN– Application pix2pix - Image-to-Image Translation with Conditional Adversarial Networks, CVPR 2017

- 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 - Image-to-Image Translation with Conditional Adversarial Networks, CVPR 2017

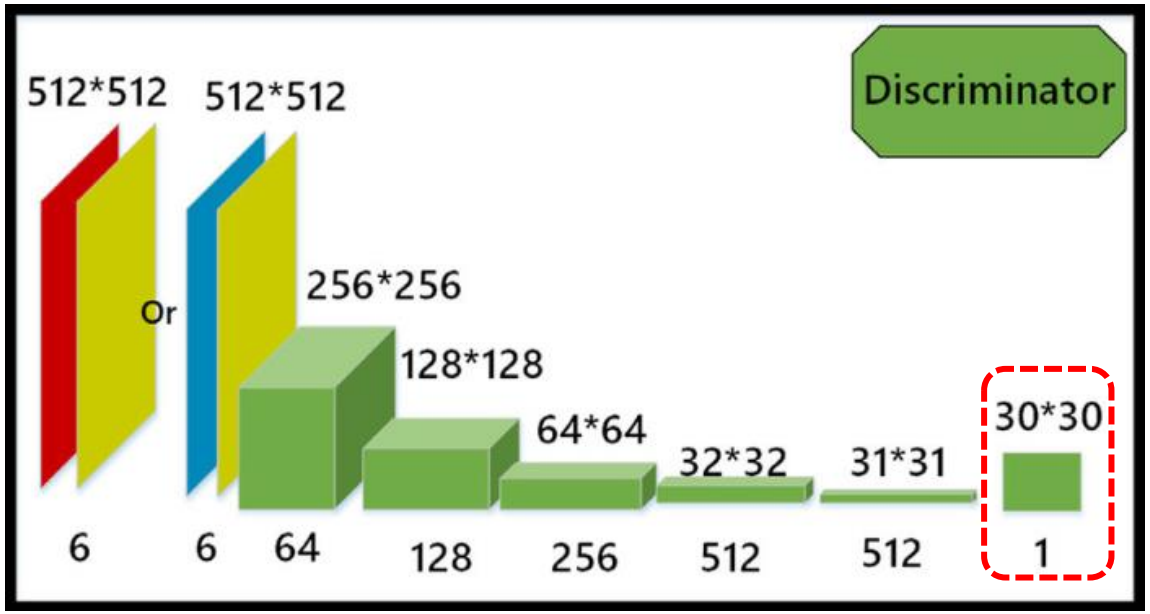
› Pix2Pix – Two alternatives for generator:
  – With (Right)/without(Left) skip connection



CGAN– Application pix2pix - Image-to-Image Translation with Conditional Adversarial Networks, CVPR 2017

› PatchGAN idea:

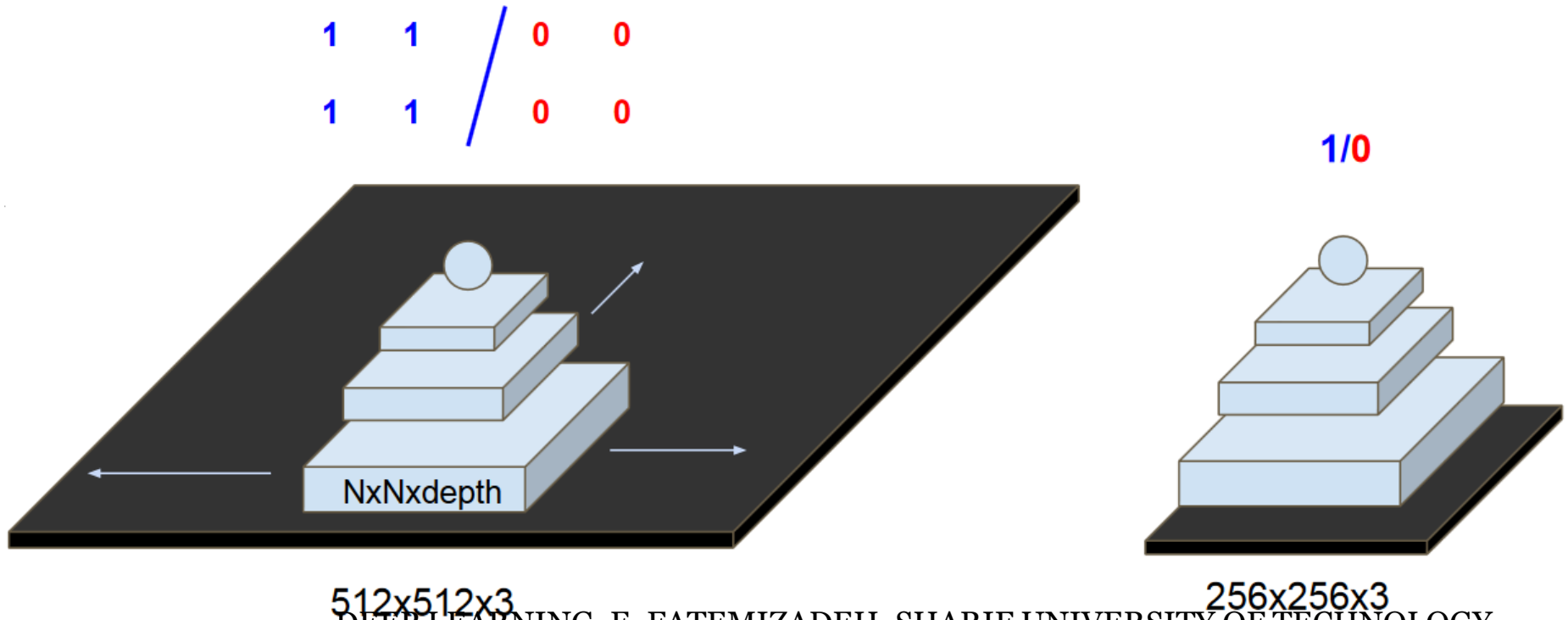
– Instead of producing decision output as single scalar vector (real vs. fake) it generates an NxN array.



CGAN– Application pix2pix - Image-to-Image Translation with Conditional Adversarial Networks, CVPR 2017

› PatchGAN idea:

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

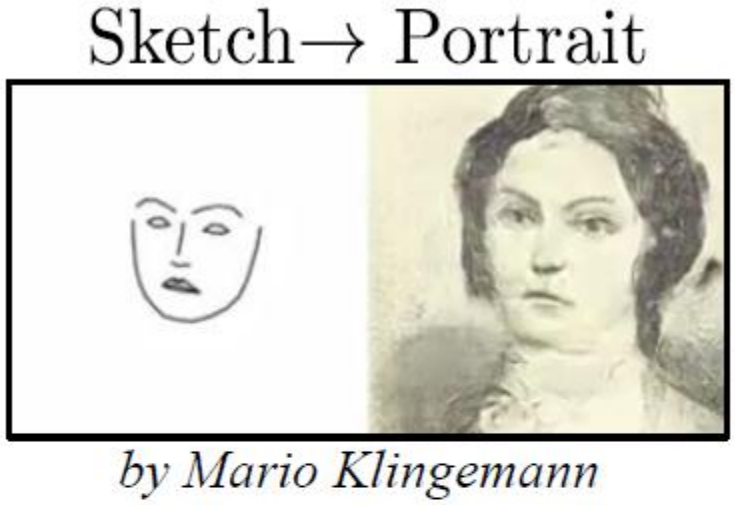


CGAN– Application pix2pix - Image-to-Image Translation with Conditional Adversarial Networks, CVPR 2017

› Pix2Pix (Edge2Photo):

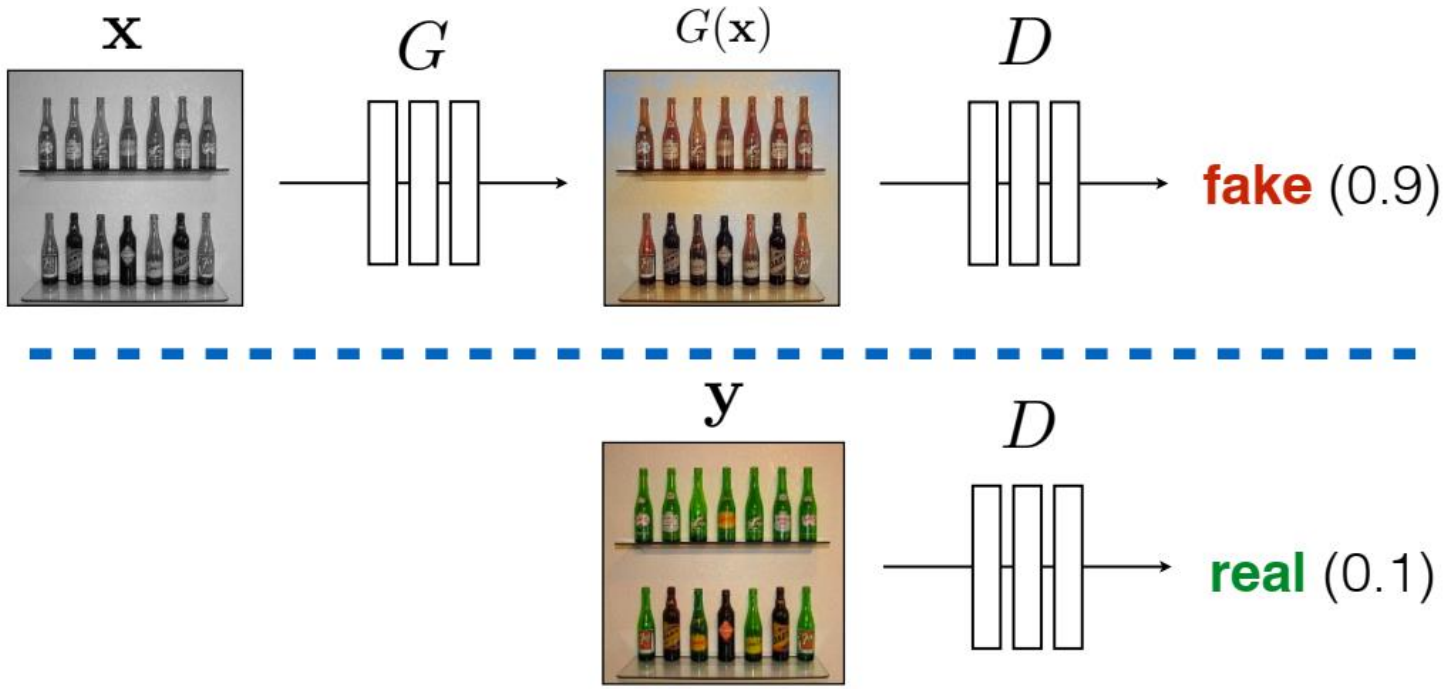
CGAN– Application pix2pix - Image-to-Image Translation with Conditional Adversarial Networks, CVPR 2017

› Pix2Pix:



CGAN– Application pix2pix - Image-to-Image Translation with Conditional Adversarial Networks, CVPR 2017

› Pix2Pix (Bw2COLOR):



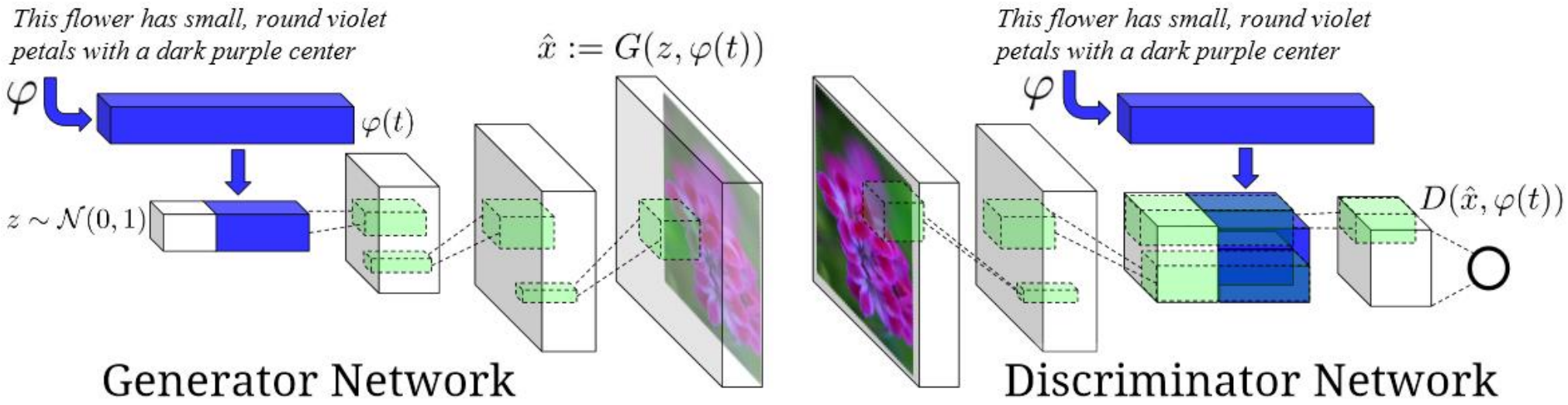
$$\arg\max_D \mathbb{E}_{\mathbf{x},\mathbf{y}}[\log D(G(\mathbf{x})) + \log(1 - D(\mathbf{y}))]$$

CGAN– Application pix2pix - Image-to-Image Translation with Conditional Adversarial Networks, CVPR 2017

› Pix2Pix (Aerialphoto2Map):

CGAN – Generative Adversarial Text to Image Synthesis, 2016

› Text-to-Image-Synthesis (G/D)



CGAN – Generative Adversarial Text to Image Synthesis, 2016

- Text-to-Image-Synthesis
- Training

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

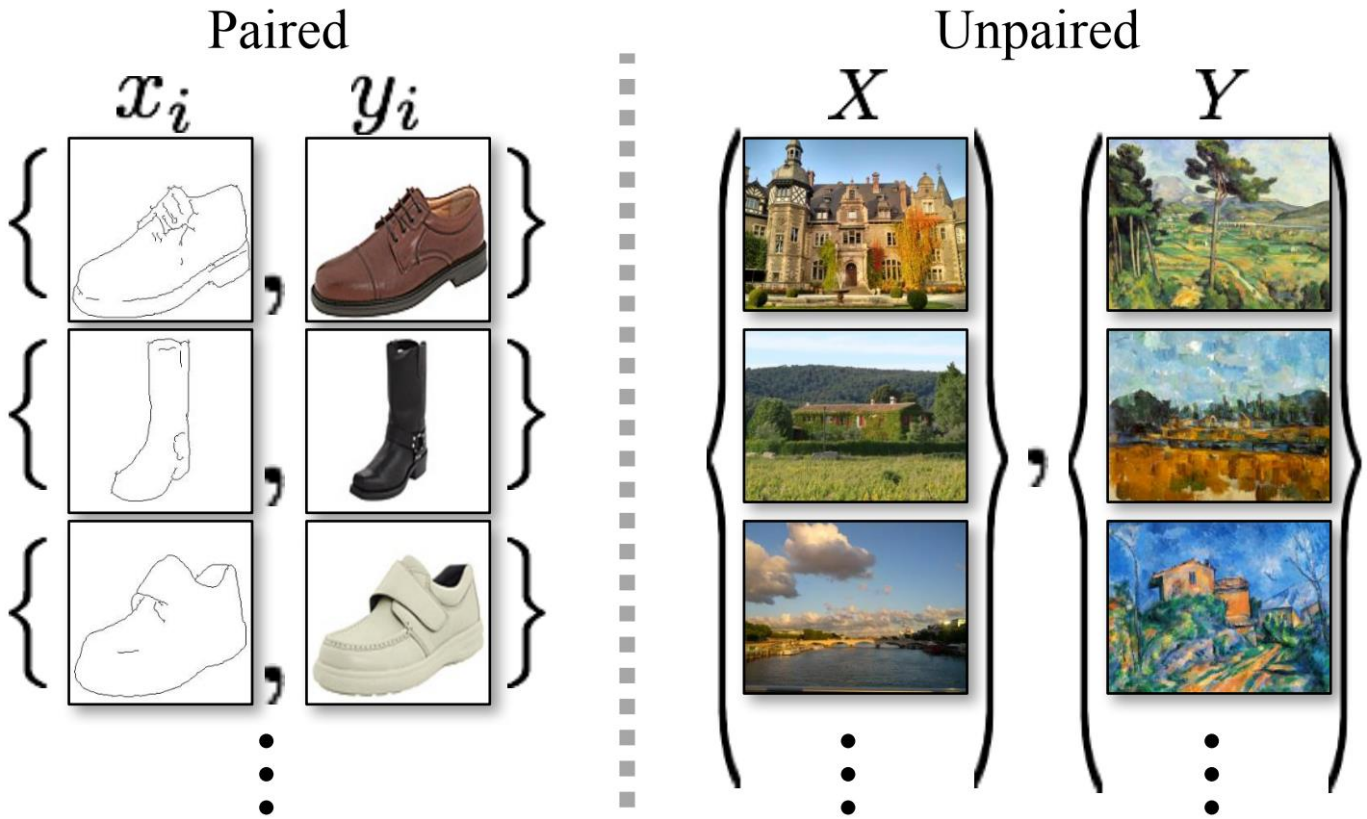
1: **Input:** minibatch images x, matching text t, mismatching $\hat{t}$, number of training batch steps S
2: **for** $n = 1$ **to** S **do**
3: $h \leftarrow \varphi(t)$ {Encode matching text description}
4: $\hat{h} \leftarrow \varphi(\hat{t})$ {Encode mis-matching text description}
5: $z \sim \mathcal{N}(0,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

CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, ICCV2017

› Image Translation (CycleGan Magic):

CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, ICCV2017

› Paired-Unpaired Data:



CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, ICCV2017

- **Main Goal**:

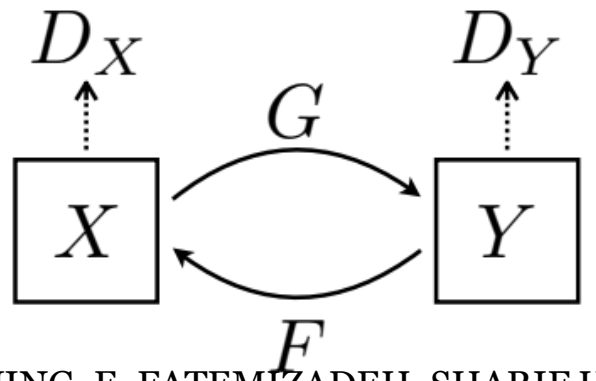
  An algorithm that can learn to translate between two domains without paired input-output examples.

- **Assumption**: There is some relationship between the domains

- **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 - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, ICCV2017

- Basic Structure:
- Two Mapping (two *Generators and thus two Discriminator*):
  - G: X→Y, $\hat{y} = G(x), x \in X$
  - F: Y→ 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 - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, ICCV2017

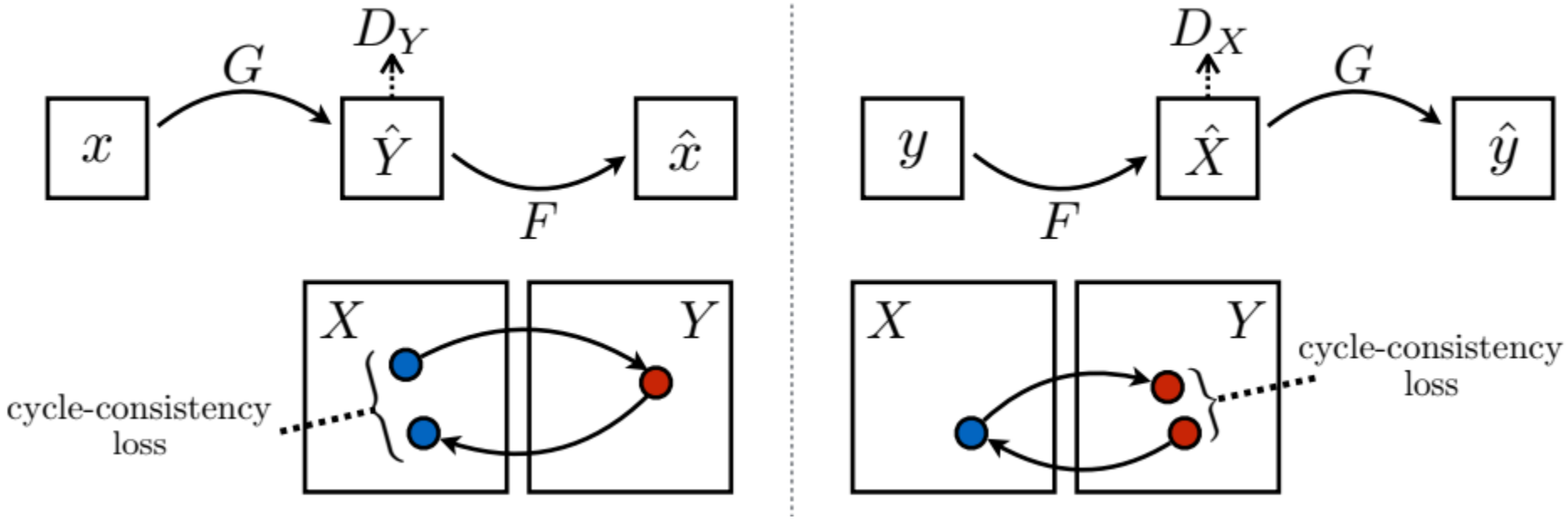
- Basic Structure:
- Add Cycle Consistency criteria

  G and F should be inverses of each other.

- We need a loss term for simultaneously:
  - $F\big(G(x)\big) \approx x$
  - $G\big(F(y)\big) \approx y$

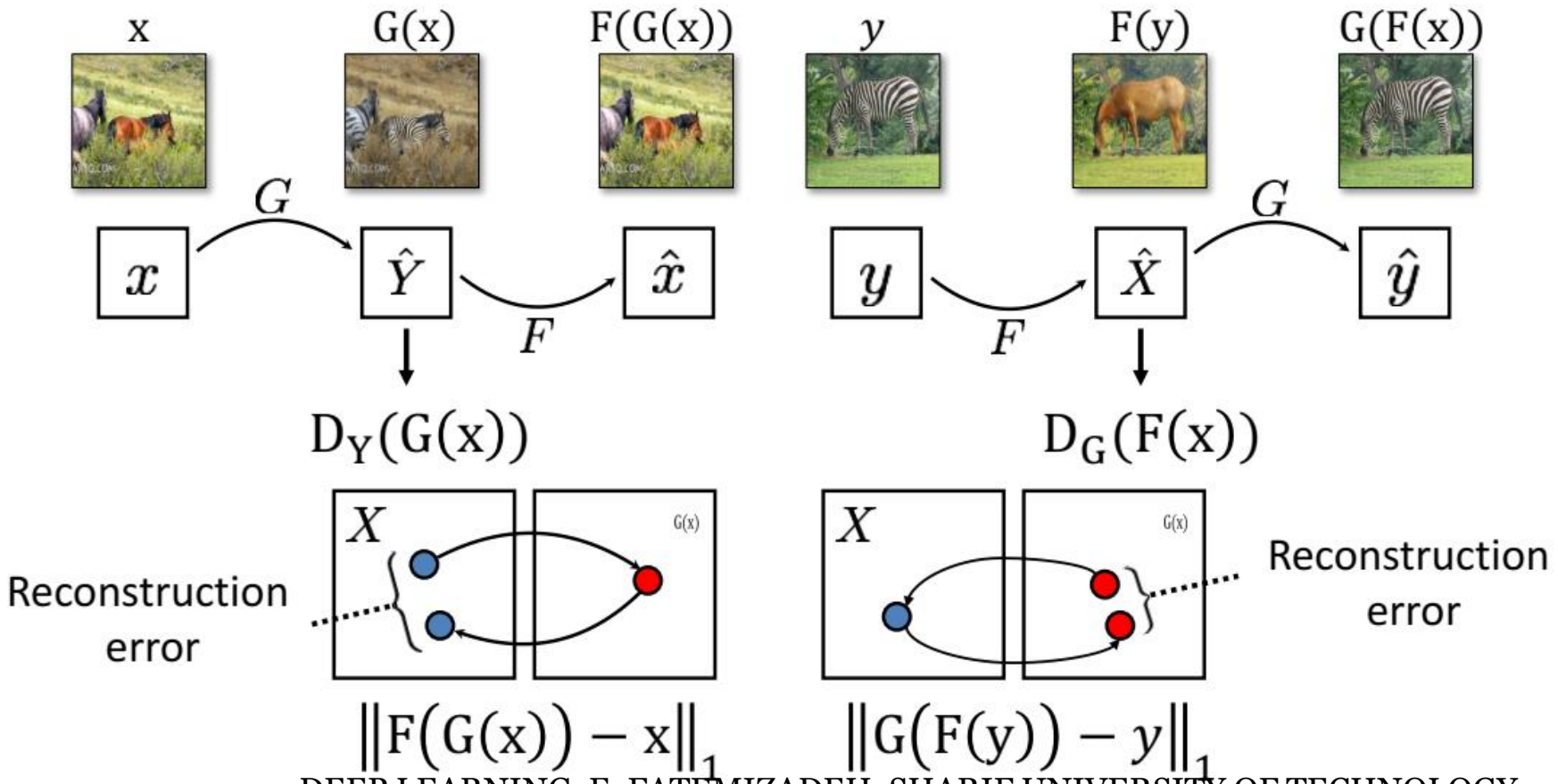
CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, ICCV2017

› Cycle Consistency Loss:



CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, ICCV2017

› Generator(s)/Discriminator(s):



CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, ICCV2017

- "Cycle Consistency Loss" and "Adversary Loss"
- Adversary Loss (Vanilla GAN or others):

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

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

- Cycle Consistency Loss:

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

- Total Loss:

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

CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, ICCV2017

› Total Loss Optimization:

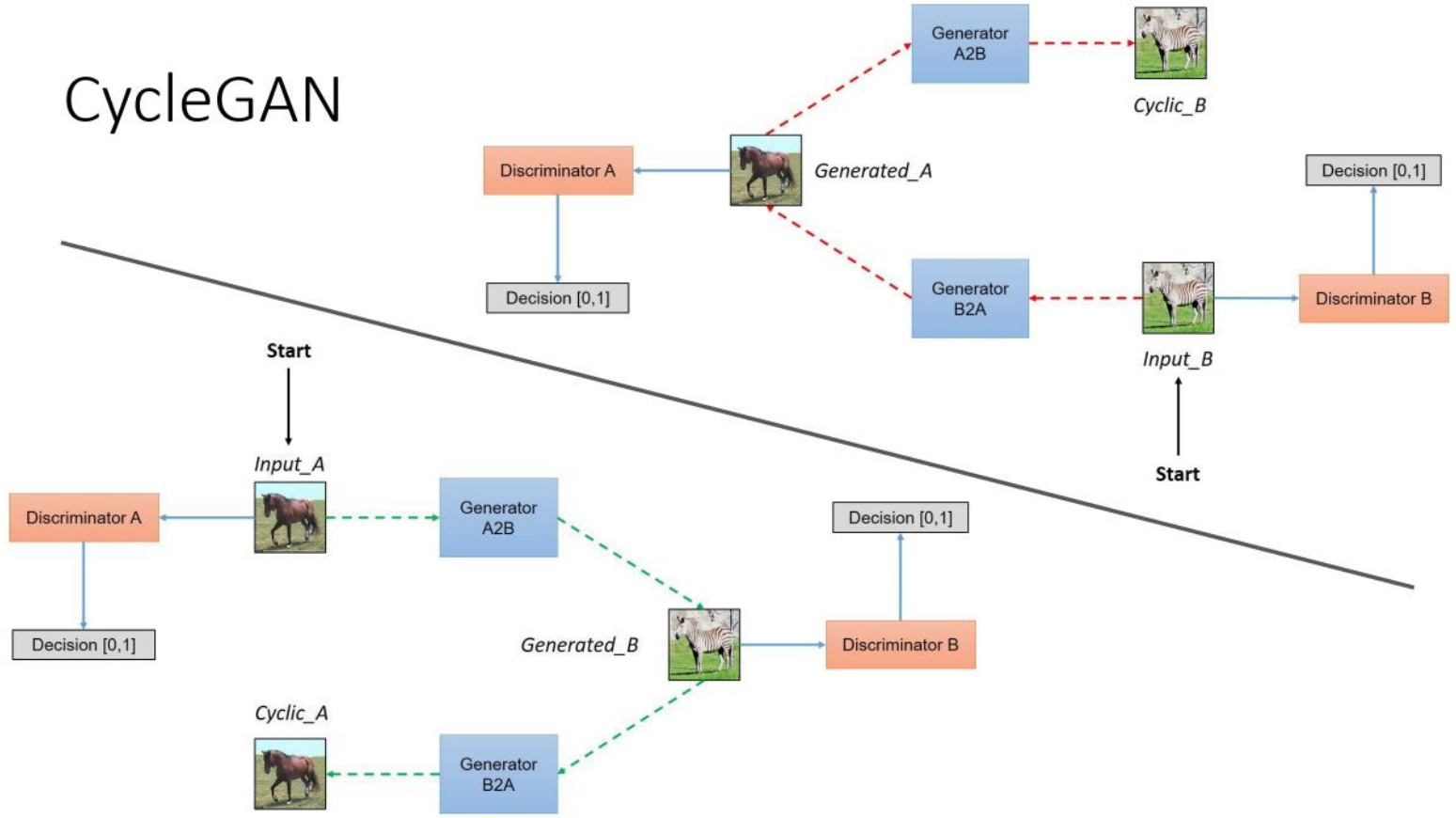
$$\{G^*, F^*\} = arg \underbrace{min}_{G,F} \underbrace{max}_{D_X, D_Y} \mathcal{L}(G, F, D_X, D_Y)$$

› Equivalent Problem: Design two autoencoder:

$$FoG: X \rightarrow X \text{ and } GoF: Y \rightarrow Y$$

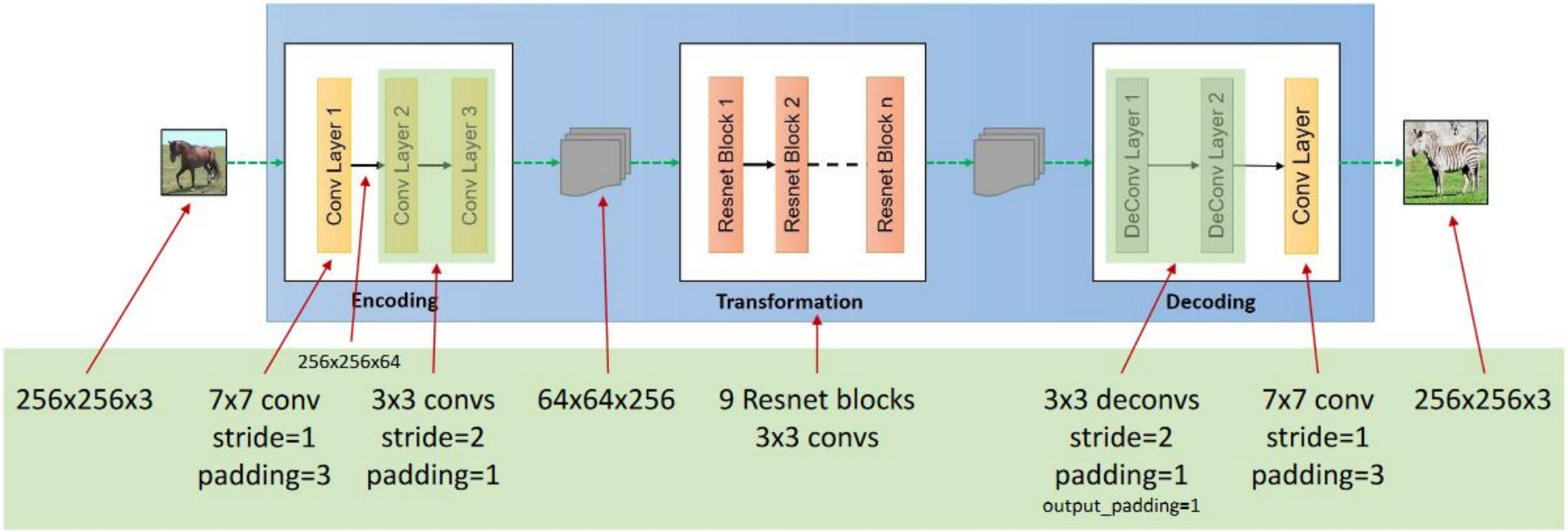
CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, ICCV2017

› CycleGAN



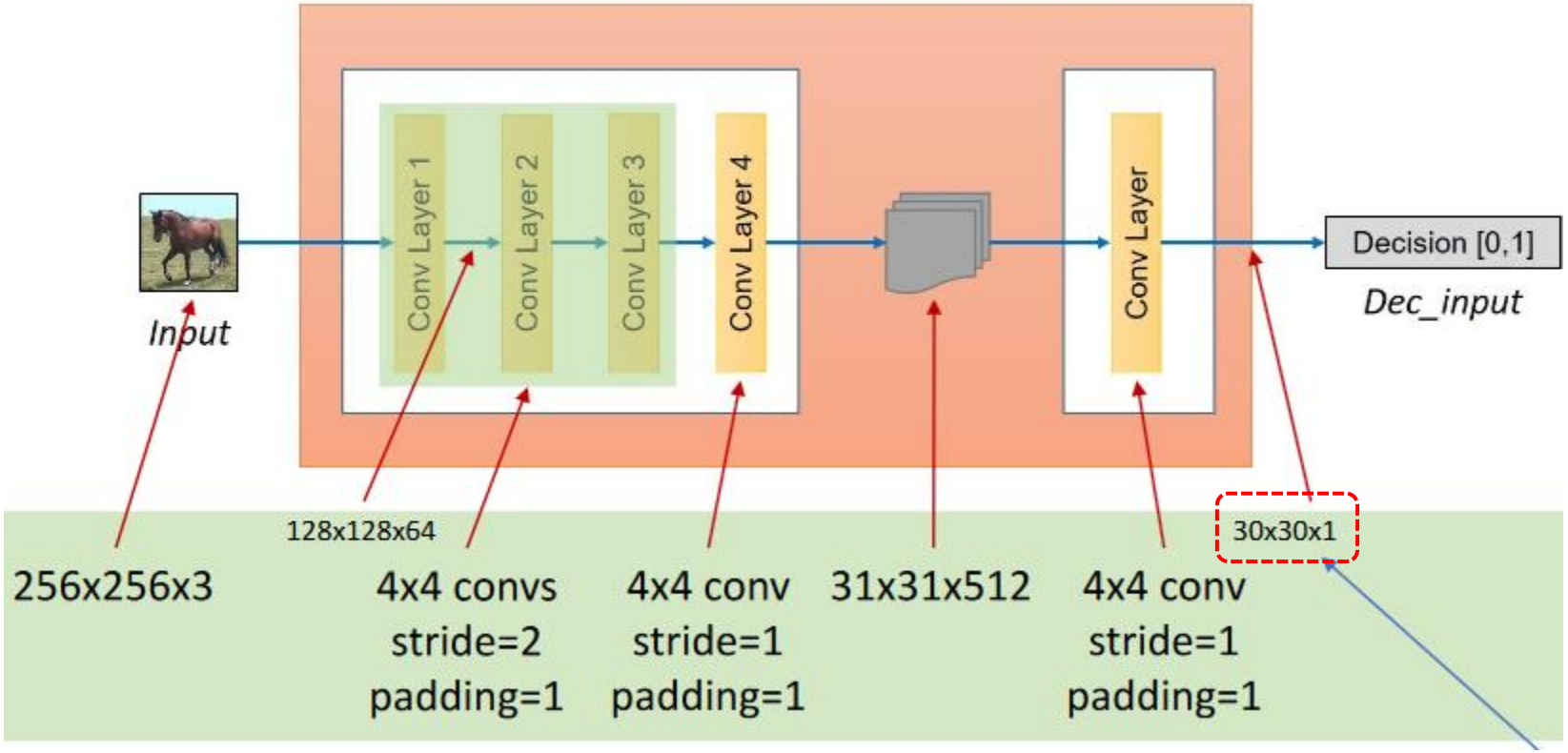
CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, ICCV2017

› GAN Generators:



CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, ICCV2017

› GAN Discriminators:



CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, ICCV2017

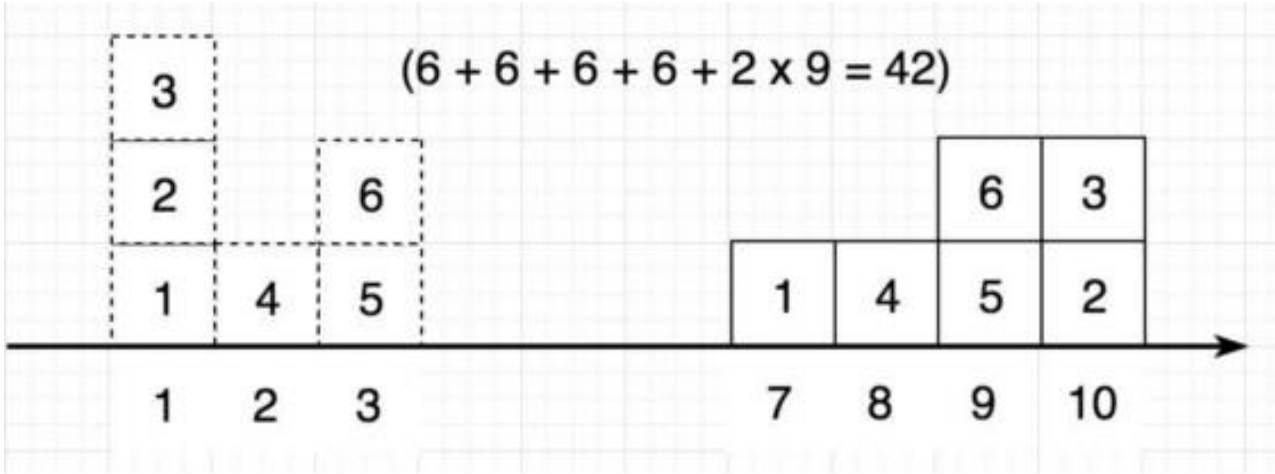
- Training:
- Generate image from generator X using image from domain X, Similarly generate an image from generator Y using image from domain Y.
- Train discriminator X on batch using images from domain X and images generated from generator Y as real and fake image respectively.
- Train discriminator 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 - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, ICCV2017

› Results:

Wasserstein GAN (WGAN) - Wasserstein GAN, 2017

- 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

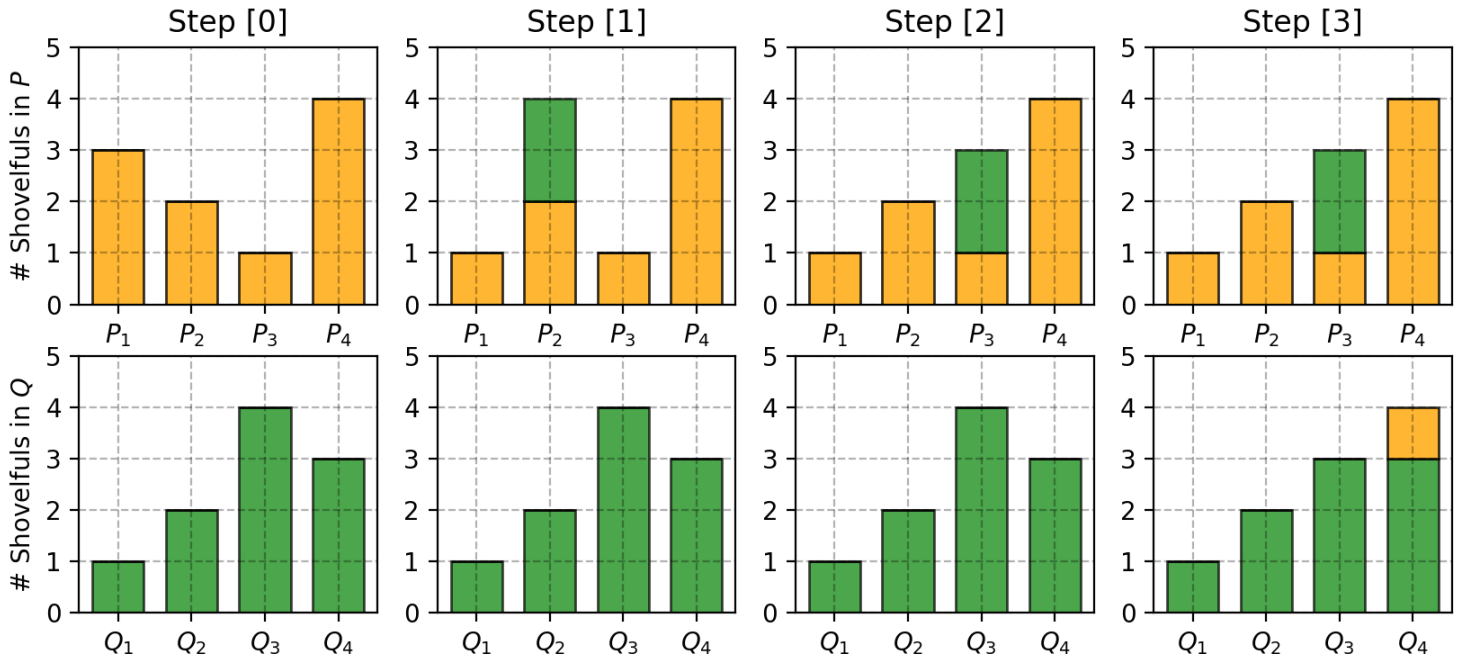


  - Question: What is minimum cost, if we know source and target distributions

Wasserstein GAN (WGAN) - Wasserstein GAN, 2017

- 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



Wasserstein GAN (WGAN) - Wasserstein GAN, 2017

› 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 are respectively P_r and P_g.

- $\sum_x \gamma(x, y) = p_g(y)$
- $\sum_y \gamma(x, y) = p_r(x)$

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

Wasserstein GAN (WGAN) - Wasserstein GAN, 2017

› 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:

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

› The Jenson-Shannon (JS) distance:

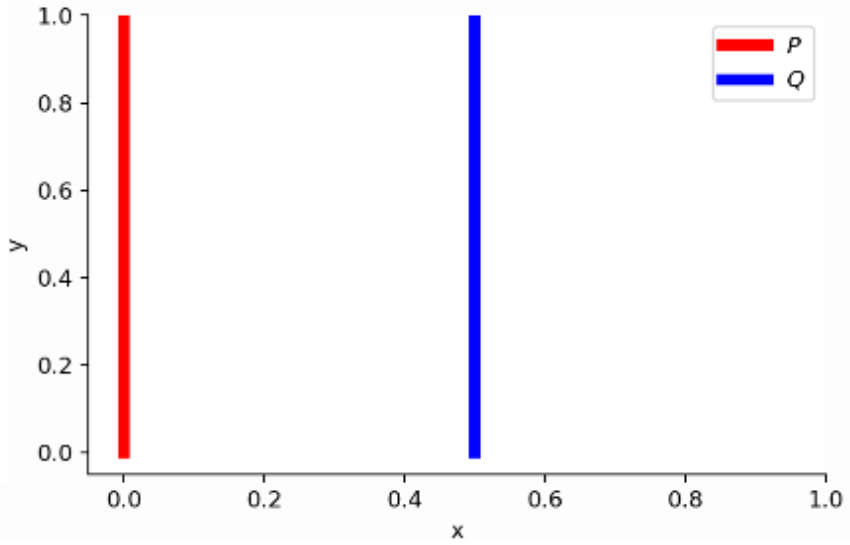
$$JS(P_r, P_g) = \frac{1}{2} KL(P_r \| P_m) + \frac{1}{2} KL(P_g \| P_m)$$

Wasserstein GAN (WGAN) - Wasserstein GAN, 2017

› Results:

- $W(\mathbb{P}_0, \mathbb{P}_\theta) = |\theta|,$
- $JS(\mathbb{P}_0, \mathbb{P}_\theta) = \begin{cases} \log 2 & \text{if } \theta \neq 0 \ , \\ 0 & \text{if } \theta = 0 \ , \end{cases}$
- $KL(\mathbb{P}_\theta \| \mathbb{P}_0) = KL(\mathbb{P}_0 \| \mathbb{P}_\theta) = \begin{cases} +\infty & \text{if } \theta \neq 0 \ , \\ 0 & \text{if } \theta = 0 \ , \end{cases}$
- and $\delta(\mathbb{P}_0, \mathbb{P}_\theta) = \begin{cases} 1 & \text{if } \theta \neq 0 \ , \\ 0 & \text{if } \theta = 0 \ . \end{cases}$

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



› W is Continuous (for continuous P_θ)!

Wasserstein GAN (WGAN) - Wasserstein GAN, 2017

› Kantorovich-Rubinstein duality:

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

› Where f is K-**L**ipschitz continuous:

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

› Where is Deep? f may be a deep (discriminator)

Wasserstein GAN (WGAN) - Wasserstein GAN, 2017

› WGAN:

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

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

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

Wasserstein GAN (WGAN) - Wasserstein GAN, 2017

› WGAN:

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

$$\begin{aligned} \nabla_\theta W(P_r, P_\theta) &= \nabla_\theta (\mathbb{E}_{x \sim P_r}[f_w(x)] - \mathbb{E}_{z \sim Z}[f_w(g_\theta(z))]) \\ &= -\mathbb{E}_{z \sim Z}[\nabla_\theta f_w(g_\theta(z))] \end{aligned}$$

› What for "K-Lipschitz continuous" condition for $f_w(x)$:

– Clipping [−c,c], c~ 0.01

Wasserstein GAN (WGAN) - Wasserstein GAN, 2017

› 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.

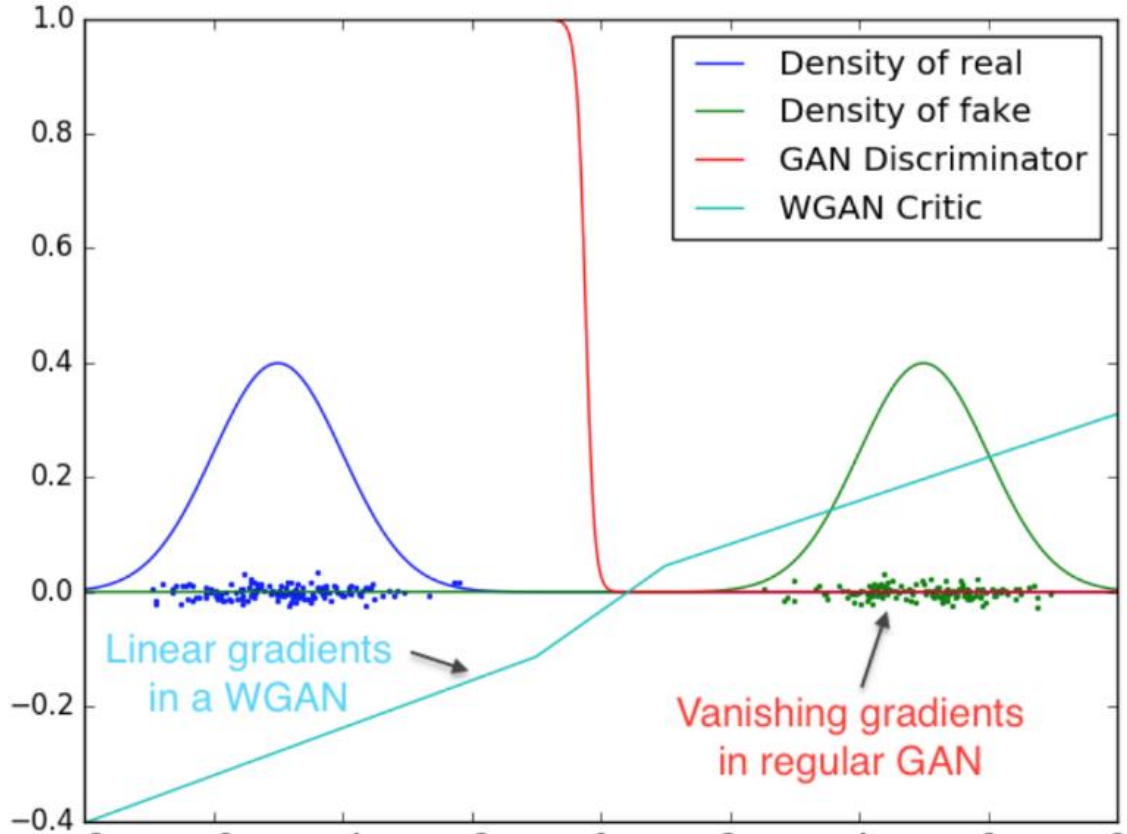
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$

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

end while

Wasserstein GAN (WGAN) - Wasserstein GAN, 2017

› D(x) plot for two Gaussian (fake/real)



Wasserstein GAN (WGAN) - Wasserstein GAN, 2017

› 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.

GAN Tips and Tricks- Improved Training of Generative Adversarial Networks using Representative Features, ICML2018

- Improve GAN training Stability:
  - Change *log(1-D)*→*-log(D)* (Gradient Vanishing)
  - DCGAN (Gradient Vanishing)
  - Unrolled GAN (Mode Collapse)
  - F-divergence (Training instability)
  - RFGAN: Better Training Stability (Mode Collapse, Better Visual Output)

GAN – Roots of Problems

- GD/SGD is not well solver for “two player game”
- Simultaneous updates needs G←→D balance
  - Moving target for both **G** and **D**
- 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 → Worse situation for generator
  - Over–trained discriminator → 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 $D_{loss} > D_{THR}$ then train D
    - if $G_{loss} > G_{THR}$ 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

Deep Learning Last Words

- Suggested works:
  - InfoGAN
  - PixelCNN/PixelRNN
  - BiGAN
  - ….

Deep Learning Last Words

- Not Covered:
  - Restricted/Deep Boltzmann Machine (RBM/DBM), why?
    - Less application today and need to pass/have Statistical Graphical Models
  - Reinforcement Learning, why?
    - Need a complete course for RK and then Deep RL
- Not Covered Well:
  - Visualization and Representation, Style Transfer, GAN Painter, … why?
    - Time!