Generative Adversarial Networks (GAN)

ISTD 50.035

Computer Vision

Acknowledgement: Some images are from various sources: UCF, Stanford cs231n, National Taiwan University, etc.

Generative Model

- Unsupervised learning techniques
- Train a model to generate data
- Input: random noise/ random vector
- Output: synthetic images

Training dataset:

…

$[0.2, 0.3, \ldots]^T$ → Generator →

Generative Model

- Unsupervised learning techniques
- Train a model to generate data
- Input: random noise/ random vector
- Output: synthetic images

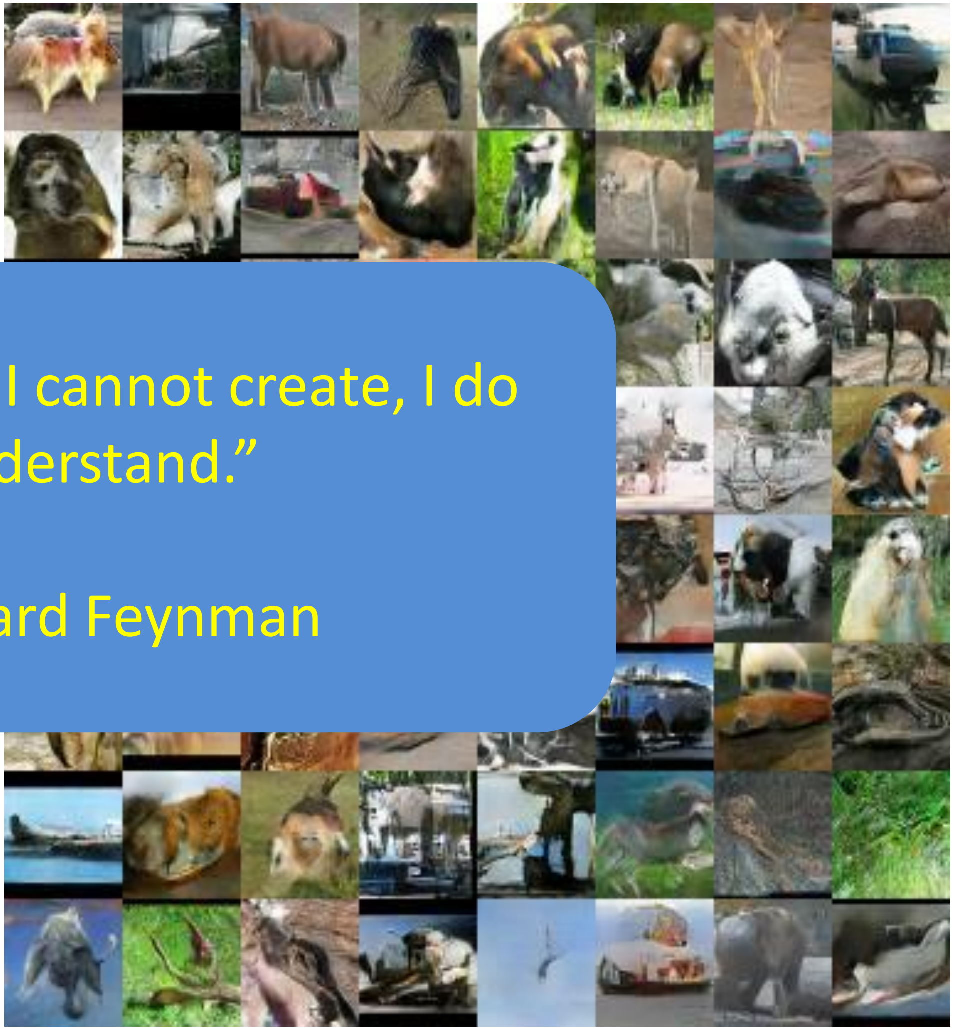
[Tran, Bui, Cheung; 2018]

Synthetic images:

Generative Model

- Deep understanding of the data and the essence

[Tran, Bui, Cheung; 2018]

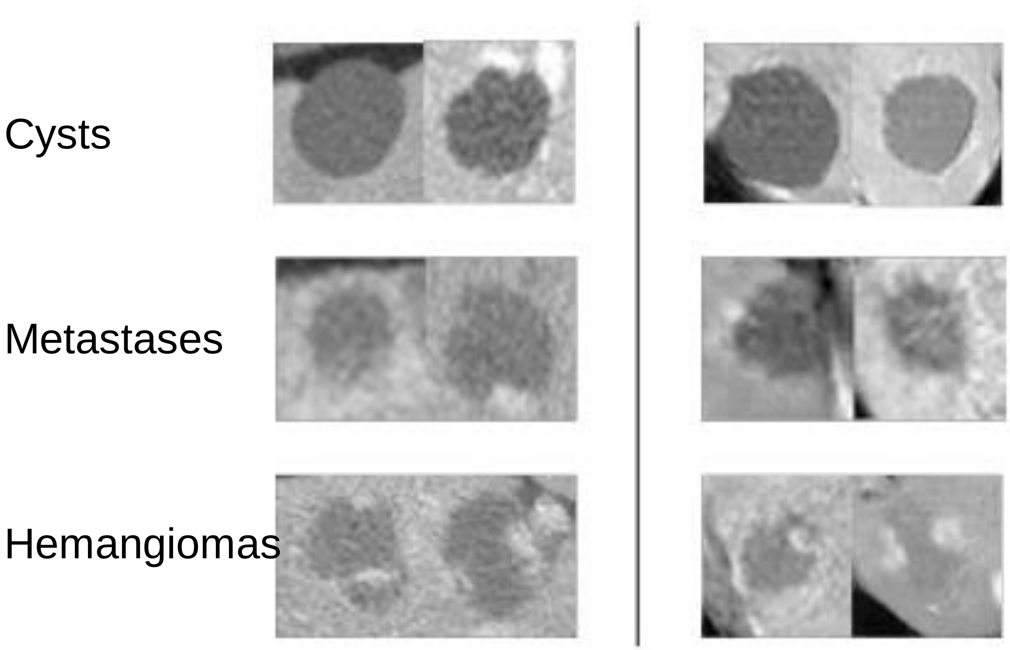


"What I cannot create, I do not understand."

-- Richard Feynman

Generative Model

- Liver lesion classification with synthetic image augmentation



Real lesions Synthetic lesions

	Sensitivity	Specificity
Baseline	78.6%	88.4%
With synthetic data	85.7%	92.4%

[Frid-Adar et al.; 2018]

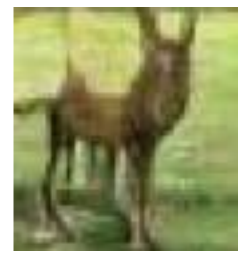
Generative Adversarial Nets (GAN)

[Goodfellow et al.; 2014]

Two networks: Generator and discriminator

$[0.2, 0.3, \ldots]^T$

Generator



Random noise vector (e.g. 100-d uniform distributed random vector)

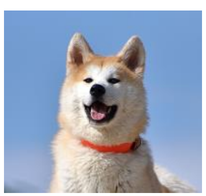
Image (vector with increased number of dimension)

Generative Adversarial Nets (GAN)

Two networks: Generator and discriminator

Image → Discriminator → Scalar score (large value means real; small value means fake)

Scalar score = P(input is real)

 → Discriminator → 1.0

 → Discriminator → 0.0

Generative Adversarial Nets (GAN)

Training:

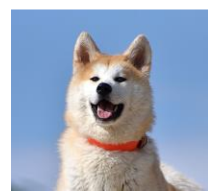
Initialize G and D

At each training iteration:

Train D to output high scores to real images and low scores to fake images

Step 1: Fix generator G, update discriminator D

update



…

D

Labeled with score 1

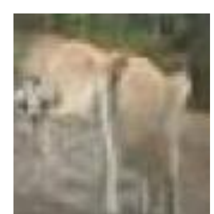
Output: 1 for real, 0 for fake

$[0.2, 0.3, \ldots]^T$

$[0.1, 0.4, \ldots]^T$

fix

G



…

Labeled with score 0

Generative Adversarial Nets (GAN)

Training:

Initialize G and D

At each training iteration:

Step 2: Fix discriminator D, update G

Train G to ‘fool’ the discriminator: maximize the D score for its generated images

fix

D

0 -> 1

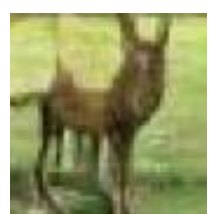
$[0.2, 0.3, \ldots]^T$

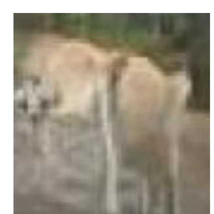
$[0.1, 0.4, \ldots]^T$

update

G

update

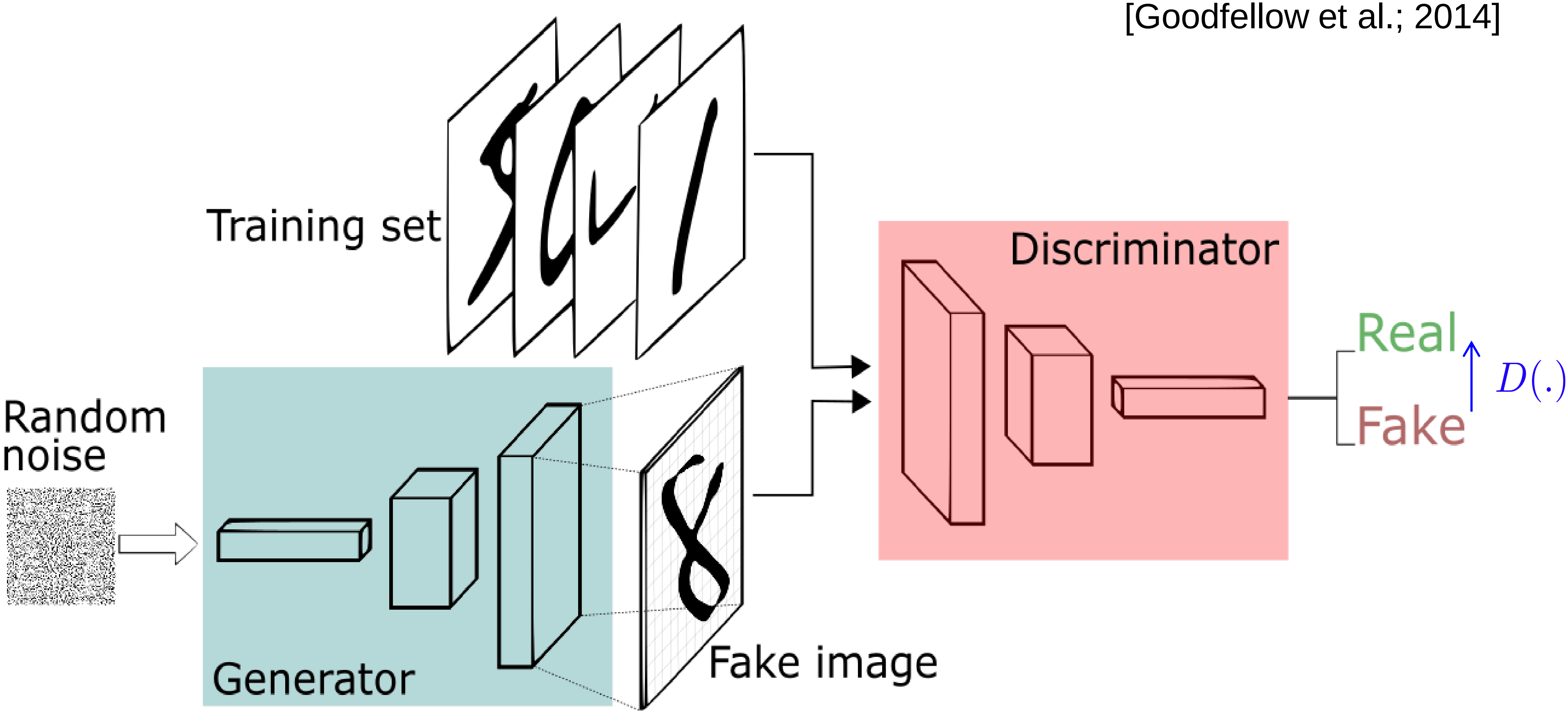




…

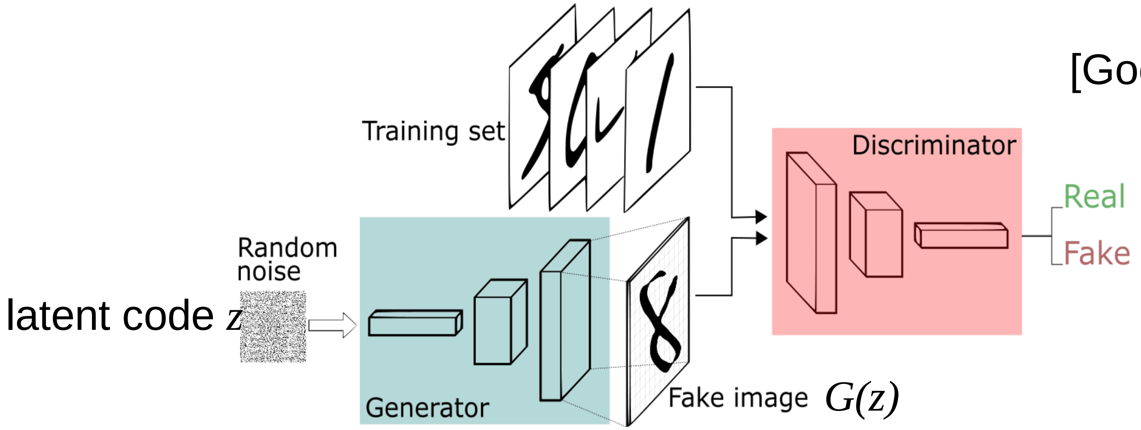
Generative Adversarial Nets (GAN)

[Goodfellow et al.; 2014]



Generative Adversarial Nets (GAN)

[Goodfellow et al.; 2014]

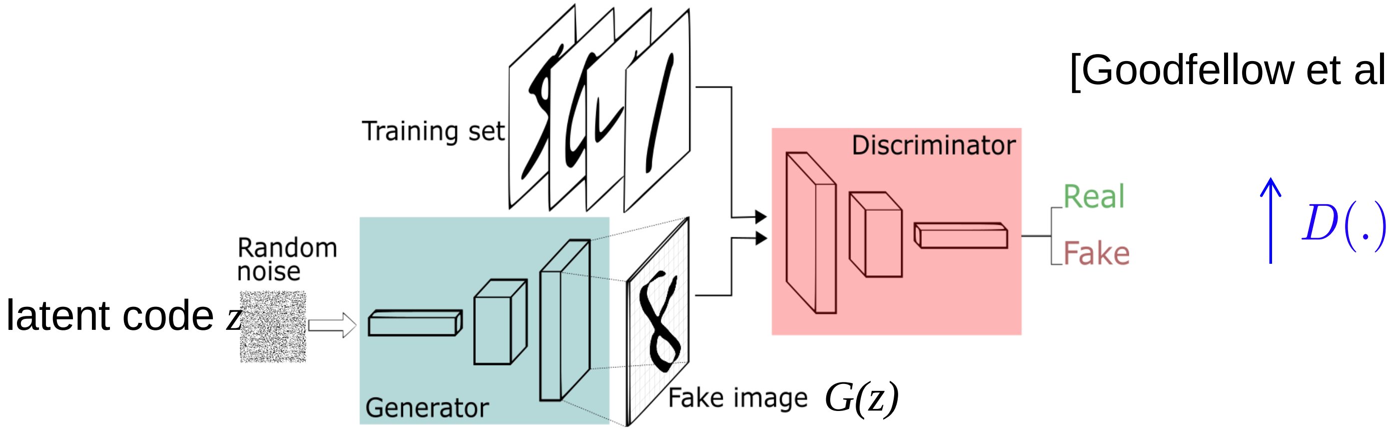


$\uparrow D(.)$

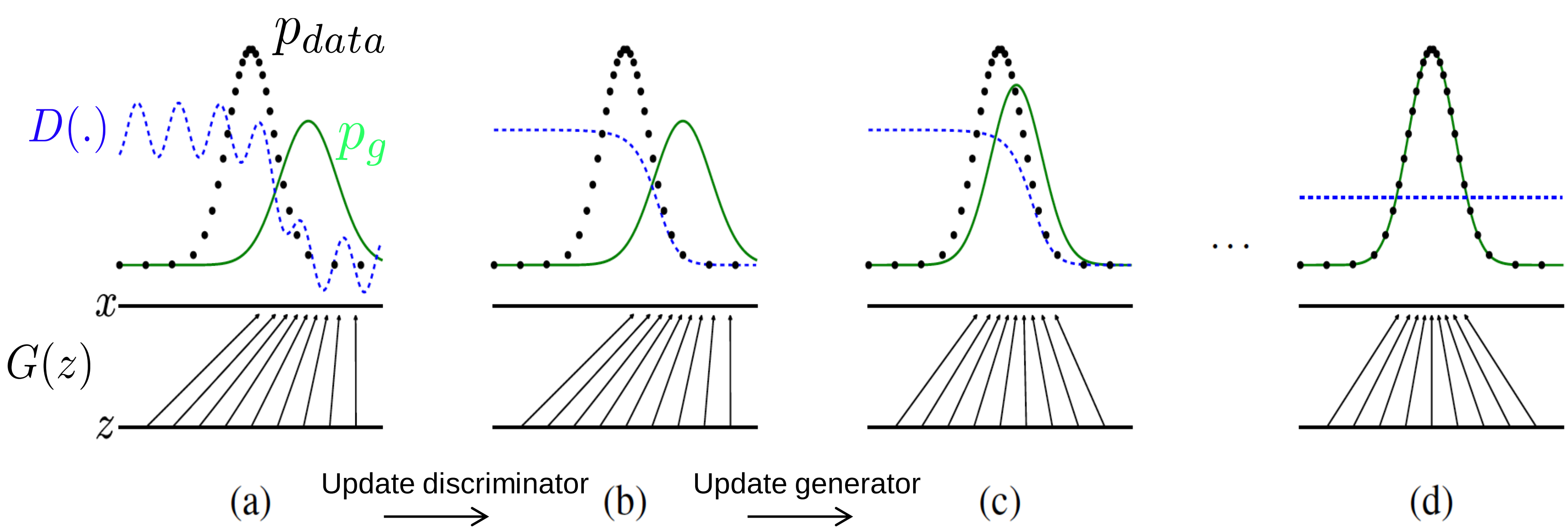
$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Generative Adversarial Nets (GAN)

[Goodfellow et al.; 2014]



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



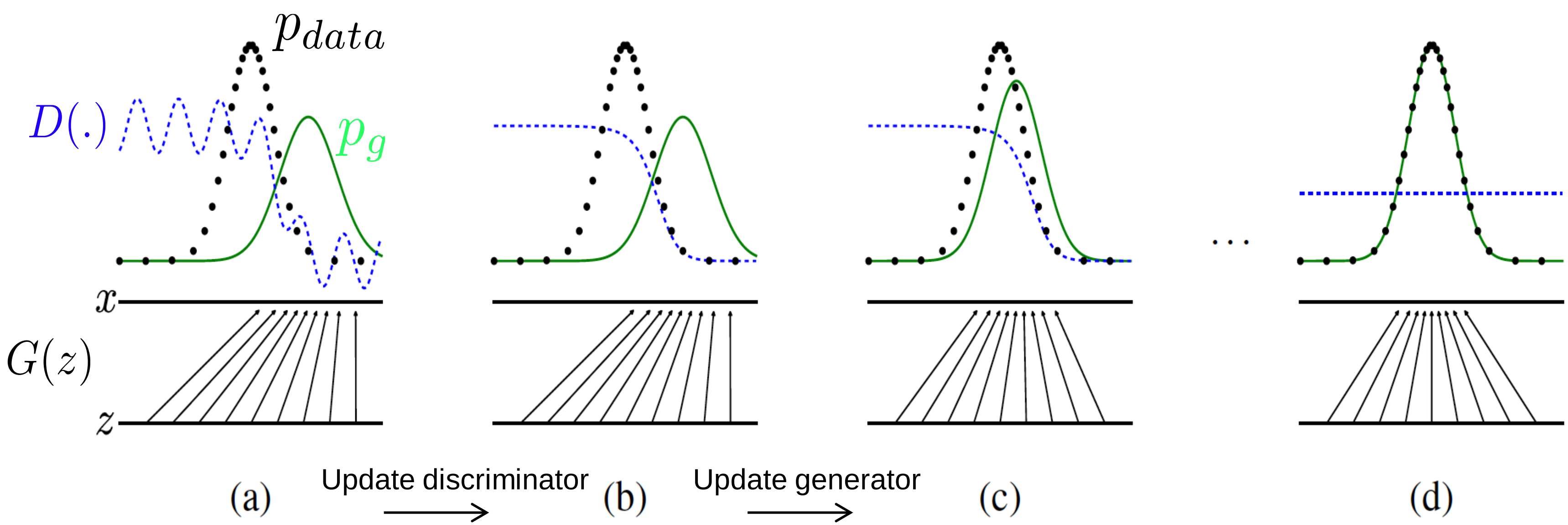
Generative Adversarial Nets (GAN)

-Mapping $G(.)$ imposes p_g

-Optimal $D(.)$ at each step is: $$\frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x})+p_g(\boldsymbol{x})}$$

-Gradient of D is used to guide G to move to regions that are more likely to be classified as real

-At the end of an ideal training: p_g=p_{data}; $D(.)$ = 0.5



Generative Adversarial Nets (GAN)

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

---

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

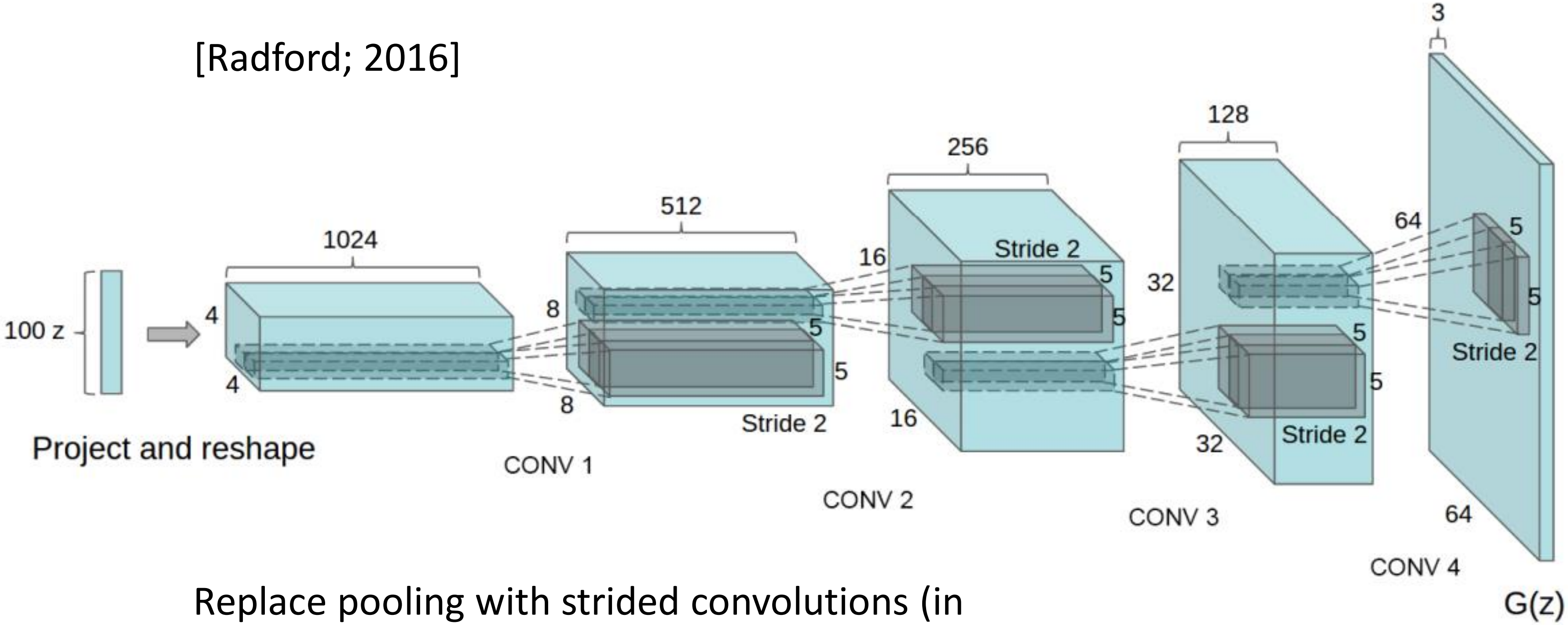
- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

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

end for

Network Architecture: DCGAN

[Radford; 2016]

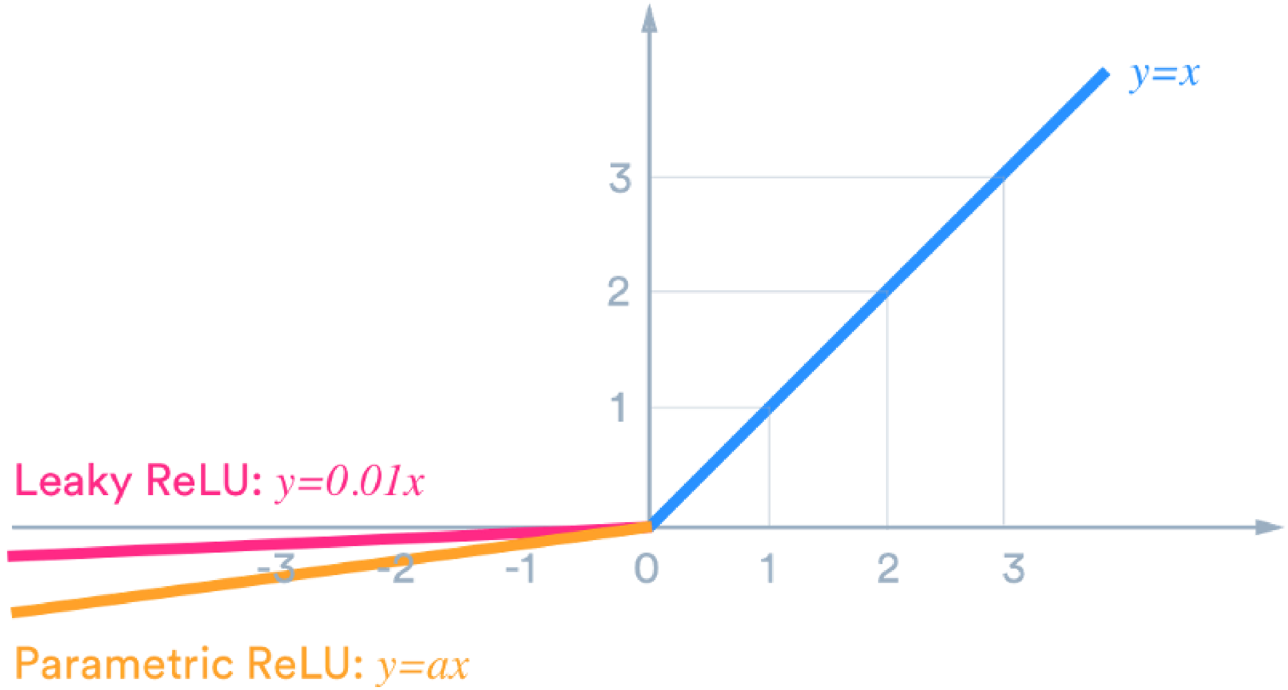


Replace pooling with strided convolutions (in discriminator) and fractional-strided (transpose) convolutions (in generator)

Use batchnorm

Use LeakyReLU in discriminator

Leaky ReLU and parametric ReLU (PReLU)

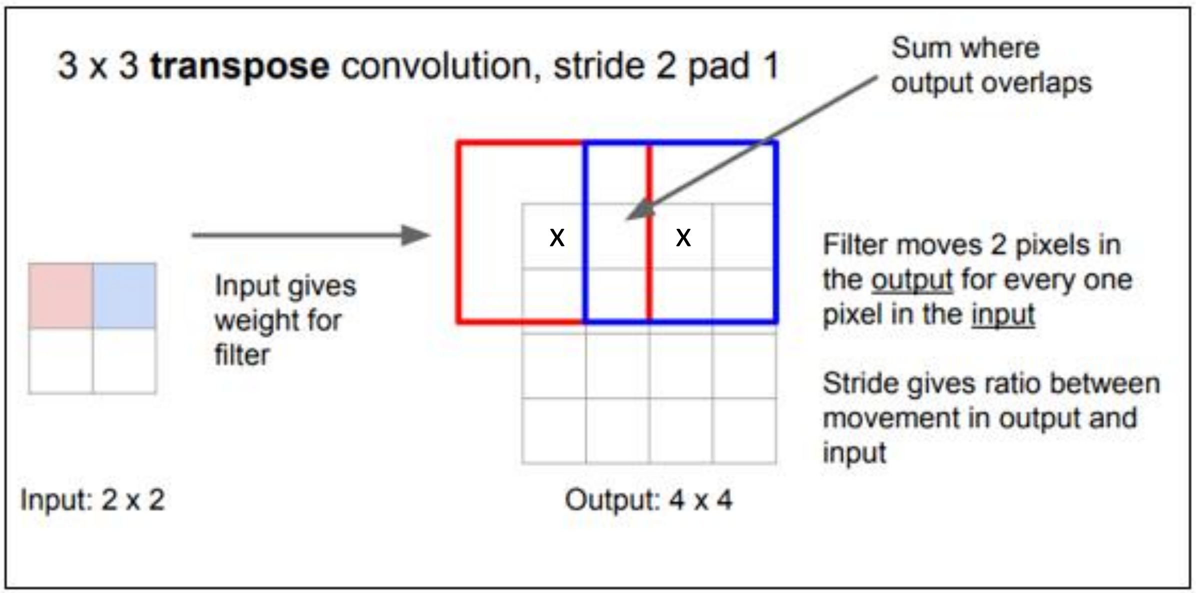


Leaky ReLU: a small slope for negative values, instead of all zero

PReLU: slope is a parameter

Upsampling by transpose convolution

Transpose convolution

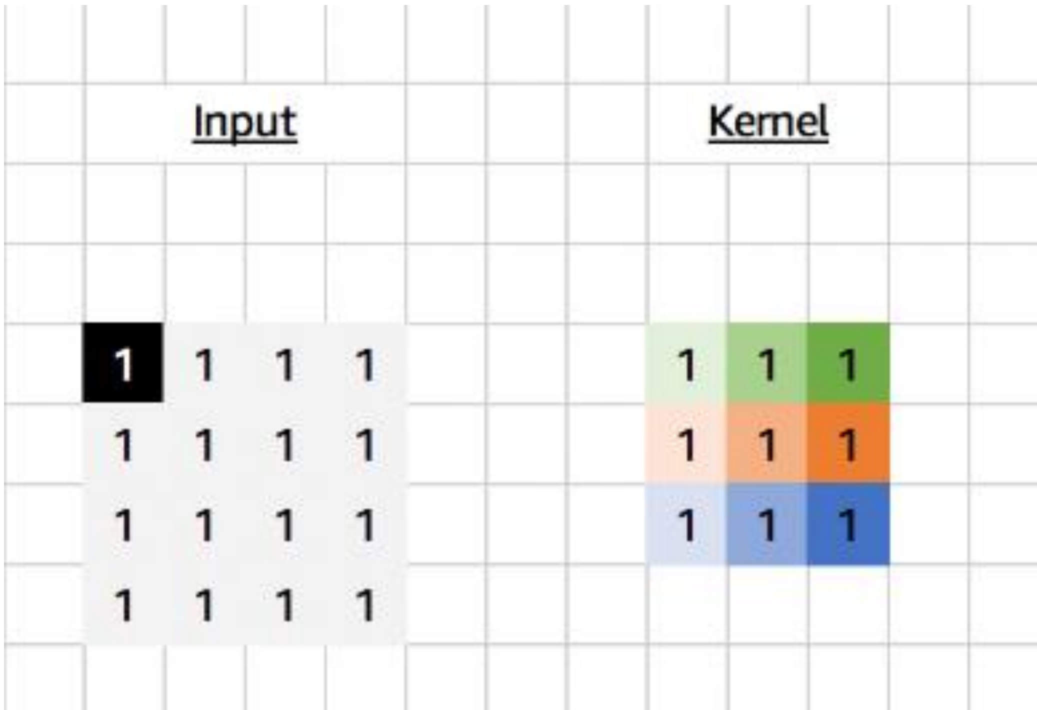


Standard convolution: input (matrix) * filter (matrix) -> output (scalar)
Transpose convolution: input (scalar) * filter (matrix) -> output (matrix)

Learn filter mask for optimal upsampling

Upsampling by transpose convolution

Transpose convolution



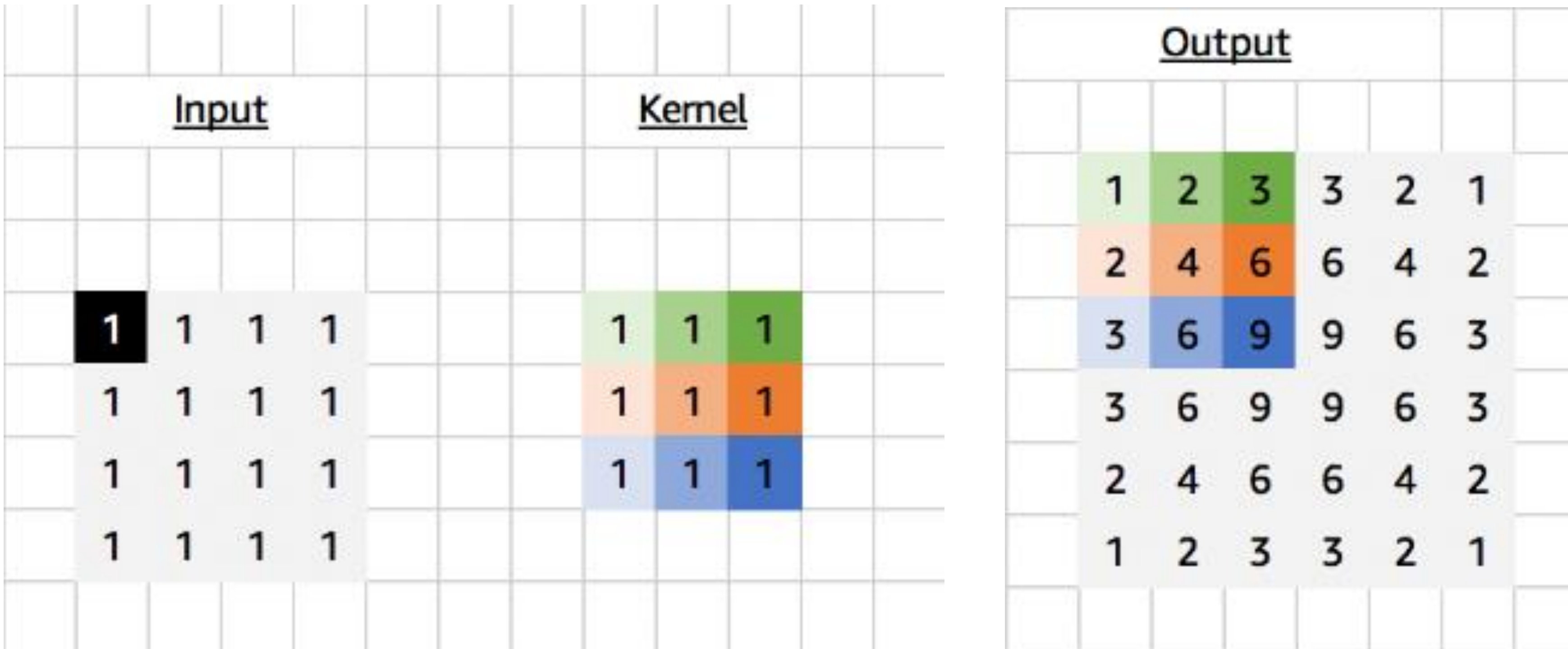
Output? (stride 1, pad 1, 6x6 output)

Standard convolution: input (matrix) * filter (matrix) -> output (scalar)
Transpose convolution: input (scalar) * filter (matrix) -> output (matrix)

Learn filter mask for optimal upsampling

Upsampling by transpose convolution

Transpose convolution



Standard convolution: input (matrix) * filter (matrix) -> output (scalar)
Transpose convolution: input (scalar) * filter (matrix) -> output (matrix)

Learn filter mask for optimal upsampling

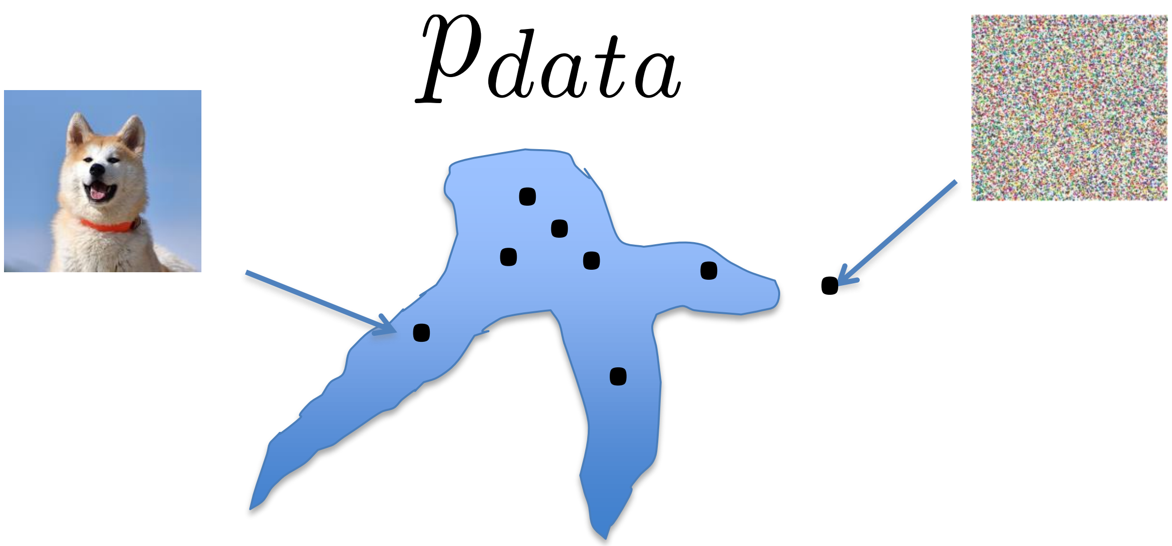
Images as samples from a probability distribution

Image as a HxWxC sample point

E.g. 256x256x3 for a 256x256 RGB image

Each pixel value -> value in one dimension

Real image -> small portion in this high dimensional space

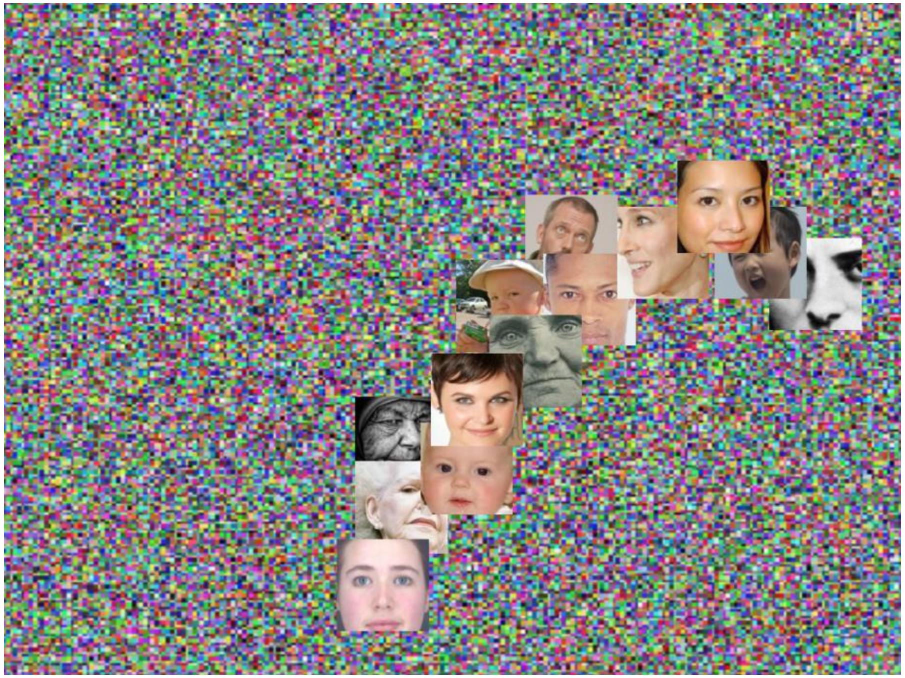


Images as samples from a probability distribution

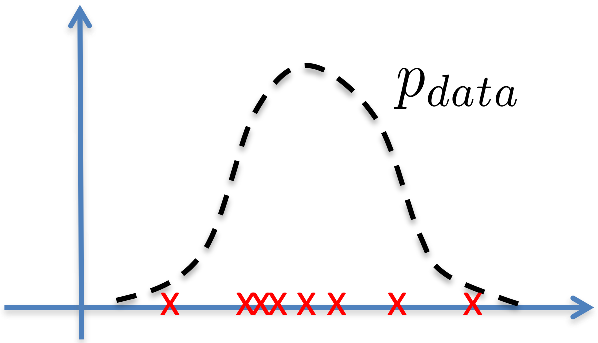
- The manifold hypothesis: natural data in high dimensional spaces concentrates close to lower dimensional manifolds

Natural images occupy a tiny fraction of the space $\mathbb{R}^{H \cdot W \cdot C}$

Smooth transformation from one image to another: continuous path along lower dimensional manifold

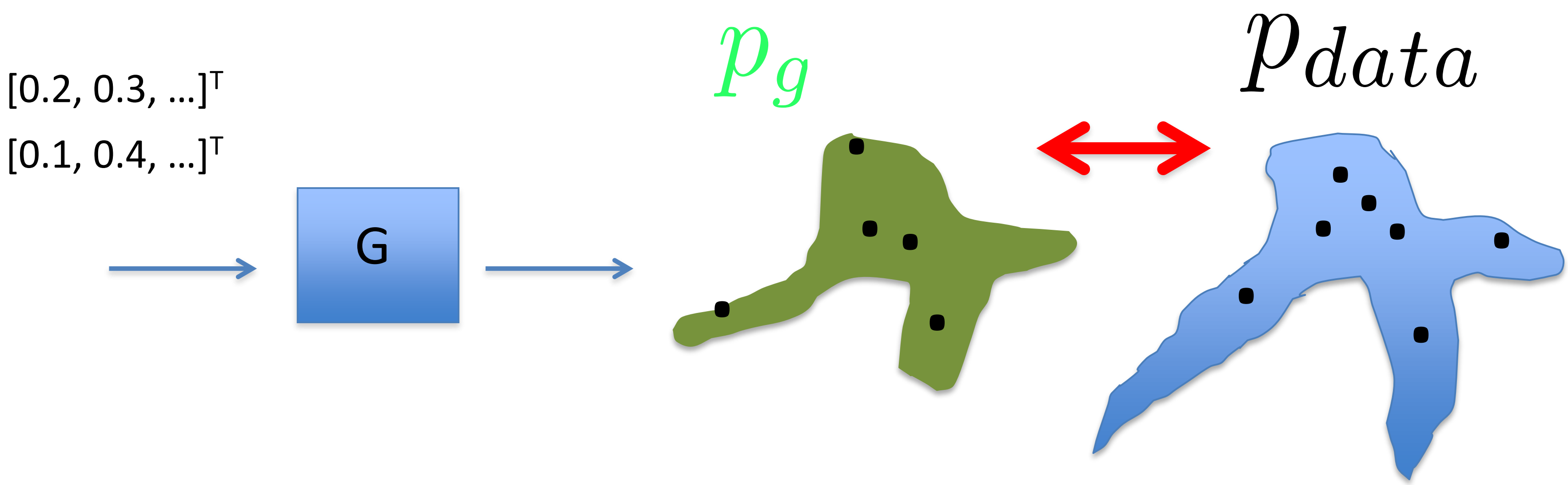


[P. Vincent; 2015]



1-D illustration: images occupy a tiny fraction of the space

Generator network G imposes a probability distribution

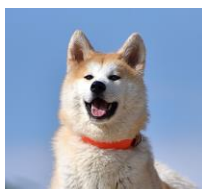


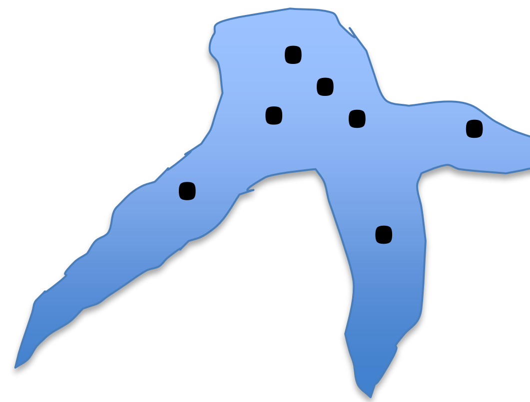
Goal for training G: Minimize the difference between p_g and p_{data}

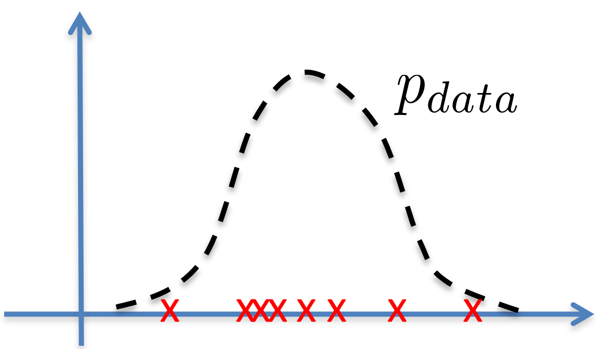
Generative model as probability estimation

Given image samples, estimate the underlying probability distribution

$$p_{data}$$



...





1-D illustration

Generative model as probability estimation

Given image samples, estimate the underlying probability distribution

Maximum likelihood estimation:

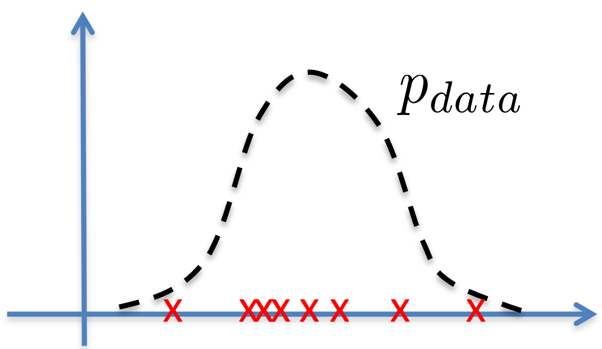
p_{model} as an estimate of the probability distribution, parameterized by some parameters $\boldsymbol{\theta}$

Likelihood: the probability that the model assigns to the observed (training) data

$$\prod_{i=1}^{m} p_{\text{model}}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right)$$

MLE:

$$\begin{aligned}\boldsymbol{\theta}^* &= \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{\text{model}}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right) \\ &= \arg\max_{\boldsymbol{\theta}} \log \prod_{i=1}^{m} p_{\text{model}}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right) \\ &= \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\text{model}}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right)\end{aligned}$$



For images, not easy to define p_{model}

Generative model as probability estimation

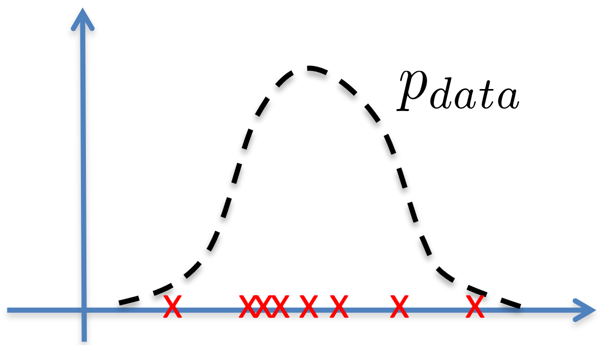
Given image samples, estimate the underlying probability distribution

Maximum likelihood estimation: Explicitly define the model p_{model} parameterized by some $\boldsymbol{\theta}$

GAN: Implicitly define the model, from which samples can be drawn

MLE:

$$\begin{aligned}\boldsymbol{\theta}^* &= \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{\text{model}}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right) \\ &= \arg\max_{\boldsymbol{\theta}} \log \prod_{i=1}^{m} p_{\text{model}}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right) \\ &= \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\text{model}}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right)\end{aligned}$$



For images, not easy to define p_{model}

GAN as JSD minimization

- Recall GAN objective

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

- With optimal discriminator, $V(D^*,G)$ is:

[Goodfellow et al.; 2014]

$$-\log(4) + 2 \cdot JSD\left(p_{\text{data}} \| p_g\right)$$

- Thus, objective for G is to minimize JSD

$$\min_G JSD(p_{data}||p_g)$$

Jensen-Shannon divergence between two distributions

GAN as JSD minimization

★ : data sampled from P_{data}

★ : data sampled from P_G

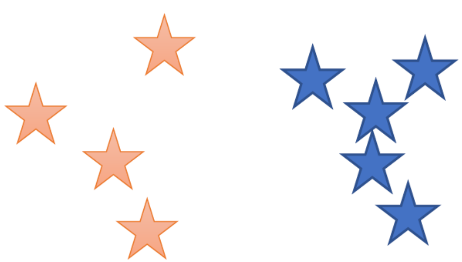
Training:

$$D^* = arg\,\boxed{\max_D V(D,G)}$$

small divergence

train → Discriminator

hard to discriminate
(cannot make objective large)



large divergence

train → Discriminator

easy to discriminate

Divergence

Divergence: measures the difference between probability distributions

Kullback-Leibler divergence (non-symmetric):

$$D_{\mathrm{KL}}(P\|Q) = -\sum_i P(i)\,\log\frac{Q(i)}{P(i)}$$

Jensen-Shannon divergence (symmetric):

$$\mathrm{JSD}(P \parallel Q) = \frac{1}{2}D(P \parallel M) + \frac{1}{2}D(Q \parallel M)$$

where $M = \frac{1}{2}(P + Q)$

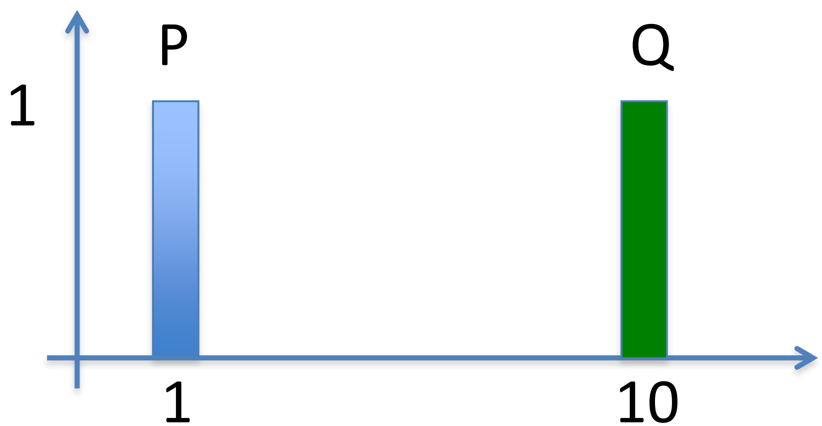
Divergence

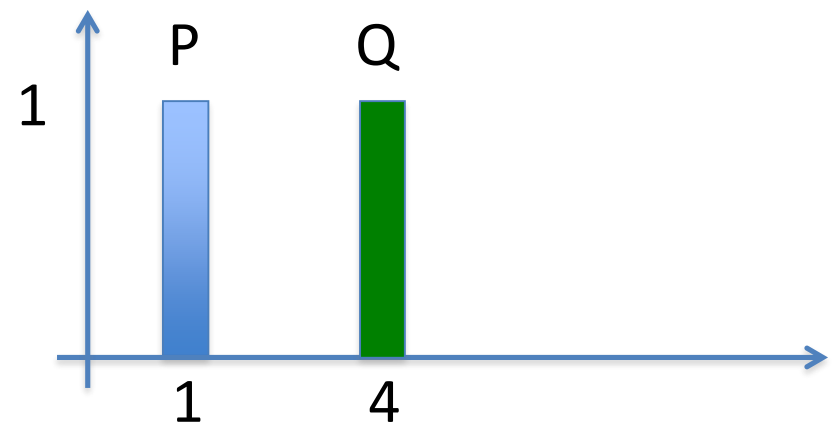
Divergence: measures the difference between probability distributions

Jensen-Shannon divergence (symmetric):

$$\mathrm{JSD}(P \parallel Q) = \frac{1}{2} D(P \parallel M) + \frac{1}{2} D(Q \parallel M)$$

where $M = \frac{1}{2}(P + Q)$





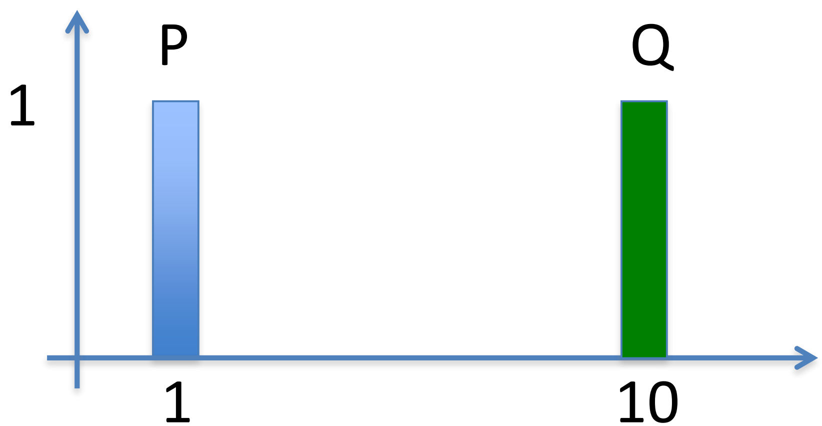
Divergence

Divergence: measures the difference between probability distributions

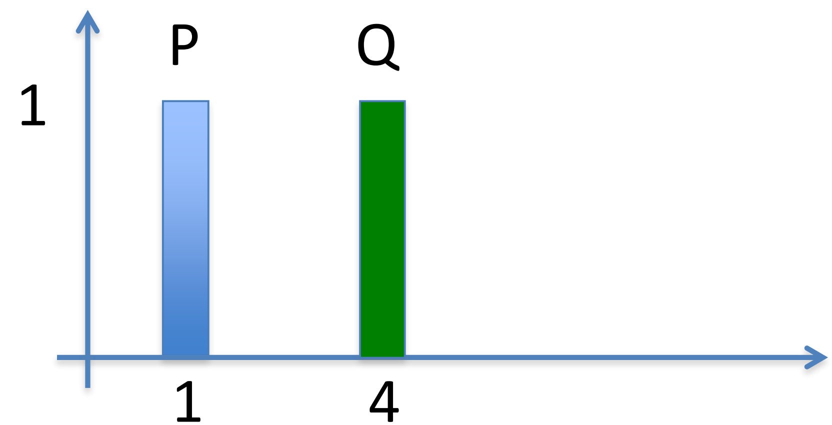
Jensen-Shannon divergence (symmetric):

$$\mathrm{JSD}(P \parallel Q) = \frac{1}{2}D(P \parallel M) + \frac{1}{2}D(Q \parallel M)$$

where $M = \frac{1}{2}(P+Q)$



JSD=ln(2)

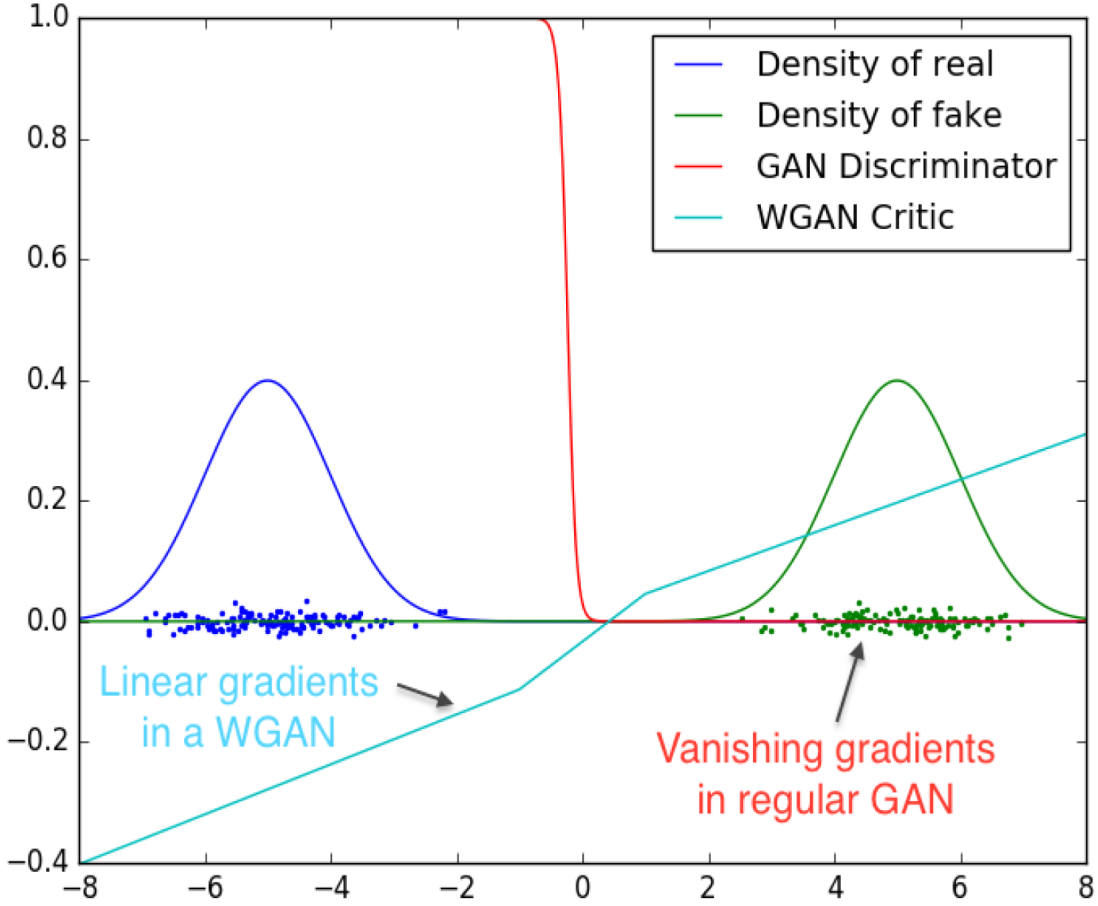


JSD=ln(2)

Recall: $\min_G JSD(p_{data}||p_g)$

Thus, unable to guide G to improve when G is not good (gradient vanishing)

Issue in training GAN: Gradient vanishing



Issue in training GAN: Mode collapse

$G(z)$