SC 4001 CE/CZ 4042 Neural Network and Deep Learning

Last update: 17 March 2024

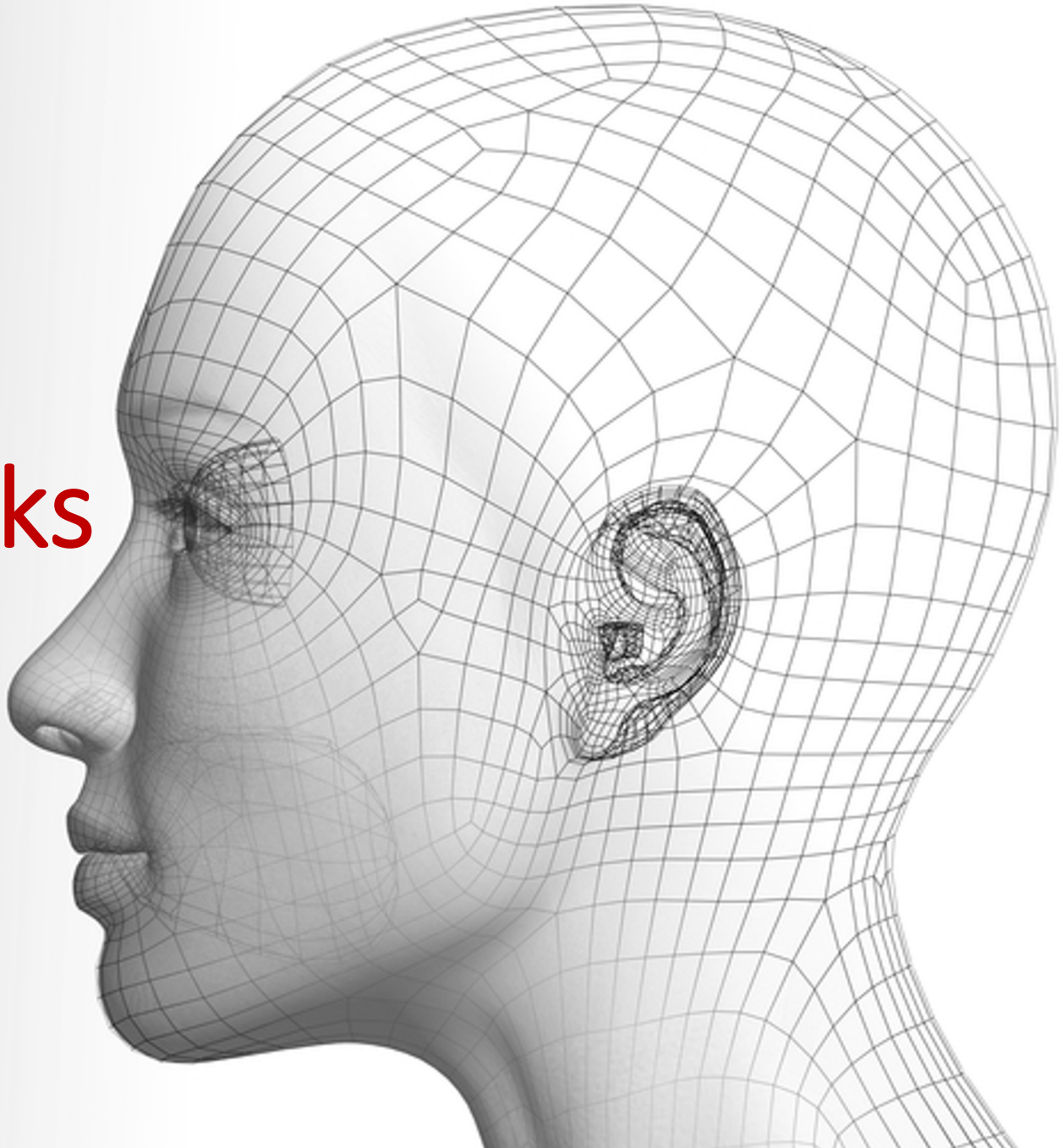
Generative Adversarial Networks

Xingang Pan

潘新钢

https://xingangpan.github.io/

https://twitter.com/XingangP



Outline

- GAN Basics
- GAN Training
- DCGAN
- Mode Collapse

Supervised learning

Data: (x, y)
x is data, y is label

Goal: Learn a *function* to map $x \rightarrow y$

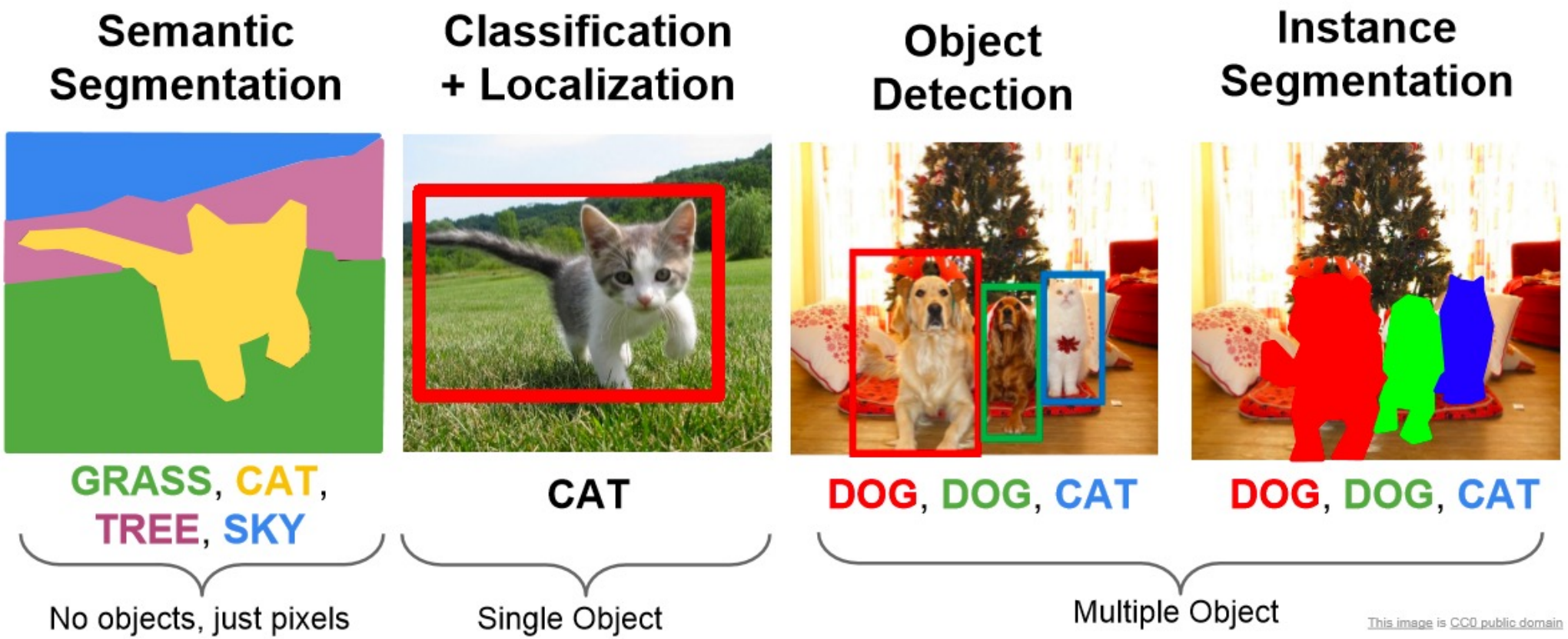
Examples:

Classification,
regression
object detection
semantic segmentation,
image captioning, etc.

Cat

Classification

Applications of supervised learning



Why generative models?

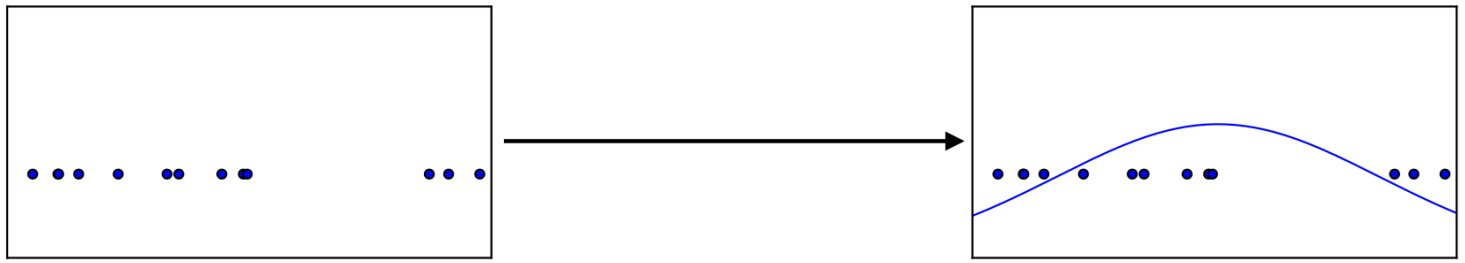
- **We've only seen *discriminative mod*els so far**
  - Given an data $\boldsymbol{x}$, predict a label $\boldsymbol{y}$
  - Estimates $p(\boldsymbol{y}|\boldsymbol{x})$

- **Discriminative models have several key limitations**
  - Can't model $p(\boldsymbol{x})$, i.e., the probability of seeing a certain data
  - Thus, can't sample from $p(\boldsymbol{x})$, i.e., **can't generate new data**

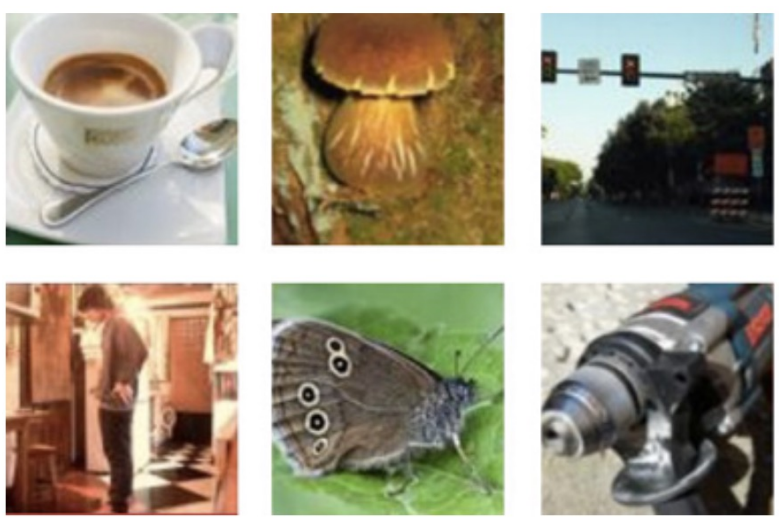
- ***Generative models* (in general) cope with all of above**
  - Can model $p(\boldsymbol{x})$
  - Can generate new data or images

Generative modeling

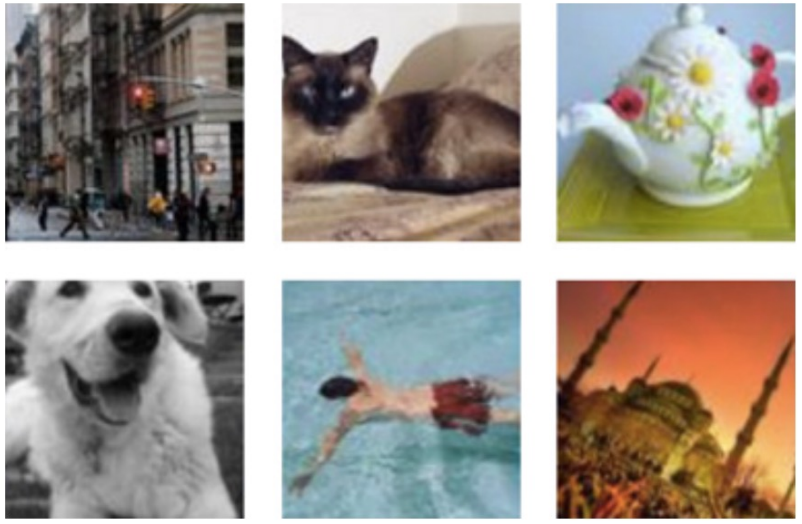
Density estimation – a core problem in unsupervised learning



Sample generation



Training samples – $p_{data}(x)$



Model samples – $p_{model}(x)$
(ideally $p_{model}(x) = p_{data}(x)$)

Image generated - ImageNet

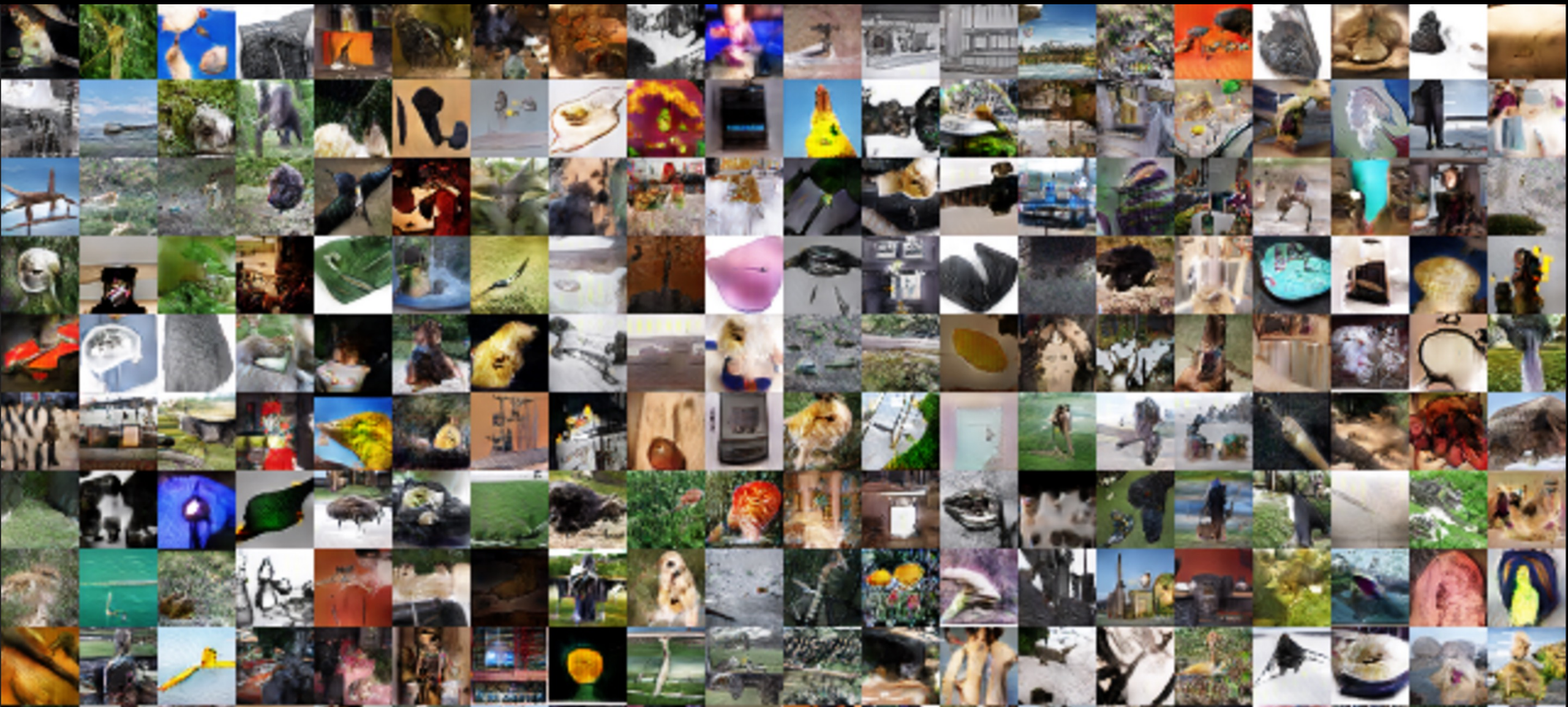
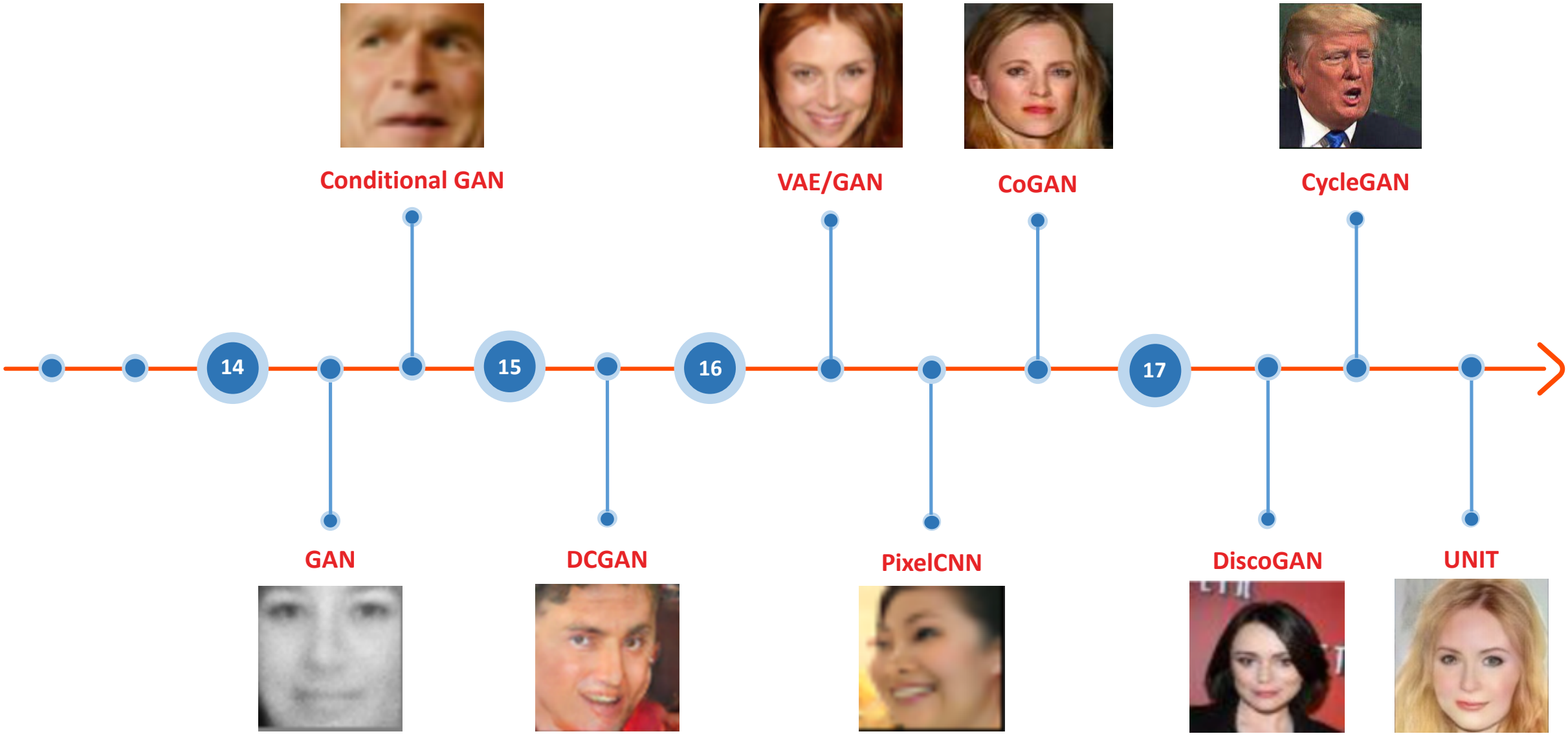


Image generated - Bedrooms

Image generated - Albums

The GAN Revolution 2014-2017



The GAN Revolution 2018 - Present

StarGAN

ELEGANT

PGGAN

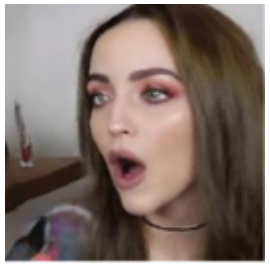
BeautyGlow

StyleGAN

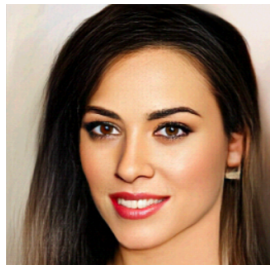
18

19

ReenactGAN



Glow



pix2pixHD

Make a Face

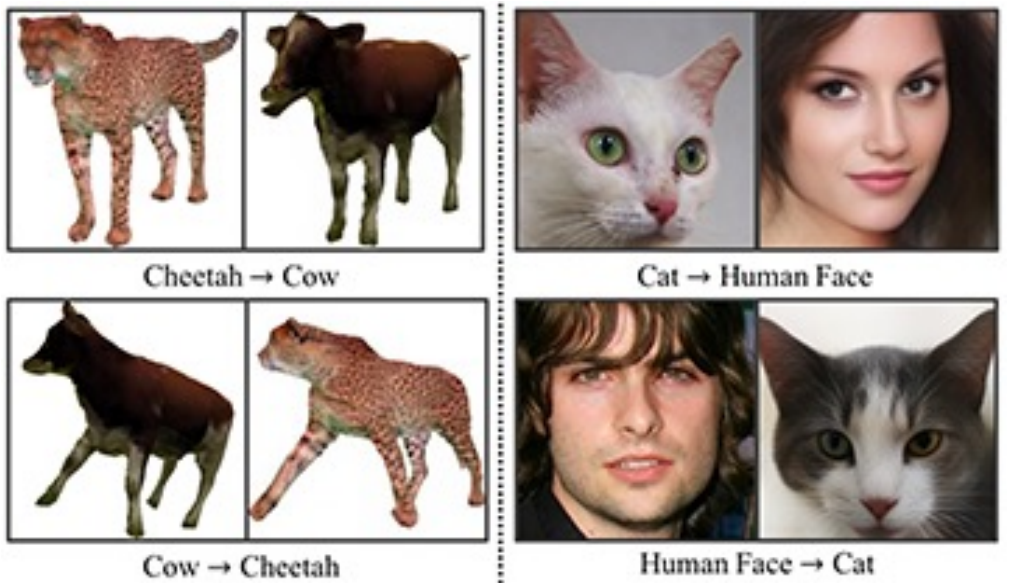
Results of StyleGAN

Credit: Tero Karras et al., A Style-Based Generator Architecture for Generative Adversarial Networks, CVPR 2019

Almost Everything can be Forged!

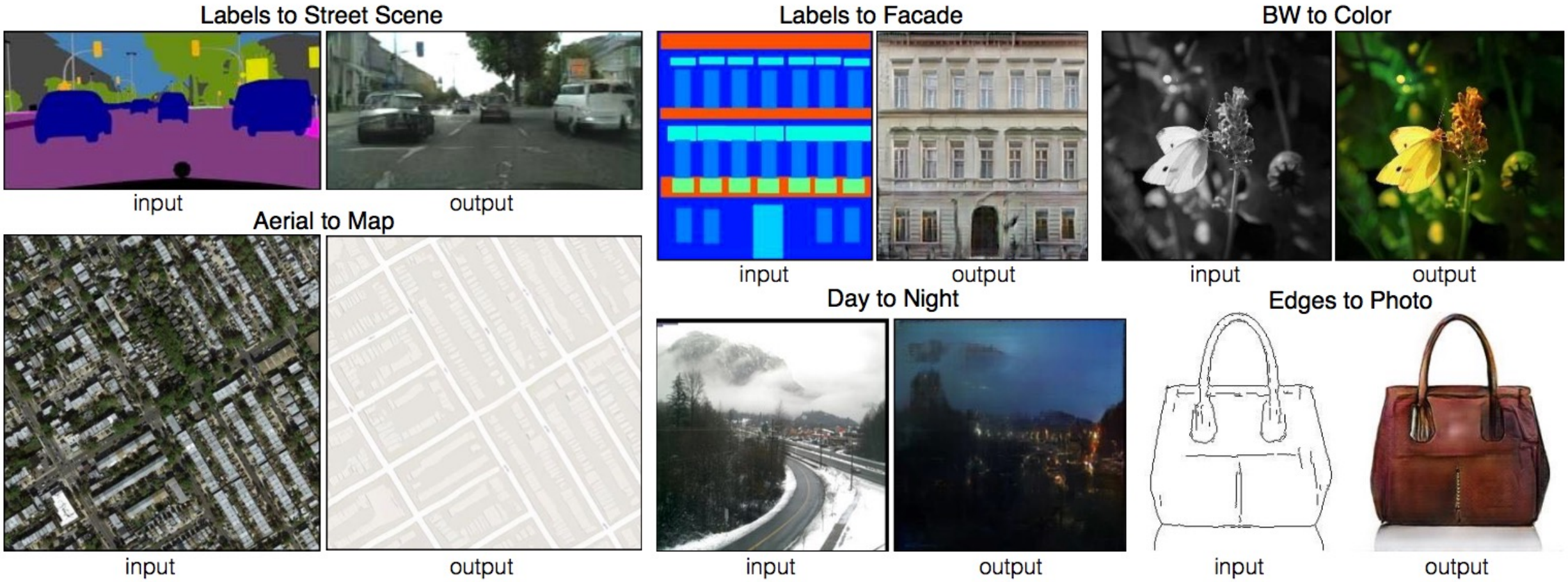
CycleGAN
[Zhu et al., ICCV 2017]

StarGAN
[Choi et al., CVPR 2018]



TransGaGa
[Wu et al., CVPR 2019]

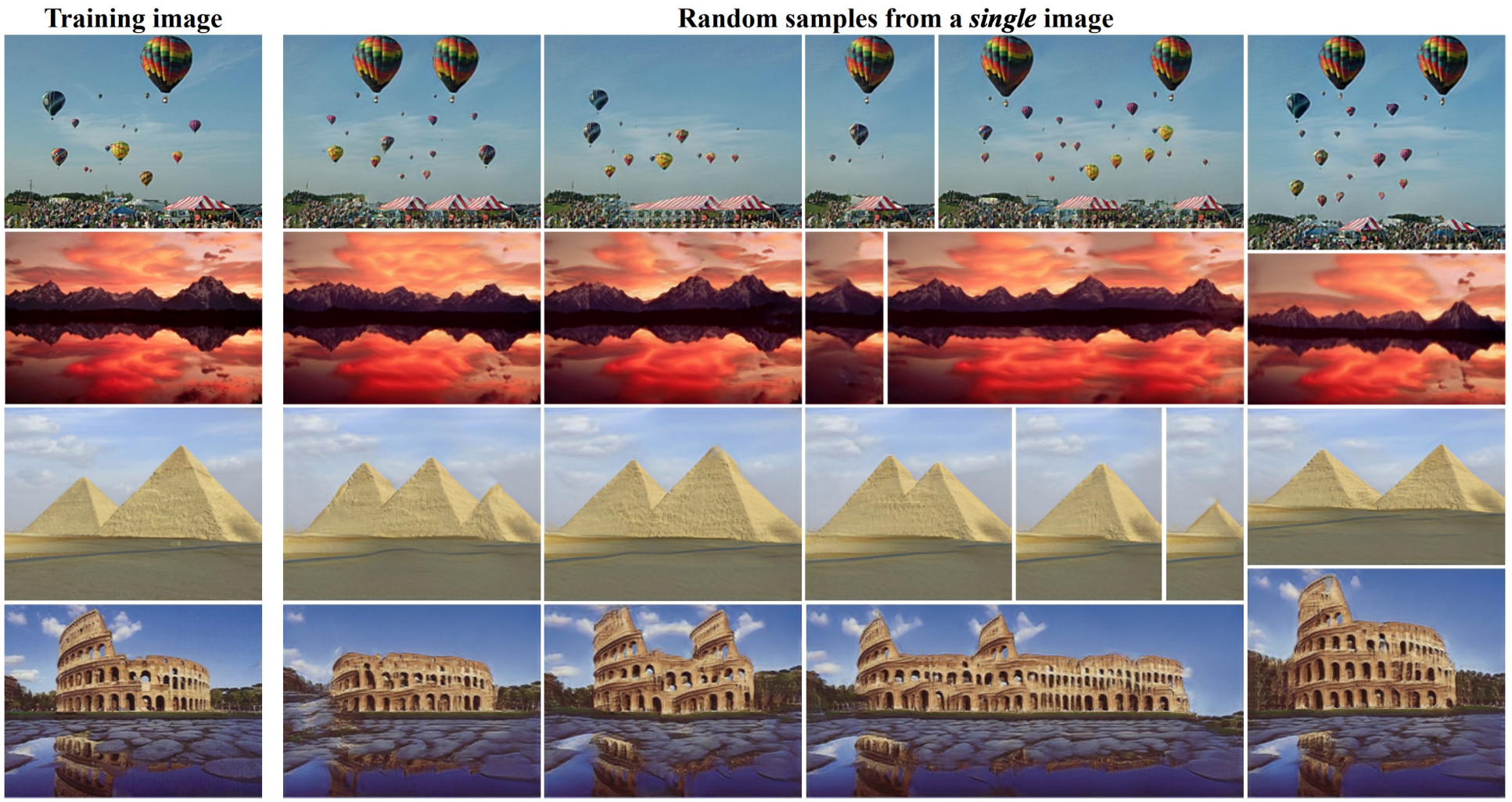
Image-to-Image Translation



Isola et al. “Image-to-Image Translation with Conditional Adversarial Networks”, arXiv:1611.07004

https://phillipi.github.io/pix2pix/

SinGAN: Learning a Generative Model from a Single Natural Image



ICCV 2019 Best Paper

Audio-driven Emotional Face Manipulation

Angry

Contempt

Fear

Input Video

Reference Style

Our Toonification Result

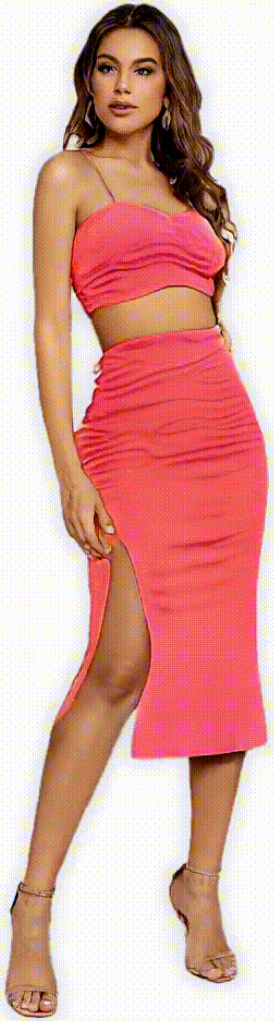
https://huggingface.co/spaces/PKUWilliamYang/VToonify

Edit upper length (StyleSpace)

Edit bottom length (StyleSpace)

Edit upper length (InterFaceGAN)

Edit bottom length (InterFaceGAN)



https://huggingface.co/spaces/hysts/StyleGAN-Human

Outline

- GAN Basics
- GAN Training
- DCGAN
- Mode Collapse

GAN Basics

Generative Adversarial Networks (GAN)

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

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

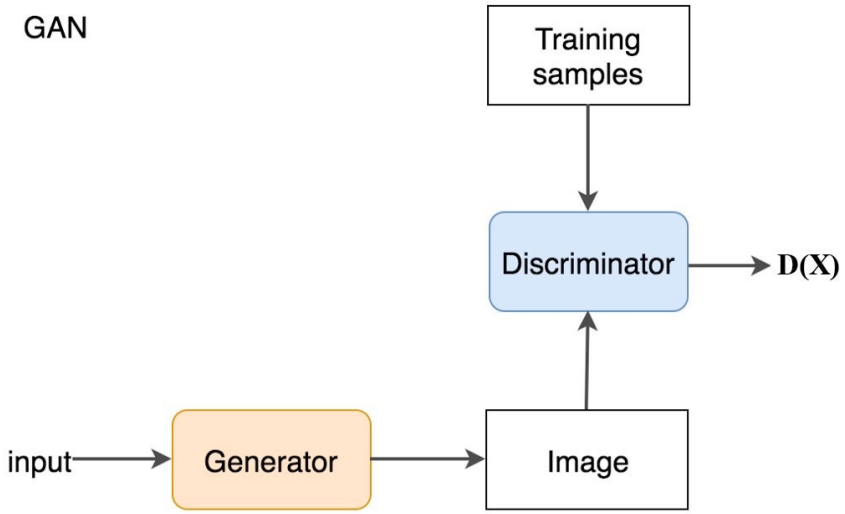
or

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

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

GAN samples noise ***z*** using normal or uniform distribution and utilizes a deep network generator ***G*** to create an image ***x (x=G(z))***

Generative Adversarial Networks (GAN)

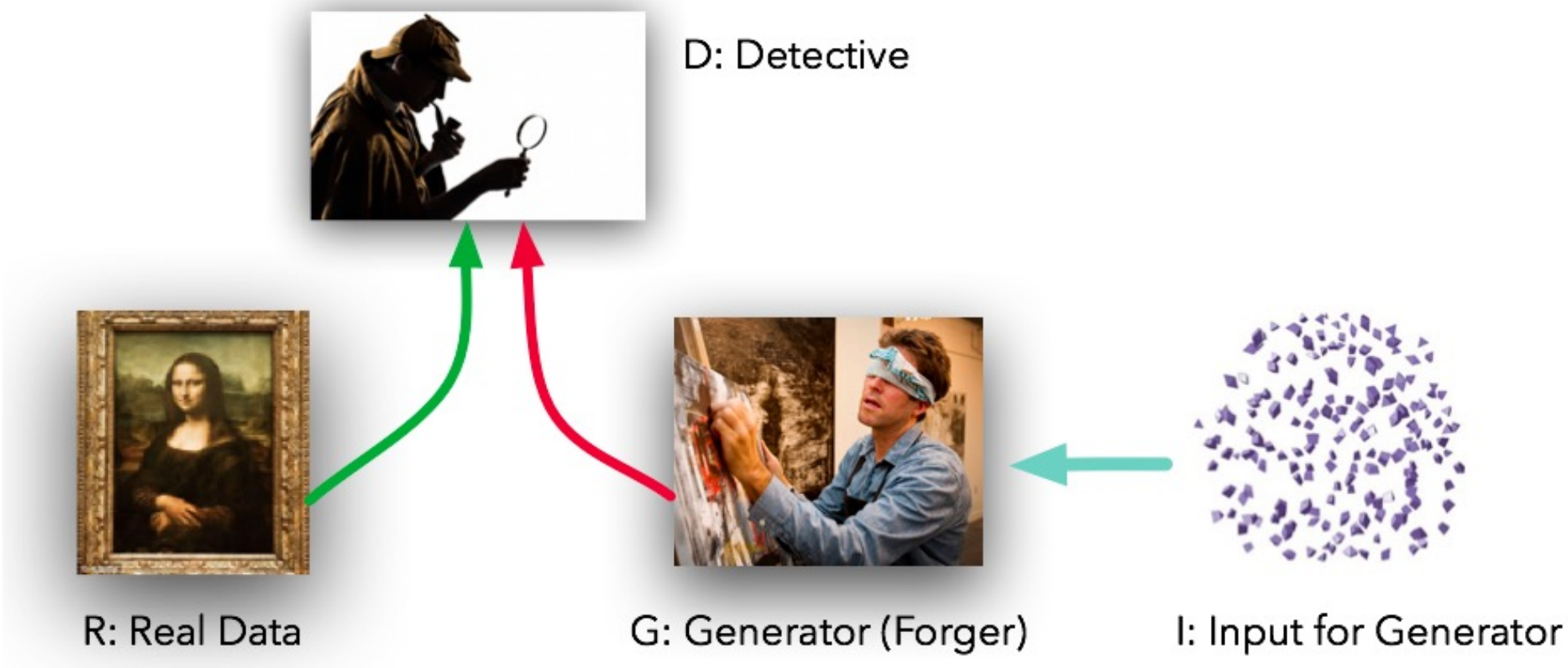


In GAN, we add a discriminator to distinguish whether the discriminator input is real or generated. It outputs a value ***D(x)*** to estimate the chance that the input is real.

D(x) = 1 suggests x is real

D(x) = 0 suggests x is fake

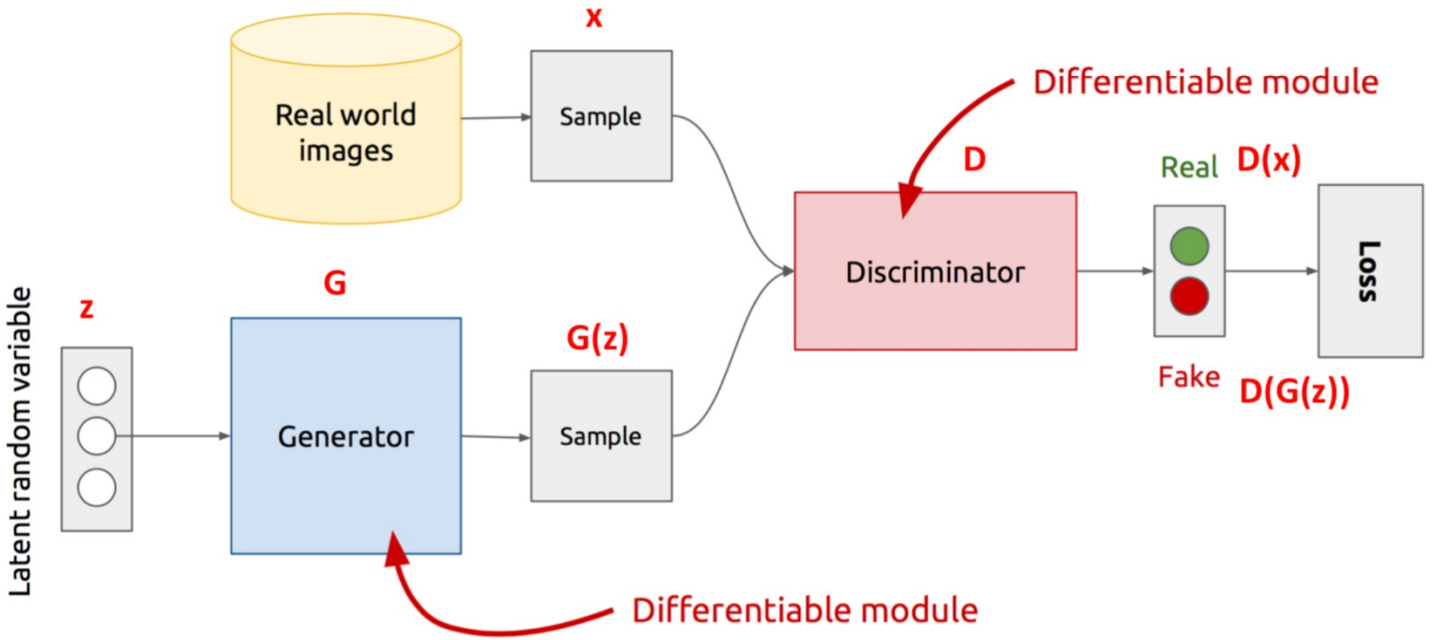
Generative Adversarial Networks (GAN)



Goodfellow et al., “Generative adversarial networks”, NIPS 2014

Generative adversarial networks (GAN)

- Generative model G:
  - Captures data distribution
  - Fool $D(G(z))$
  - Generate an image $G(z)$ such that $D(G(z))$ is wrong (i.e. $D(G(z)) = 1$)
- Discriminative model D:
  - Distinguishes between real and fake samples
  - $D(x) = 1$ when x is a real image, and otherwise



z is some random noise (Gaussian/Uniform).

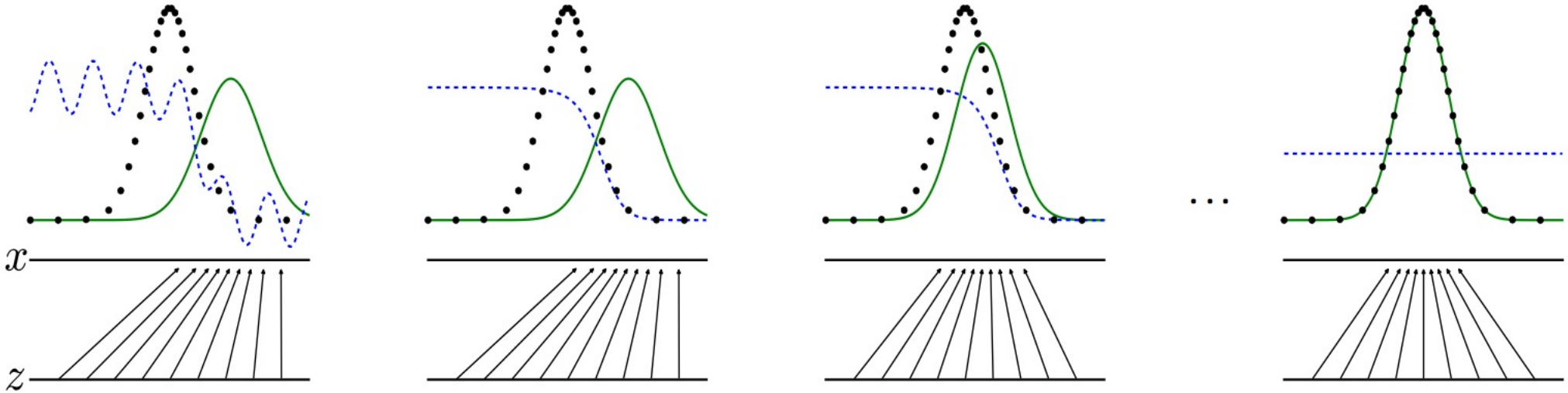
z can be thought as the latent representation of the data.

Generative adversarial networks (GAN)

data generating distribution

generative distribution

discriminative distribution



(a)

An adversarial pair near convergence

(b)

D is trained to discriminate samples from data

(c)

After an update to G, gradient of D has guided G(z) to flow to regions that are more likely to be classified as data

(d)

After several steps of training, G and D will reach a point at which both cannot improve because $p_g = p_{data}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(x) = 0.5$.

Generative adversarial networks (GAN)

- D and G play the following two-player minimax game with value function $V(G, D)$

Generator: input = random vector z
Output: $G(z)$ as synthetic image

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

Discriminator: input = generated/real images
Output: $D(x) = 1$ if real image

Outline

- GAN Basics
- GAN Training
- DCGAN
- Mode Collapse

GAN Training

Training procedure

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: maximize

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

  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: minimize

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

end for

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

D(x) = probability that x is a real image
0 = generated image; 1 = real image

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

end for

Uniform noise vector (random numbers)

Real images

maximize

Generator: input=*random numbers*, output=*synthetic image*

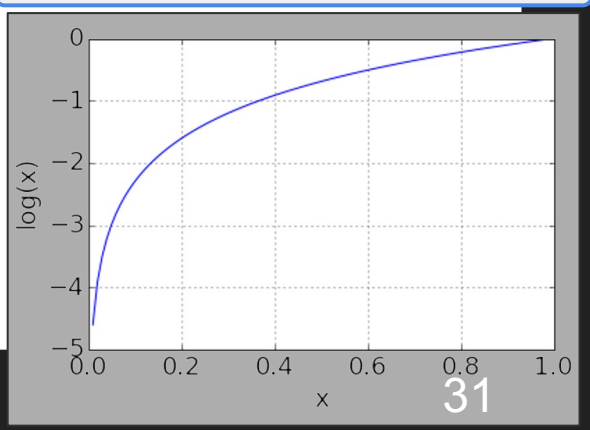
Gradient w.r.t the parameters of the Discriminator

Average over *m* samples

Discriminator: (input=*generated/real image*, output=*prediction of real image*)

Let's do an example

Note showing ln(x) not log(x)



D(*x*) = probability that *x* is a real image
0 = generated image; 1 = real image

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

end for

Uniform noise vector (random numbers)

Real images

maximize

Generator: input=*random numbers*, output=*synthetic image*

Gradient w.r.t the parameters of the Discriminator

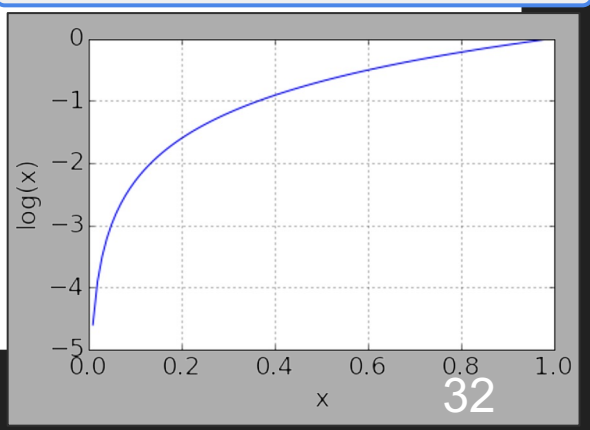
Discriminator: (input=*generated/real image*, output=*prediction of real image*)

Average over *m* samples

D(x) = 0.8
log(0.8) = -0.2

Imagine for a real image D(x) scores 0.8 that it is a real image (good)

Note showing ln(x) not log(x)



D(*x*) = probability that *x* is a real image
0 = generated image; 1 = real image

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

Uniform noise vector (random numbers)

maximize

Real images

Generator: input=*random numbers*, output=*synthetic image*

Gradient w.r.t the parameters of the Discriminator

Average over *m* samples

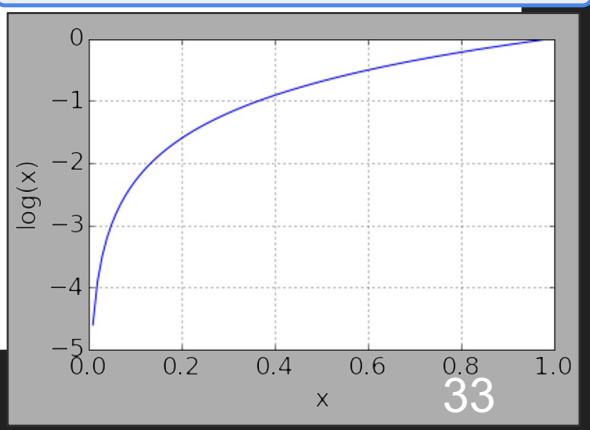
Discriminator: (input=*generated/real image*, output=*prediction of real image*)

D(x) = 0.8
log(0.8) = -0.2

Then for a **generated** image, D(x) scores 0.2 that it is a real image (good)

D(G(z)) = 0.2
log(1-0.2) = log(0.8) = -0.2

Note showing ln(x) not log(x)



D(*x*) = probability that *x* is a real image
0 = generated image; 1 = real image

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

end for

Uniform noise vector (random numbers)

Real images

maximize

Generator: input=*random numbers*, output=*synthetic image*

Gradient w.r.t the parameters of the Discriminator

Discriminator: (input=*generated/real image*, output=*prediction of real image*)

Average over *m* samples

D(x) = 0.8
log(0.8) = -0.2

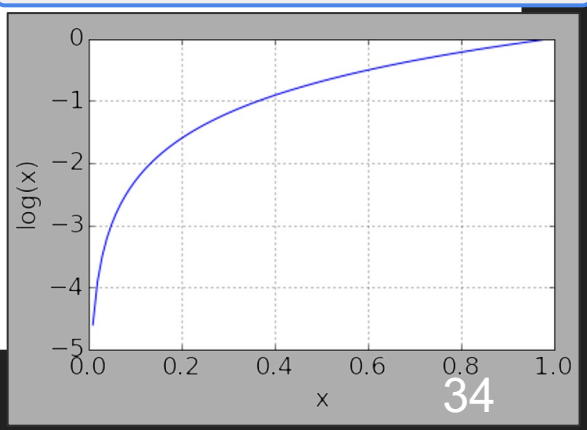
D(G(z)) = 0.2
log(1-0.2) = log(0.8) = -0.2

-0.2 + -0.2 = -0.4

We add them together and this gives us a fairly high (-0.4) loss (note we ascend so we want to maximize this).

Also note that we are adding two negative numbers so 0 is the upper bound

Note showing ln(x) not log(x)



D(*x*) = probability that *x* is a real image
0 = generated image; 1 = real image

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

end for

Uniform noise vector (random numbers)

Real images

maximize

Generator: input=*random numbers*, output=*synthetic image*

Gradient w.r.t the parameters of the Discriminator

Discriminator: (input=*generated/real image*, output=*prediction of real image*)

Average over *m* samples

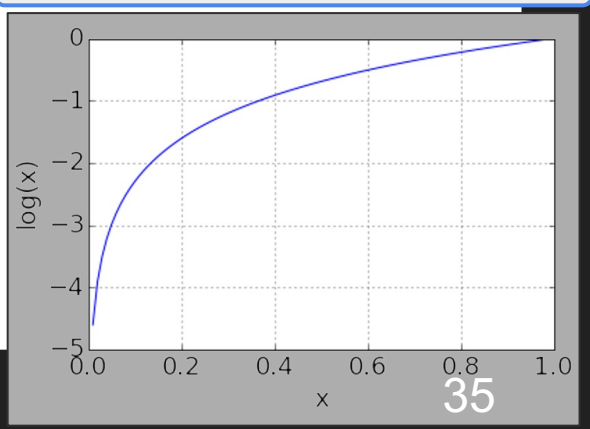
Let's do another example

D(x) = 0.8
log(0.8) = -0.2

D(G(z)) = 0.2
log(1-0.2) = log(0.8) = -0.2

-0.2 + -0.2 = -0.4

Note showing ln(x) not log(x)



D(*x*) = probability that *x* is a real image
0 = generated image; 1 = real image

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

end for

Uniform noise vector (random numbers)

Real images

maximize

Generator: input=*random numbers*, output=*synthetic image*

Gradient w.r.t the parameters of the Discriminator

Average over *m* samples

Discriminator: (input=*generated/real image*, output=*prediction of real image*)

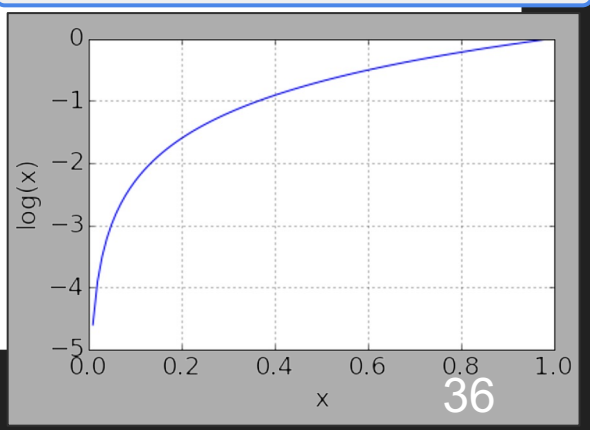
D(x) = 0.8
log(0.8) = -0.2

D(G(z)) = 0.2
log(1-0.2) = log(0.8) = -0.2
-0.2 + -0.2 = -0.4

D(x) = 0.2
log(0.2) = -1.6

D(x) scores 0.2 that a real image is a real image (bad)

Note showing ln(x) not log(x)



D(*x*) = probability that *x* is a real image
0 = generated image; 1 = real image

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

Uniform noise vector (random numbers)

Real images

maximize

Generator: input=*random numbers*, output=*synthetic image*

Gradient w.r.t the parameters of the Discriminator

Average over *m* samples

Discriminator: (input=*generated/real image*, output=*prediction of real image*)

D(x) = 0.8
log(0.8) = -0.2

D(x) = 0.2
log(0.2) = -1.6

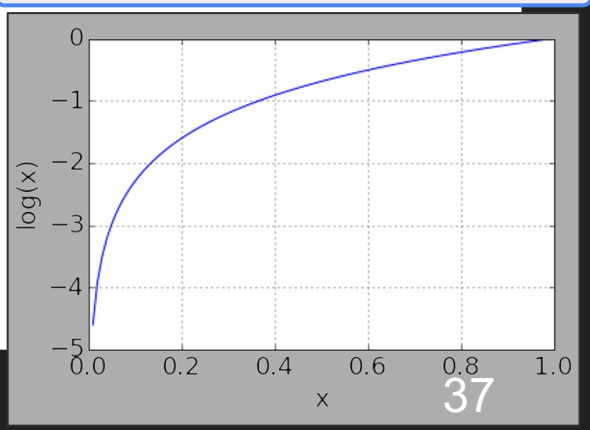
D(x) scores 0.8 that the generated image is a real image (bad)

D(G(z)) = 0.8
log(1-0.8) = log(0.2) = -1.6

-0.2 + -0.2 = -0.4

end for

Note showing ln(x) not log(x)



D(*x*) = probability that *x* is a real image
0 = generated image; 1 = real image

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

end for

Uniform noise vector (random numbers)

Real images

maximize

Generator: input=*random numbers*, output=*synthetic image*

Gradient w.r.t the parameters of the Discriminator

Average over *m* samples

Discriminator: (input=*generated/real image*, output=*prediction of real image*)

D(x) = 0.8
log(0.8) = -0.2

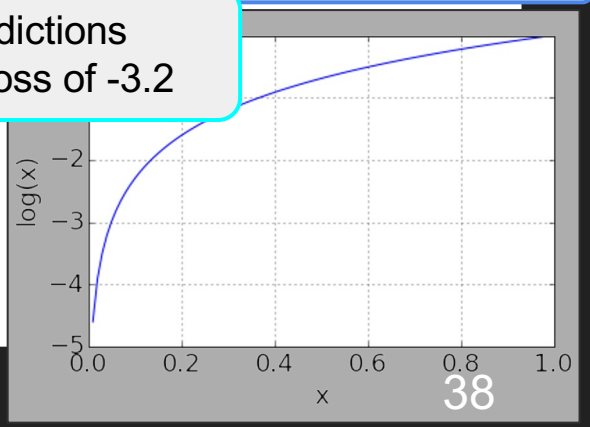
D(G(z)) = 0.2
log(1-0.2) = log(0.8) = -0.2
-0.2 + -0.2 = -0.4

D(x) = 0.2
log(0.2) = -1.6

D(G(z)) = 0.8
log(1-0.8) = log(0.2) = -1.6
-1.6 + -1.6 = -3.2

These "bad" predictions combined gives a loss of -3.2

Note showing ln(x) not log(x)



D(*x*) = probability that *x* is a real image
0 = generated image; 1 = real image

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

Uniform noise vector (random numbers)

Real images

maximize

Generator: input=*random numbers*, output=*synthetic image*

Gradient w.r.t the parameters of the Discriminator

Discriminator: (input=*generated/real image*, output=*prediction of real image*)

Average over *m* samples

D(x) = 0.8
log(0.8) = -0.2

D(G(z)) = 0.2
log(1-0.2) = log(0.8) = -0.2

-0.2 + -0.2 = -0.4

D(x) = 0.2
log(0.2) = -1.6

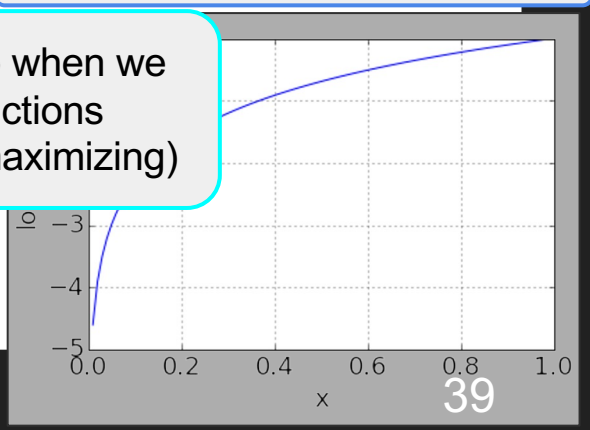
D(G(z)) = 0.8
log(1-0.8) = log(0.2) = -1.6

-1.6 + -1.6 = -3.2

end for

Compare the loss to when we had "good" predictions (remember we are maximizing)

Note showing ln(x) not log(x)



D(x) = probability that x is a real image
0 = generated image; 1 = real image

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:

end for

Uniform noise vector (random numbers)

maximize

Real images

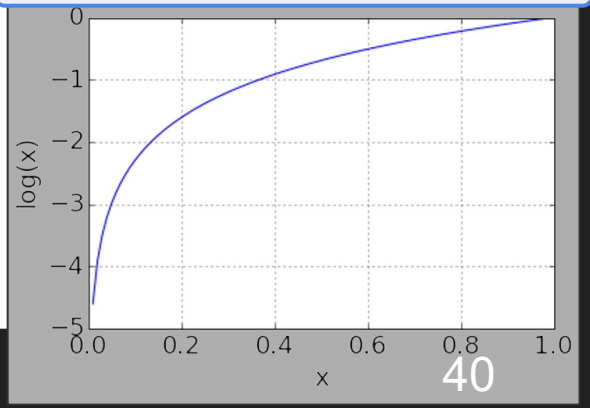
Generator: input=*random numbers*, output=*synthetic image*

Gradient w.r.t the parameters of the Discriminator

Discriminator: (input=*generated/real image*, output=*prediction of real image*)

Average over *m* samples

Note showing ln(x) not log(x)



D(x) = probability that x is a real image
0 = generated image; 1 = real image

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

Uniform noise vector (random numbers)

Real images

maximize

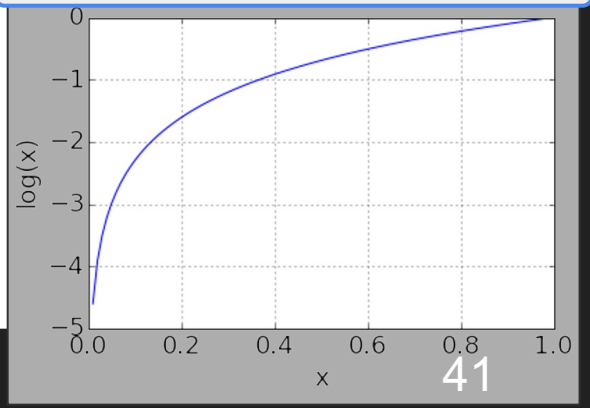
Generator: input=*random numbers*, output=*synthetic image*

Gradient w.r.t the parameters of the Discriminator

Discriminator: (input=*generated/real image*, output=*prediction of real image*)

Average over *m* samples

Note showing ln(x) not log(x)



D(*x*) = probability that *x* is a real image
0 = generated image; 1 = real image

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:

Uniform noise vector (random numbers)

Real images

maximize

Generator: input=*random numbers*, output=*synthetic image*

Gradient w.r.t the parameters of the Discriminator

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

Average over *m* samples

Discriminator: (input=*generated/real image*, output=*prediction of real image*)

  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:

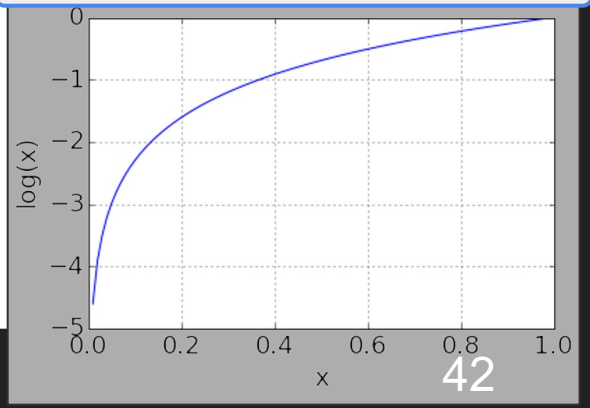
D(G(z)) = 0.2

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

D(G(z)) scores 0.2 that a generated image is a real image = bad :(We didn't fooled D(.)

  end for

Note showing ln(x) not log(x)



D(*x*) = probability that *x* is a real image
0 = generated image; 1 = real image

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:

Uniform noise vector (random numbers)

maximize

Real images

Generator: input=*random numbers*, output=*synthetic image*

Gradient w.r.t the parameters of the Discriminator

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

Discriminator: (input=*generated/real image*, output=*prediction of real image*)

Average over *m* samples

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:

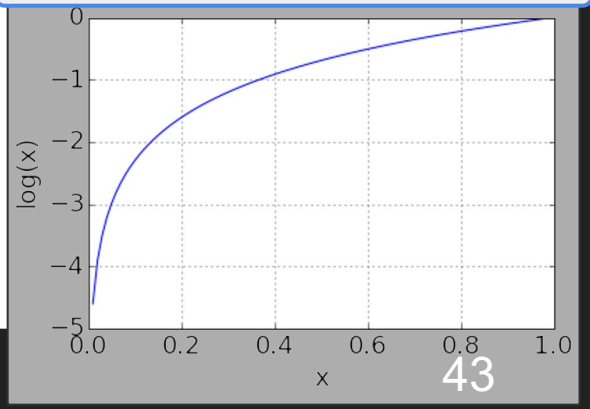
D(G(z)) = 0.2
log(1-0.2) = log(0.8) = -0.2

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

Gets assigned a loss of -0.2

end for

Note showing ln(x) not log(x)



D(*x*) = probability that *x* is a real image
0 = generated image; 1 = real image

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:

Uniform noise vector (random numbers)

Real images

maximize

Generator: input=*random numbers*, output=*synthetic image*

Gradient w.r.t the parameters of the Discriminator

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

Average over *m* samples

Discriminator: (input=*generated/real image*, output=*prediction of real image*)

  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:

minimize

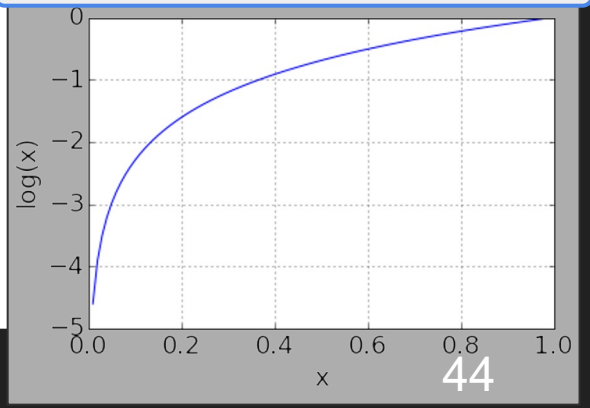
D(G(z)) = 0.2
log(1-0.2) = log(0.8) = -0.2

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

Notice here we want to minimize this loss function (not maximize like before)

  end for

Note showing ln(x) not log(x)



D(*x*) = probability that *x* is a real image
0 = generated image; 1 = real image

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:

Uniform noise vector (random numbers)

maximize

Real images

Generator: input=*random numbers*, output=*synthetic image*

Gradient w.r.t the parameters of the Discriminator

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

Discriminator: (input=*generated/real image*, output=*prediction of real image*)

Average over *m* samples

  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:

D(G(z)) = 0.2
log(1-0.2) = log(0.8) = -0.2

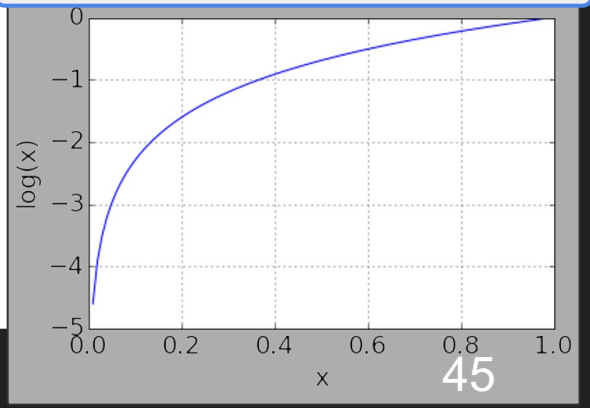
minimize

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

One more example ...

  end for

Note showing ln(x) not log(x)



D(*x*) = probability that *x* is a real image
0 = generated image; 1 = real image

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

Uniform noise vector (random numbers)

Real images

maximize

Generator: input=*random numbers*, output=*synthetic image*

Gradient w.r.t the parameters of the Discriminator

Discriminator: (input=*generated/real image*, output=*prediction of real image*)

Average over *m* samples

minimize

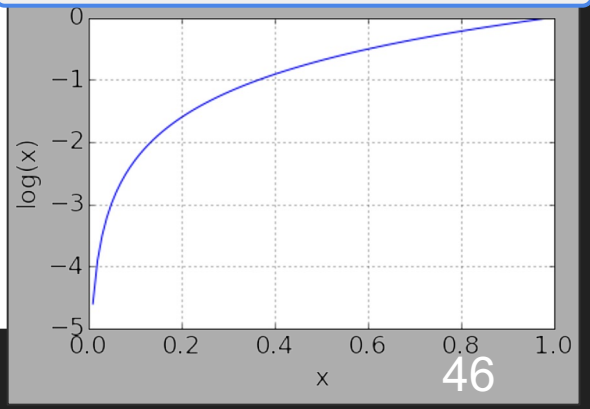
D(G(z)) = 0.2

log(1-0.2) = log(0.8) = -0.2

D(G(z)) = 0.8

D(G(z)) scores 0.8 that a generated image is a real image = good :) We fooled D(.)!

Note showing ln(x) not log(x)



D(*x*) = probability that *x* is a real image
0 = generated image; 1 = real image

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:

Uniform noise vector (random numbers)

Real images

maximize

Generator: input=*random numbers*, output=*synthetic image*

Gradient w.r.t the parameters of the Discriminator

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

Average over *m* samples

Discriminator: (input=*generated/real image*, output=*prediction of real image*)

  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:

minimize

D(G(z)) = 0.2
log(1-0.2) = log(0.8) = -0.2

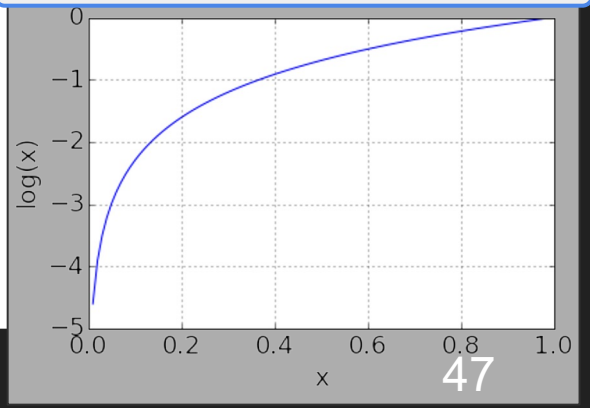
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

D(G(z)) = 0.8
log(1-0.8) = log(0.2) = -1.6

This gives loss of -1.6

  end for

Note showing ln(x) not log(x)



D(*x*) = probability that *x* is a real image
0 = generated image; 1 = real image

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:

Uniform noise vector (random numbers)

Real images

maximize

Generator: input=*random numbers*, output=*synthetic image*

Gradient w.r.t the parameters of the Discriminator

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

Average over *m* samples

Discriminator: (input=*generated/real image*, output=*prediction of real image*)

  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:

minimize

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

D(G(z)) = 0.2

log(1-0.2) = log(0.8) = -0.2

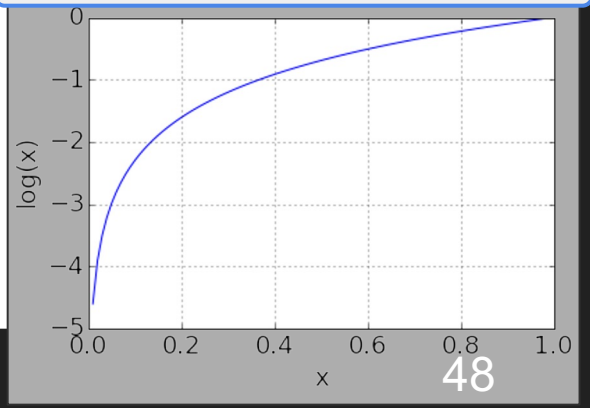
D(G(z)) = 0.8

log(1-0.8) = log(0.2) = -1.6

So minimizing this loss means to generate images that fool D(.)

  end for

Note showing ln(x) not log(x)



D(x) = probability that x is a real image
0 = generated image; 1 = real image

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:

Uniform noise vector (random numbers)

Real images

maximize

Generator: input=*random numbers*, output=*synthetic image*

Gradient w.r.t the parameters of the Discriminator

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

Average over *m* samples

Discriminator: (input=*generated/real image*, output=*prediction of real image*)

  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:

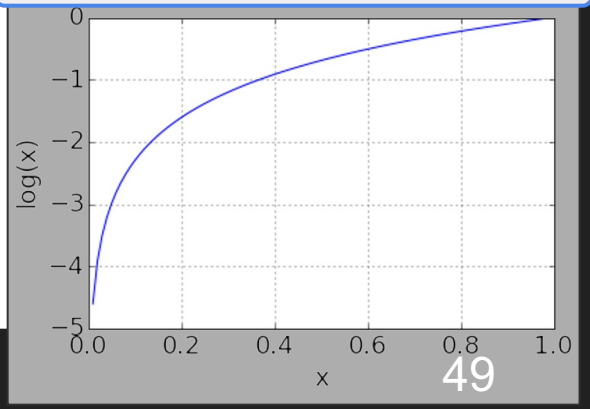
minimize

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

Gradient w.r.t the parameters of the Generator

end for

Note showing ln(x) not log(x)

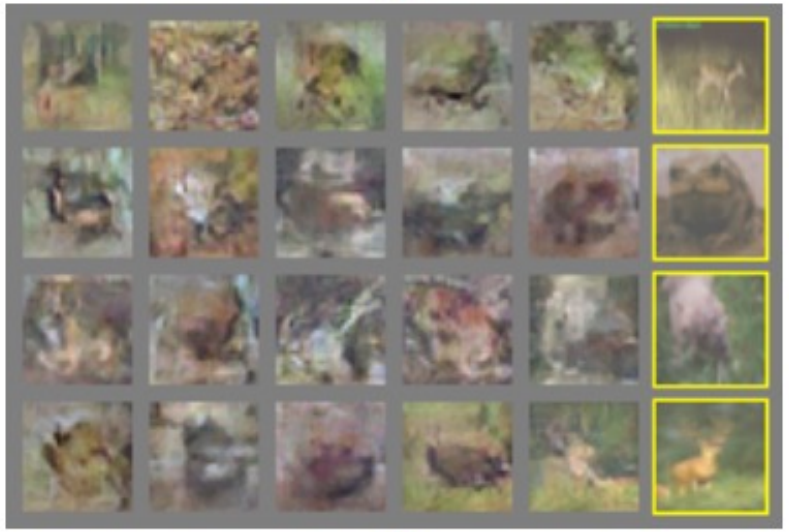


Results from GAN

a)

b)

c)



d)

Training a system for rendering arbitrary images (arbitrary within the bounds of the subject matter that we trained the system on)

Example 1

Design a GAN to learn Gaussian distributed data with mean = 1.0 and standard deviation = 1.0.

The generator receives uniformly distributed 1-dimensional inputs in the range [-1, +1].

Discriminator is a 4-layer DNN:

- Input dimension = 1
- Number of hidden neurons in hidden-layer 1 = 10 (activation: Tanh)
- Number of hidden neurons in hidden-layer 2 = 10 (activation: Tanh)
- Output dimension = 1 (binary classification)

Generator is a 4-layer DNN:

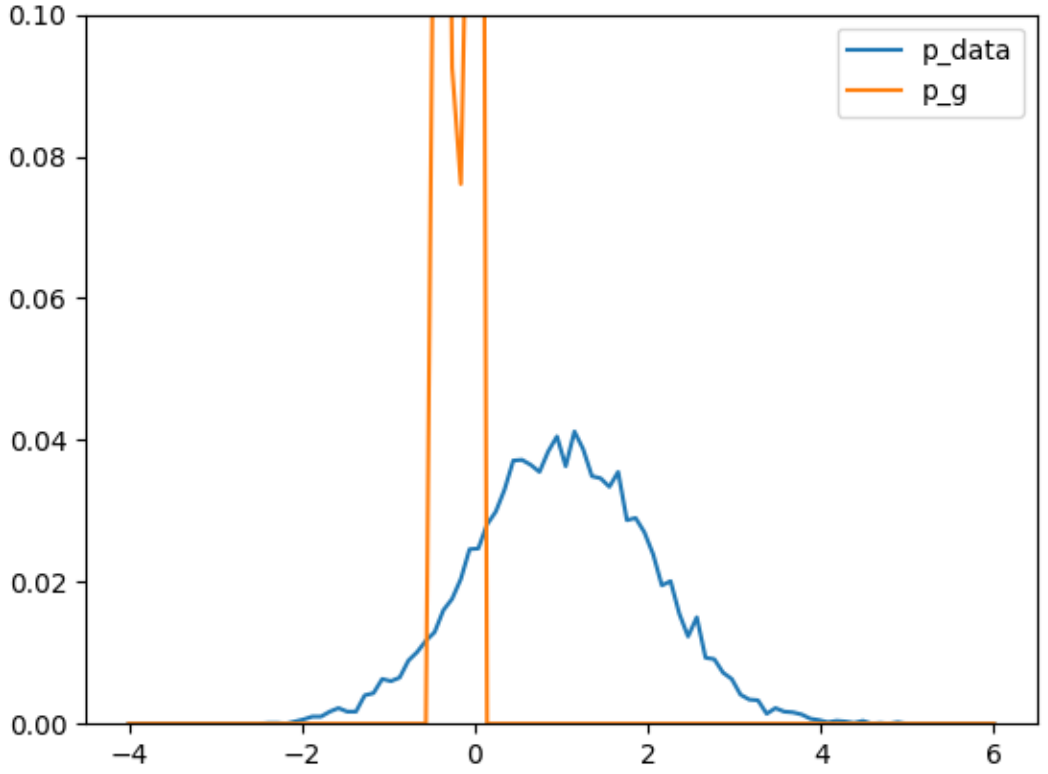
- Input dimension = 1
- Number of hidden neurons in hidden-layer 1 = 5 (activation: Tanh)
- Number of hidden neurons in hidden-layer 2 = 5 (activation: Tanh)
- Output dimension = 1 (activation: None)

eg11.1.ipynb

Example 1

Histogram of 10000 randomly drawn points from U[-1, 1]

Before training

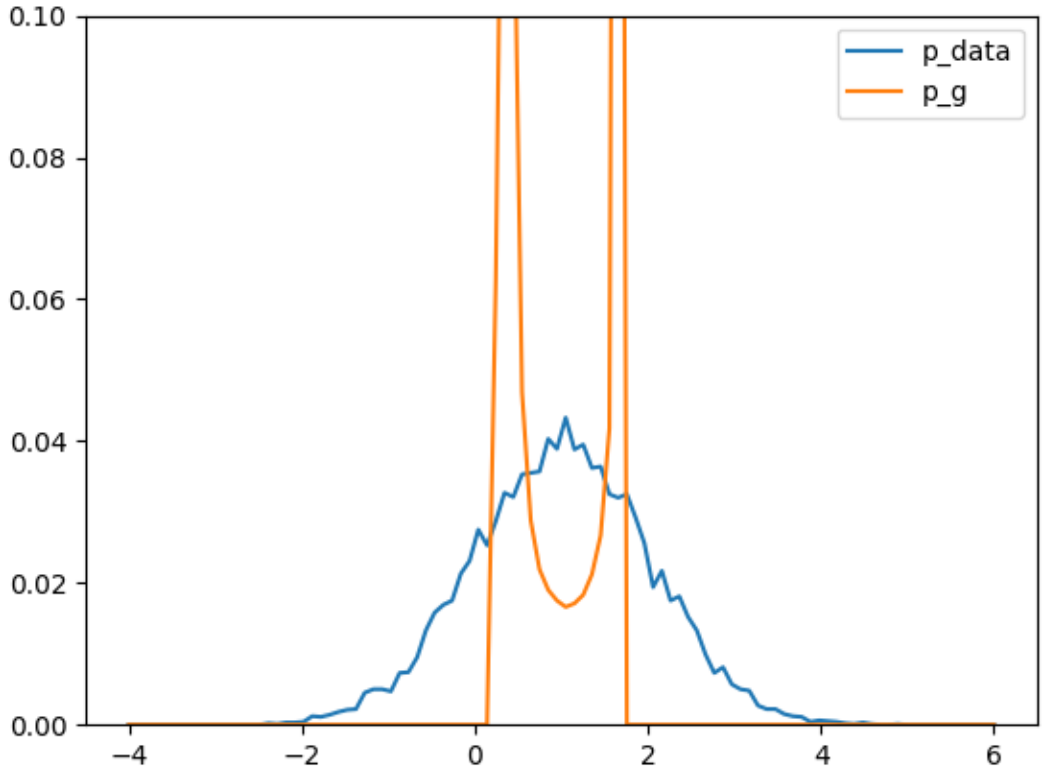


Centered at 0

Example 1

Histogram of 10000 randomly drawn points from U[-1, 1]

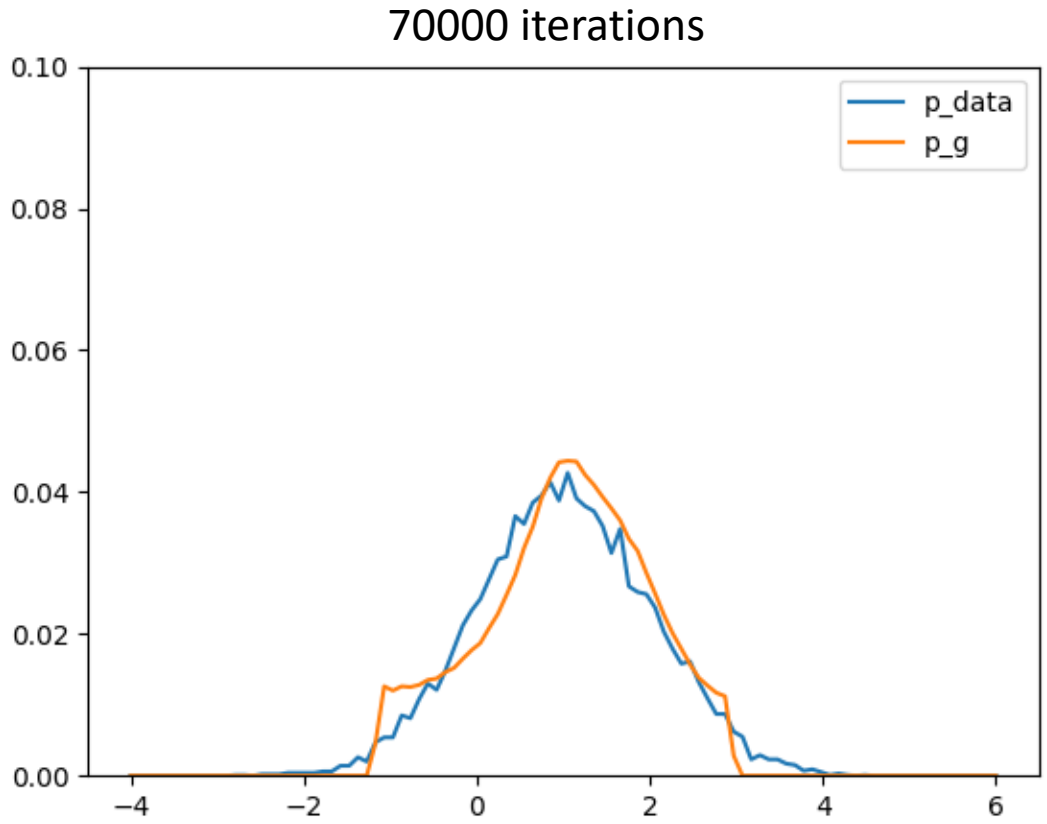
15000 iterations



Symmetric about 1, but not same distribution

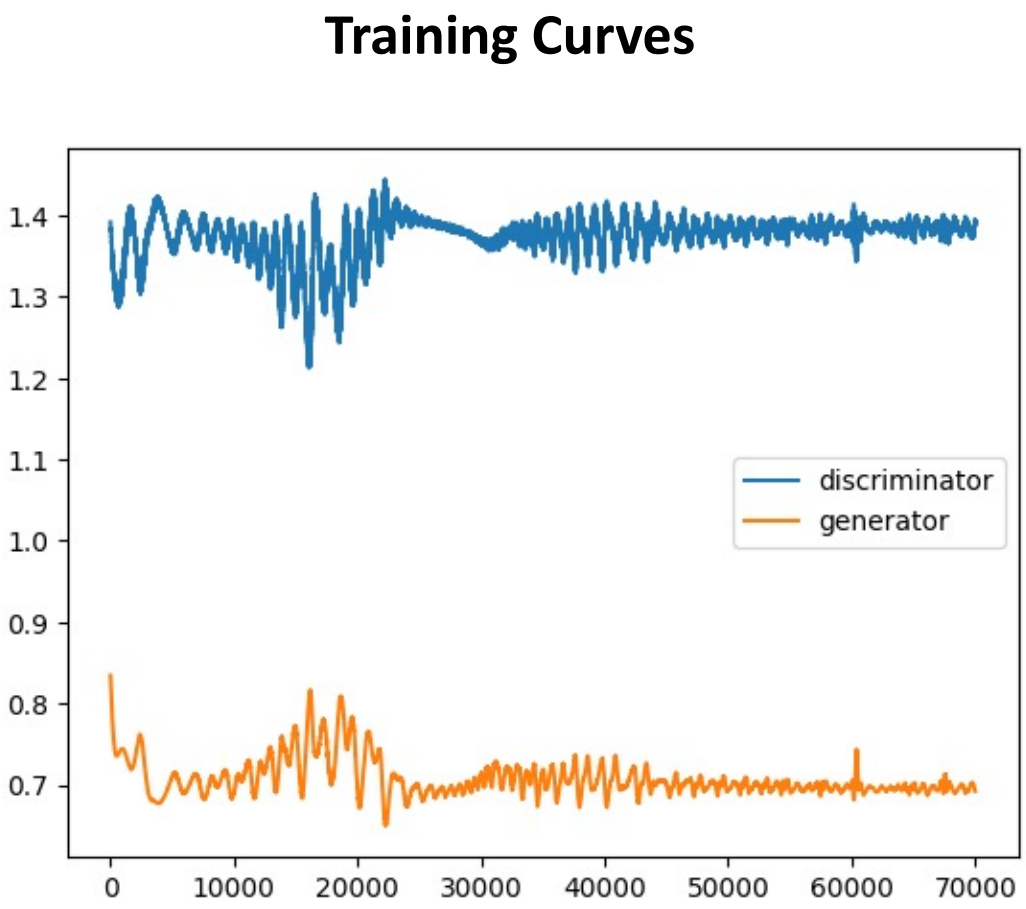
Example 1

Histogram of 10000 randomly drawn points from U[-1, 1]



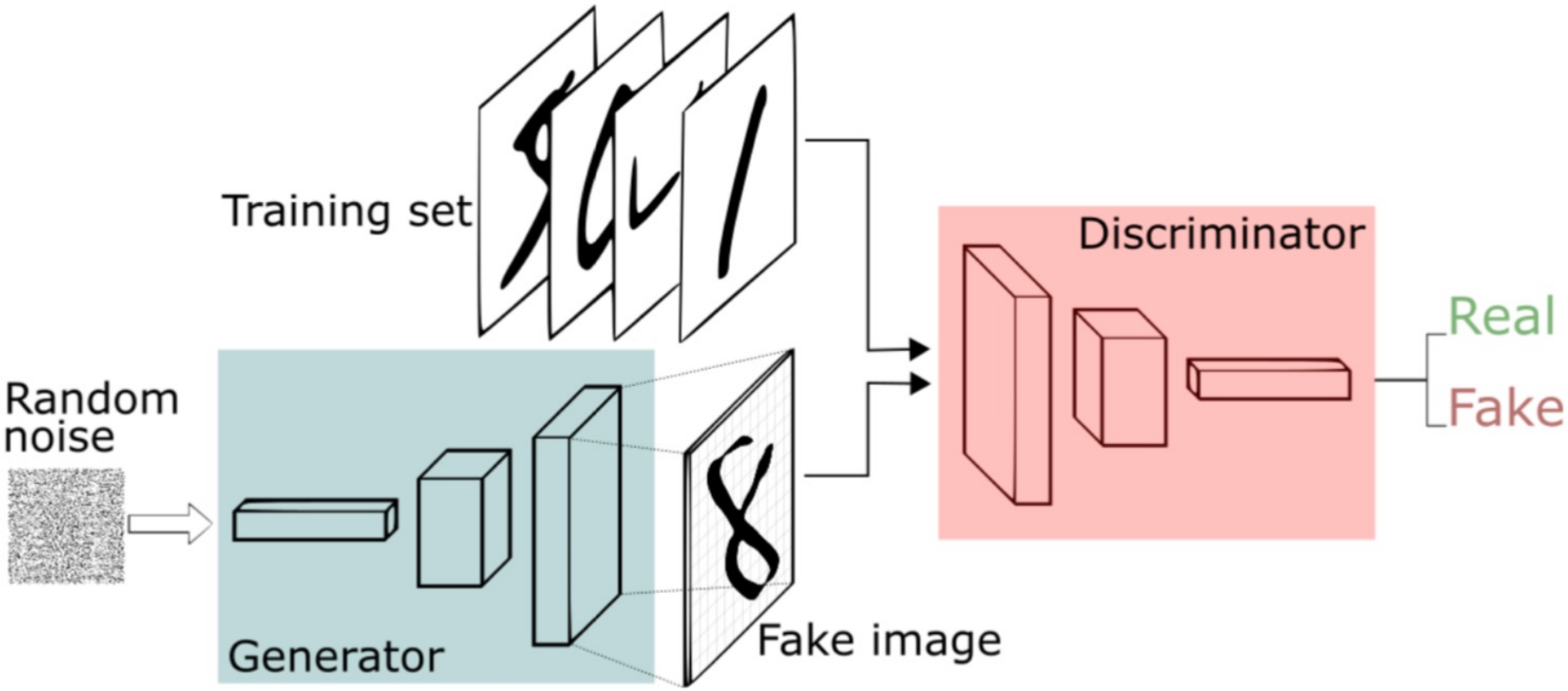
Not perfect, but somehow similar to the data distribution

Example 1


Training Curves
1.4
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0
10000
20000
30000
40000
50000
60000
70000
discriminator
generator

Example 2

Design a GAN to generate MNIST images from a uniformly distributed noise vector of 100 dimensions.



Example 2

Generator:

- Input dimension = 100, drawn from a uniform distribution U(-1, 1)
- Hidden layers
  - Number of hidden neurons = 256 (activation: ReLU)
  - Number of hidden neurons = 512 (activation: ReLU)
  - Number of hidden neurons = 1024 (activation: ReLU)
- Output dimension = 784 (activation: Tanh)

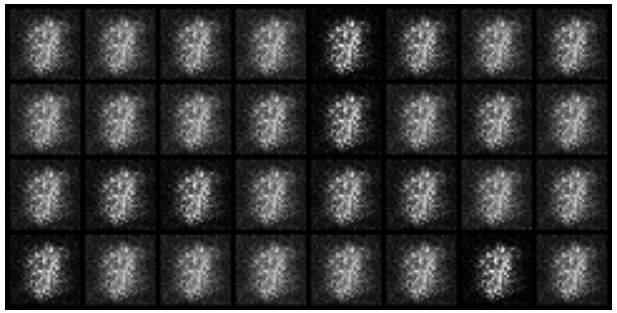
Discriminator:

- Input dimension = 784
- Hidden layers
  - Number of hidden neurons = 1024 (activation: ReLU)
  - Number of hidden neurons = 512 (activation: ReLU)
  - Number of hidden neurons = 256 (activation: ReLU)
- Output dimension = 1 (activation: Sigmoid, binary classification)

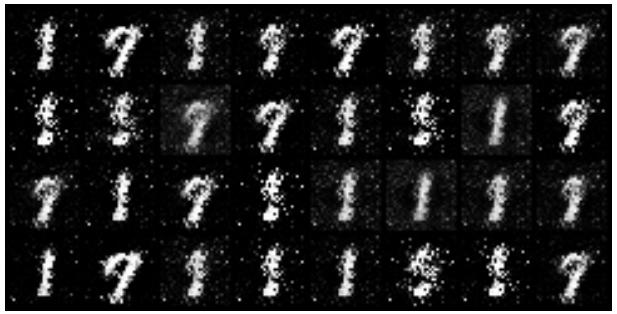
eg11.2.ipynb

Example 2

Start of training



10th epoch



20th epoch

30th epoch

40th epoch

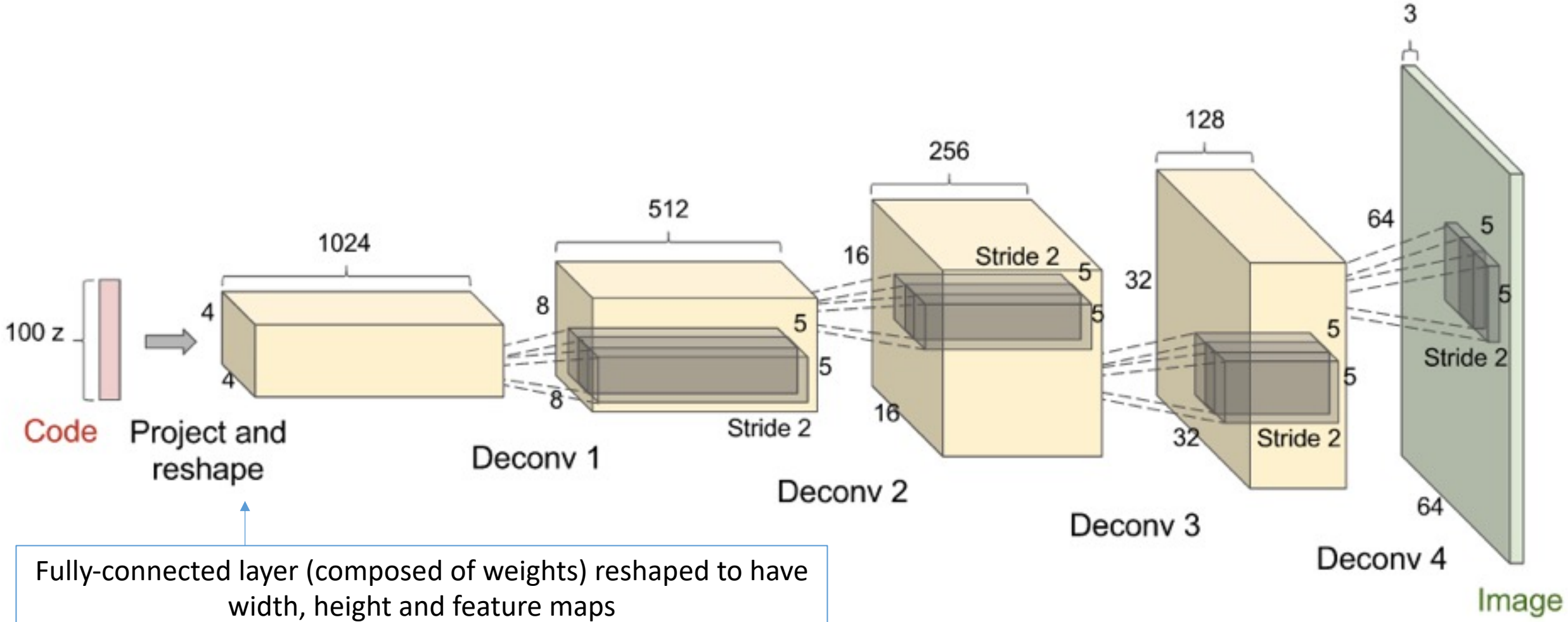
50th epoch

Outline

- GAN Basics
- GAN Training
- DCGAN
- Mode Collapse

DCGAN

Deep Convolutional GAN (DCGAN)

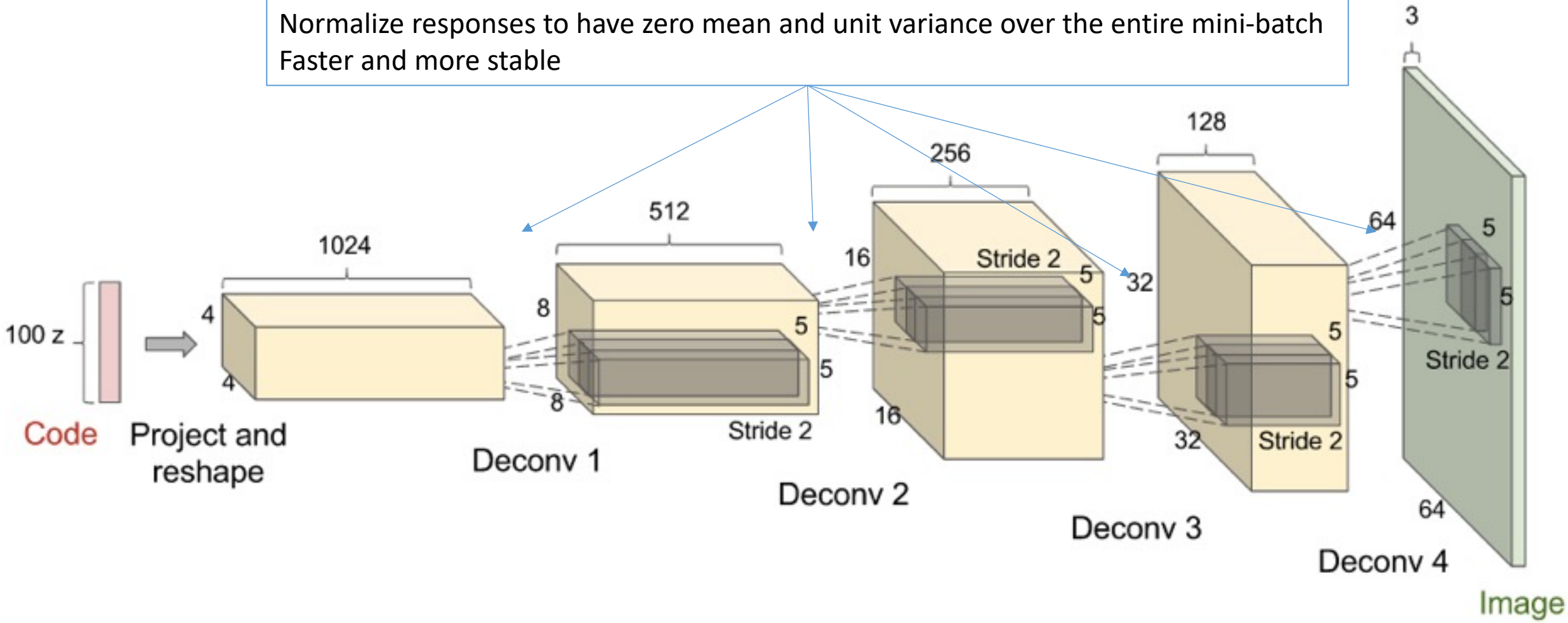


Fully-connected layer (composed of weights) reshaped to have width, height and feature maps

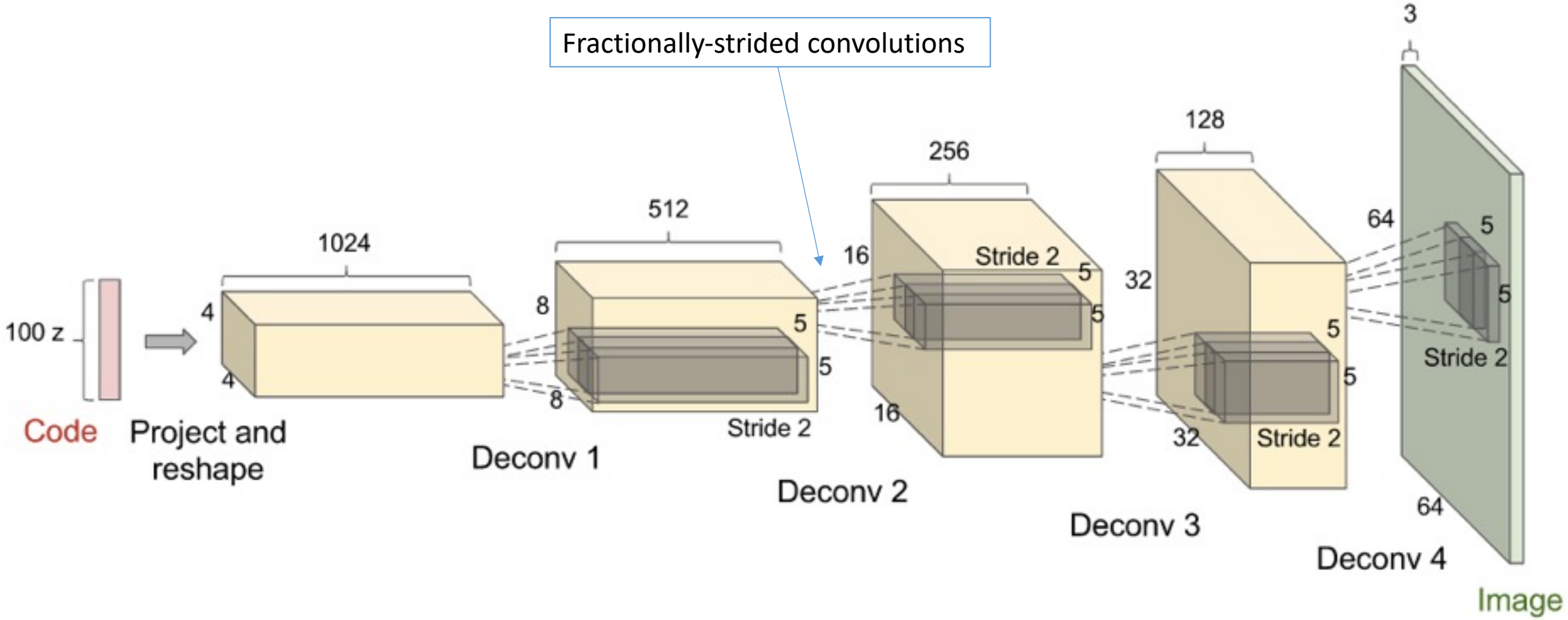
We will go through the code in the tutorial

Deep Convolutional GAN (DCGAN)

Most “deconvs” are batch normalized:
Normalize responses to have zero mean and unit variance over the entire mini-batch
Faster and more stable

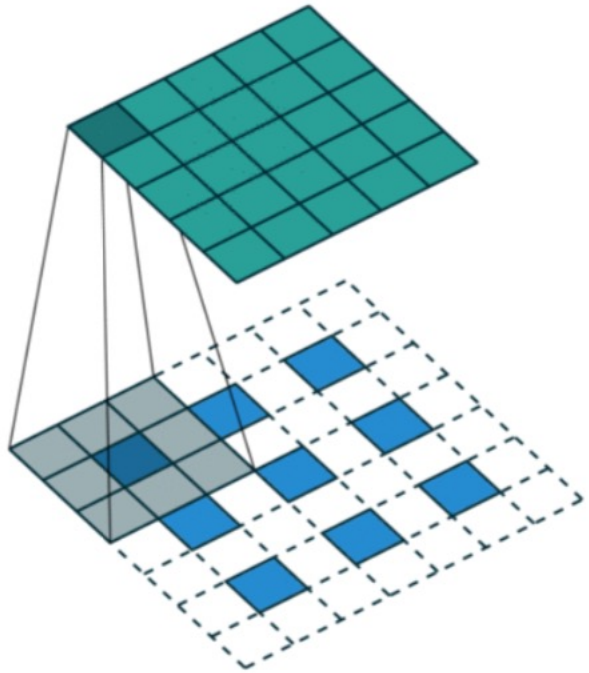


Deep Convolutional GAN (DCGAN)



Fractionally-strided convolutions

Fractional strides involve: zeros are inserted *between* input units, which makes the kernel move around at a slower pace than with unit strides



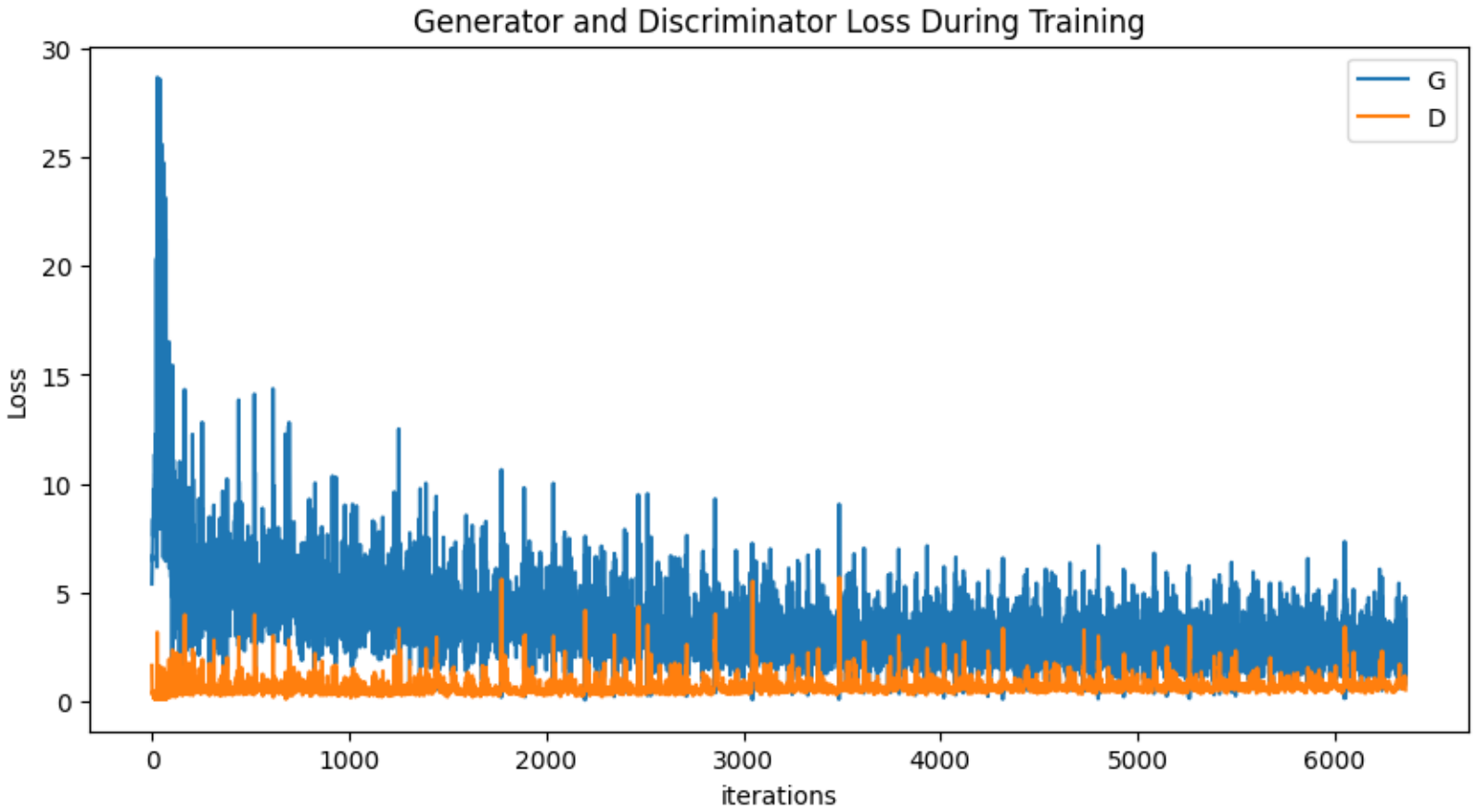
https://pytorch.org/docs/stable/generated/torch.nn.ConvTranspose2d.html

Up-sampling by fractional striding
Input 3x3, output = 5x5
stride =1convolution window = 3 x 3 (with respect to output)

Other tricks proposed by DCGAN

- Adam optimizer (adaptive moment estimation) = similar to SGD but with less parameter tuning
- Momentum = 0.5 (usually is 0.9 but training oscillated and was unstable)
- Low learning rate = 0.0002

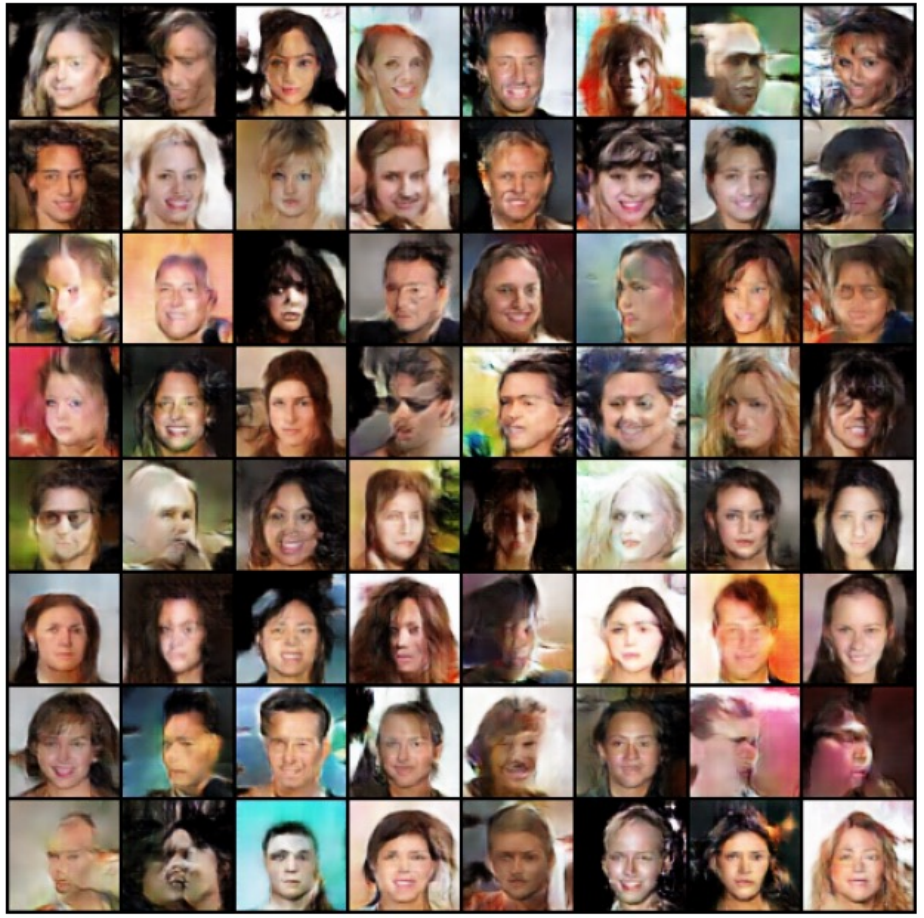
DCGAN


Generator and Discriminator Loss During Training
Loss
iterations
G
D
30
25
20
15
10
5
0
0
1000
2000
3000
4000
5000
6000

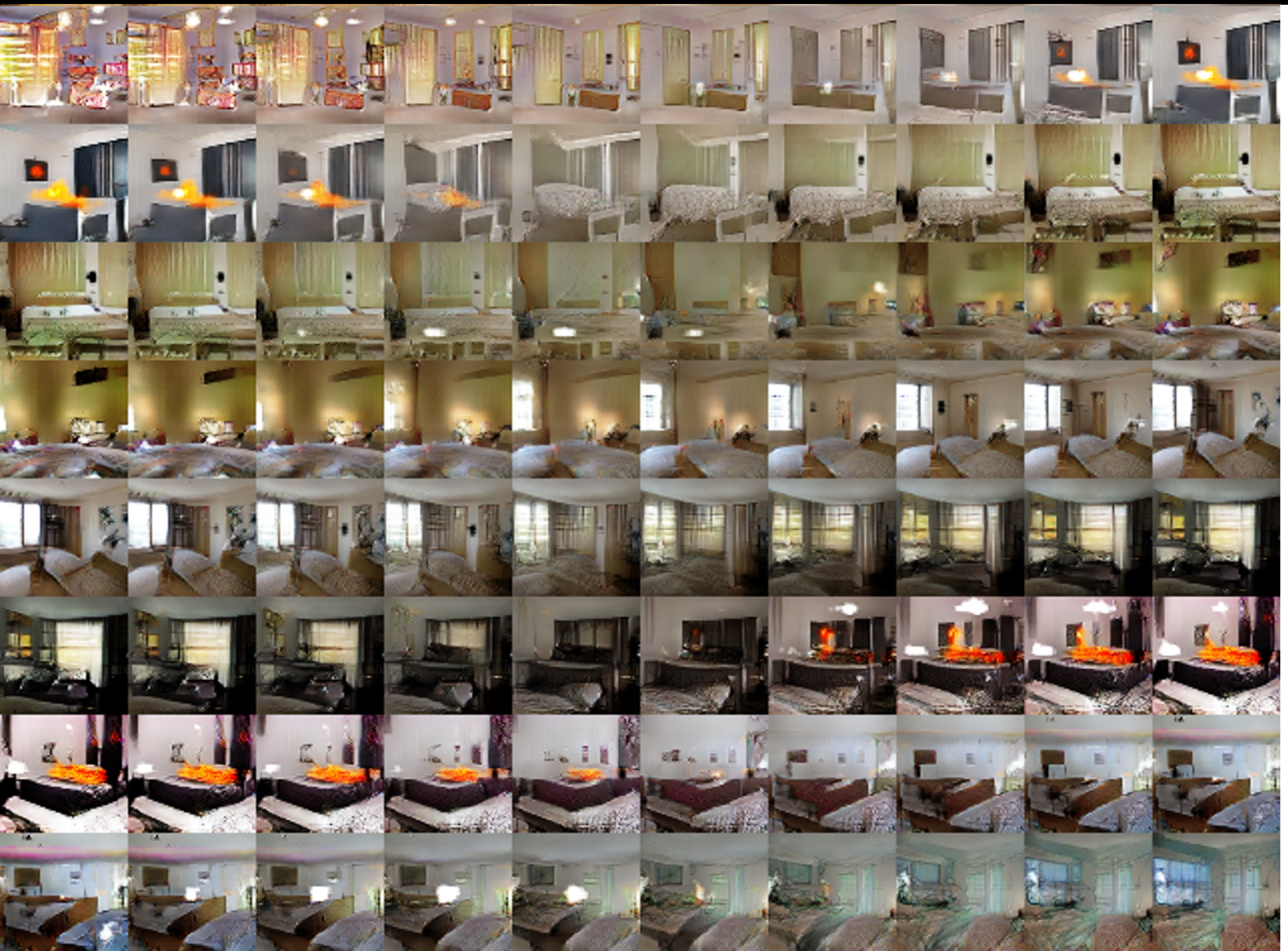
DCGAN

Real Images

Fake Images



Sanity check by walking on the manifold



Interpolating between a series of random points in $\boldsymbol{z}$
($\boldsymbol{z}$ = the 100-d random numbers)
e.g.,

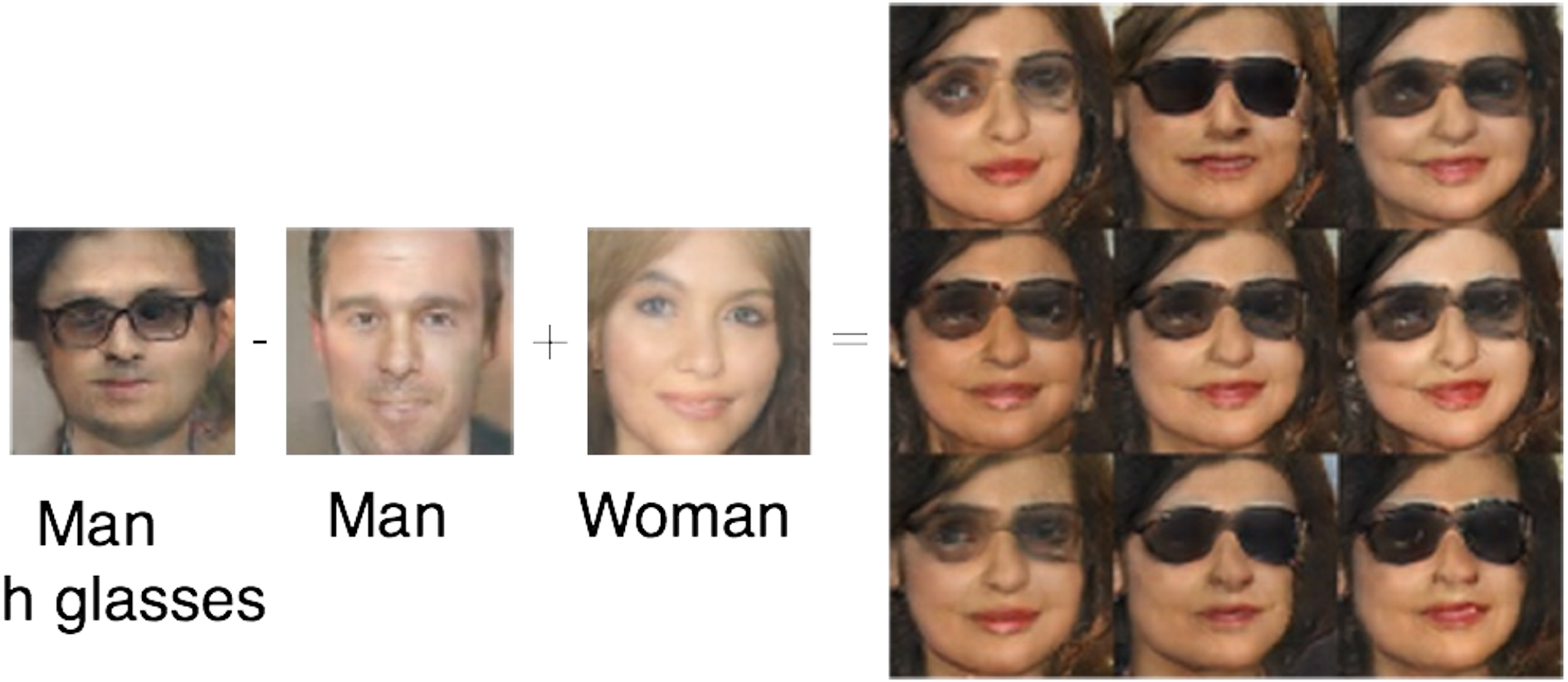
$$\boldsymbol{z}_{new} = m\,\boldsymbol{z}_1 + (1-m)\boldsymbol{z}_2$$

Note the smooth transition between each scene

←the window slowly appearing

This indicates that the space learned has smooth transitions!
(and is not simply memorizing)

Sanity check by vector space arithmetic



Outline

- GAN Basics
- GAN Training
- DCGAN
- Mode Collapse

Mode Collapse

Advantages of GANs

- **Plenty of existing work on Deep Generative Models**
  - Boltzmann Machine
  - Deep Belief Nets
  - Variational AutoEncoders (VAE)

- **Why GANs?**
  - Sampling (or generation) is straightforward.
  - Training doesn't involve Maximum Likelihood estimation (i.e. finding the best model parameters that fit the training data the most), believed to cause poorer quality of images (blurry images)
  - Robust to overfitting since the generator never sees the training data.

Problems with GAN

- **Probability Distribution is Implicit**
  - Not straightforward to compute $p(x)$
  - Thus **Vanilla GANs** are only good for Sampling/Generation.

- **Training is Hard**
  - Non-Convergence
  - Mode-Collapse

Non-convergence

- **Deep Learning models (in general) involve a single player**
  - The player tries to maximize its reward (minimize its loss).
  - Use SGD (with Backpropagation) to find the optimal parameters.
  - SGD has convergence guarantees (under certain conditions).
  - **Problem:** With non-convexity, we might converge to local optima.

$$\min_G L(G)$$

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

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

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

Non-convergence example

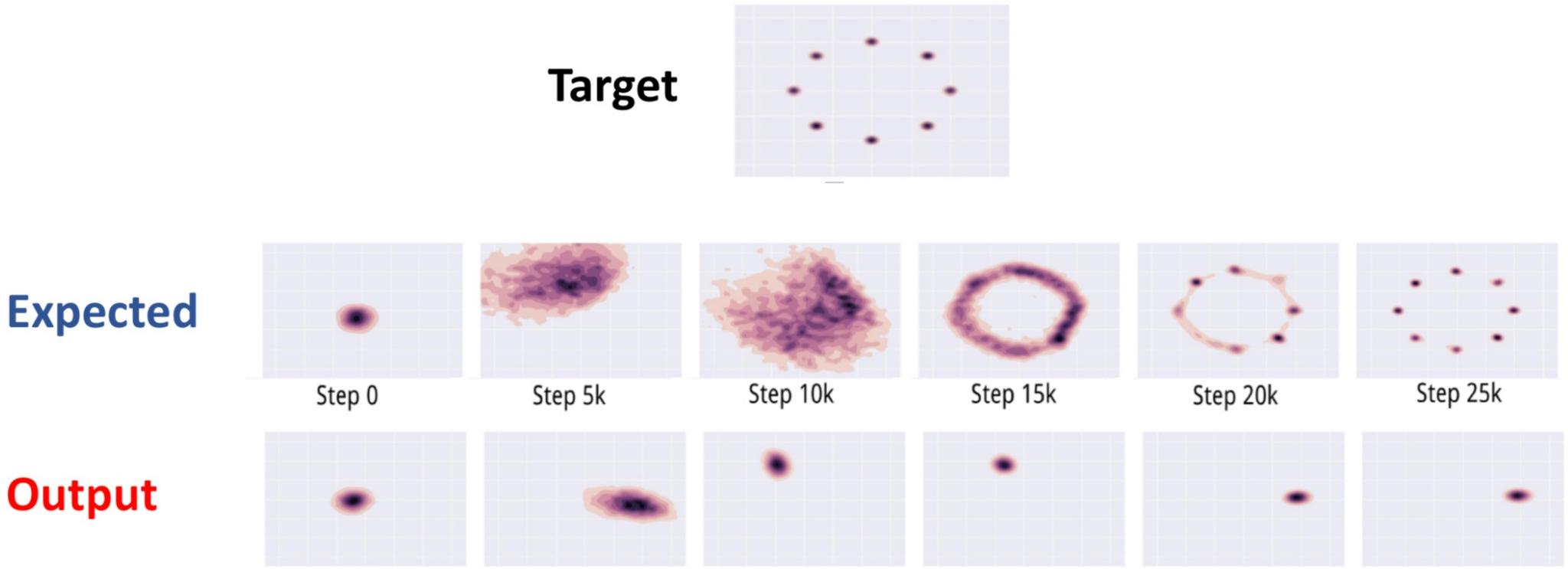
$$\min_x \max_y V(x, y)$$

Let $V(x, y) = xy$

• State 1:	x > 0	y > 0	V > 0	Increase y	Decrease x
• State 2:	x < 0	y > 0	V < 0	Decrease y	Decrease x
• State 3:	x < 0	y < 0	V > 0	Decrease y	Increase x
• State 4 :	x > 0	y < 0	V < 0	Increase y	Increase x
• State 5:	x > 0	y > 0	V > 0 == State 1	Increase y	Decrease x

Mode Collapse

Generator fails to output diverse samples



Metz, Luke, et al. "Unrolled Generative Adversarial Networks." arXiv preprint arXiv:1611.02163 (2016)

How to reward sample diversity

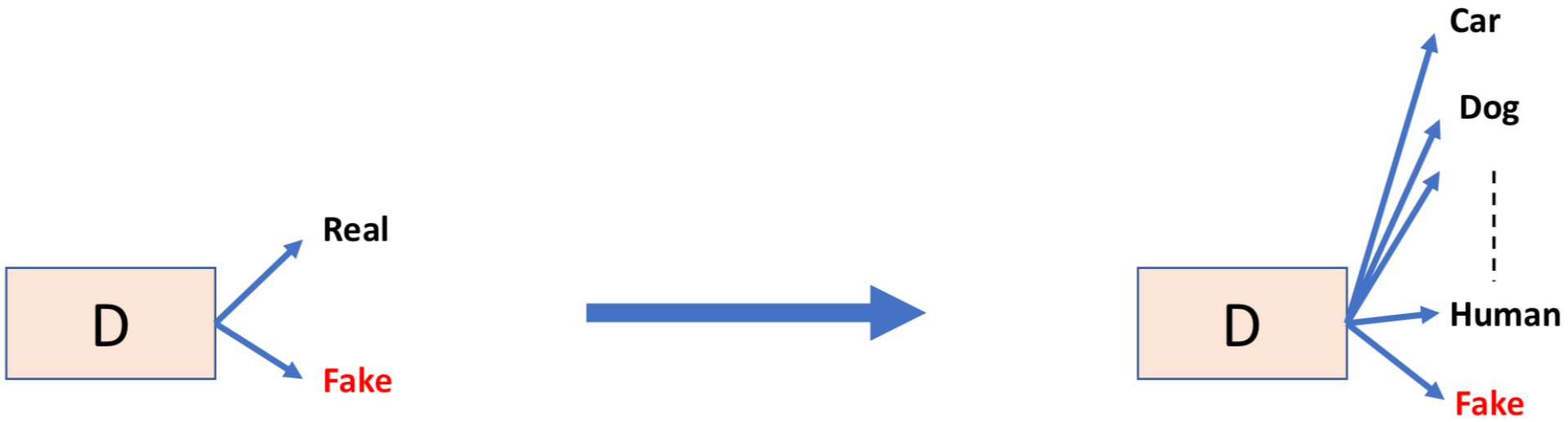
- **At Mode Collapse,**
  - Generator produces good samples, but a very few of them.
  - Thus, Discriminator can't tag them as fake.
- **To address this problem,**
  - Let the Discriminator know about this edge-case.
- **More formally,**
  - Let the Discriminator look at the entire batch instead of single examples
  - If there is lack of diversity, it will mark the examples as fake
- **Thus,**
  - Generator will be forced to produce diverse samples.

Mini-Batch GANs

- **Extract features that capture diversity in the mini-batch**
  - For e.g. L2 norm of the difference between all pairs from the batch
  - That is, the errors at an intermediate layer for real and fake samples are minimized.
- **Feed those features to the discriminator along with the image**
- **Feature values will differ b/w diverse and non-diverse batches**
  - Thus, Discriminator will rely on those features for classification
- **This in turn,**
  - Will force the Generator to match those feature values with the real data
  - Will generate diverse batches

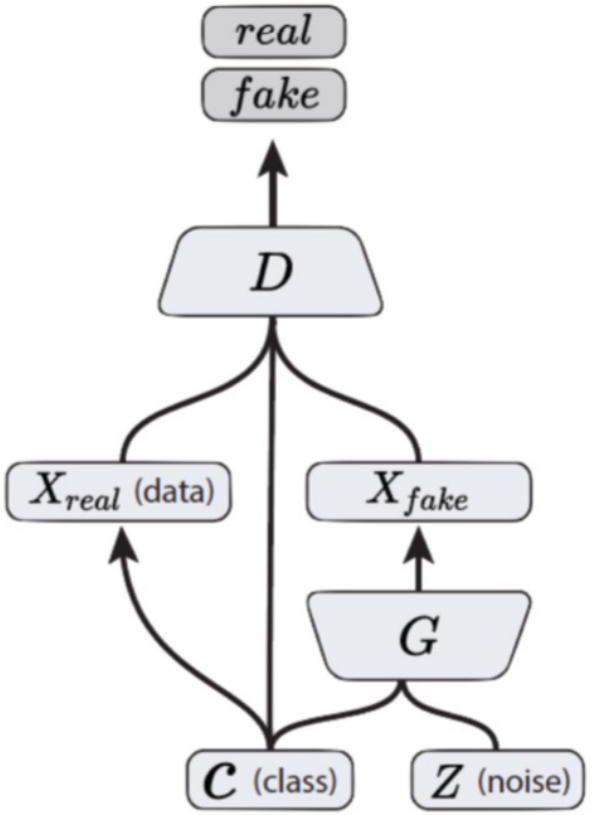
Supervision with Labels

- Label information of the real data might help



- Empirically generates much better samples

Conditional GANs



Conditional GAN
(Mirza & Osindero, 2014)

Conditions the inputs to the generator and discriminator with the labels

Conditional GANs

MNIST digits generated conditioned on their class label.

$[1, 0, 0, 0, 0, 0, 0, 0, 0, 0] \rightarrow$

$[0, 1, 0, 0, 0, 0, 0, 0, 0, 0] \rightarrow$

$[0, 0, 1, 0, 0, 0, 0, 0, 0, 0] \rightarrow$

$[0, 0, 0, 1, 0, 0, 0, 0, 0, 0] \rightarrow$

$[0, 0, 0, 0, 1, 0, 0, 0, 0, 0] \rightarrow$

$[0, 0, 0, 0, 0, 1, 0, 0, 0, 0] \rightarrow$

$[0, 0, 0, 0, 0, 0, 1, 0, 0, 0] \rightarrow$

$[0, 0, 0, 0, 0, 0, 0, 1, 0, 0] \rightarrow$

$[0, 0, 0, 0, 0, 0, 0, 0, 1, 0] \rightarrow$

$[0, 0, 0, 0, 0, 0, 0, 0, 0, 1] \rightarrow$

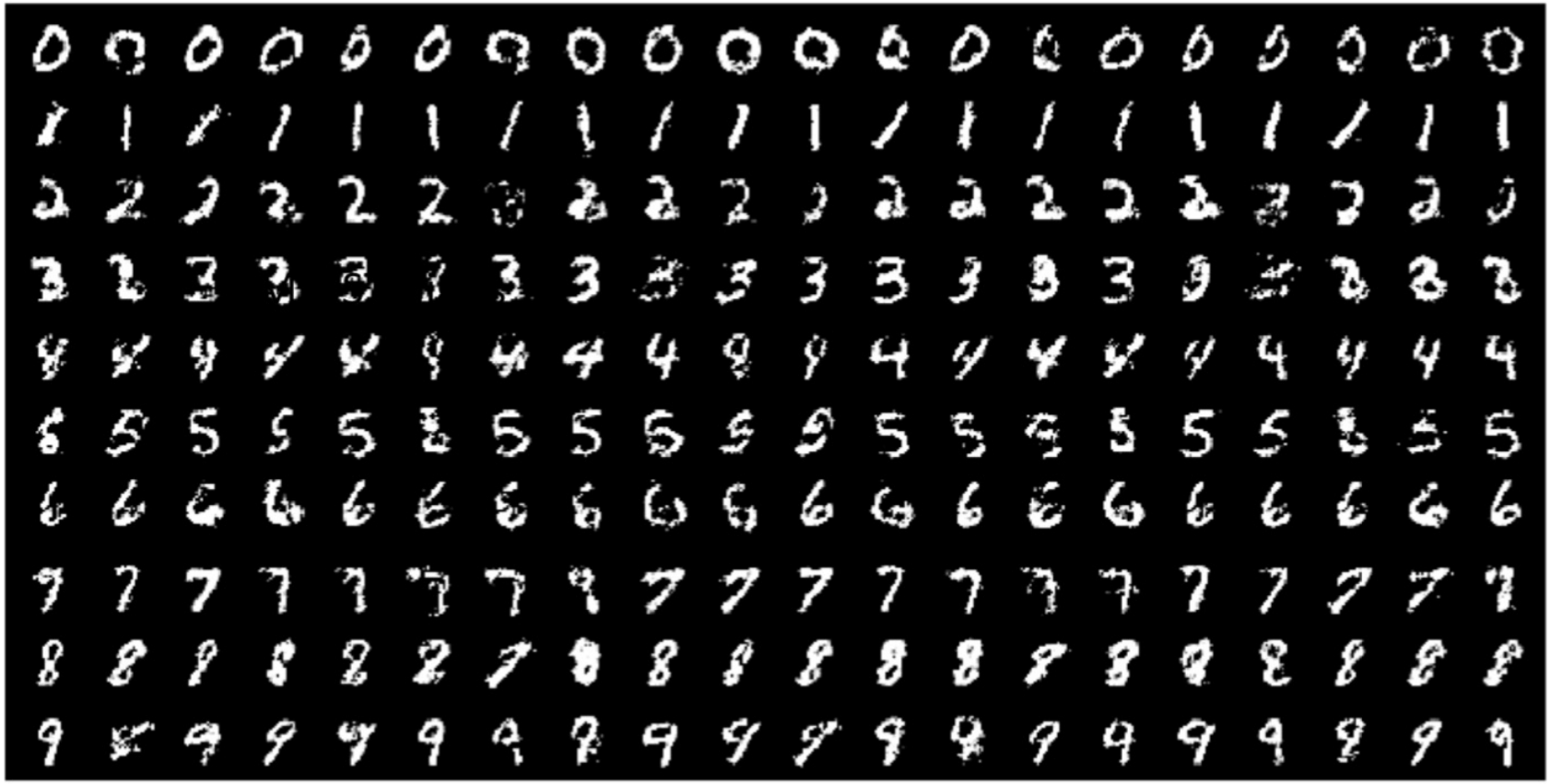
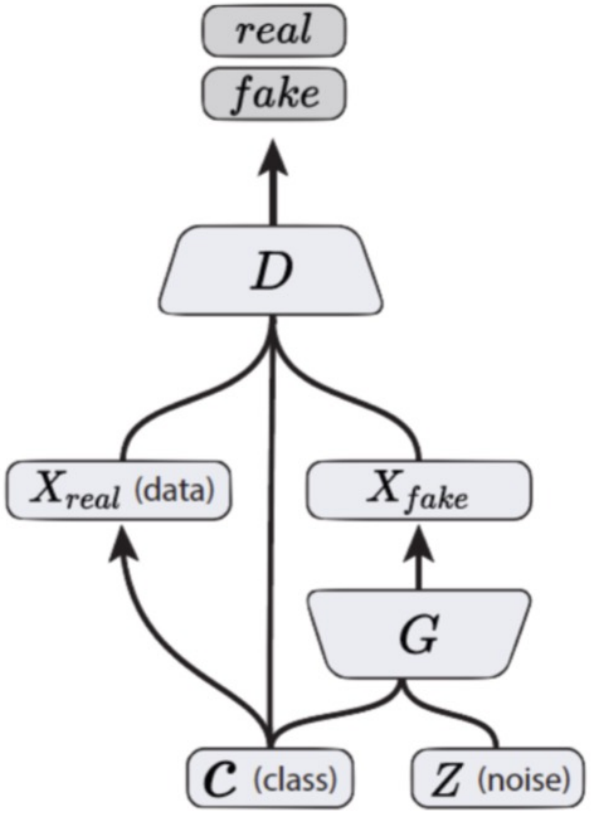


Figure 2 in the original paper.

Mirza, Mehdi, and Simon Osindero. Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784 (2014).

Conditional GANs

- Simple modification to the original GAN framework that conditions the model on *additional information* for better multi-modal learning.
- Lends to many practical applications of GANs when we have explicit *supervision* available.



Conditional GAN
(Mirza & Osindero, 2014)

Image Credit: Figure 2 in Odena, A., Olah, C. and Shlens, J., 2016. Conditional image synthesis with auxiliary classifier GANs. *arXiv preprint arXiv:1610.09585*.