Introduction to Generative Adversarial Networks (GANs)

Nguyen Tuan Duong
Nguyen Phuoc Tat Dat
(2017/9/9)

Agenda

- Generative Modeling
- Idea of GAN
- Training GAN
- Distribution matching
- Example
- Variants
- Applications of Generative Models

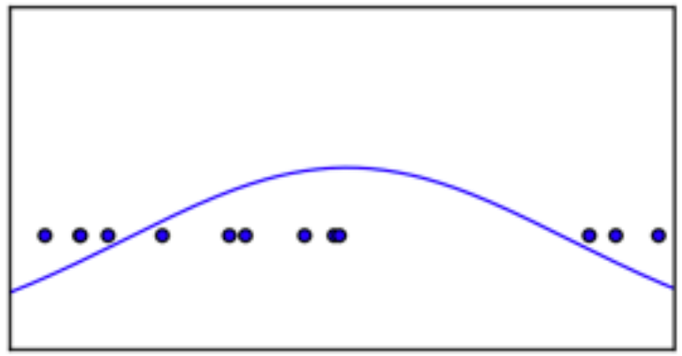
Generative Modeling

Generative Modeling

Training samples

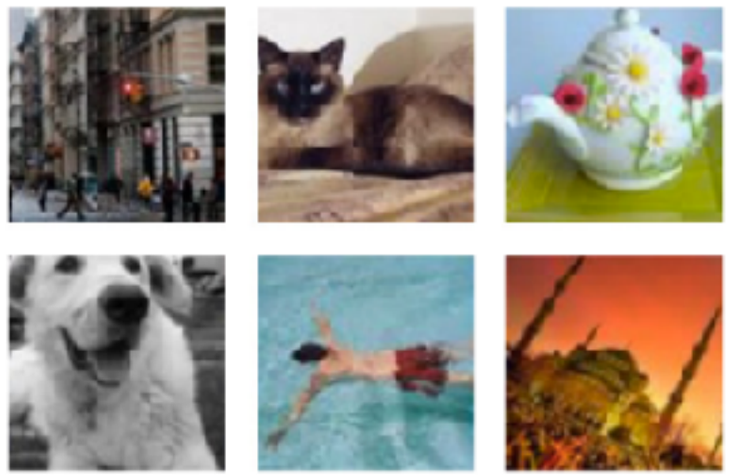
Density estimation

(Explicit or implicit)



p_{model}

p_{data}

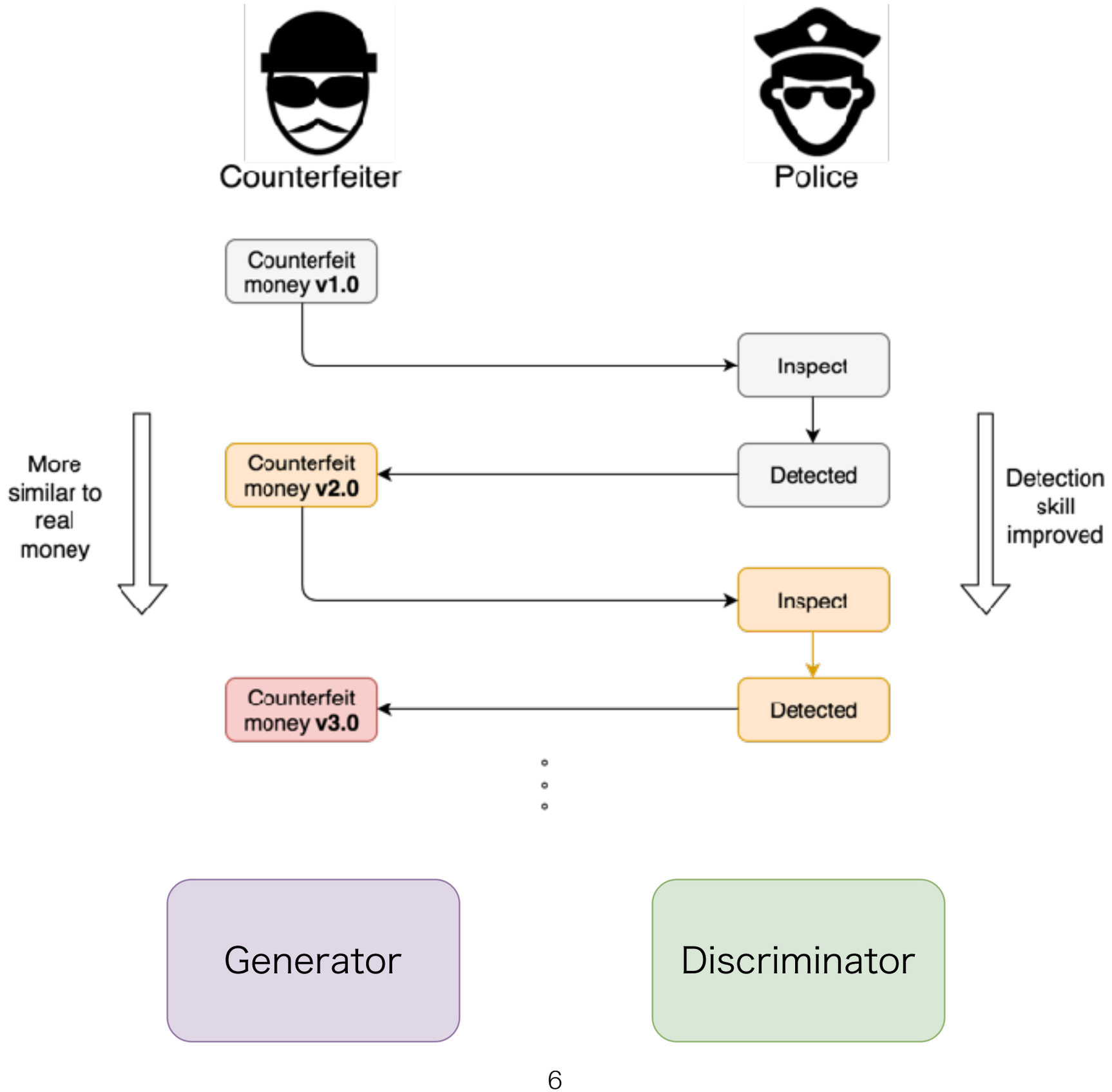


Model samples

(Goodfellow 2016)

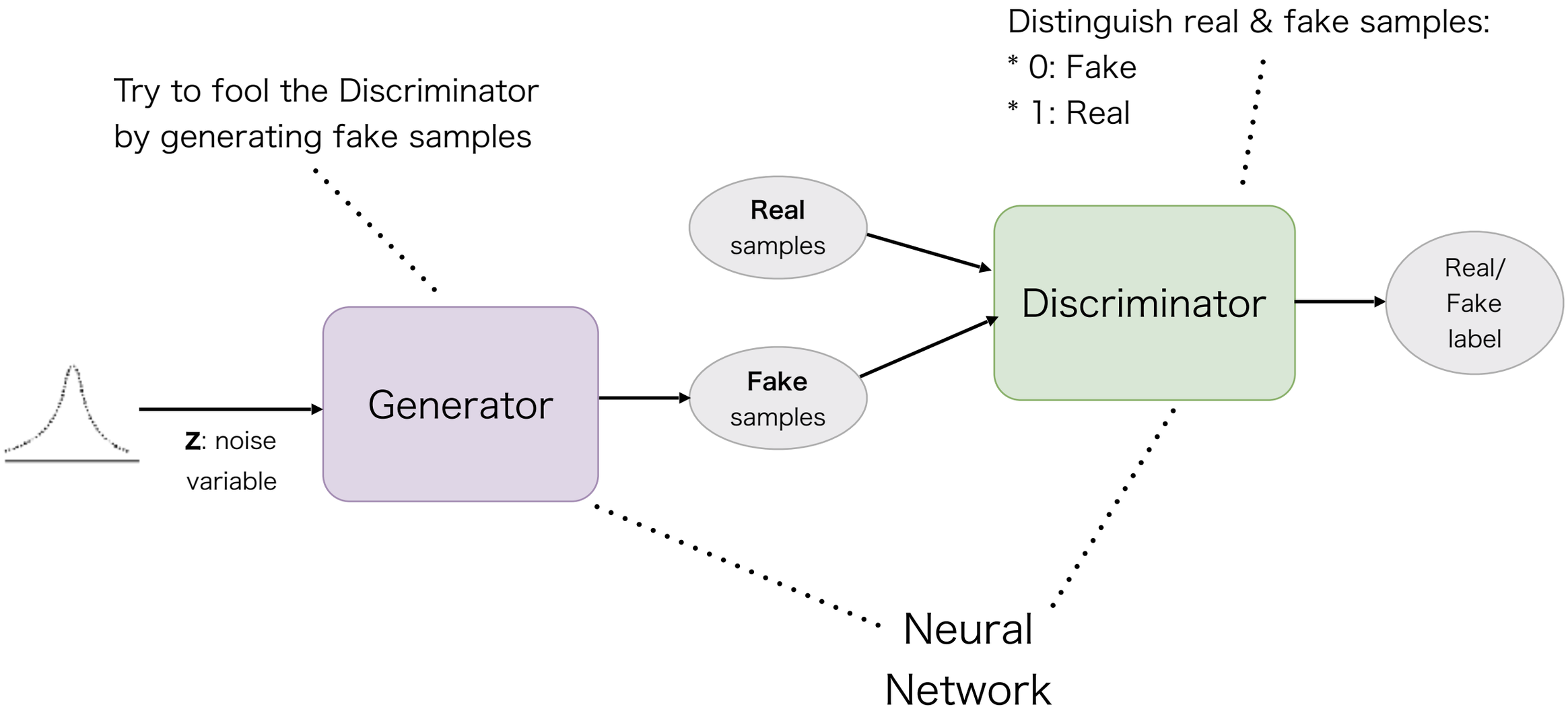
The idea of GAN

The idea of GAN



Training GAN

Framework

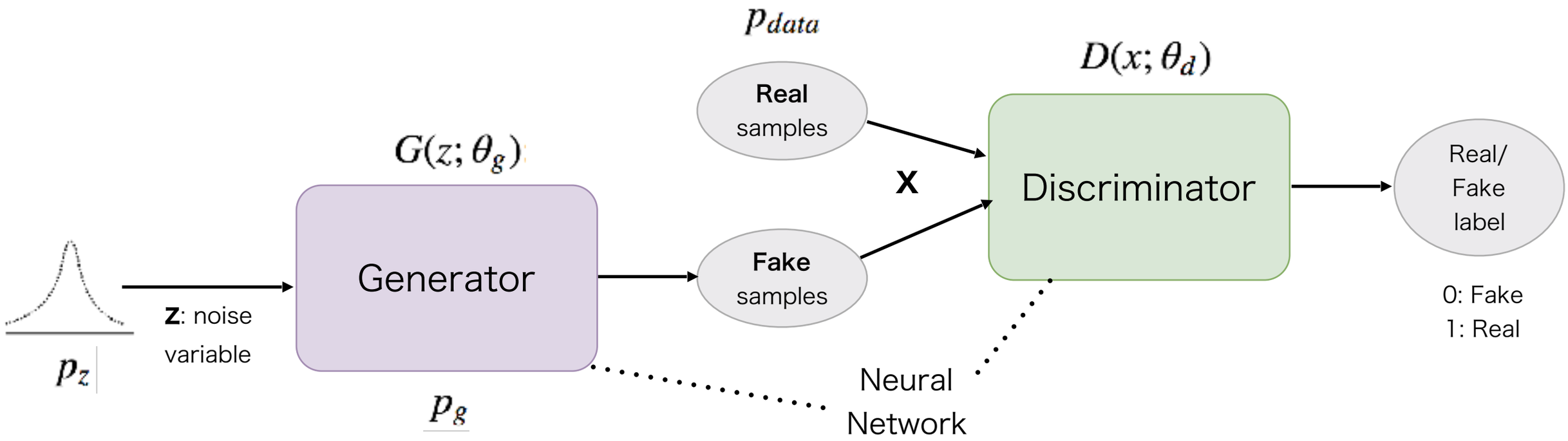


Generative Adversarial Networks

Notation

p_{data}	Distribution of training data
p_z	Distribution of the noise variable z
$G(z;\theta_g)$	Differentiable function with parameters θ_g , which receives z as input, and outputs an fake image.
$D(x;\theta_d)$	Function with parameters θ_d , which receives image x as input, and outputs a probability that x came from p_{data}
p_g	Generator's distribution
m	Number of training samples

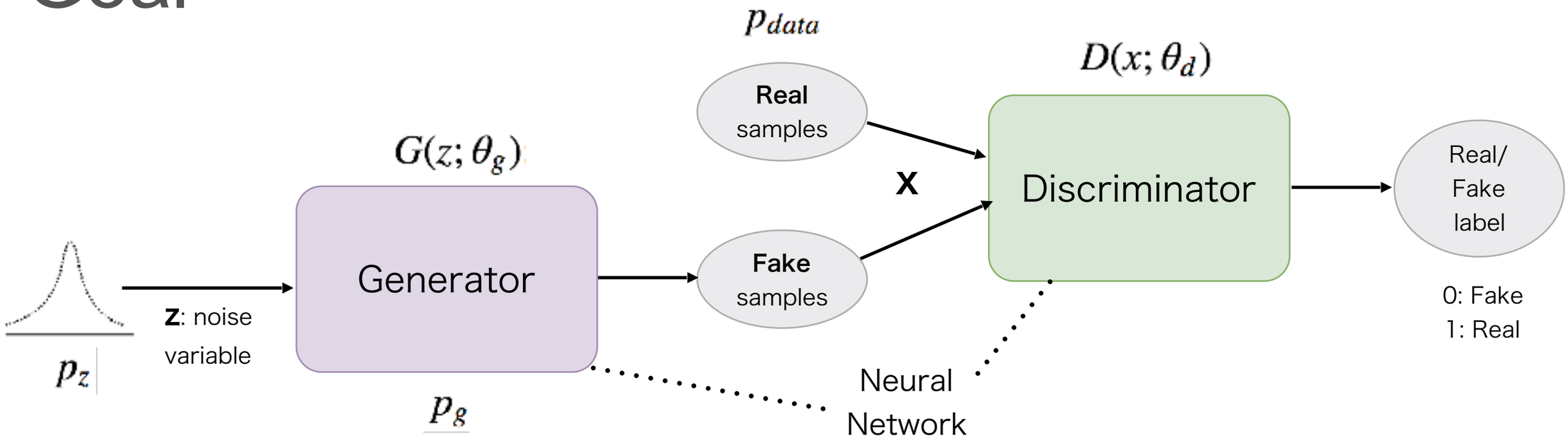
Goal



Ultimate goal:

* Train a good generator G which can generate an item x such that it seems to come from real samples.

Goal



- Minimize the loss function of G

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

- Maximize the loss function of the D

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Minimax game

- The training process is playing the minimax game between G and D:

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

(Goodfellow 2014)

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:

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

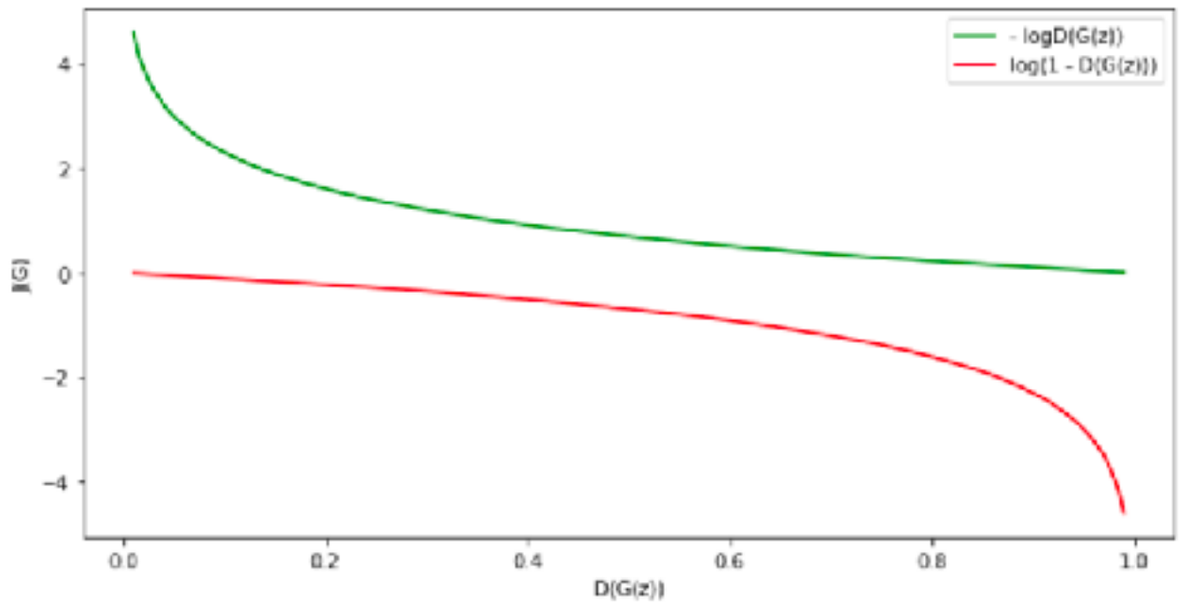
end for

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

Gradient saturation in minimax game

- The inner loop (with k) will optimize D. We hope this step makes D(G(z)) near to 0

- In minimax game, the loss function of G will be optimized with the target log(1-D(G(z))), which gives low gradient when D(G(z)) near to 0



- One trick to solve the low gradient problem is to change the target when optimizing the loss function of G:

$$\min_{\theta_g}[-E_{z\sim p(z)} log(D_{\theta_d}(G_{\theta_g}(z)))]$$

Improved version

- Minimize the loss function of G

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

- Maximize the loss function of the D

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Improved version

- Minimize the loss function of G

$$\min_{\theta_g} [-E_{z \sim p(z)} log(D_{\theta_d}(G_{\theta_g}(z)))]$$

- Maximize the loss function of the D

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

(Goodfellow 2016)

Distribution matching

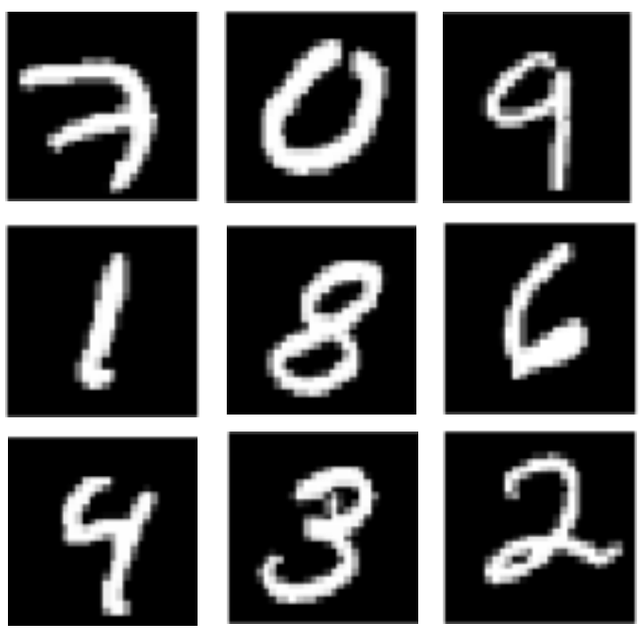
(Jupyter notebook)

Example

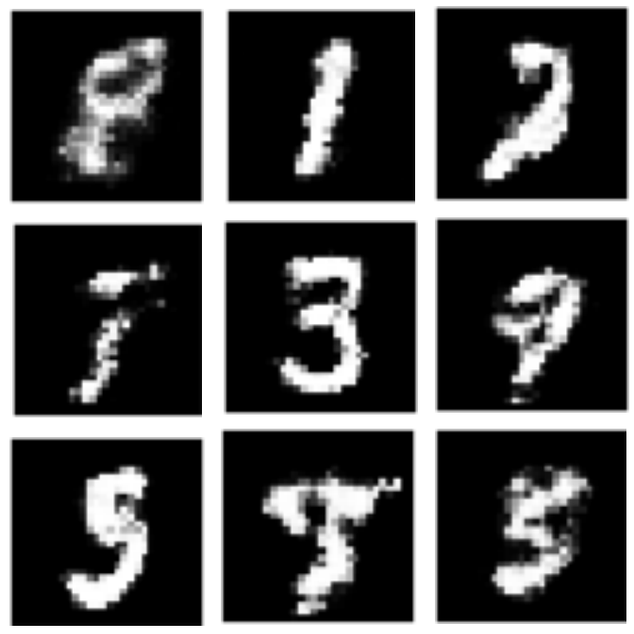
Generating handwritten digits

Example

- Generate hand-writing digit trained on MNIST dataset
- Train the models on MNIST dataset
  - Dataset of hand-written digit greyscale images
  - 60,000 training and 10,000 testing images of size 28x28



Real hand-writing digits
(real images)



Generated hand-writing digits
(fake images)

Change the loss function optimization for practice

- Minimize the loss function of G

Improved version

$$\min_{\theta_g}[-E_{z\sim p(z)} log(D_{\theta_d}(G_{\theta_g}(z)))]$$

- Maximize the loss function of the D

$$\max_{\theta_d}\left[\mathbb{E}_{x\sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z\sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))\right]$$

- Minimize the loss function of G

In practice

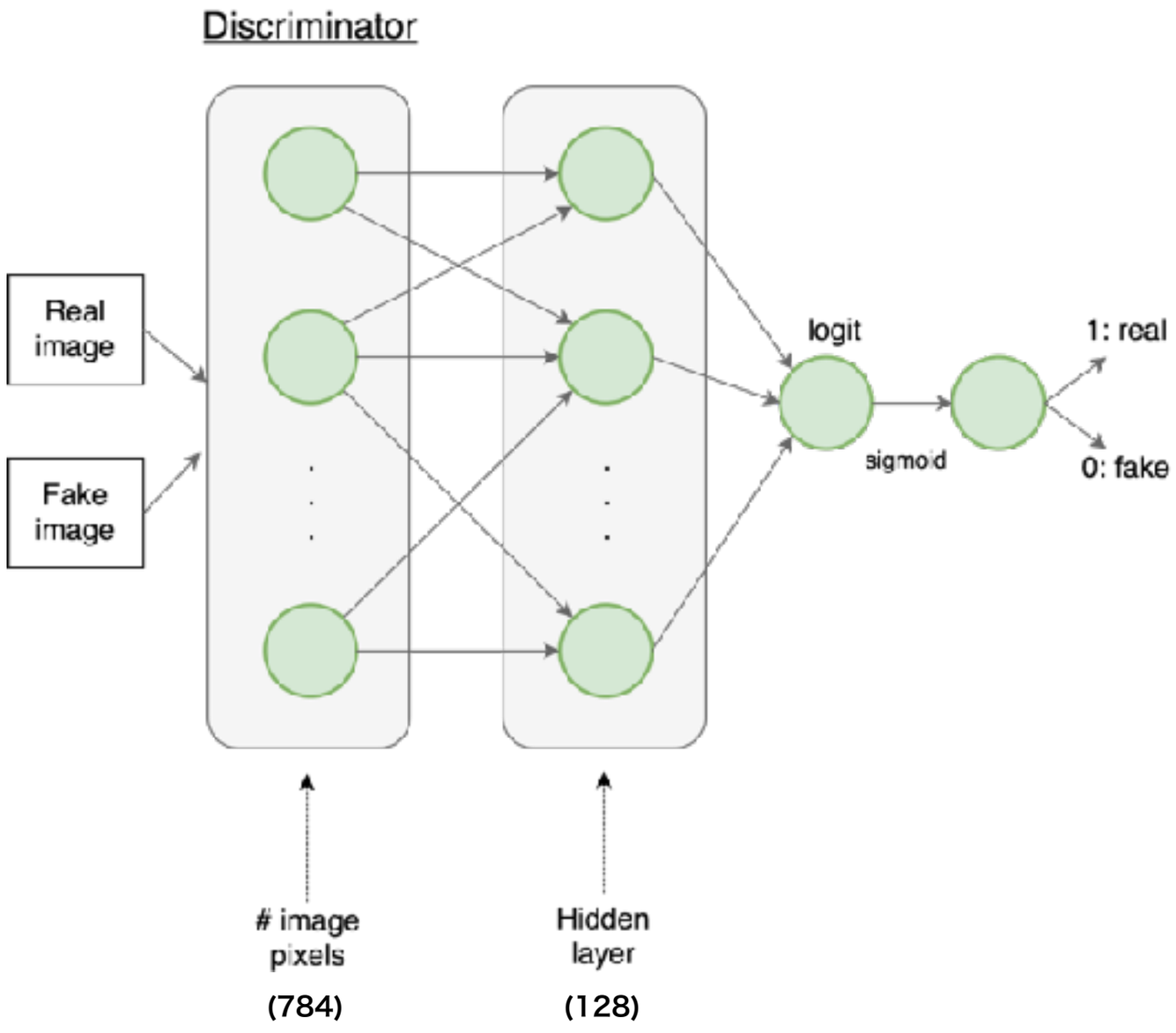
$$\min_{\theta_g}[-E_{z\sim p(z)} log(D_{\theta_d}(G_{\theta_g}(z)))]$$

- Minimize the loss function of the D

$$\min_{\theta_d}[-E_{x\sim p_{data}} log(D_{\theta_d}(x)) - E_{z\sim p(z)} log(1 - D_{\theta_d}(G_{\theta_g}(z)))]$$

(Goodfellow 2016)

Network Architecture - Discriminator

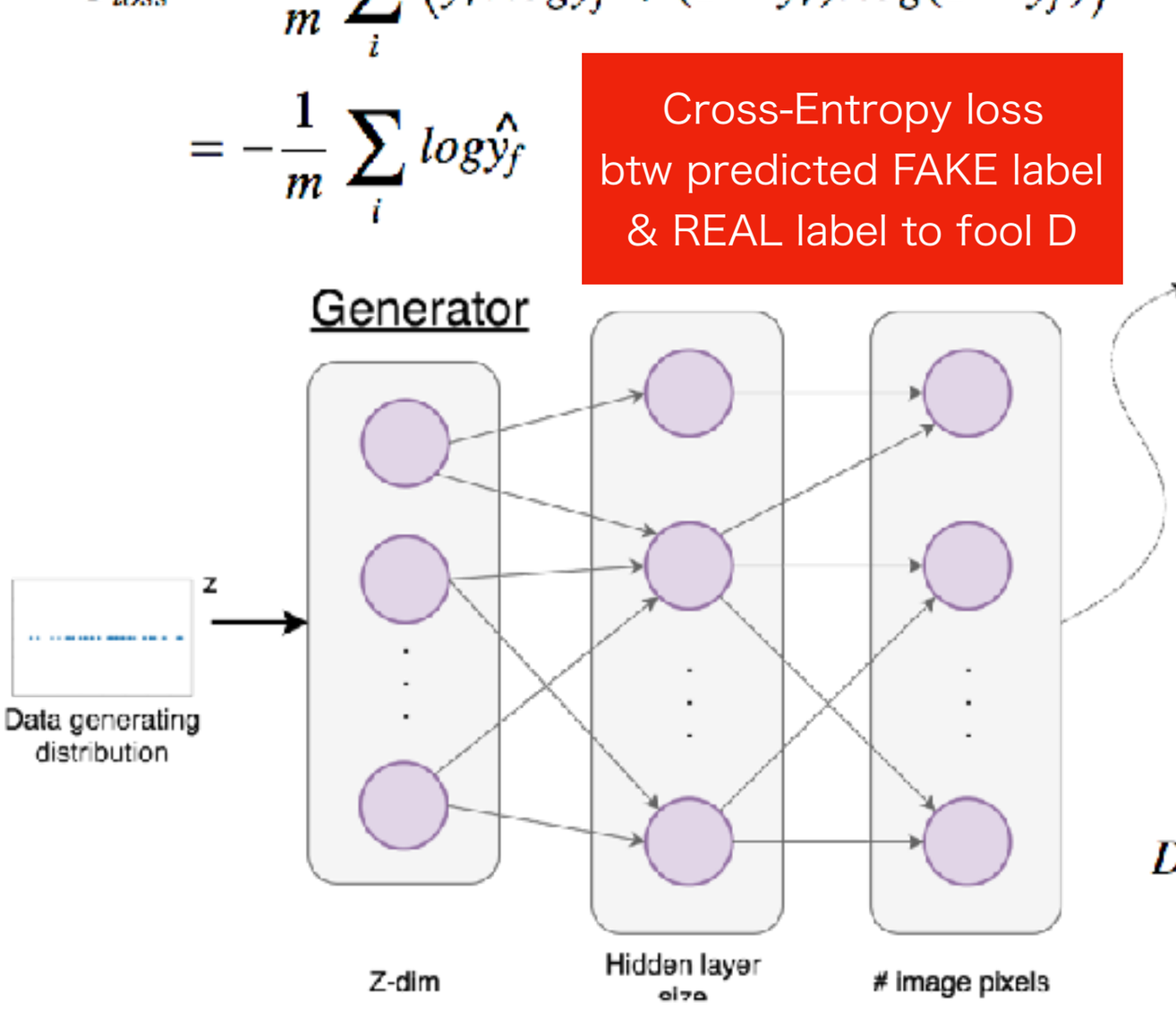


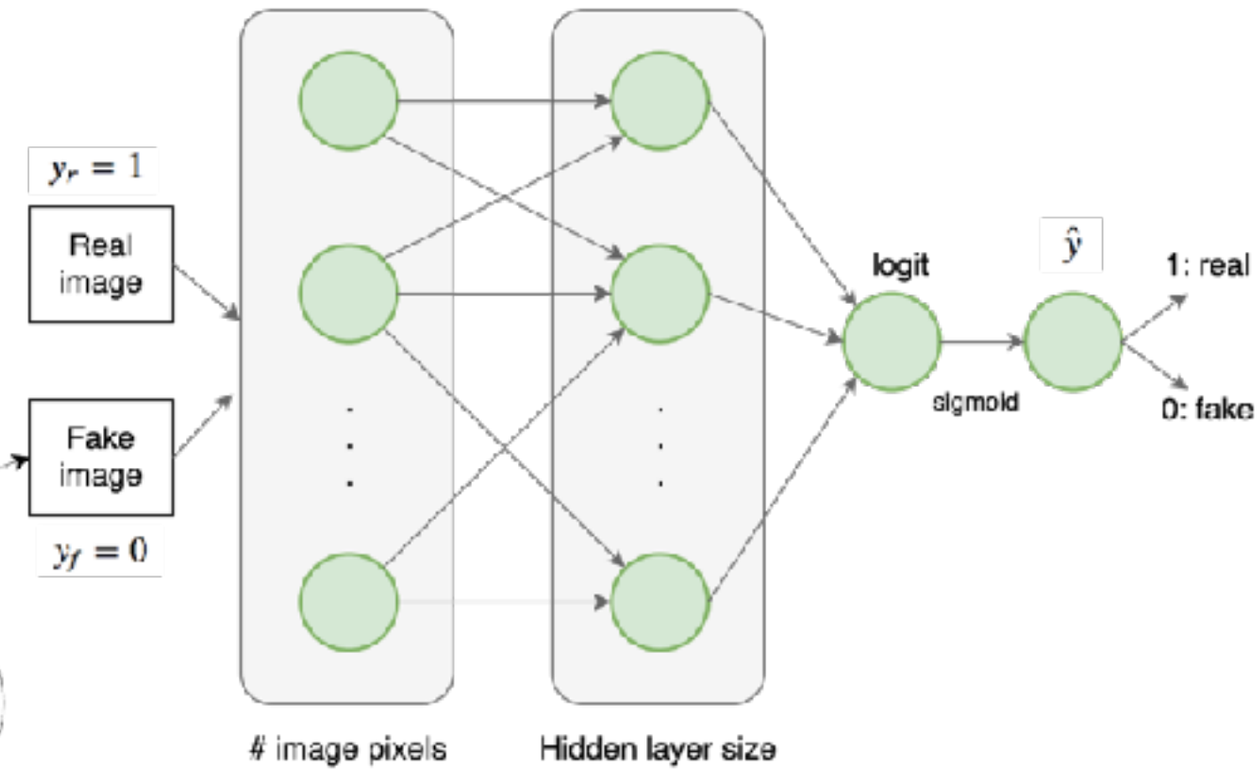
Network Architecture

$$G_{loss} = -\frac{1}{m}\sum_i \left(y_r . log\hat{y}_f + (1 - y_r) . log(1 - \hat{y}_f)\right)$$

$$= -\frac{1}{m}\sum_i log\hat{y}_f$$

Cross-Entropy loss
btw predicted FAKE label
& REAL label to fool D





$$D_{loss} = D_{loss\_real} + D_{loss\_fake}$$

Cross-Entropy loss

$$D_{loss\_real} = -\frac{1}{m}\sum_i \left(y_r . log\hat{y}_r + (1 - y_r) . log(1 - \hat{y}_r)\right)$$

$$= -\frac{1}{m}\sum_i log\hat{y}_r$$

$$D_{loss\_fake} = -\frac{1}{m}\sum_i \left(y_f . log\hat{y}_f + (1 - y_f) . log(1 - \hat{y}_f)\right)$$

$$= -\frac{1}{m}\sum_i log(1 - \hat{y}_f)$$

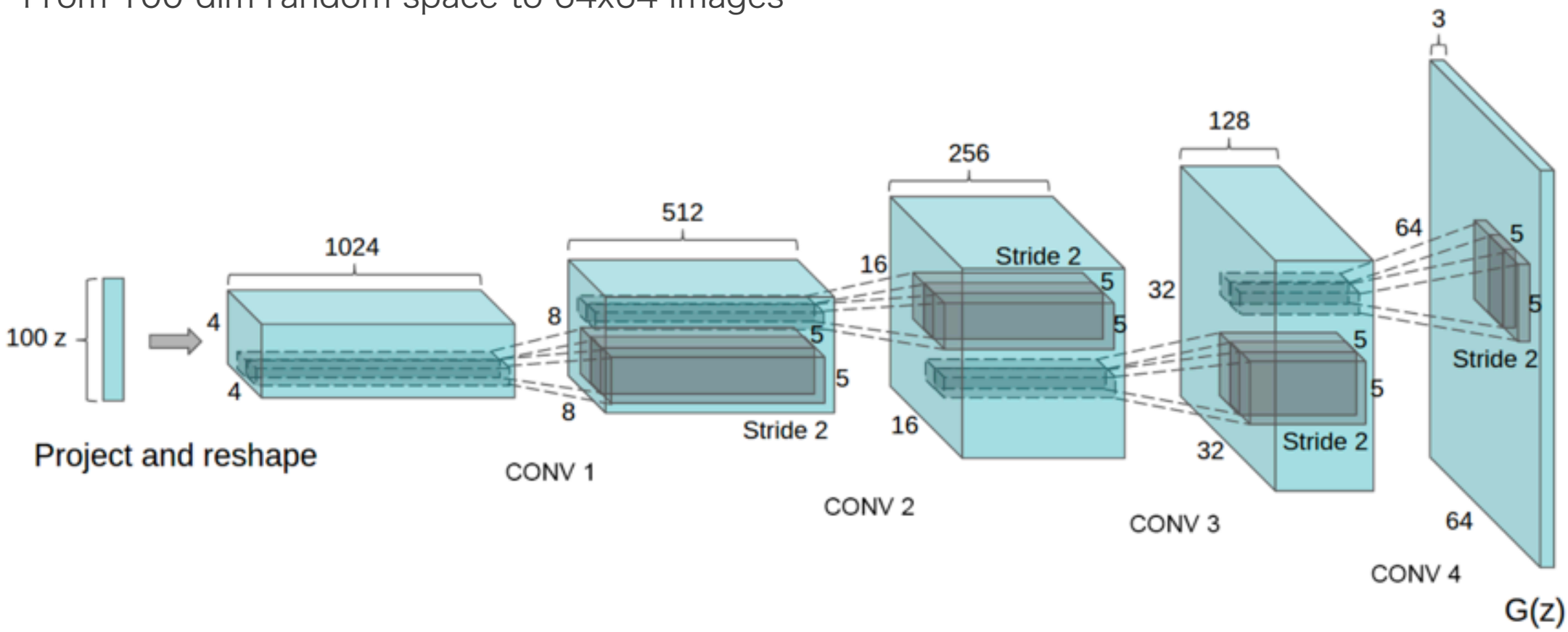
Demo

(Jupyter notebook)

Variants

DCGAN

- DCGAN architecture for the generator used for Large-scale Scene Understanding (LSUN)
- From 100-dim random space to 64x64 images



(Radford et al, ICLR 2016)

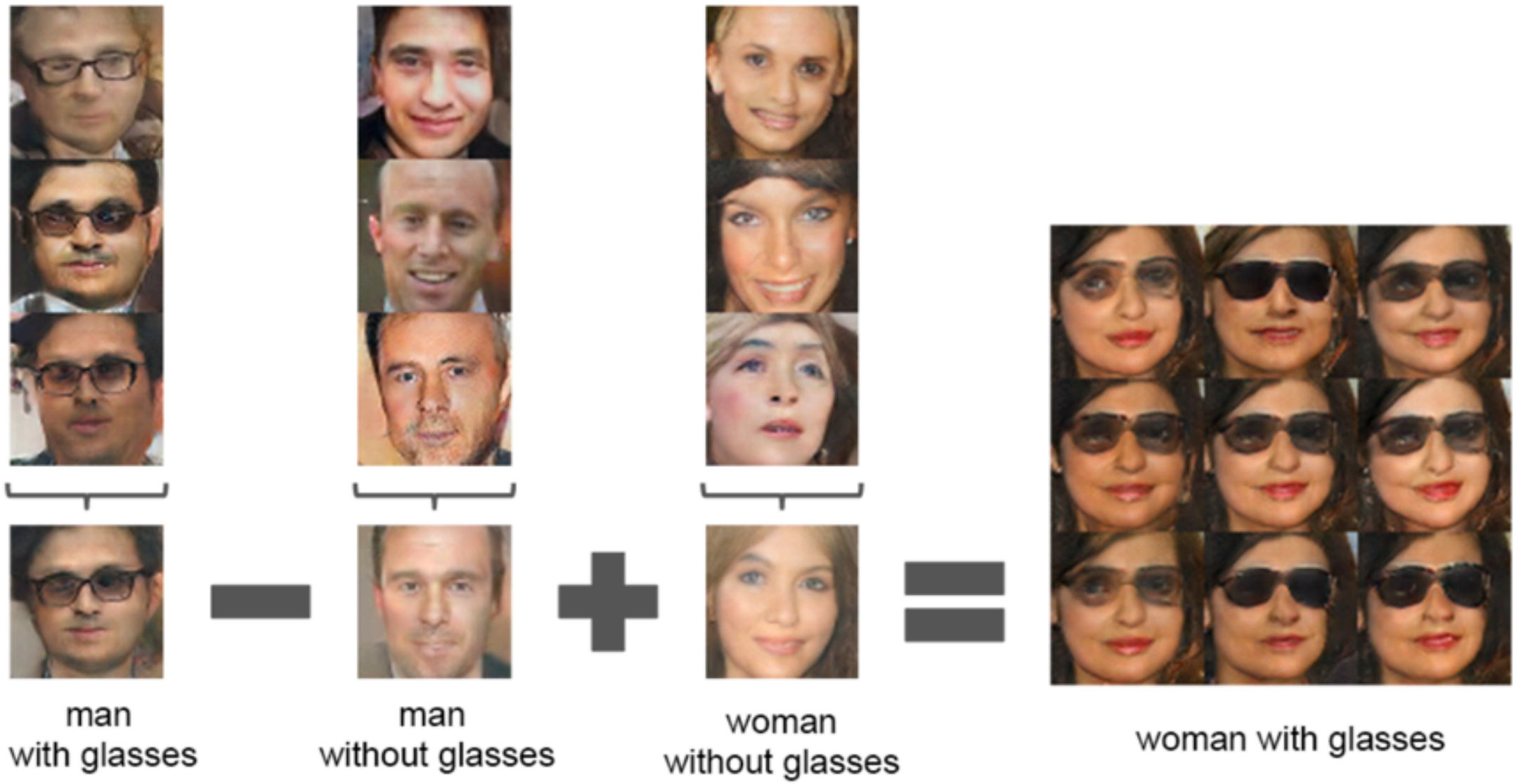
DCGAN

- Generated samples after training on LSUN dataset

(Radford et al, ICLR 2016)

DCGAN

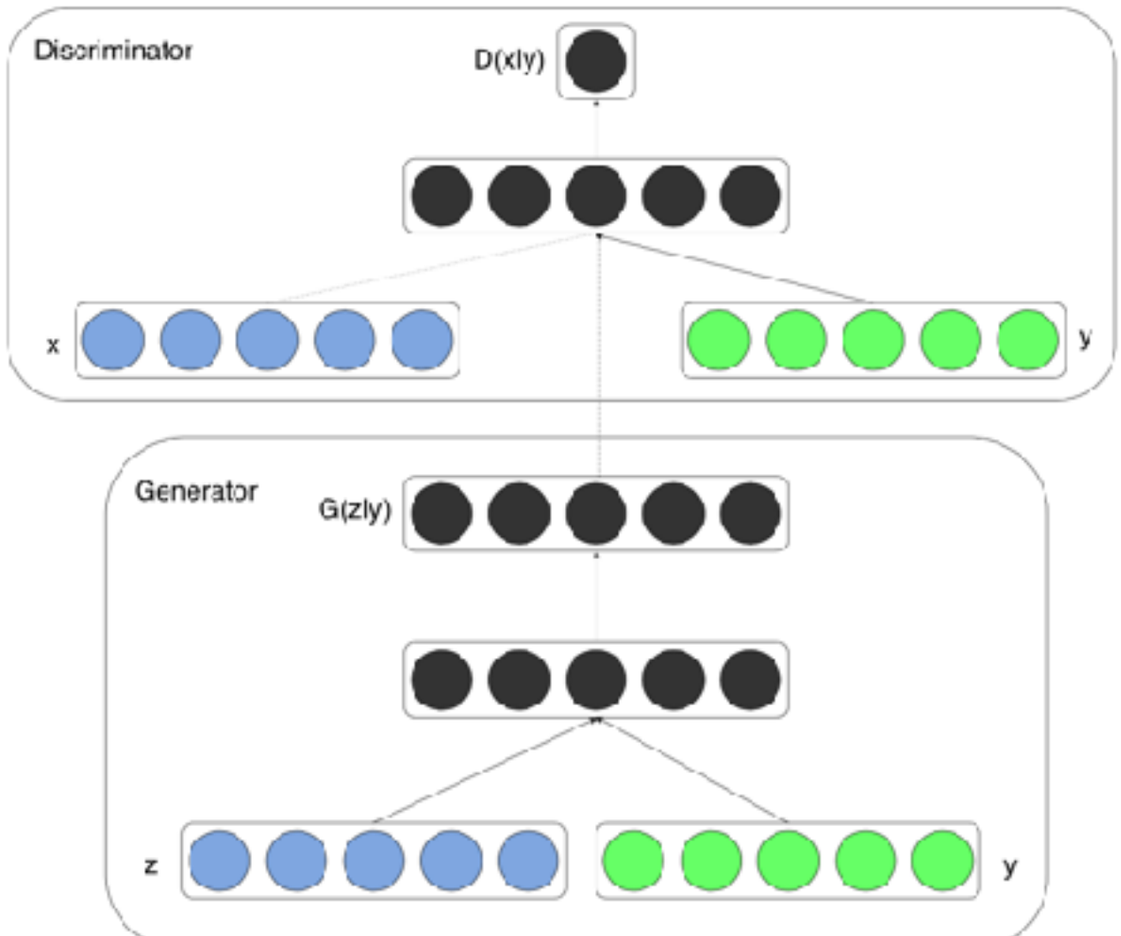
- Vector arithmetic for visual concepts



(Radford et al, ICLR 2016)

Conditional GAN

(Mirza et al, 2014)



- Class labels are encoded as one-hot vectors
- Control class labels of images by feeding one-hot vectors to both Discriminator & Generator

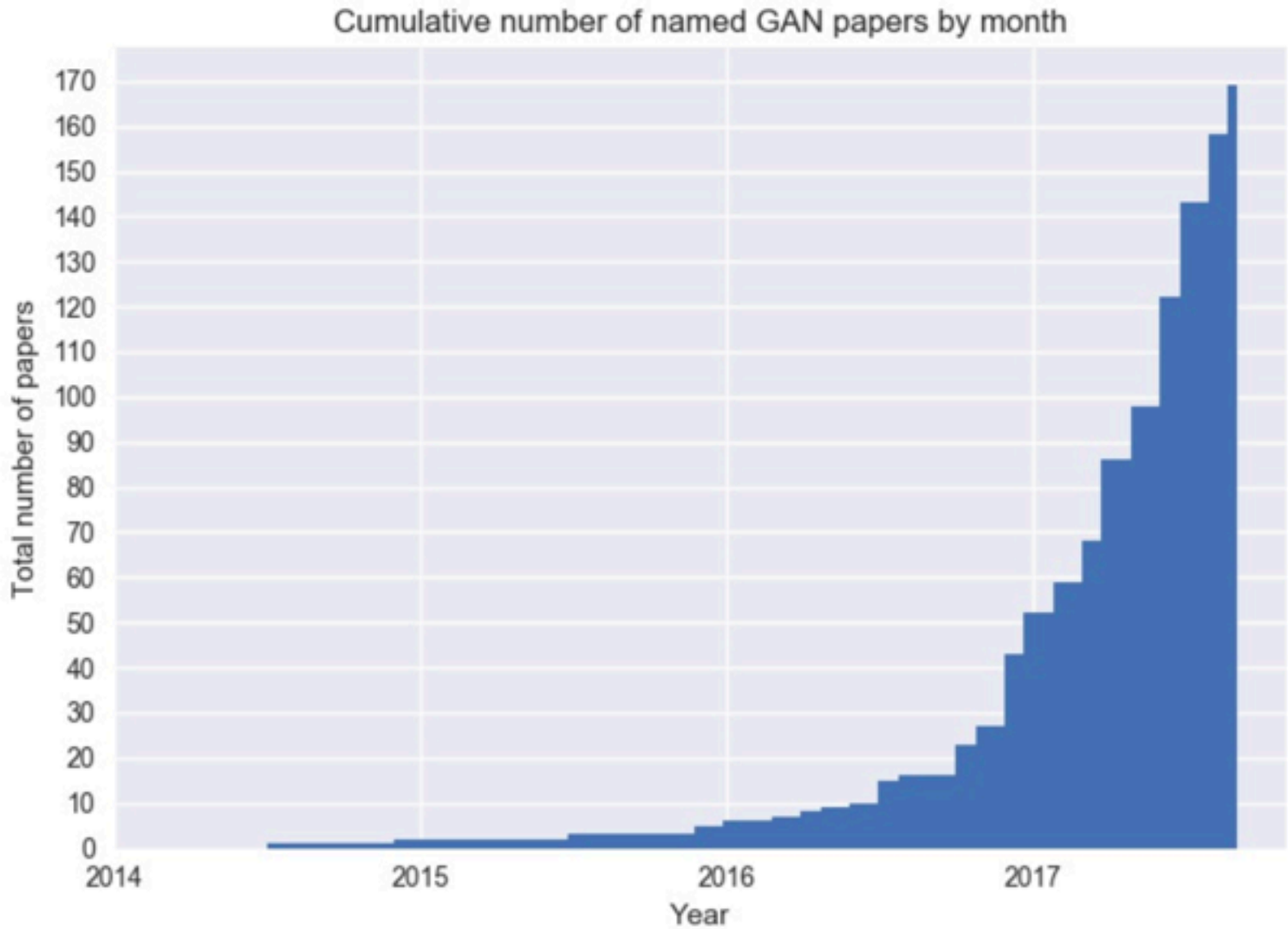
$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

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

Conditions on
the instances for both D & G

GAN Zoo

- Near to 200 GAN variants since the vanilla GAN was proposed in 2014



- The GAN Zoo:
  - https://deephunt.in/the-gan-zoo-79597dc8c347

Questions

Is there anyway to evaluate the generated images automatically?

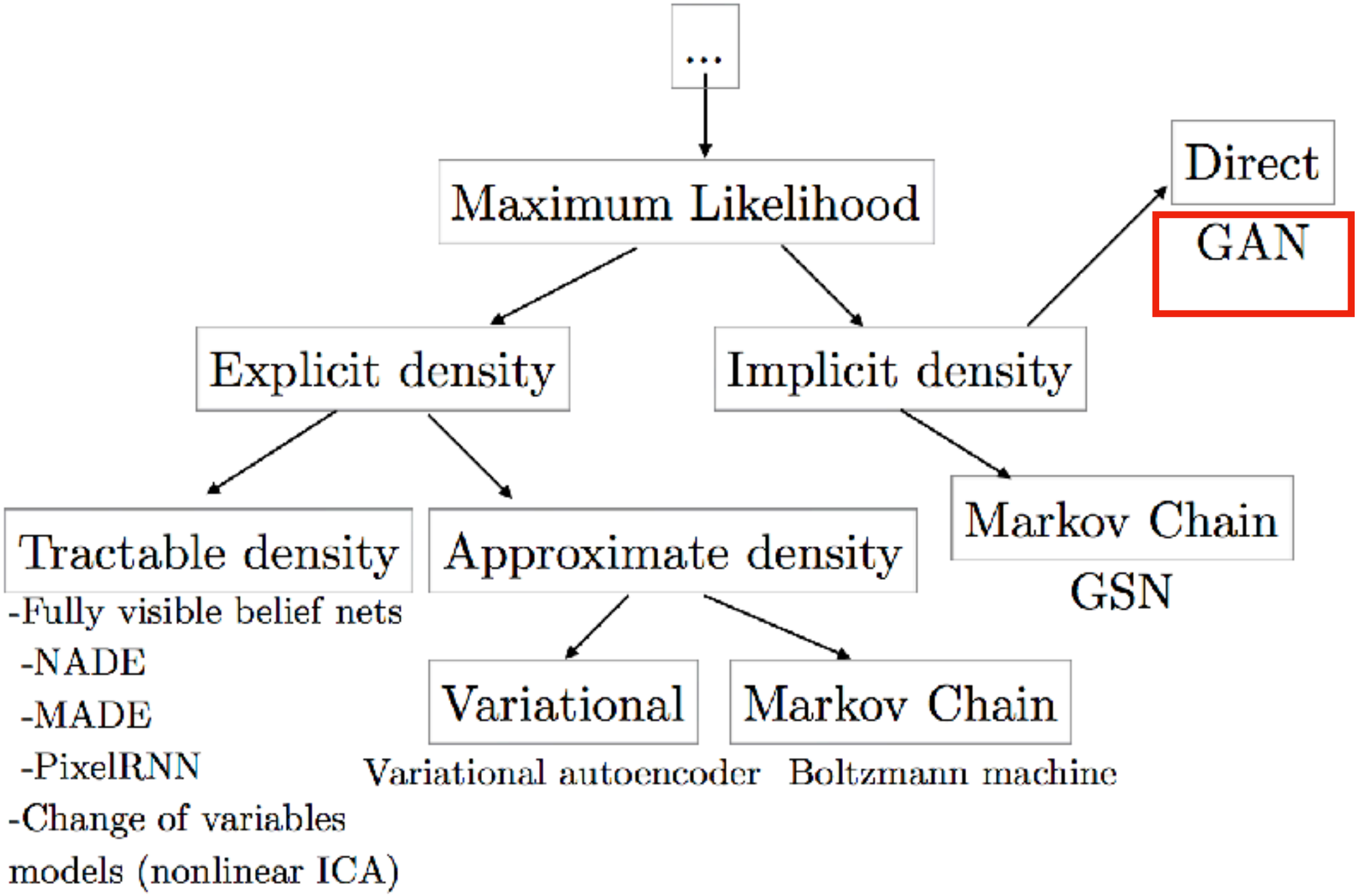
(so hard with training <-> looking)

Applications of Generative Models

Applications of Generative Models

- Create art works
- Build simulation environments
- Fix corrupted data…

Generative Modeling methods



(Goodfellow 2016)

Summary

- GAN trains a generator along with a discriminator by an adversarial process:
  - Discriminator aims to differentiate between real data and generated data
  - Generator tries to fool the discriminator by generating data
- Both of discriminator and generator are usually neural networks (although they can be other models)
- Proposed in 2014, GAN opened a new and hot research trend in Generative Modeling

Thank you!