ETH zürich

Generative Adversarial Networks (GANs)

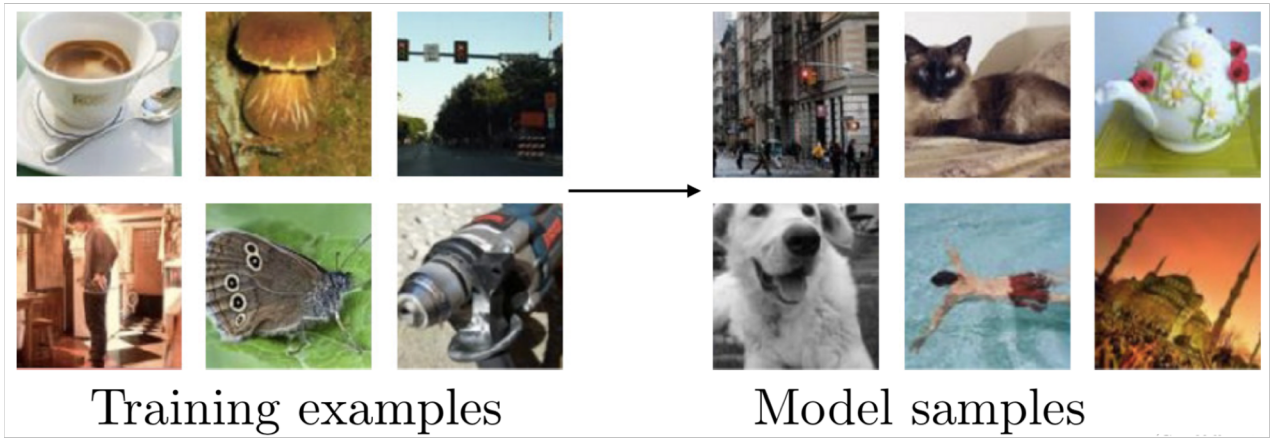
Javier Montoya (montoya@ethz.ch)

Arequipa, 9th January 2019

What is a generative model?

- **Key Idea:**
  - It cares about what distribution **Pdata(X)** generated the data **X**

  - It generates new samples from the learned distribution **Pmodel(X)**.



See NIPS 2016 Tutorial: Generative Adversarial Networks (I. Goodfellow)

Generative vs. Discriminative models

- **Discriminative Models:**
  - Given an image **X**, predict its label **Y.**
  - Estimates **P(Y|X).**

- **Limitations of discriminative models:**
  - Can't model **P(X)**, i.e. the prob. of seeing a certain image **X**.
  - They can't sample from **P(X)**, i.e. can't generate new images.

Why generative models?

What are some recent and potentially upcoming breakthroughs in deep learning?

Answer · Follow · 217 · Request

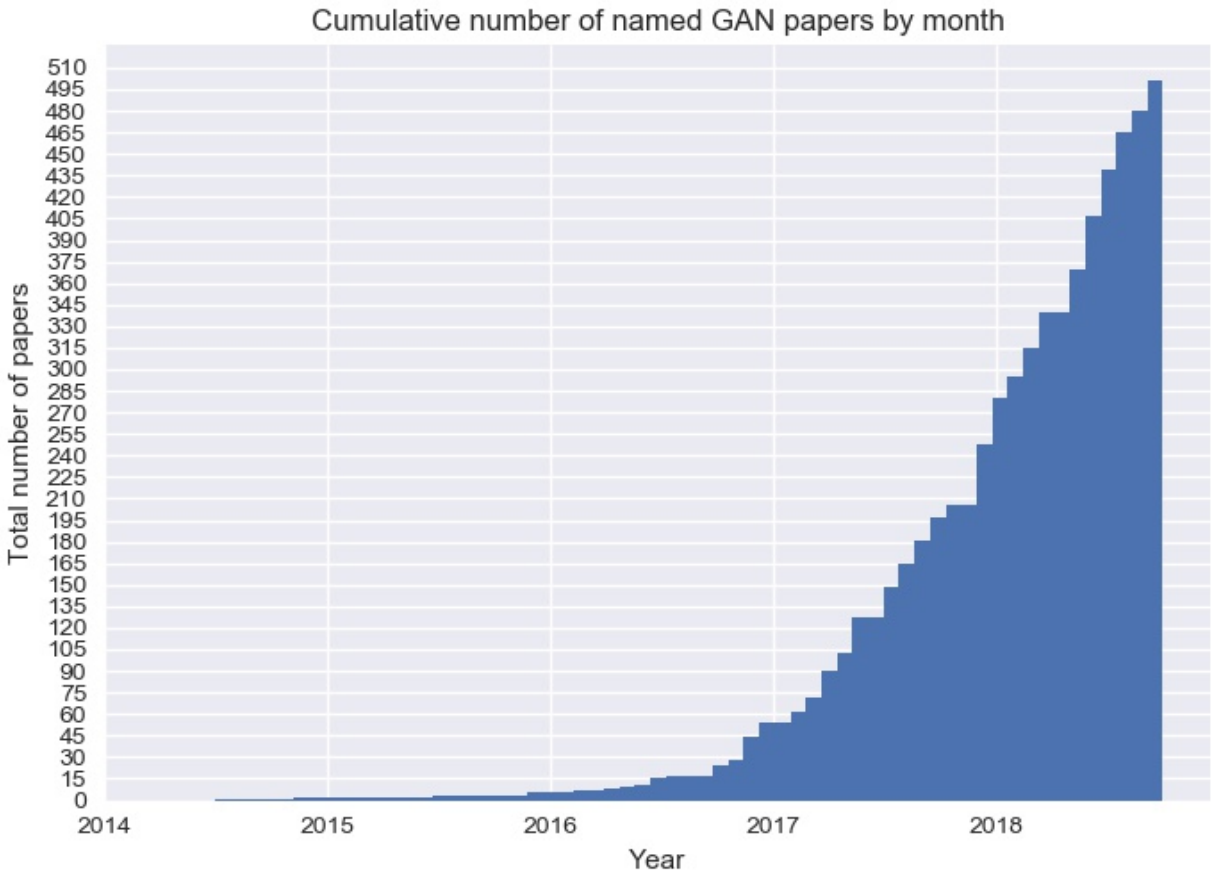
Yann LeCun, Director of AI Research at Facebook and Professor at NYU
Answered Jul 28, 2016 · Upvoted by Joaquin Quiñonero Candela, studied Machine Learning and Gokul Krishnan, M.Sc Computer Science & Machine Learning, ETH Zurich (2018)

There are many interesting recent development in deep learning, probably too many for me to describe them all here. But there are a few ideas that caught my attention enough for me to get personally involved in research projects.

The most important one, in my opinion, is adversarial training (also called GAN for Generative Adversarial Networks). This is an idea that was originally proposed by Ian Goodfellow when he was a student with Yoshua Bengio at the University of Montreal (he since moved to Google Brain and recently to OpenAI).

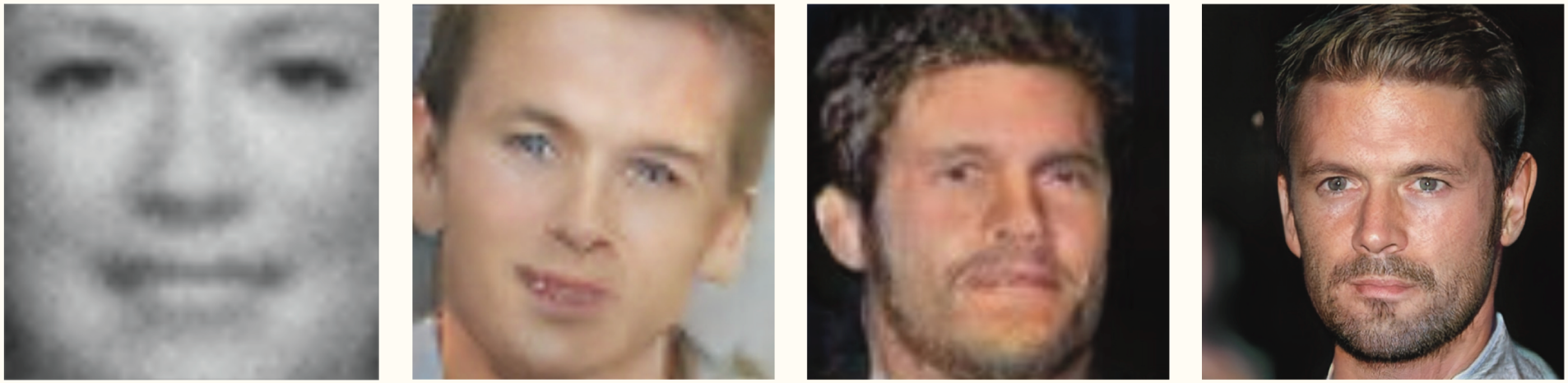
This, and the variations that are now being proposed is the most interesting idea in the last 10 years in ML, in my opinion.

Why generative models?



https://github.com/hindupuravinash/the-gan-zoo

Why generative models?

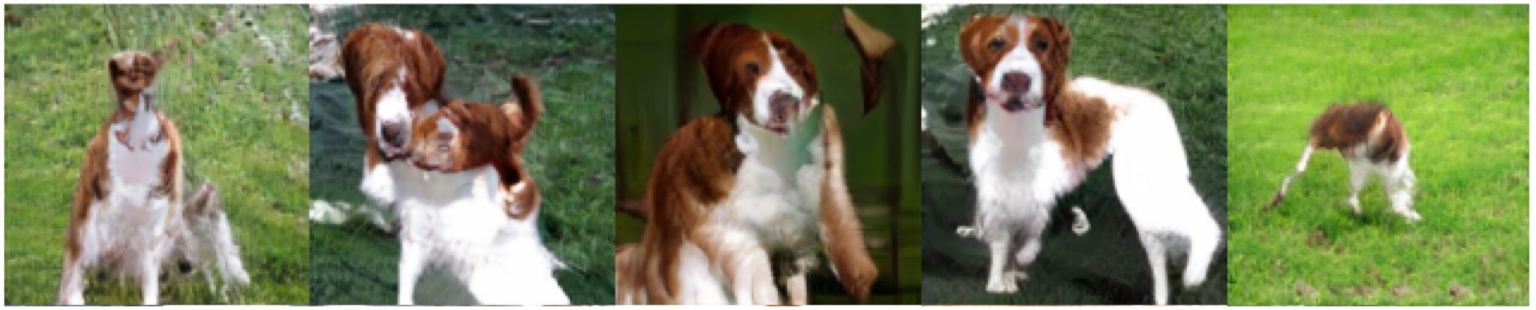


Brundage et al, 2018

Why generative models?

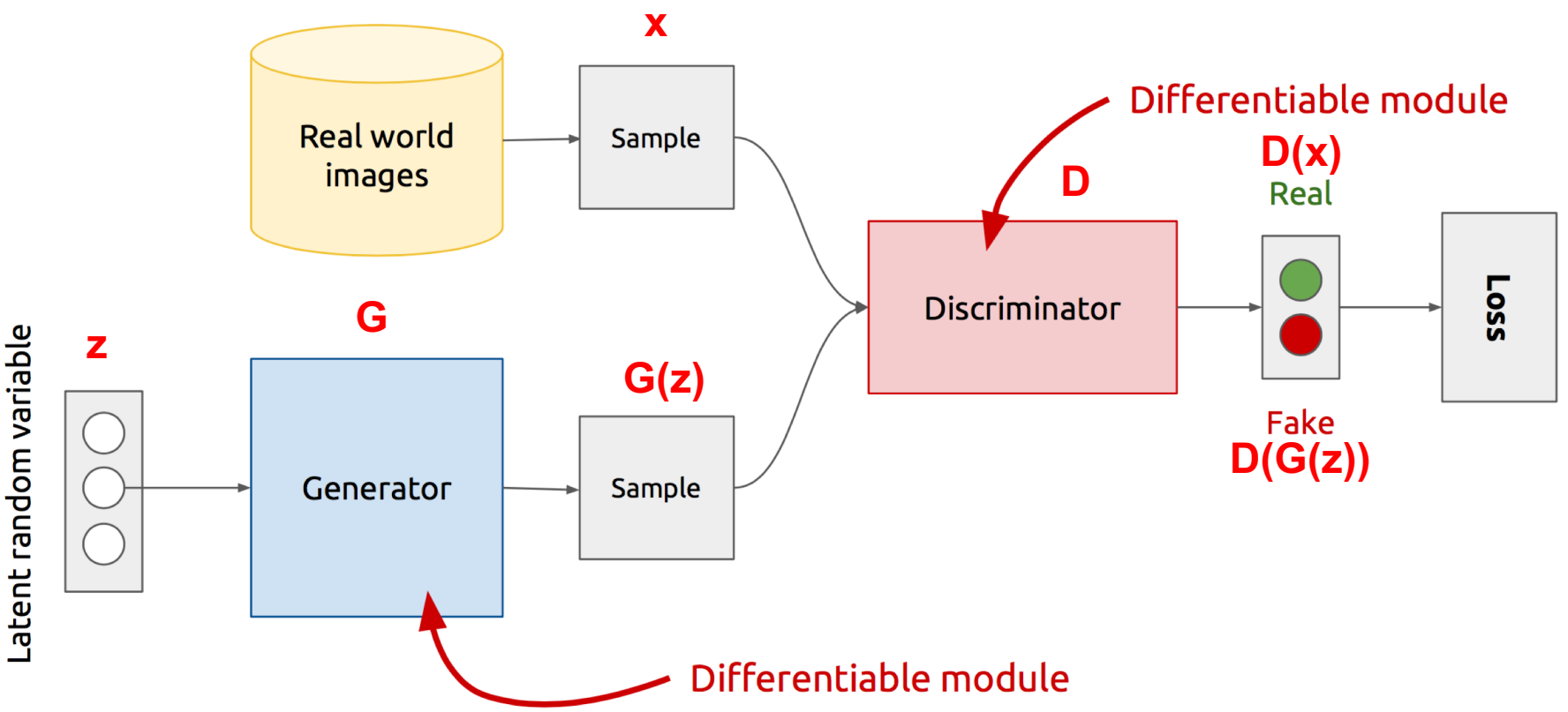
Odena et al 2016

Miyato et al 2017



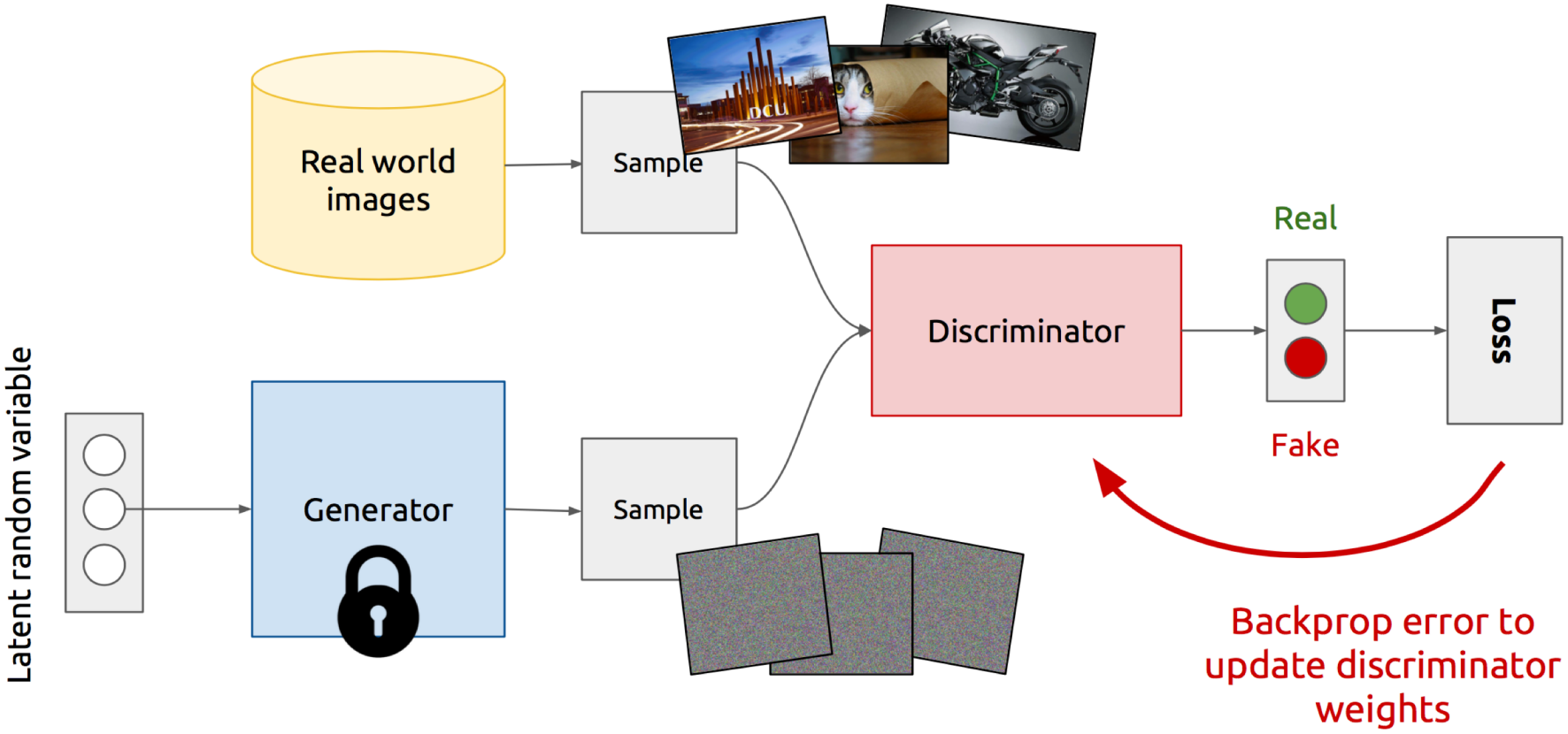
Zhang et al 2018

GAN's Architecture

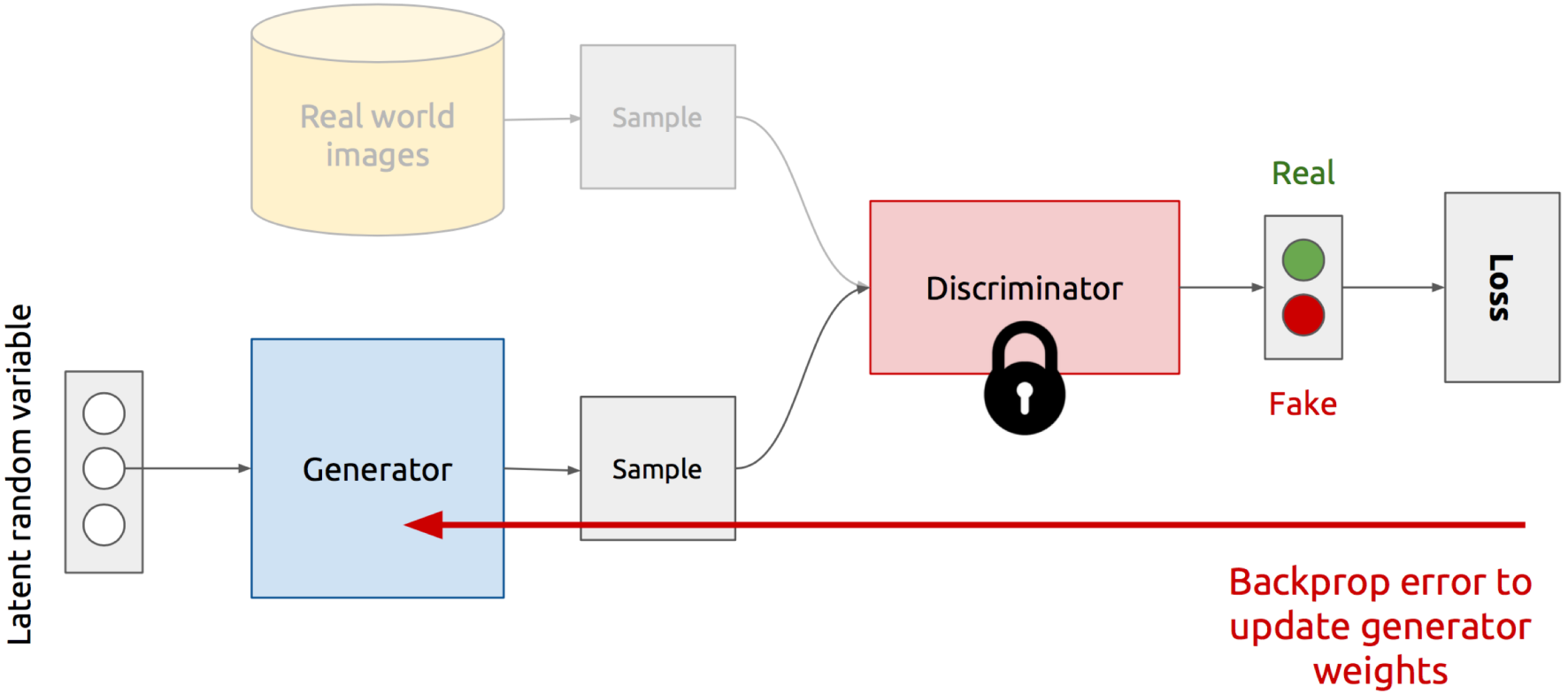


- **Z** is some random noise (Gaussian/Uniform).
- **Z** can be thought as the latent representation of the image.

GAN's Architecture

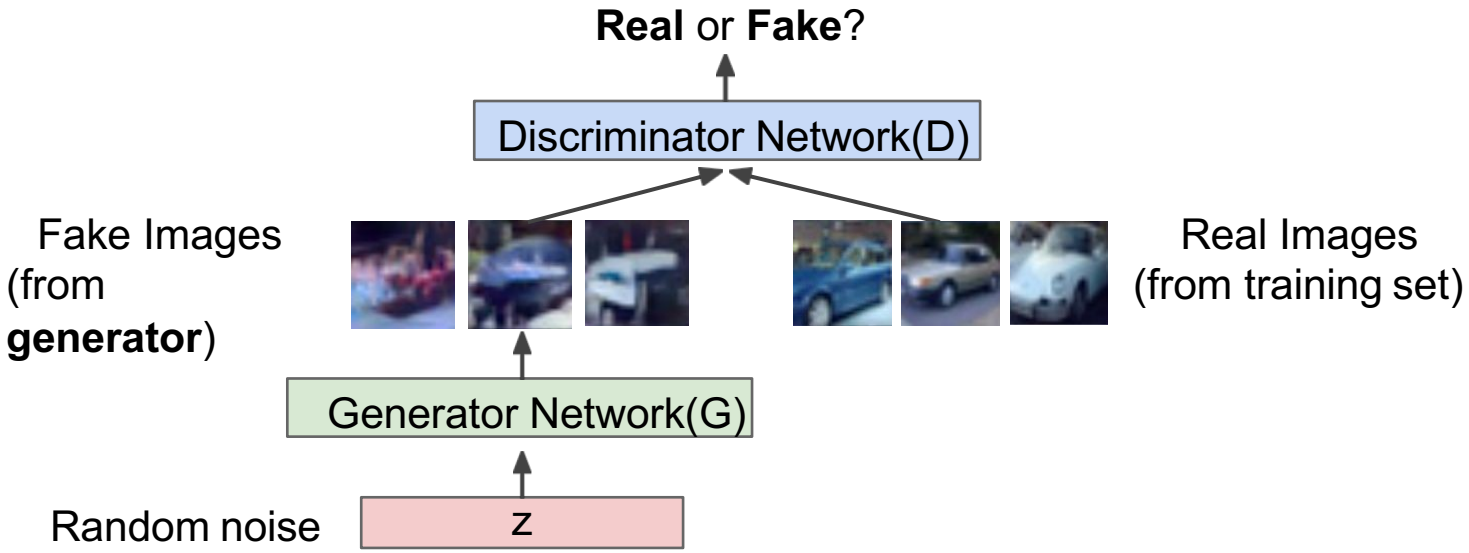


GAN's Architecture



Training GANs: Two-player game

- **Generator network (G):** tries to **fool** the **discriminator** by generating real-looking images.
- **Discriminative network (D):** tries to distinguish between **real** and **fake** images.



Training GANs: Two-player game

- **Generator network (G):** tries to **fool** the **discriminator** by generating real-looking images.
- **Discriminative network (D):** tries to distinguish between **real** and **fake** images.
- Train jointly in **minimax game**

Minimax objective function:

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

Training GANs: Two-player game

- **Generator network (G):** tries to **fool** the **discriminator** by generating real-looking images.
- **Discriminative network (D):** tries to distinguish between **real** and **fake** images.
- Train jointly in **minimax game**

Minimax objective function:

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

Discriminator output for **real** data x

Discriminator output for generated **fake** data G(z)

Training GANs: Two-player game

- **Generator network (G):** tries to **fool** the **discriminator** by generating real-looking images.
- **Discriminative network (D):** tries to distinguish between **real** and **fake** images.
- Train jointly in **minimax game**

Minimax objective function:

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

Discriminator output for **real** data x

Discriminator output for generated **fake** data G(z)

- **Discriminator** (θ_d) wants to **maximize objective** such that D(x) is close to 1 (**real**) and D(G(z)) is close to 0 (**fake**)
- **Generator** (θ_g) wants to **minimize objective** such that D(G(z)) is close to 1 (discriminator is fooled into thinking generated G(z) is real)

Training GANs: Two-player game

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

Training GANs: Two-player game

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:

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

Some find k=1 more stable, other use k>1, no best rule.

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

Training GANs: Two-player game

for number of training iterations **do**

Discriminator (D)

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

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

Loss function to maximize for the **Discriminator**

end for

Generator (G)

- 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

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

Loss function to minimize for the **Generator**

end for

Training GANs: Two-player game

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

Gradient w.r.t the parameters of the **Discriminator**

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

Loss function to maximize for the **Discriminator**

  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

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

Loss function to minimize for the **Generator**

end for

Gradient w.r.t the parameters of the **Generator**

Training GANs: Two-player game

for number of training i

for k steps **do**

- Sample minibatc … e 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:

[interpretation] compute the gradient of the loss function, and then update the parameters to min/max the loss function (gradient descent/ascent)

maximize

Gradient w.r.t the parameters of the **Discriminator**

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

Loss function to maximize for the **Discriminator**

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

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

Loss function to minimize for the **Generator**

end for

Gradient w.r.t the parameters of the **Generator**

Training GANs: Two-player game

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

Uniform noise vector (random numbers)

Real images

end for

Training GANs: Two-player game

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

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

Training GANs: Two-player game

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

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

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

Training GANs: Two-player game

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

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

Real image

Generated image

Training GANs: Two-player game

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

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*)

Training GANs: Two-player game

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)

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*)

Let's do an example

Training GANs: Two-player game

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

Gradient w.r.t the parameters of the Discriminator

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

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)

Training GANs: Two-player game

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

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

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

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

end for

Training GANs: Two-player game

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

Gradient w.r.t the parameters of the Discriminator

Average over *m* samples

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

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

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

Training GANs: Two-player game

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

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

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

end for -0.2 + -0.2 = -0.4

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 another example

Training GANs: Two-player game

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

Gradient w.r.t the parameters of the Discriminator

Average over *m* samples

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

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

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

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

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

Training GANs: Two-player game

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 $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
    - Sample minibatch of m examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
    - 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

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

Average over *m* samples

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

  end for

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

end for -0.2 + -0.2 = -0.4

Training GANs: Two-player game

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)

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

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

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

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

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

-1.6 + -1.6 = -3.2

Training GANs: Two-player game

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

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

Training GANs: Two-player game

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:

end for

Training GANs: Two-player game

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

Gradient w.r.t the parameters of the Discriminator

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

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

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:

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

end for

Training GANs: Two-player game

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

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

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

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

  end for

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

Training GANs: Two-player game

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

Gradient w.r.t the parameters of the Discriminator

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

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

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

Training GANs: Two-player game

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

Average over *m* samples

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

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

minimize

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

Training GANs: Two-player game

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

minimize

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

end for

One more example ...

Training GANs: Two-player game

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

Gradient w.r.t the parameters of the Discriminator

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

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

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

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

end for

Training GANs: Two-player game

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

Gradient w.r.t the parameters of the Discriminator

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

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

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

Training GANs: Two-player game

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

end for

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

Training GANs: Two-player game

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

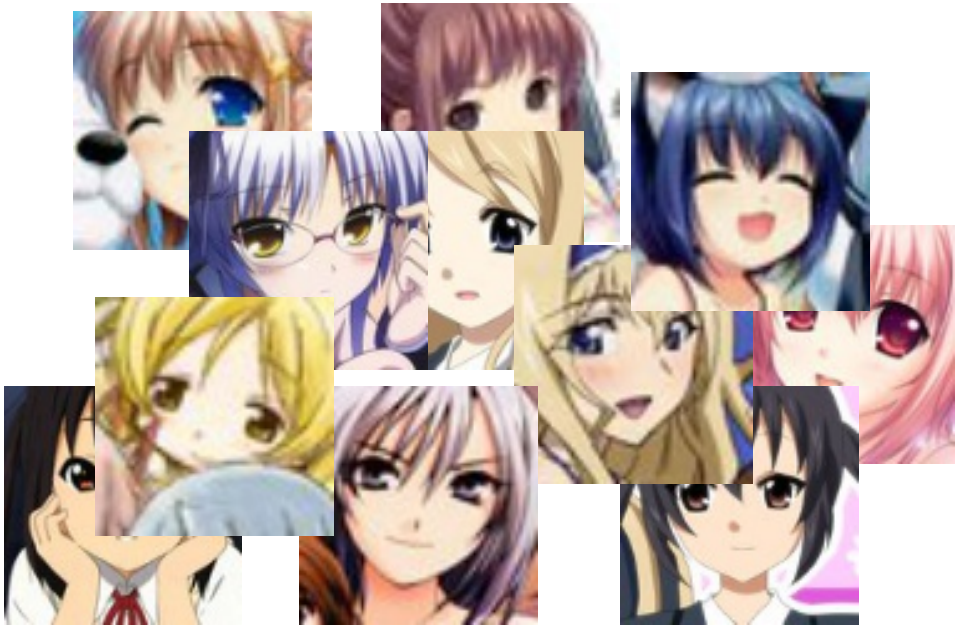
Average over *m* samples

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

minimize

Gradient w.r.t the parameters of the Generator

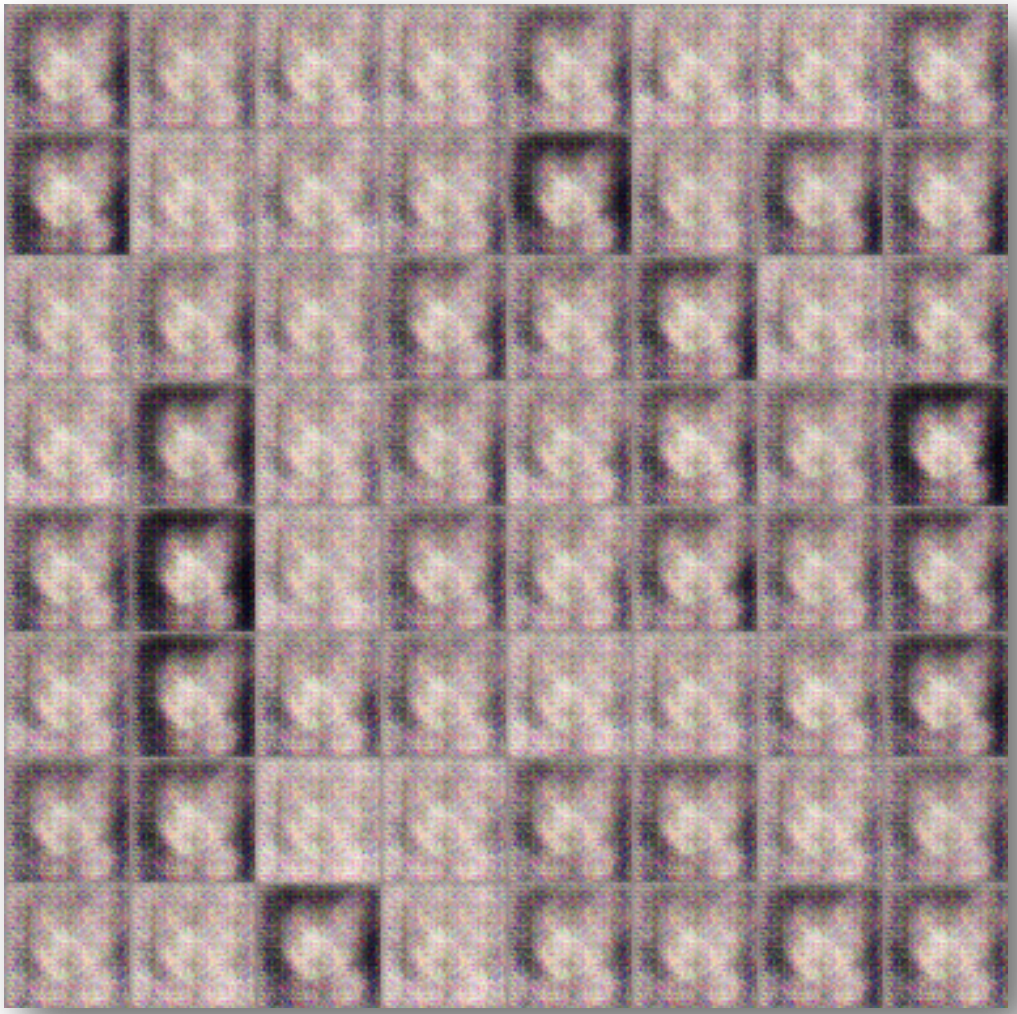
Training GANs: Iterations



Generating Anime Figures

Training GANs: Iterations

100 rounds



Training GANs: Iterations

1000 rounds

Training GANs: Iterations

2000 rounds

Training GANs: Iterations

5000 rounds

Training GANs: Iterations

10000 rounds

Training GANs: Iterations

20000 rounds

Advantages with GANs

- Generation and sampling is straightforward.
- Robust to Overfitting since Generator never sees the training data.
- Empirically, GANs are good at capturing the modes of the distribution.

Challenges with GANs

- Training is hard:
  - **Non-convergence**: the model parameters oscillate and never converge.
  - **Mode-collapse**: the generator (G) produces limited varieties of samples.
  - **Diminished-gradients**: the discriminator (D) gets too successful that the generator's gradients vanish.

Training GANs: Non-convergence

- GANs involve two (or more) players:
  - Discriminator (D) tries to maximize its reward.
  - Generator (G) tries to minimize the Discriminator's reward.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p(x)}[\log D(x)] + \mathbb{E}_{z\sim q(z)}\left[\log\left(1 - D(G(z))\right)\right]$$

  - We might not converge to the Nash equilibrium:

$$P_{data}(x) = P_{gen}(x) \;\; \forall x$$

Training GANs: Non-convergence

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

Let $V(x, y) = xy$

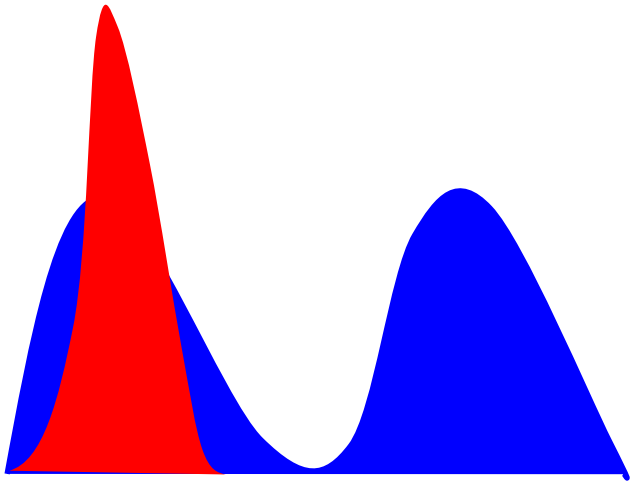
• State 1:	x > 0	y > 0	V > 0	Increase y	Decrease x
• State 2:	x < 0	y > 0	V < 0	Decrease y	Decrease x
• State 3:	x < 0	y < 0	V > 0	Decrease y	Increase x
• State 4 :	x > 0	y < 0	V < 0	Increase y	Increase x
• State 5:	x > 0	y > 0	V > 0	Increase y	Decrease x

== State 1

Training GANs: Mode-collapse

Converge to same faces

Generated Distribution

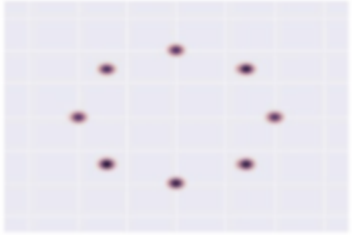


Data Distribution

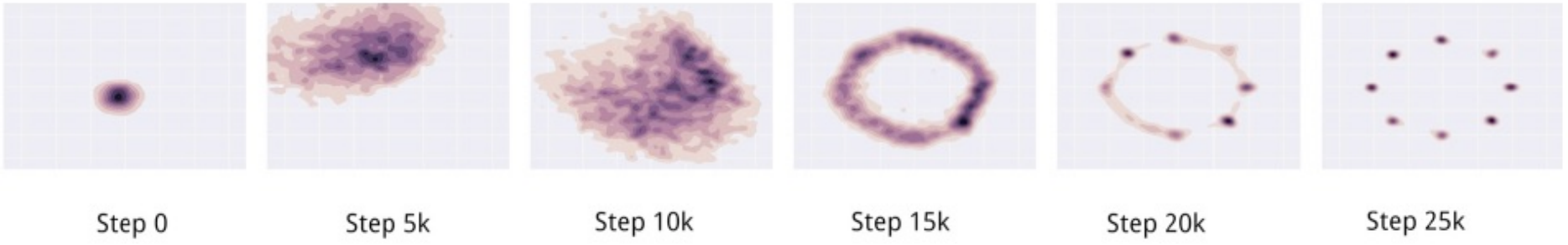
Sometimes, this is hard to tell since one sees only what's generated, but not what's missed.

Training GANs: Mode-collapse

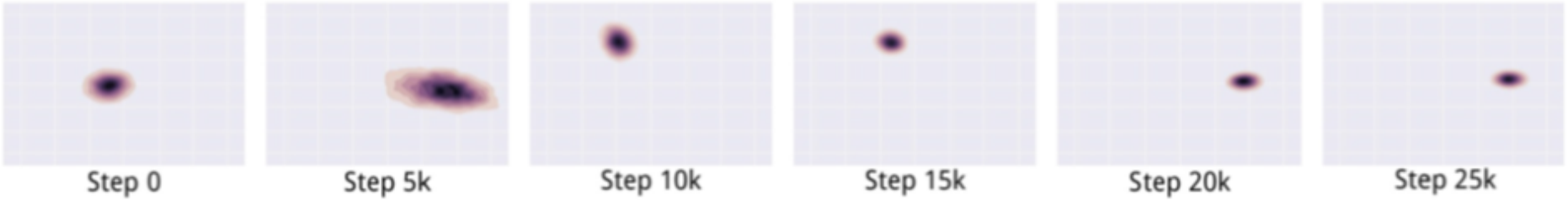
8 Gaussian distributions:



What we want …



In reality …



Training GANs: Tipps (Mini-batches)

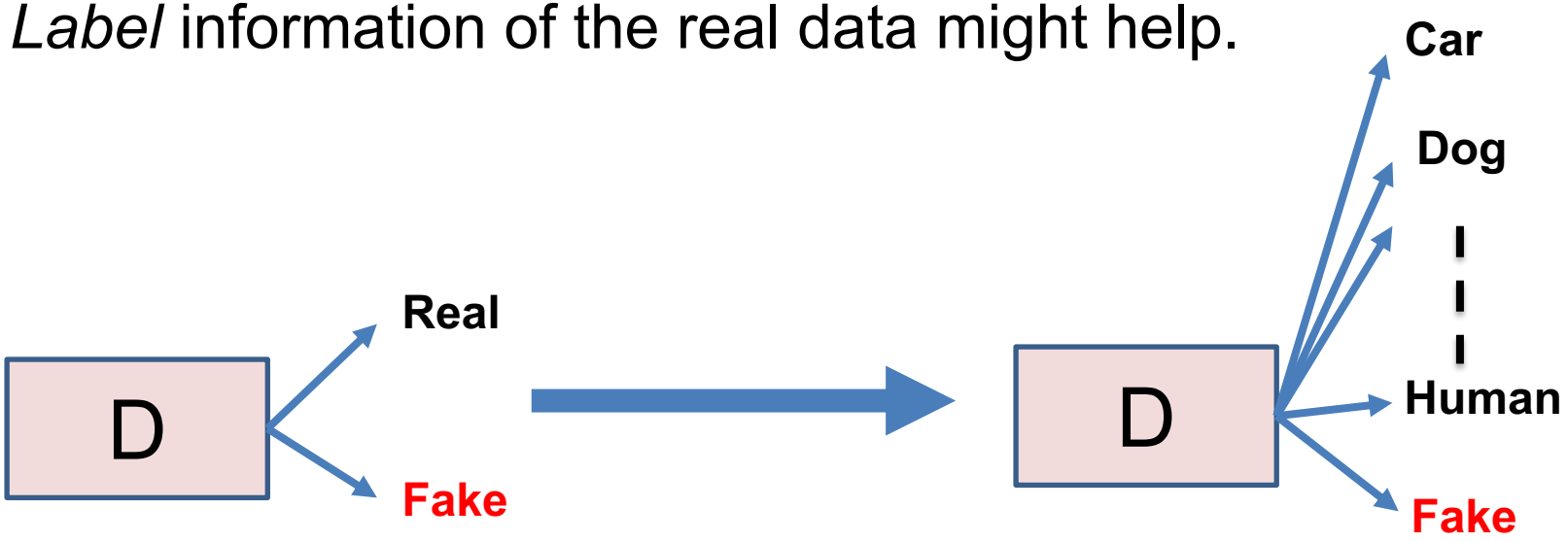
- At mode collapse:
  - Generator (G) produces good samples, but a very few of them.
  - Thus, Discriminator (D) can't tag them as fake.

- 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

Training GANs: Tipps (Multi-class)

- *Label* information of the real data might help.



- Empirically generates much better samples

Training GANs: Tipps (Loss)

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

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

$$G^* = \arg\min_G V(D,G)$$

- In this formulation, Discriminator's strategy was $D(x) \to 1$, $D(G(z)) \to 0$

- Alternatively, we can flip the binary classification labels i.e. **Fake = 1**, **Real = 0**

$$V(D,G) = \mathbb{E}_{x\sim p(x)}[\log(1 - D(x))] + \mathbb{E}_{z\sim q(z)}[\log(D(G(z)))]$$

- In this new formulation, Discriminator's strategy will be $D(x) \to 0$, $D(G(z)) \to 1$

Training GANs: Tipps (Loss)

- If all we want to encode is $D(x) \rightarrow 0, \quad D\big(G(z)\big) \rightarrow 1$

$$D^* = argmax_D \, \mathbb{E}_{x \sim p(x)} \big[\log\big(1 - D(x)\big)\big] + \mathbb{E}_{z \sim q(z)} \big[\log\big(D(G(z))\big)\big]$$

We can use this

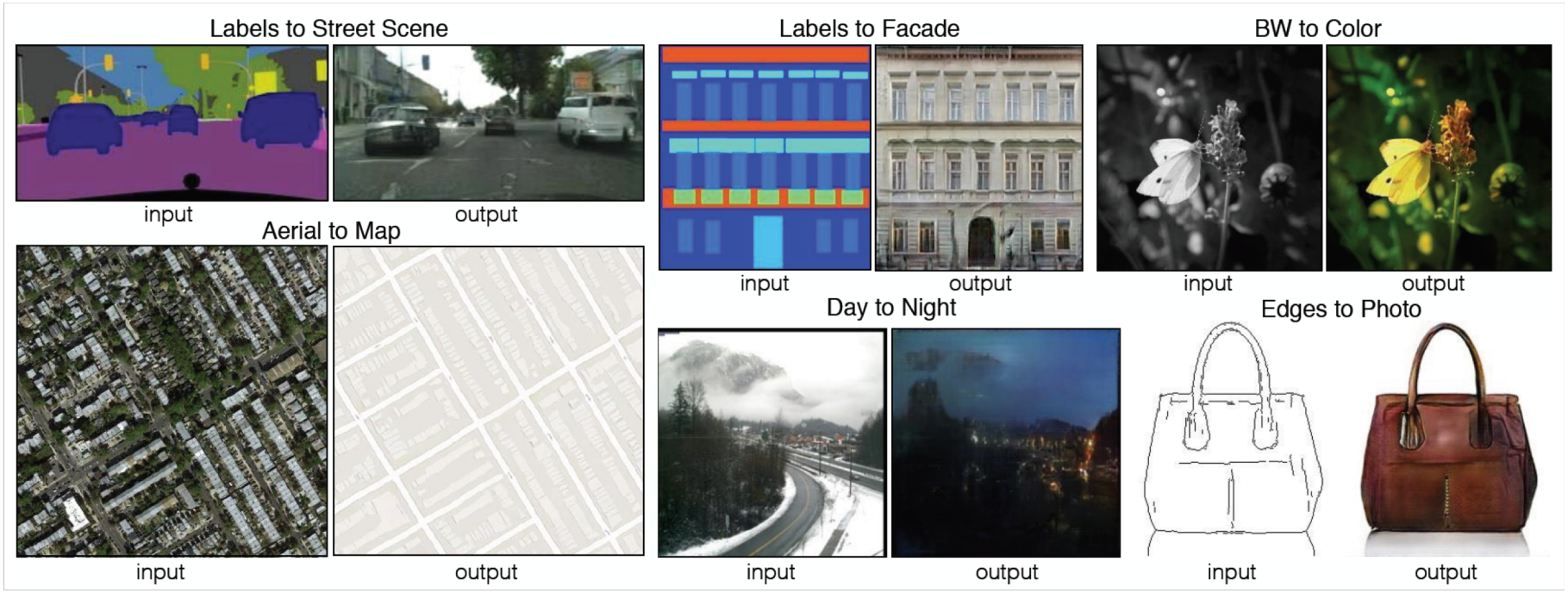
$$D^* = argmin_D \, \mathbb{E}_{x \sim p(x)} \log\big(D(x)\big) + \mathbb{E}_{z \sim q(z)} \left[\log\left(1 - D\big(G(z)\big)\right)\right]$$

- Now, we can replace cross-entropy with any loss function **(Hinge Loss)**

$$D^* = argmin_D \, \mathbb{E}_{x \sim p(x)} D(x) + \mathbb{E}_{z \sim q(z)} \max\left(0, m - D\big(G(z)\big)\right)$$

- And thus, instead of outputting probabilities, Discriminator just has to output:
  - High values for fake samples
  - Low values for real samples

Image-To-Image Translation

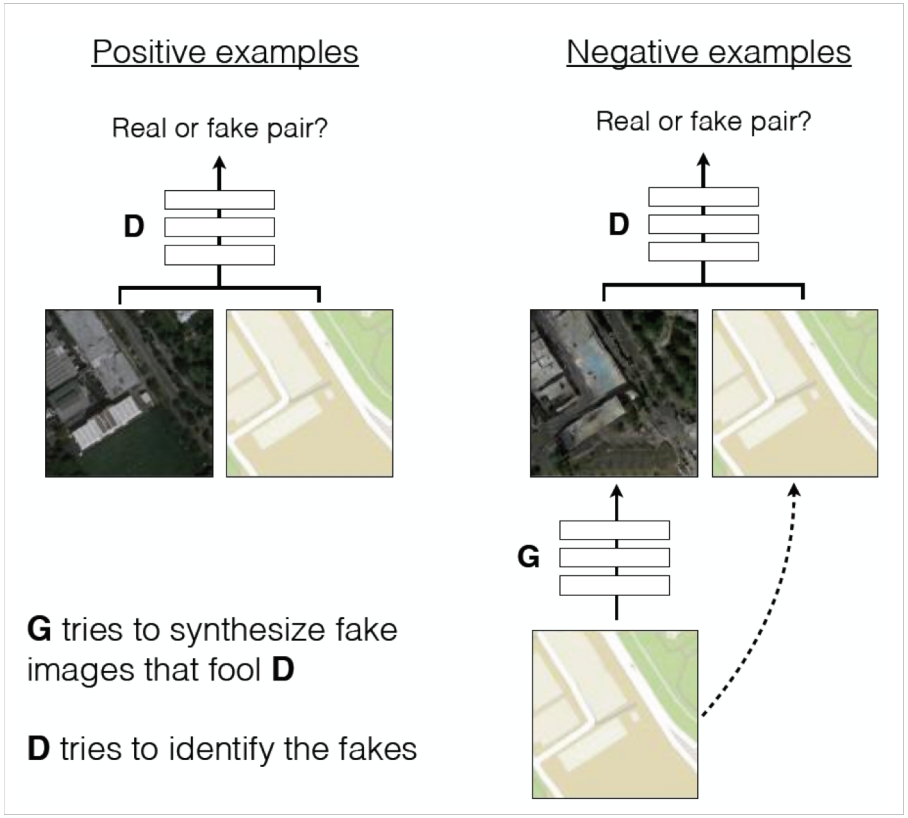


[Link to an interactive demo of this paper]

Isola, P., Zhu, J. Y., Zhou, T., & Efros, A. A. "**Image-to-image translation with conditional adversarial networks**". arXiv preprint arXiv:1611.07004. (2016).

Image-To-Image Translation

- **Architecture**
  - DCGAN-based.
- **Training:**
  - Conditioned on images from the source domain.
- **Benefits:**
  - Effective way of handling different domains.
  - No need of **structural loss**.



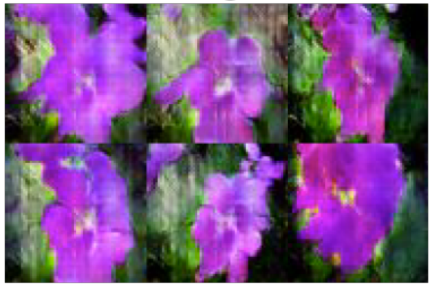
Text-To-Image Synthesis

- **Motivation**
  - Given a text description, generate closely related images.
- **Training:**
  - Use CGAN with **D** and **G** conditioned on dense “text” embedding.

this small bird has a pink breast and crown, and black primaries and secondaries.

this magnificent fellow is almost all black with a red crest, and white cheek patch.

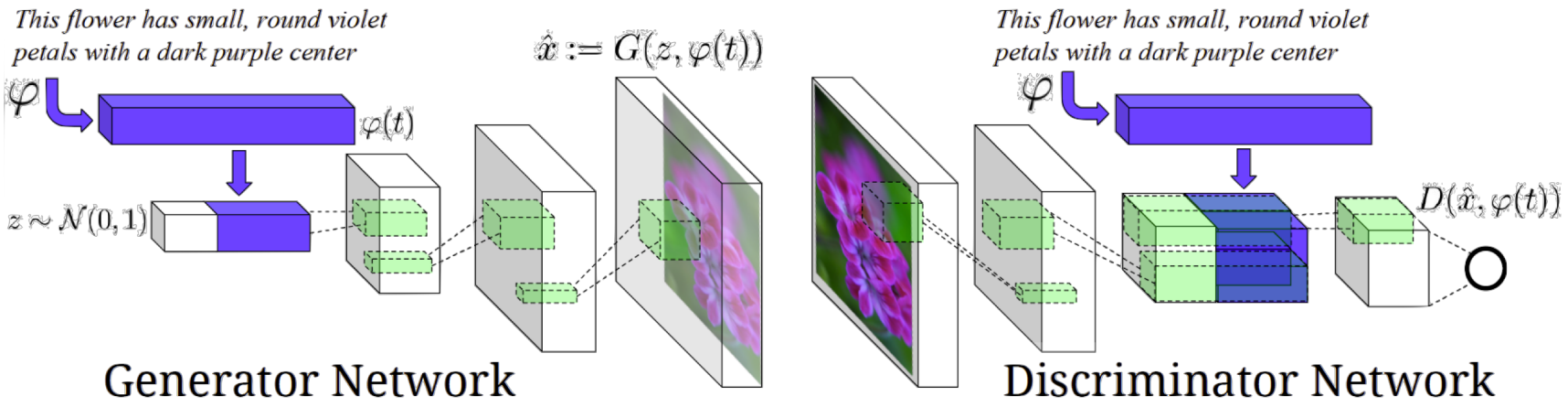
the flower has petals that are bright pinkish purple with white stigma



this white and yellow flower have thin white petals and a round yellow stamen

Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., & Lee, H. “**Generative adversarial text to image synthesis**”. ICML (2016).

Text-To-Image Synthesis



Positive Samples:

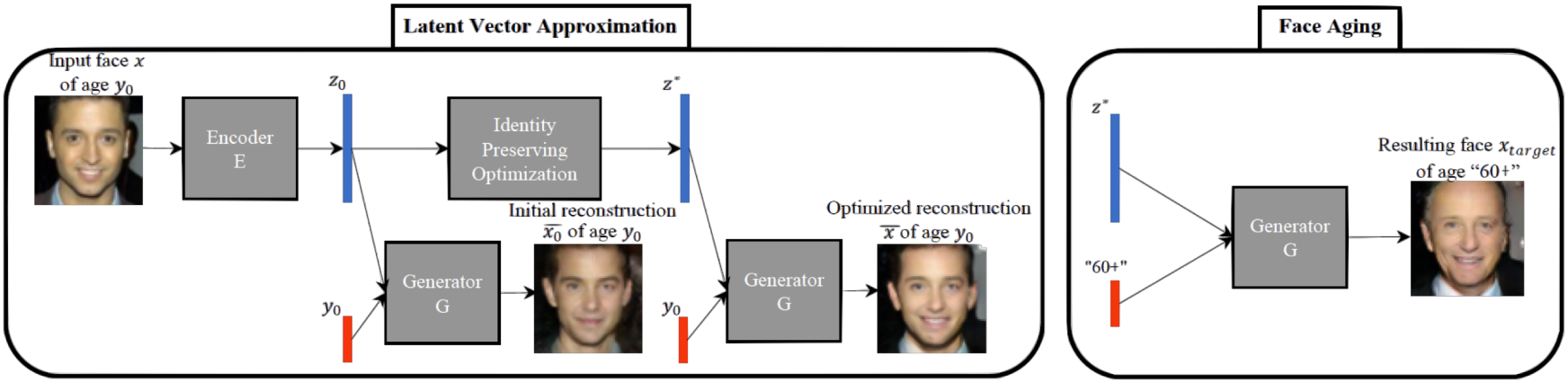
{**Real** Image, **Right** Text}

Negative Samples:

{**Real** Image, **Wrong** Text}

{**Fake** Image, **Right** Text}

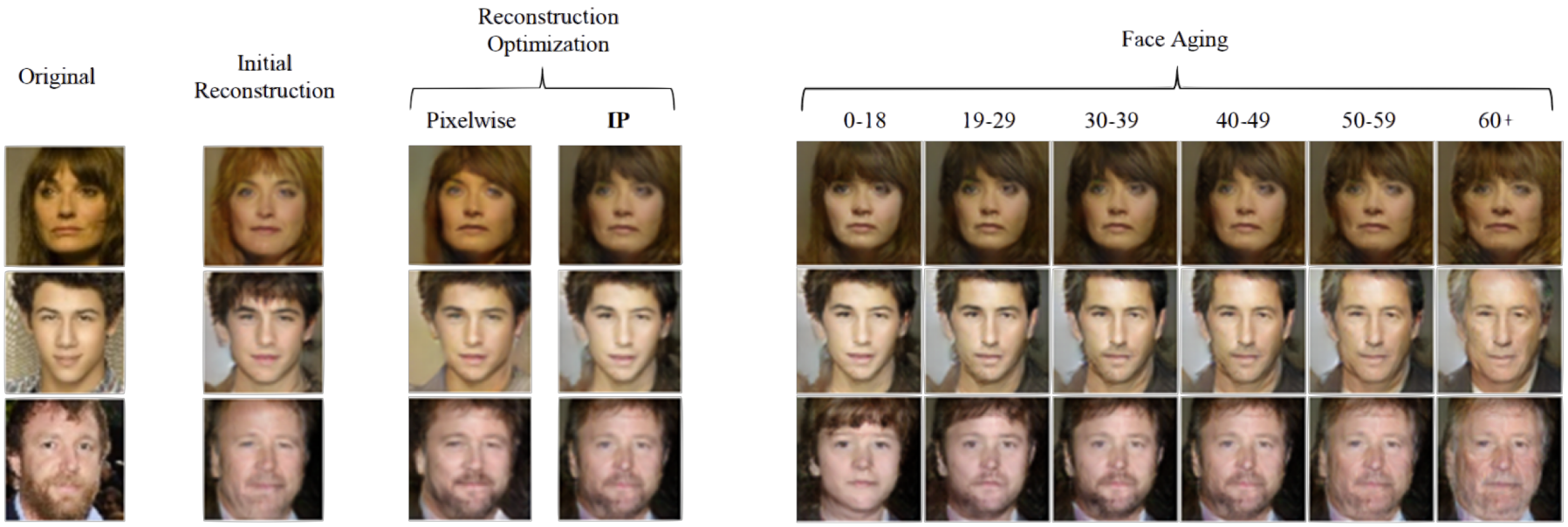
Facial Aging



- **Architecture:**
  - Given a text description, generate closely related images.
- **Training:**
  - Latent code conditioned on a discrete embedding of age categories.

Antipov, G., Baccouche, M., & Dugelay, J. L. (2017). "**Face Aging With Conditional Generative Adversarial Networks**". arXiv preprint arXiv:1702.01983.

Facial Aging



Generative Adversarial Networks (GANs)

Javier Montoya (montoya@ethz.ch)

Arequipa, 9th January 2019