Generative Adversarial Networks (GAN)

Sep. 15th 2018

Stat 991

Claim

- **This set of slides is modified from CS231n Stanford University.**
- **This is only for educational purposes.**

Overview

- **Generative Models**
- **Generative Adversarial Networks (GAN)**
  - **Two-player game**
  - **Deep convolution GAN**
  - **Vector representation in GAN**
  - **Demo**
    - **Vanilla GAN**
    - **DCGAN**
    - **Wasserstein GAN**

Generative Models

- Given training data, generate new samples from same distribution

Training data $\sim p_{data}(x)$

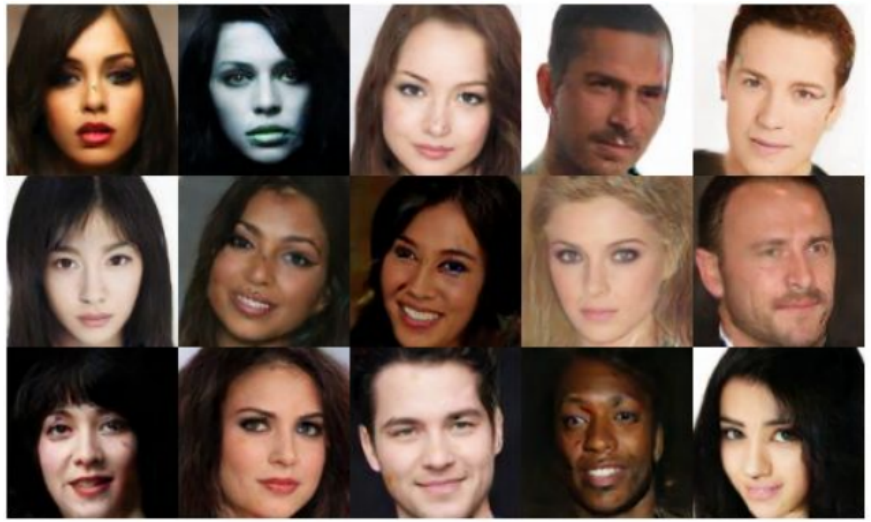
Generated samples $\sim p_{model}(x)$

Want to learn $p_{model}(x)$ similar to $p_{data}(x)$

- Addresses density estimation, a core problem in unsupervised learning
- **Several flavors:**
  - Explicit density estimation: explicitly define and solve for $p_{model}(x)$
  - Implicit density estimation: learn model that can sample from $p_{model}(x)$ w/o explicitly defining it

Why Generative Models?

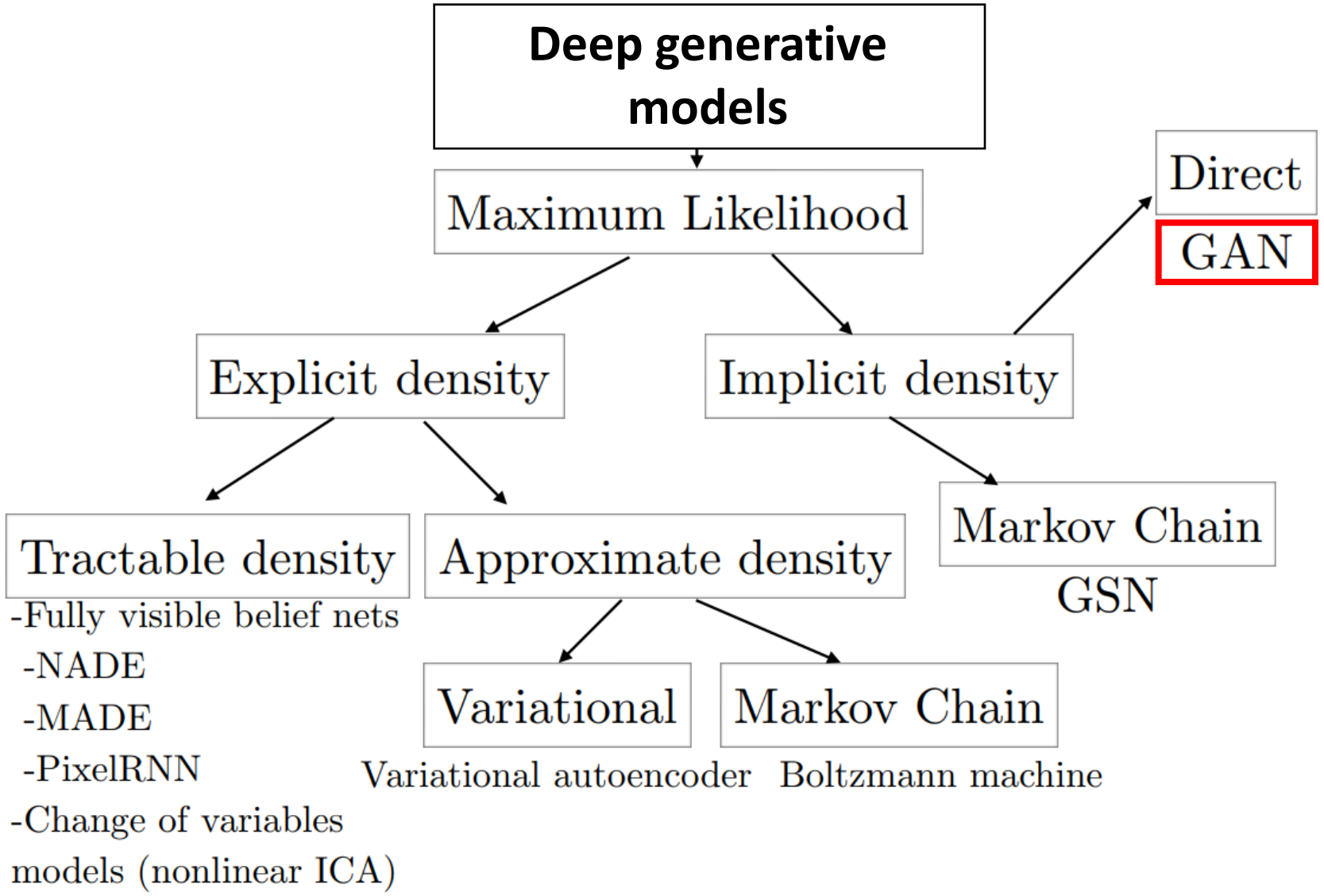
- Realistic samples for artwork, super-resolution, colorization, etc.



- Generative models of time-series data can be used for simulation and planning (reinforcement learning applications!)
- Training generative models can also enable inference of latent representations that can be useful as general features

Figures from L-R: (1) Alec Radford et al. 2016; (2) David Berthelot et al. 2017; Phillip Isola et al. 2017.

Taxonomy of Generative Models



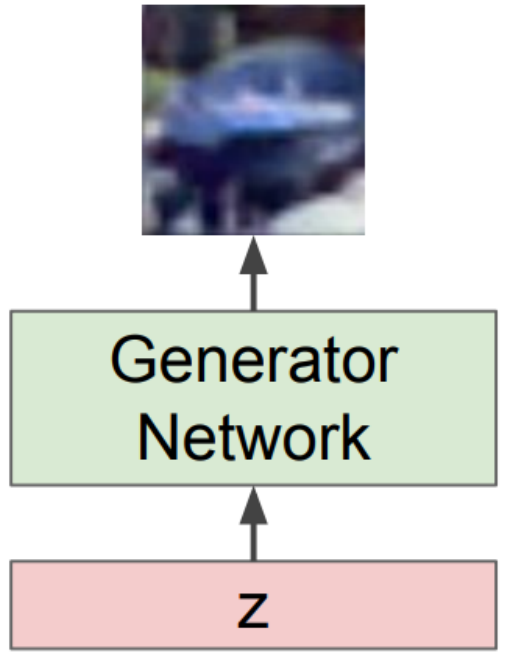
Adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Generative Adversarial Networks (GAN)

- Give up on explicitly modeling density, and just want ability to sample.
- GAN: don’t work with any explicit density function!
- Instead, take game-theoretic approach: learn to generate from training distribution through 2-player game

Generative Adversarial Networks (GAN)

- Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!
- Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.
- Q: What can we use to represent this complex transformation?
- A: A neural network!

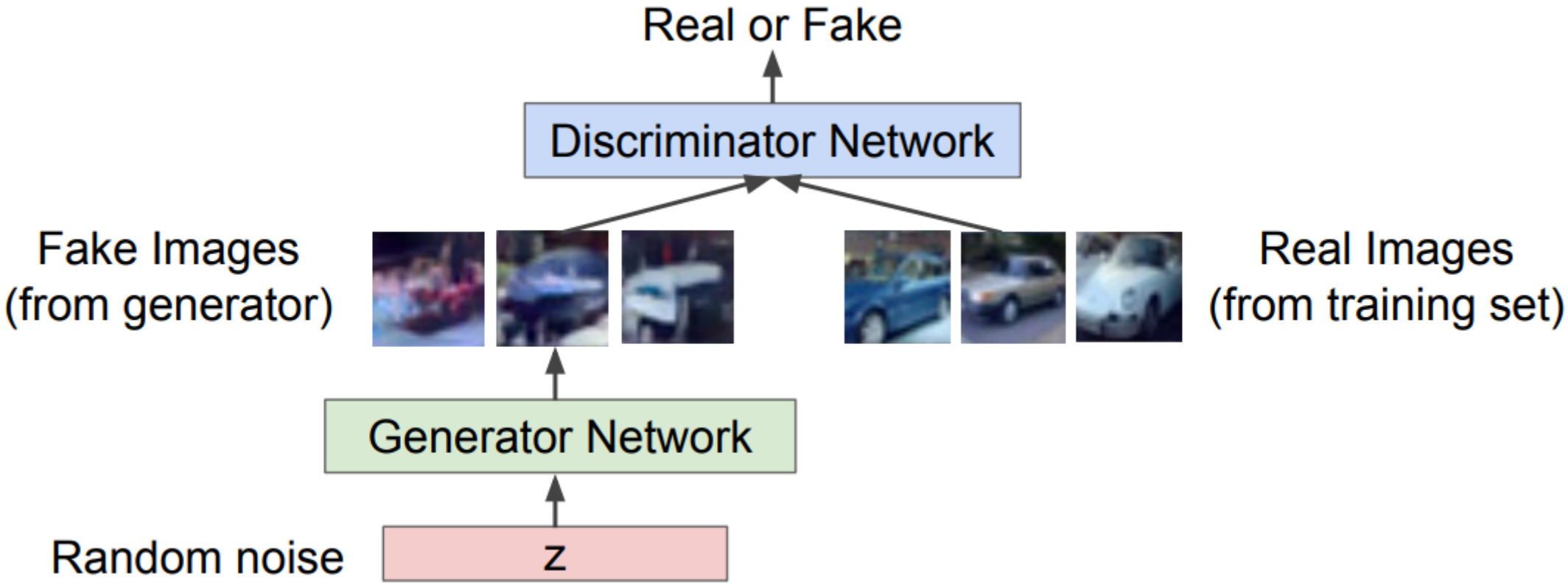


Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Training GANs: Two-player game

Figures from Emily Denton et al. 2015.

- **Generator network:** try to fool the discriminator by generating real-looking images
- **Discriminator network:** try to distinguish between real and fake images



Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

- **Generator network:** try to fool the discriminator by generating real-looking images
- **Discriminator network:** try to distinguish between real and fake images
- Train jointly **in minimax game**

Minimax objective function:

Discriminator outputs likelihood in (0,1) of real image

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

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

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Minimax objective function:

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

Alternate between

1. **Gradient ascent** on discriminator

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

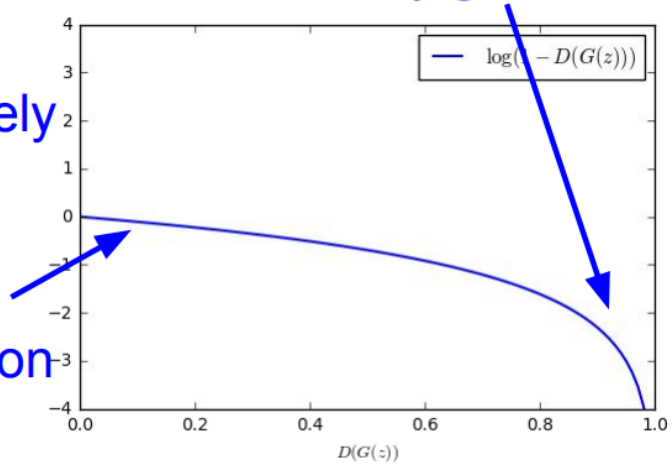
2. **Gradient descent** on generator

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

In practice, optimizing this generator objective does not work well!

Gradient signal dominated by region where sample is already good

When sample is likely fake, want to learn from it to improve generator. But gradient in this region is relatively flat!



Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Minimax objective function:

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

Alternate between

1. **Gradient ascent** on discriminator

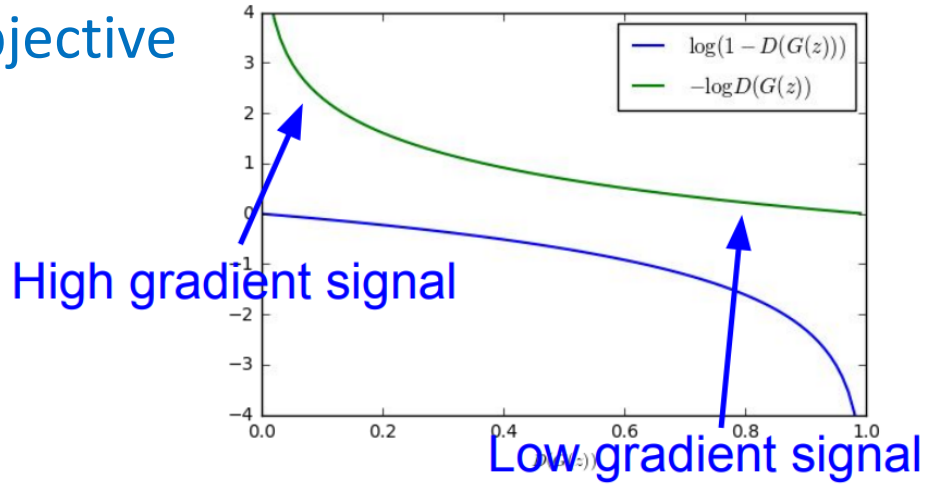
$$\max_{\theta_d} \left[\boldsymbol{E}_{x \sim p_{data}} log D_{\theta_d}(x) + \boldsymbol{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Instead: Gradient ascent** on generator, different objective

$$\max_{\theta_g} \boldsymbol{E}_{z \sim p(x)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

Now maximize likelihood of discriminator being wrong. Same objective of fooling discriminator, but now higher gradient signal for bad samples. **Standard in practice.**

Aside: Jointly training two networks is challenging, can be unstable. Choosing objectives with better loss landscapes helps training. For example, LSGAN & Wasserstein GAN



Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Putting it together: GAN training algorithm

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:

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

D

  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:

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

G

  end for

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

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

Recent work (e.g. Wasserstein GAN) alleviates this problem, better stability!

Improved Wasserstein GAN (better stablization)

Problem with current GAN

- Loss is not correlated with convergence
- Unstable
- Mode collapse
- In general P_{data} and P_{model} unlikely to have non-negligible intersection

Solve: Wasserstein GAN

- Balancing generator and discriminator. This gives a lower bound on loss and avoids collapse
- Apply random noise to real samples – creates an intersections

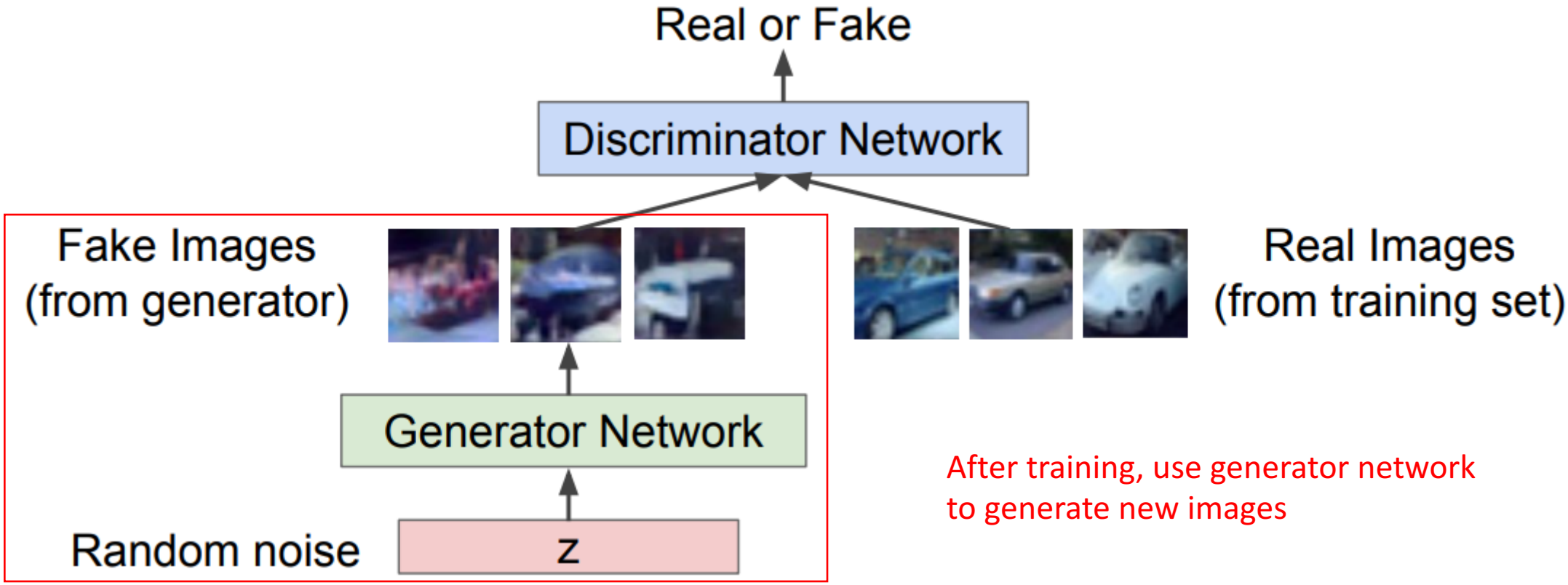
50K steps 100k steps

Gulrajani, Ishaan, et al. "Improved training of wasserstein GANs."

Training GANs: Two-player game

Figures from Emily Denton et al. 2015.

- **Generator network:** try to fool the discriminator by generating real-looking images
- **Discriminator network:** try to distinguish between real and fake images

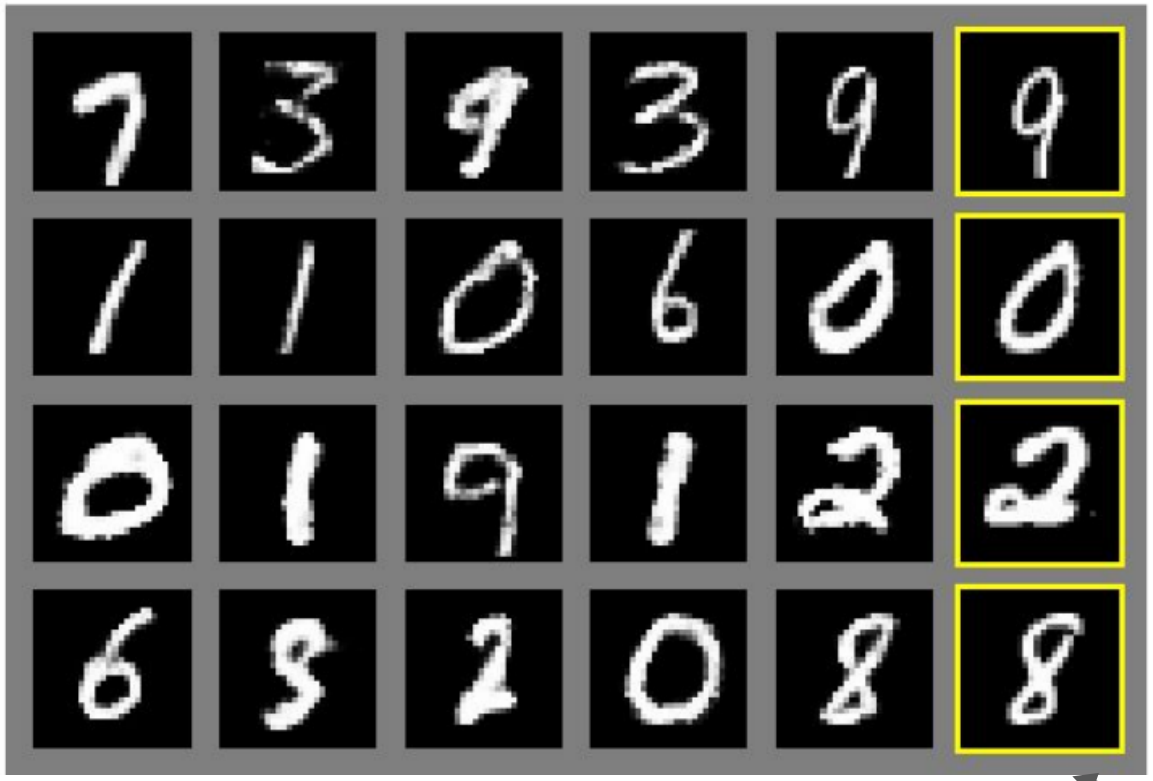


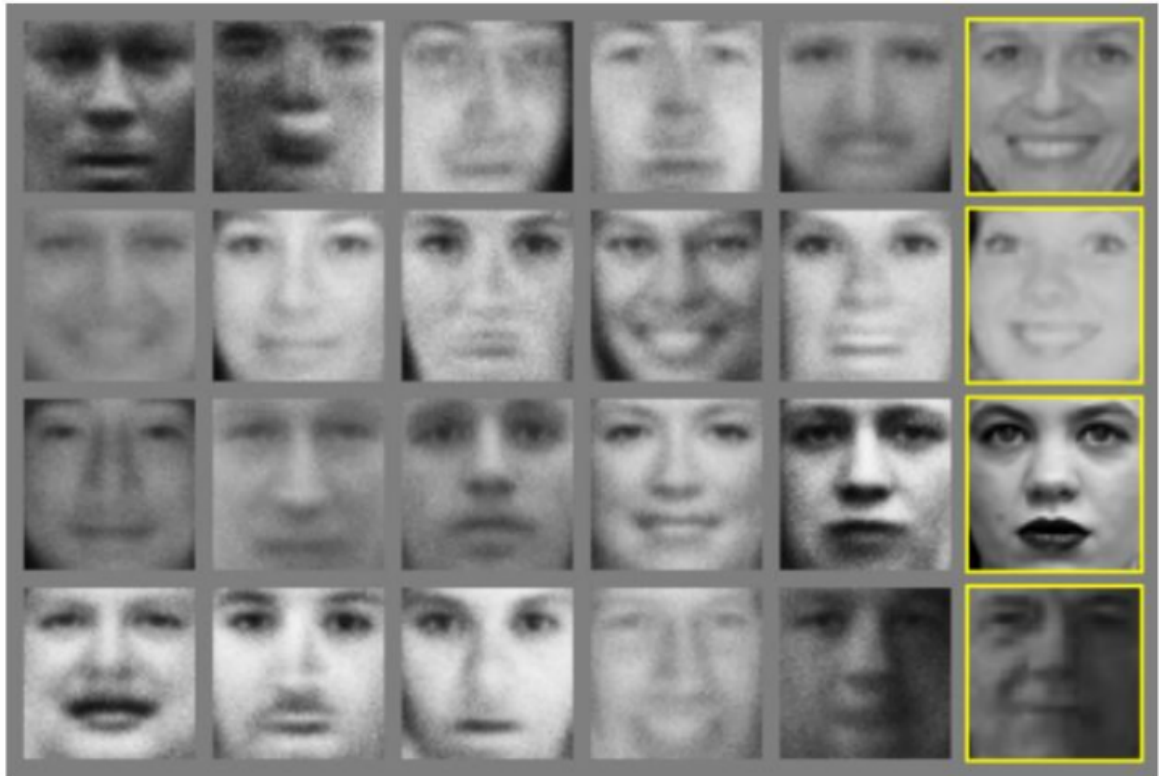
After training, use generator network to generate new images

Generative Adversarial Nets

Ian Goodfellow et al.,
"Generative Adversarial Nets",
NIPS 2014

Generated samples





Nearest neighbor from training set

Generative Adversarial Nets

Ian Goodfellow et al.,
"Generative Adversarial Nets",
NIPS 2014

Generated samples

Nearest neighbor from training set

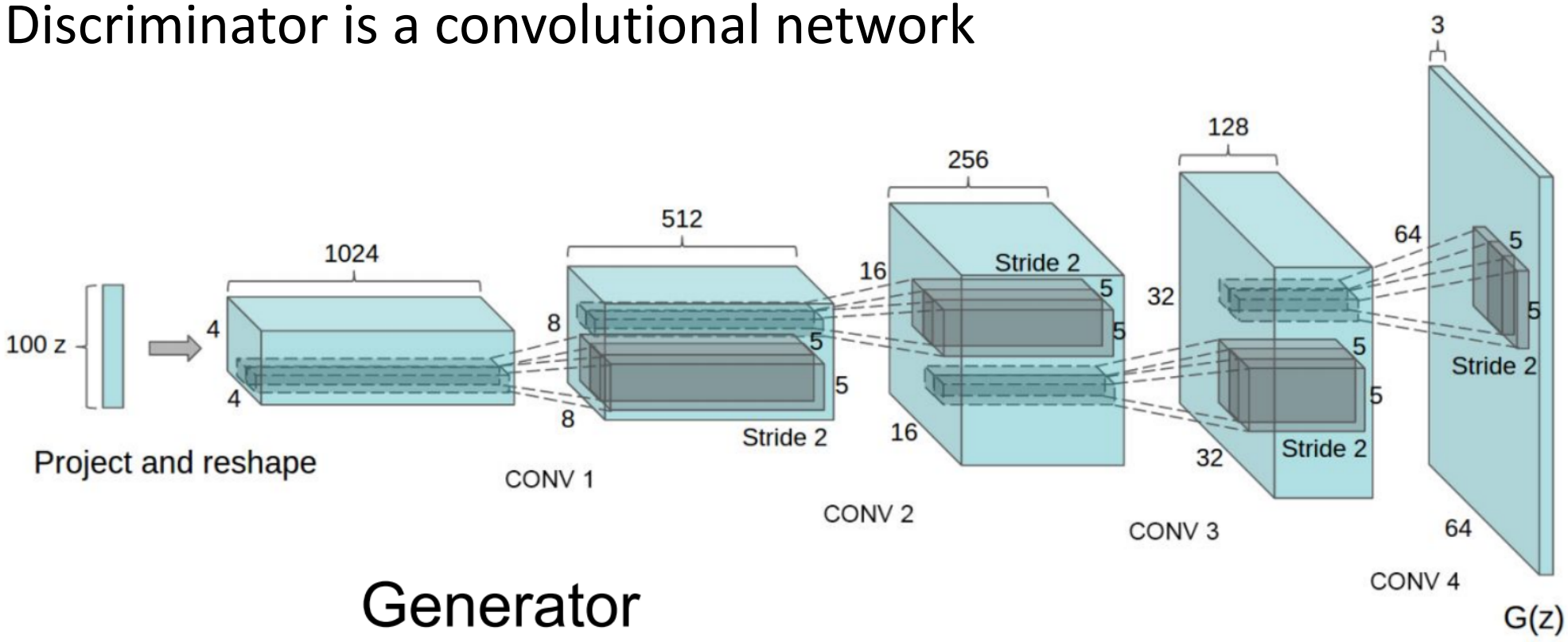
Generative Adversarial Nets: Convolutional Architectures

- **Generator is an upsampling network with fractionally-strided convolutions**
- **Discriminator is a convolutional network**
- Architecture guidelines
  - Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator)
  - Use **batchnorm** in both the generator and the discriminator
    - Reduce “internal covariate shift” X
    - significantly smooth optimization landscape
  - Remove fully connected hidden layers for deeper architectures
  - Use ReLU activation in generator for all layers except for the output, which uses Tanh
  - Use **LeakyReLU** activation in the discriminator for all layers
    - allow gradients to flow backwards through the layer unimpeded

Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

Generative Adversarial Nets: Convolutional Architectures

- Generator is an upsampling network with fractionally-strided convolutions
- Discriminator is a convolutional network



Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

Generative Adversarial Nets: Convolutional Architectures

Samples from the model look much better!

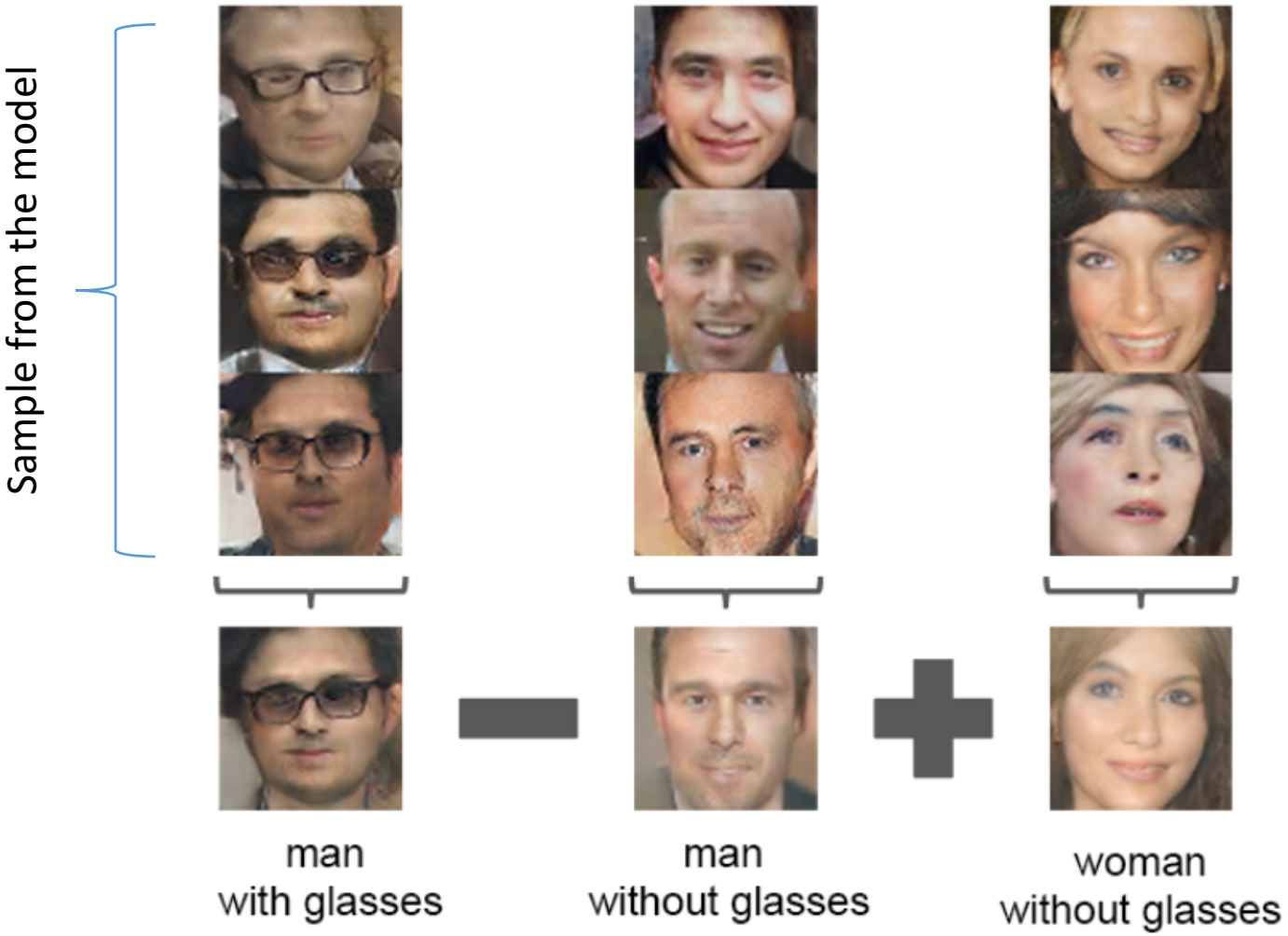
Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

Generative Adversarial Nets: Convolutional Architectures

Interpolating between random points in latent space

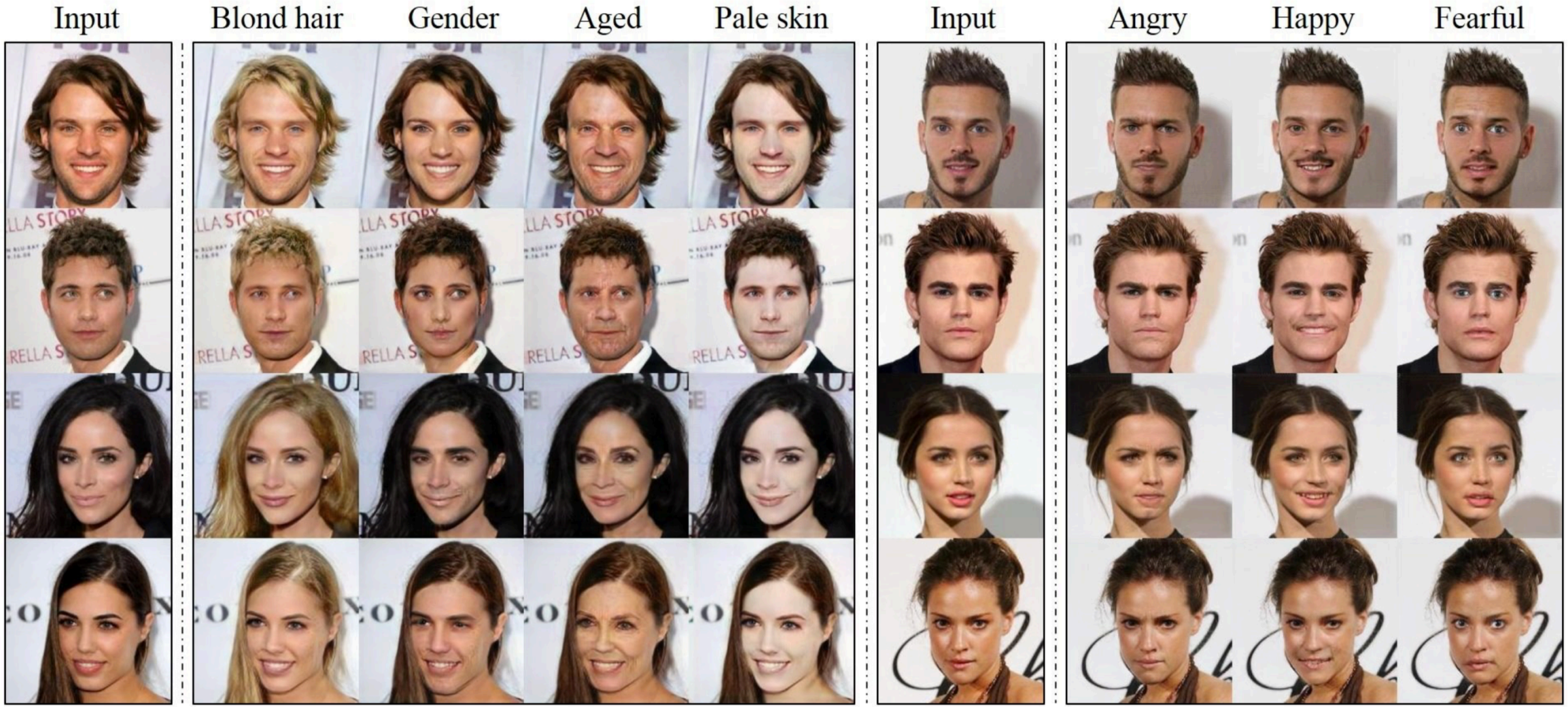
Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

Generative Adversarial Nets: Interpretable Vector Math



Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

Generative Adversarial Nets: Conditional GAN



Explosion of GANs

Source->Target domain transfer

CycleGAN. Zhu et al. 2017.

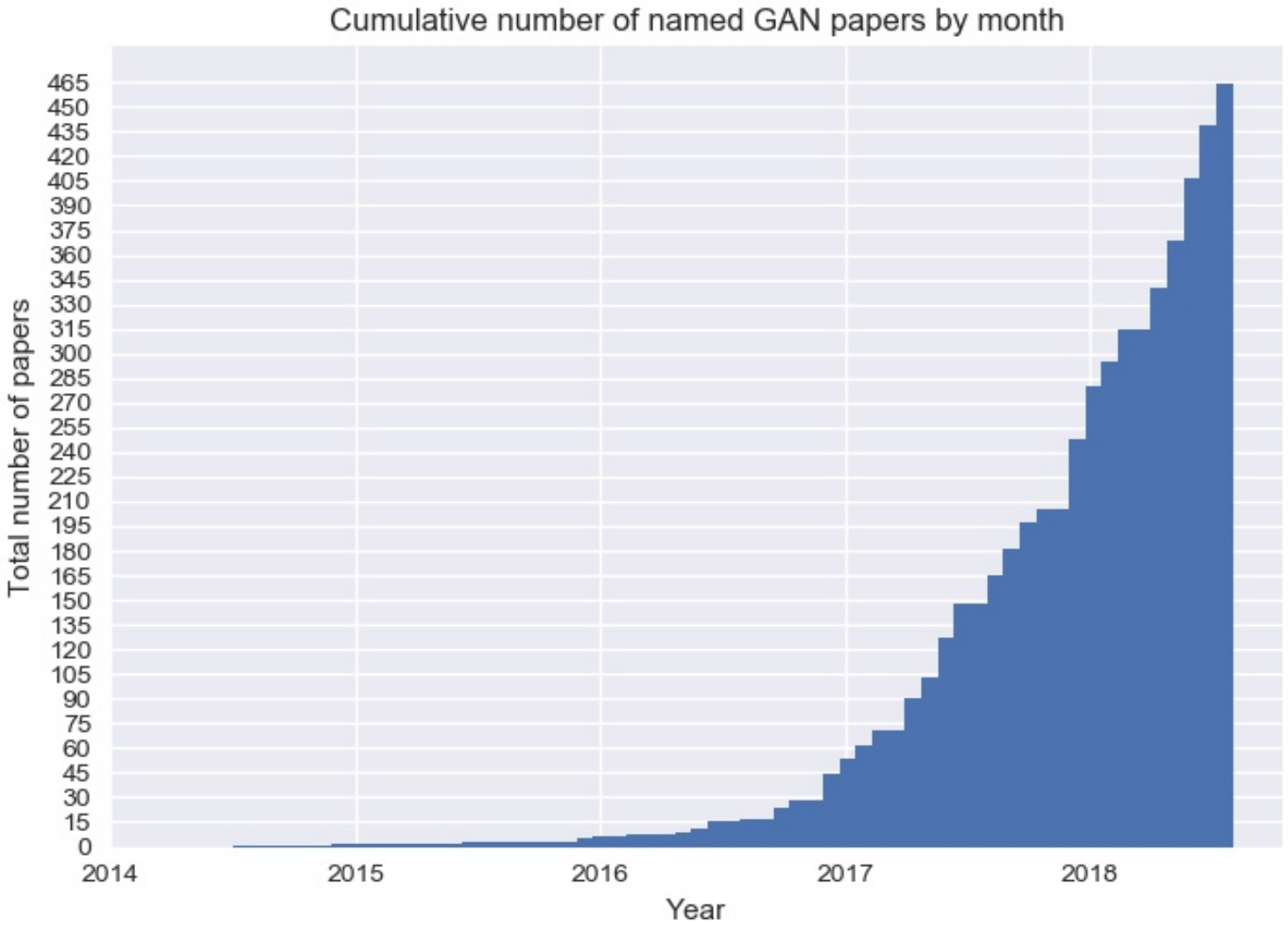
Text -> Image Synthesis

Reed et al. 2017.

Many GAN applications

Pix2pix. Isola 2017. Many examples at https://phillipi.github.io/pix2pix/

Explosion of GANs



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

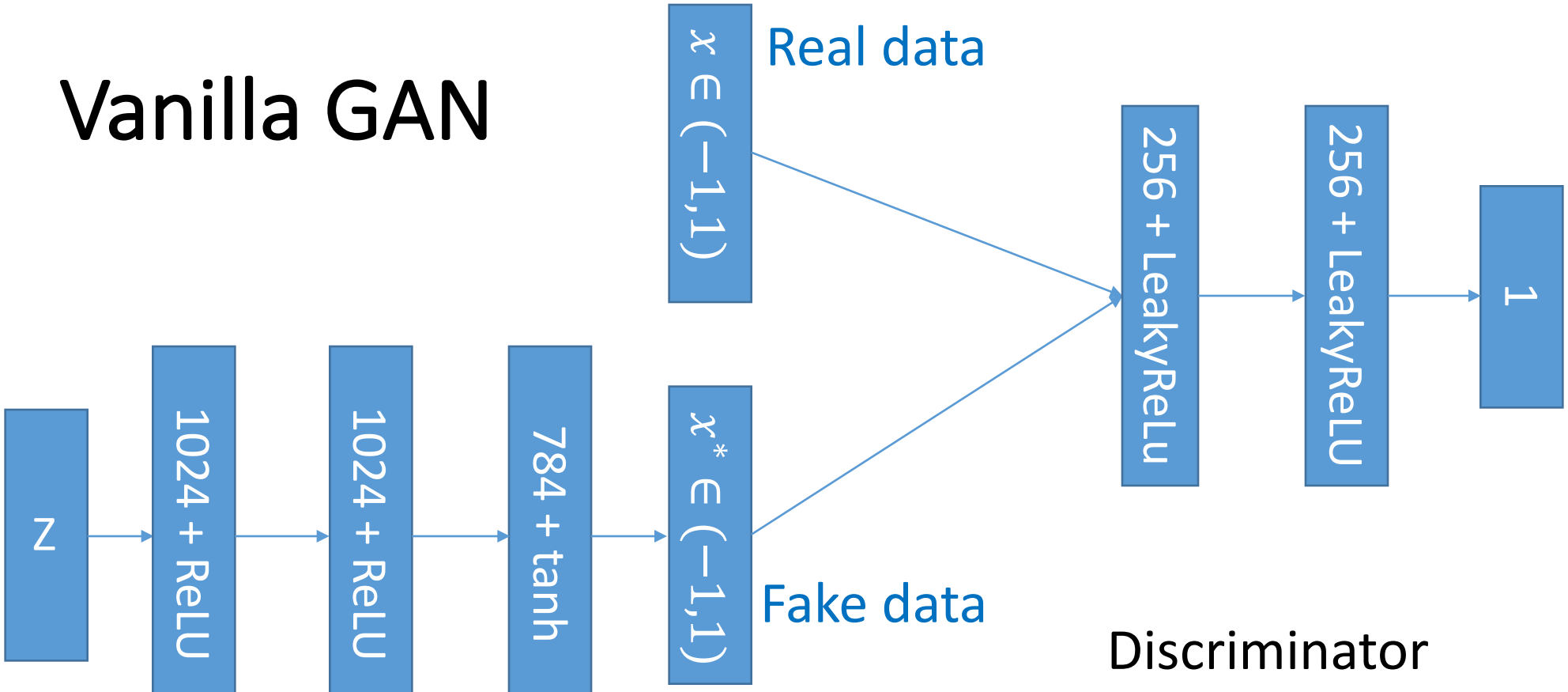
GANs

- Don’t work with an explicit density function
- Take game-theoretic approach: learn to generate from training distribution through 2-player game
- Pros:
  - Beautiful, state-of-the-art samples!
- Cons:
  - Trickier / more unstable to train
  - Can’t solve inference queries such as $p(x), p(z|x)$
- Active areas of research:
  - Better loss functions, more stable training (Wasserstein GAN, LSGAN, many others)
  - Conditional GANs, GANs for all kinds of applications

GANs

- Demo
  - Vanilla GAN
  - DCGAN
  - Wasserstein GAN

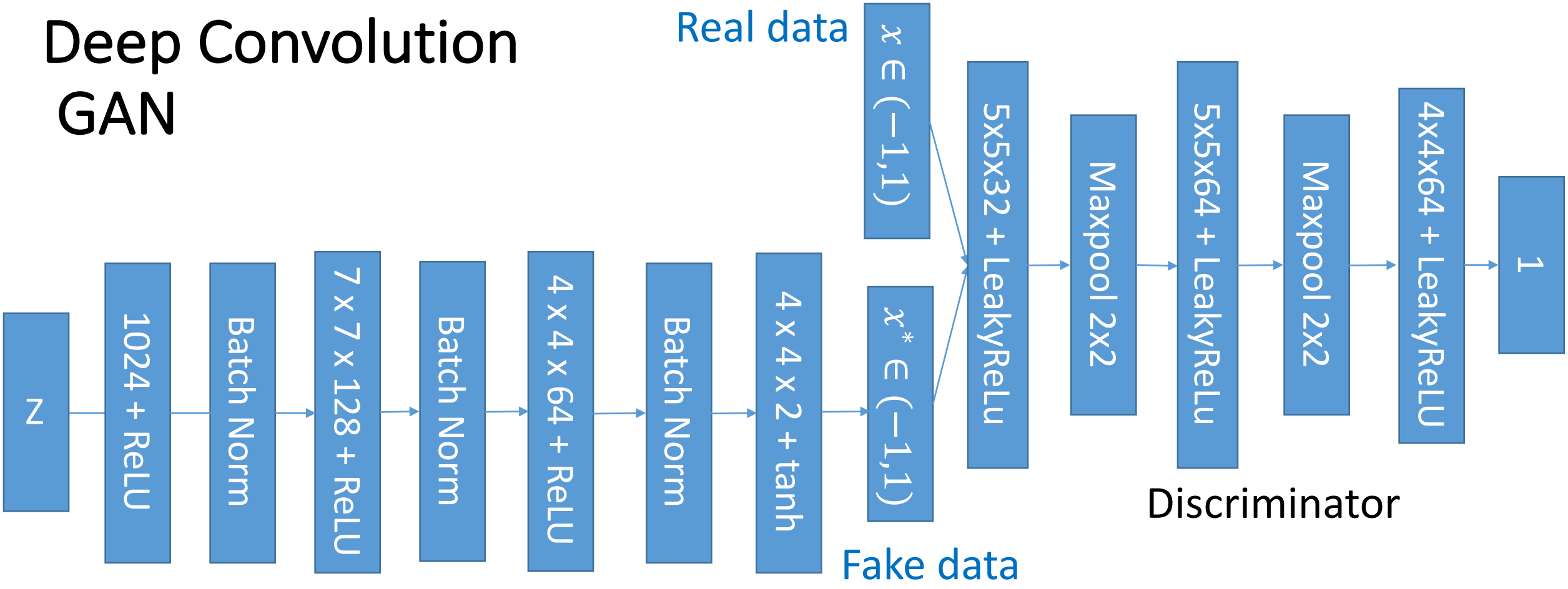
Vanilla GAN



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

$$\max_{\theta_g} \boldsymbol{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

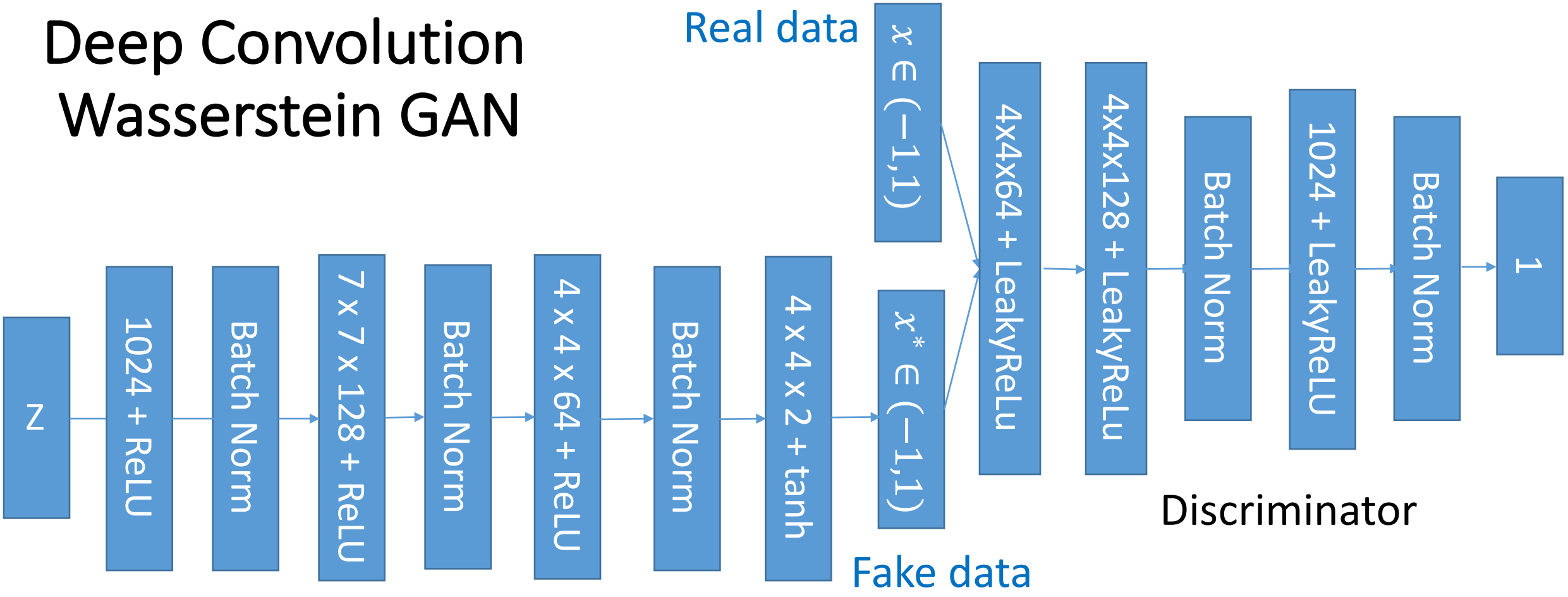
Deep Convolution GAN



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

$$\max_{\theta_g} \boldsymbol{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

Deep Convolution Wasserstein GAN



$$\max_{\theta_g} \boldsymbol{E}_{z\sim p(z)} f_w(G_{\theta_g}(z))$$

$$\max_{\theta_d} \left[\boldsymbol{E}_{z\sim p(z)} f_w(x) - \boldsymbol{E}_{x\sim p_{data}} f_w(x)\right]$$

- https://github.com/zhouzilu/cs231n/blob/master/assignment3/GANs-TensorFlow.ipynb

Thank you

- https://poloclub.github.io/ganlab/
- https://github.com/hindupuravinash/the-gan-zoo
- http://cs231n.stanford.edu/
- https://github.com/zhouzilu/cs231n