# Lecture 16:
# Generative Models:
# VQ-VAE, GANs, and Diffusion Models

# Announcement

- We will not have synchronous in-person lecture this Wednesday afternoon
- Wednesday lecture will be asynchronous and recorded research spotlight talks from several student researchers, where each talk will be 5min-10min.
- Action items for **finishing within one week (By Dec 4)**
  - Watch each talk at your own pace
  - Answer one question related to each talk
  - Raise one question or suggestion to the speaker of each talk (feel free to leave your contact info). We will send them to the speaker
- The links can be found in Bruinlearn Modules/Week9

# Last Time: Supervised vs Unsupervised Learning

**Supervised Learning**

**Data**: (x, y)

x is data, y is label

**Goal**: Learn a *function* to map x -> y

**Examples**: Classification, regression, object detection, semantic segmentation, image captioning, etc.

**Unsupervised Learning**

**Data**: x

Just data, no labels!

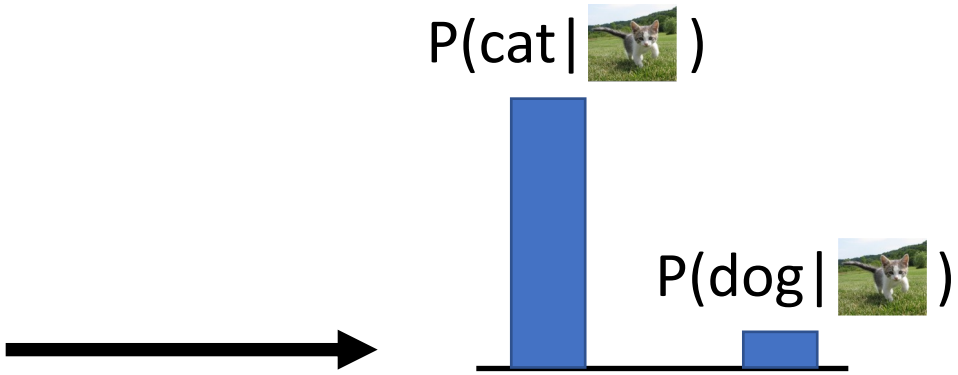
**Goal**: Learn some underlying hidden *structure* of the data

**Examples**: Clustering, dimensionality reduction, feature learning, density estimation, etc.

# Last Time: Discriminative vs Generative Models

**Discriminative Model:** Learn a probability distribution p(y|x)

**Data**: x




**Generative Model**: Learn a probability distribution p(x)

**Density Function**

p(x) assigns a positive number to each possible x; higher numbers mean x is more likely

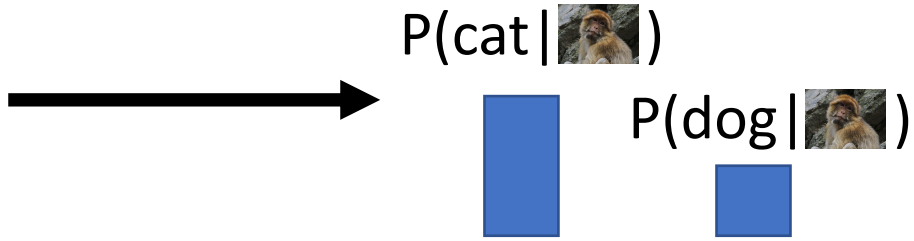
Density functions are **normalized**:

$$\int_X p(x)dx = 1$$

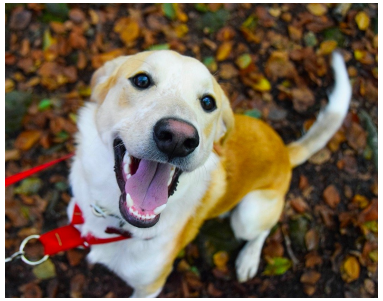
Different values of x **compete** for density

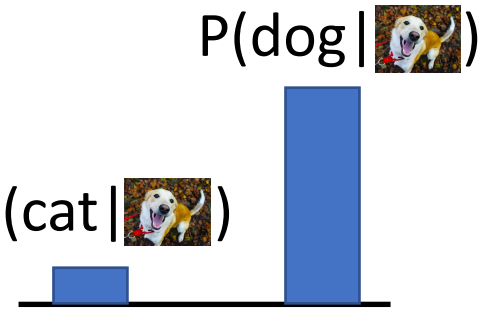
# Last Time: Discriminative vs Generative Models

**Discriminative Model:** Learn a probability distribution p(y|x)




**Generative Model**: Learn a probability distribution p(x)



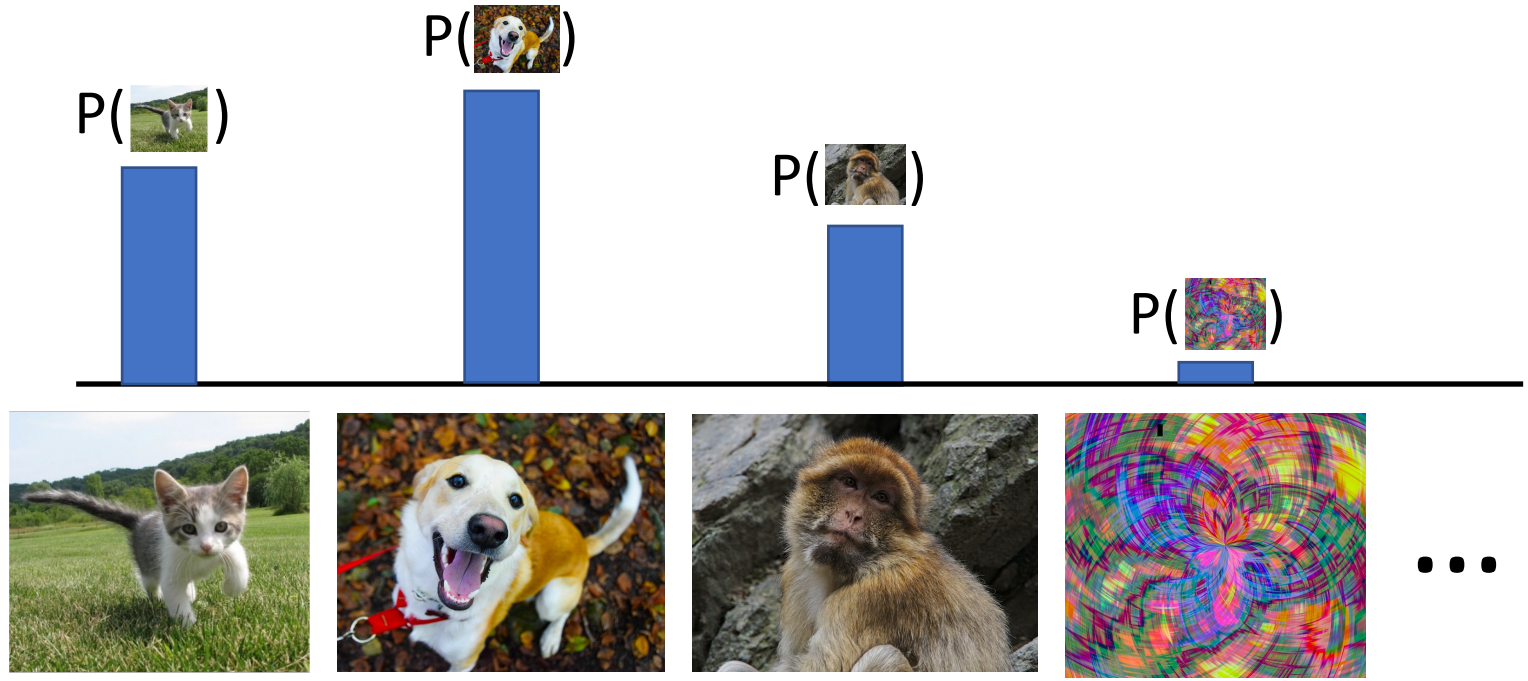



Discriminative model: No way for the model to handle unreasonable inputs; it must give label distributions for all images

# Last Time: Discriminative vs Generative Models

**Discriminative Model:** Learn a probability distribution p(y|x)

**Generative Model**: Learn a probability distribution p(x)




Generative model: All possible images compete with each other for probability mass

Requires deep image understanding!

# Last Time: Taxonomy of Generative Models

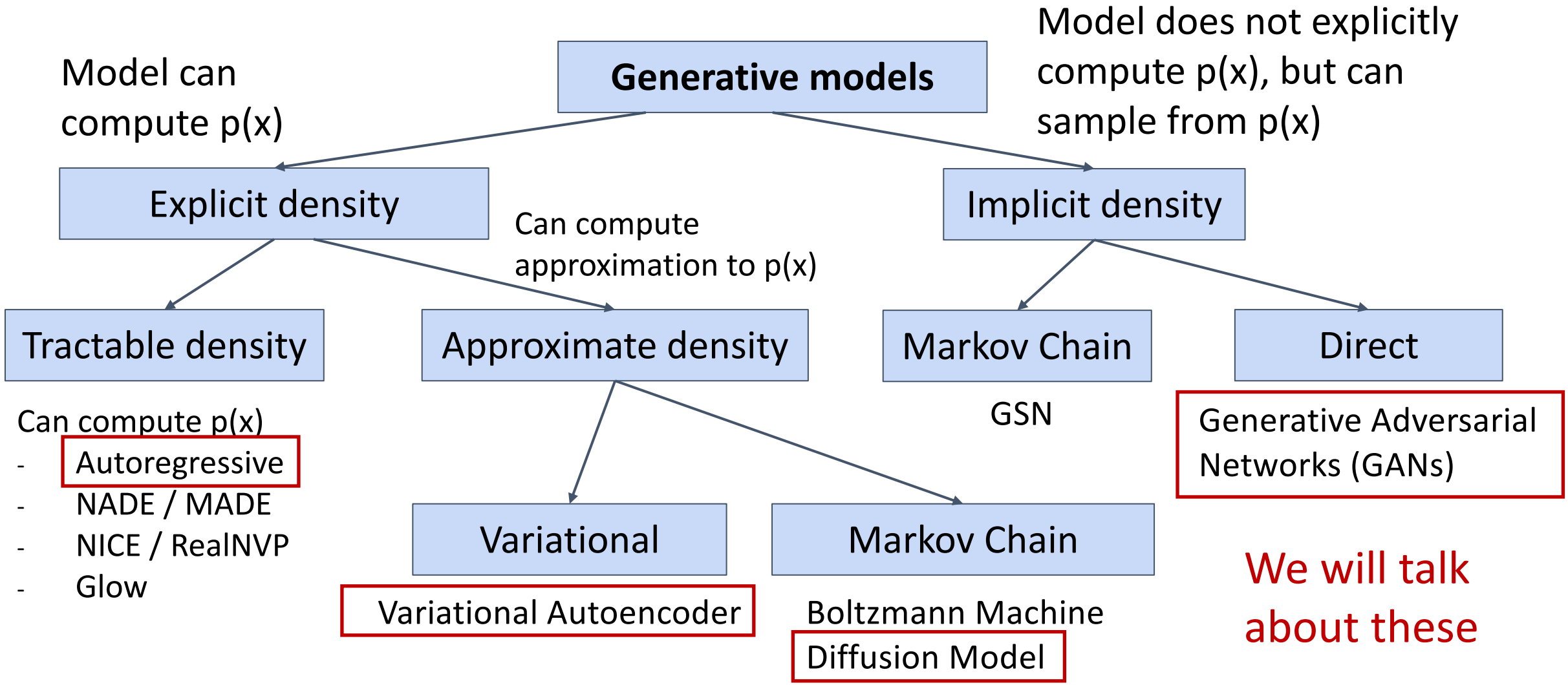



Figure adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

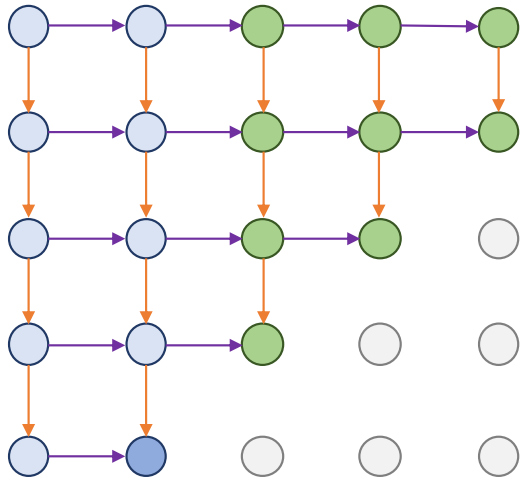
# Last Time: Autoregressive Models

**Explicit Density Function**

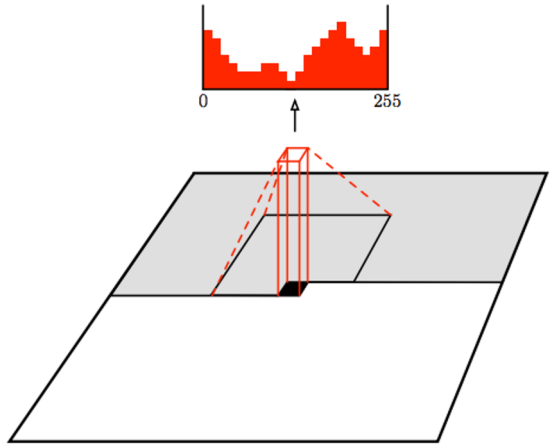
$$p(x) = p(x_1, x_2, x_3, \dots, x_T)$$
$$= p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \dots$$
$$= \prod_{t=1}^{T} p(x_t \mid x_1, \dots, x_{t-1})$$

Train by maximizing log-likelihood of training data

PixelRNN



PixelCNN




Van den Oord et al, "Pixel Recurrent Neural Networks", ICML 2016

Van den Oord et al, "Conditional Image Generation with PixelCNN Decoders", NeurIPS 2016

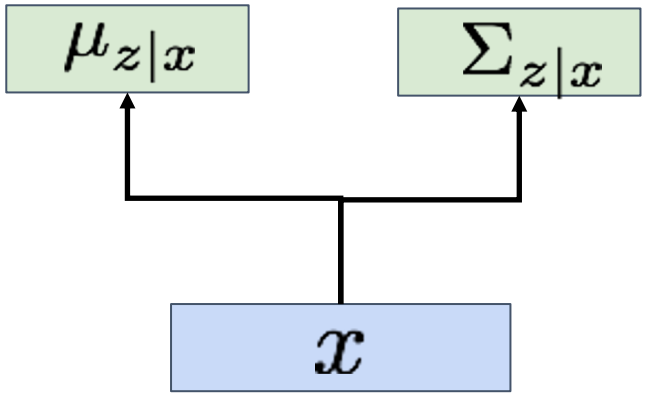
# Last Time: Variational Autoencoders

Jointly train **encoder** q and **decoder** p to maximize the **variational lower bound** on the data likelihood

$$\log p_\theta(x) \geq E_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}\left(q_\phi(z|x), p(z)\right)$$

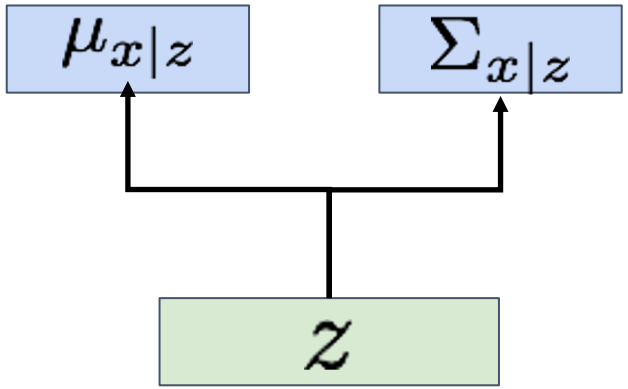
**Encoder Network**

$$q_\phi(z \mid x) = N(\mu_{z|x}, \Sigma_{z|x})$$




**Decoder Network**

$$p_\theta(x \mid z) = N(\mu_{x|z}, \Sigma_{x|z})$$



# So far: Two types of generative models

**Autoregressive models**

- Directly maximize p(data)
- High-quality generated images
- Slow to generate images
- No explicit latent codes

**Variational models**

- Maximize lower-bound on p(data)
- Generated images often blurry
- Very fast to generate images
- Learn rich latent codes

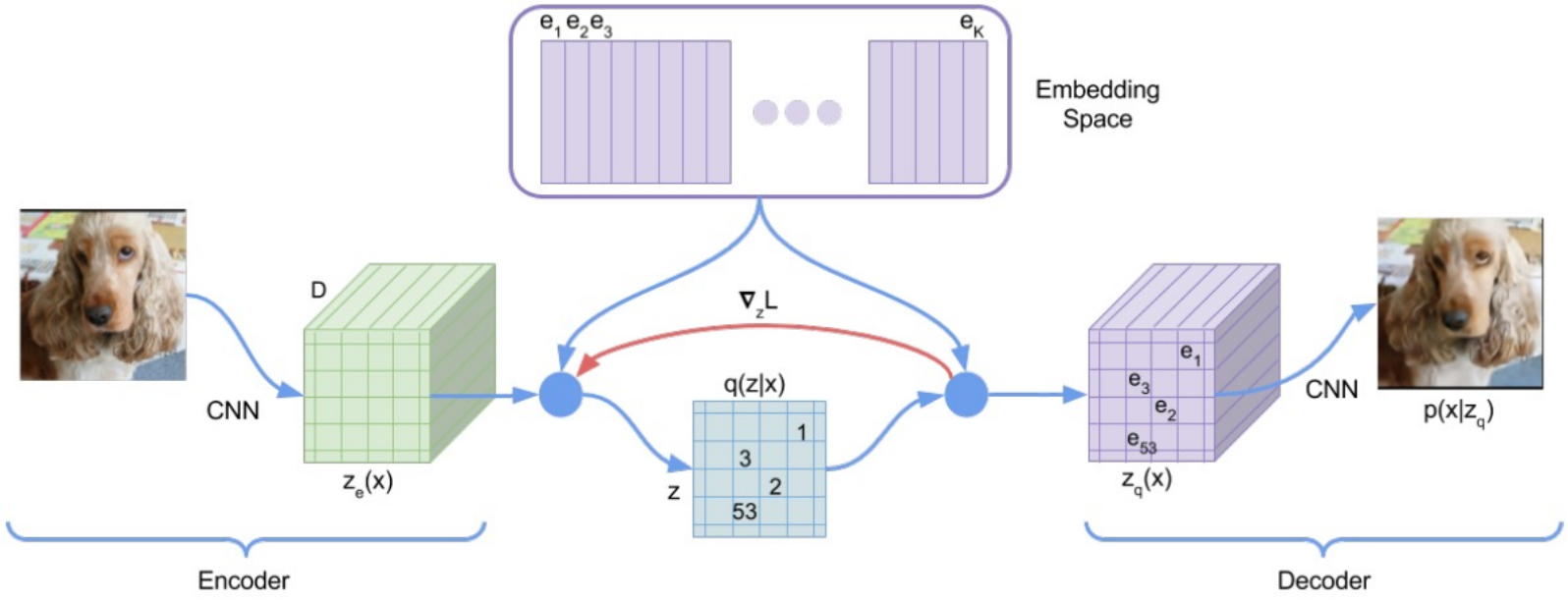
# So far: Two types of generative models

**Autoregressive models**
- Directly maximize p(data)
- High-quality generated images
- Slow to generate images
- No explicit latent codes

**Variational models**
- Maximize lower-bound on p(data)
- Generated images often blurry
- Very fast to generate images
- Learn rich latent codes

Can we combine them and get the best of both worlds?

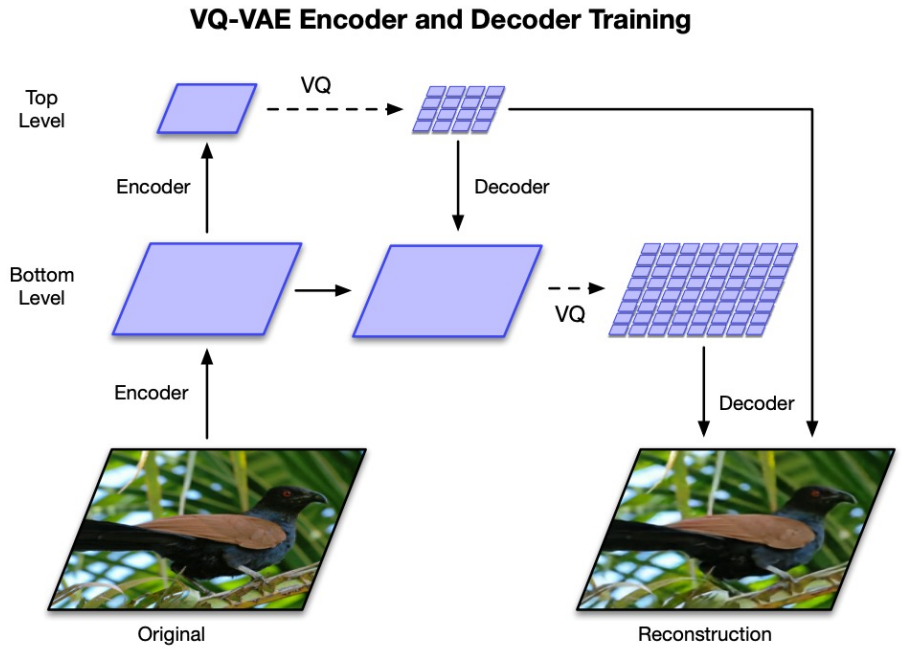
# Combining VAE + Autoregressive: Vector-Quantized Variational Autoencoder (VQ-VAE)



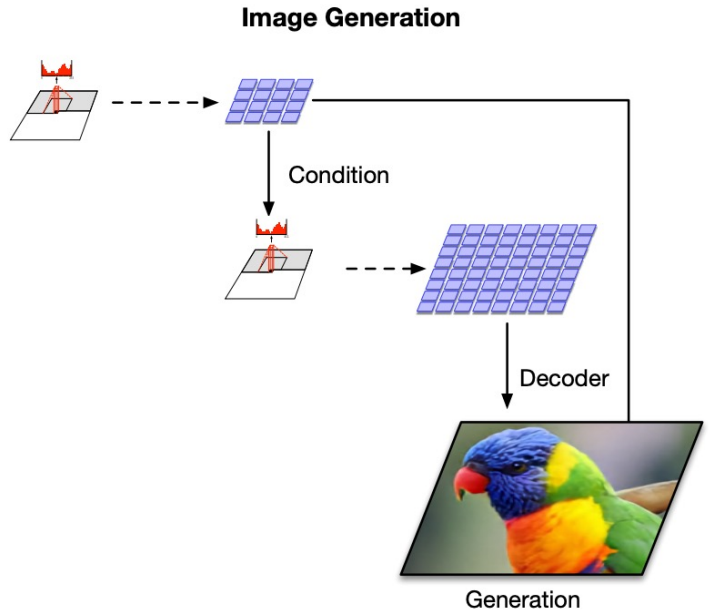

run Vector Quantization in the latent space (continuous value->discrete distribution), then learn a PixelCNN on discrete latents as a prior

# Combining VAE + Autoregressive: Vector-Quantized Variational Autoencoder (VQ-VAE2)

Train a VAE-like model to generate multiscale grids of latent codes




Use a multiscale PixelCNN to sample in latent code space




Razavi et al, “Generating Diverse High-Fidelity Images with VQ-VAE-2”, NeurIPS 2019

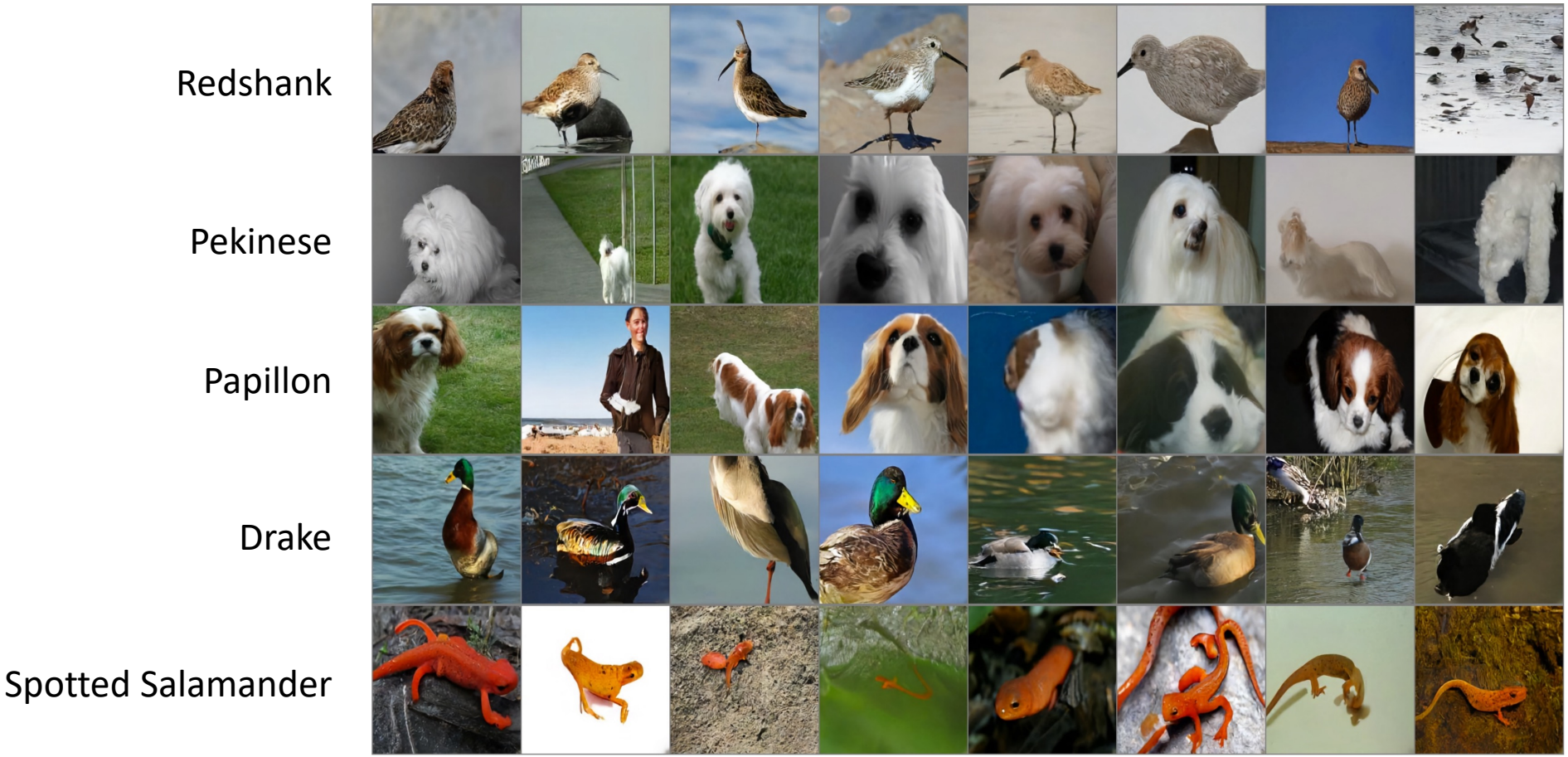
# Combining VAE + Autoregressive: VQ-VAE2

256 x 256 class-conditional samples, trained on ImageNet

Razavi et al, “Generating Diverse High-Fidelity Images with VQ-VAE-2”, NeurIPS 2019

# Combining VAE + Autoregressive: VQ-VAE2

## 256 x 256 class-conditional samples, trained on ImageNet




Razavi et al, “Generating Diverse High-Fidelity Images with VQ-VAE-2”, NeurIPS 2019

# Combining VAE + Autoregressive: VQ-VAE2

1024 x 1024 generated faces, trained on FFHQ

Razavi et al, “Generating Diverse High-Fidelity Images with VQ-VAE-2”, NeurIPS 2019

# Combining VAE + Autoregressive: VQ-VAE2

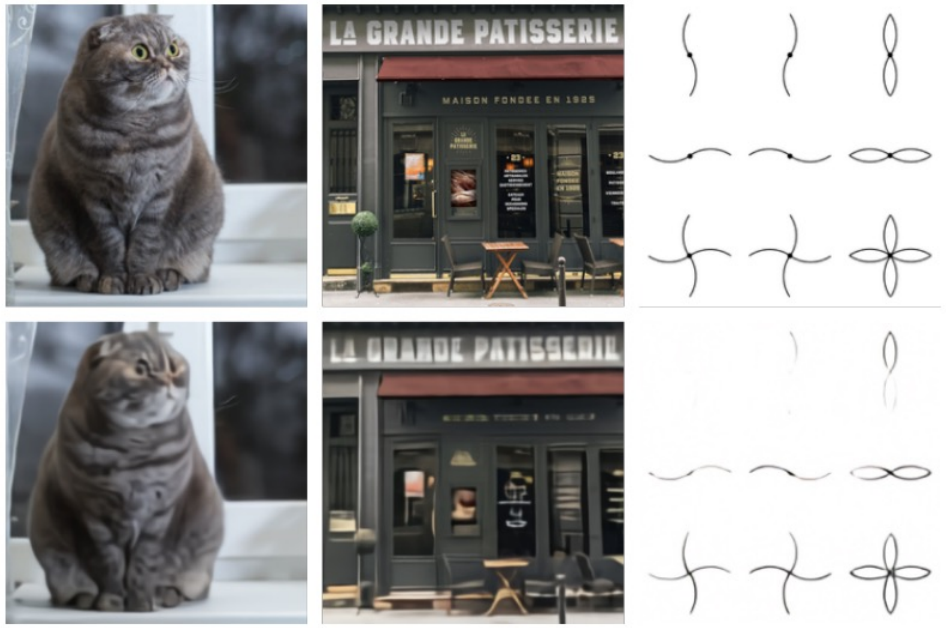
1024 x 1024 generated faces, trained on FFHQ

Razavi et al, “Generating Diverse High-Fidelity Images with VQ-VAE-2”, NeurIPS 2019

# VQ-VAE in OpenAI's DALL-E (v1)

Original images

Reconstruction




a vocabulary of 8192 in latent space of VQ-VAE

Sentence to Image translation

(a) a tapir made of accordion. a tapir with the texture of an accordion.

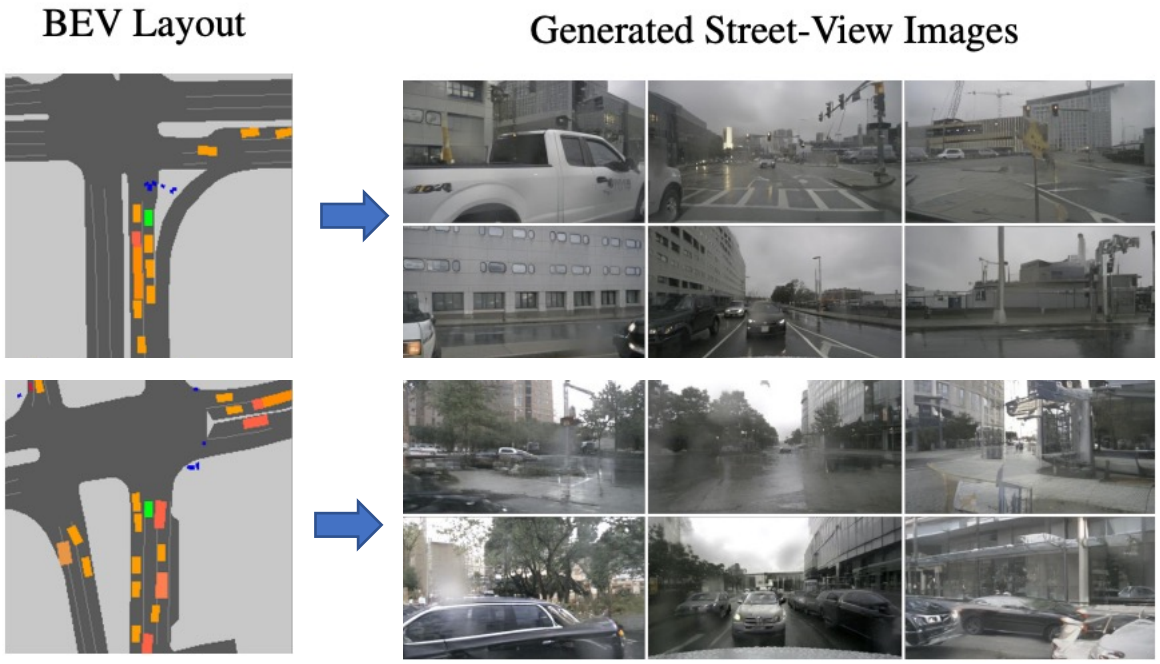
(b) an illustration of a baby hedgehog in a christmas sweater walking a dog

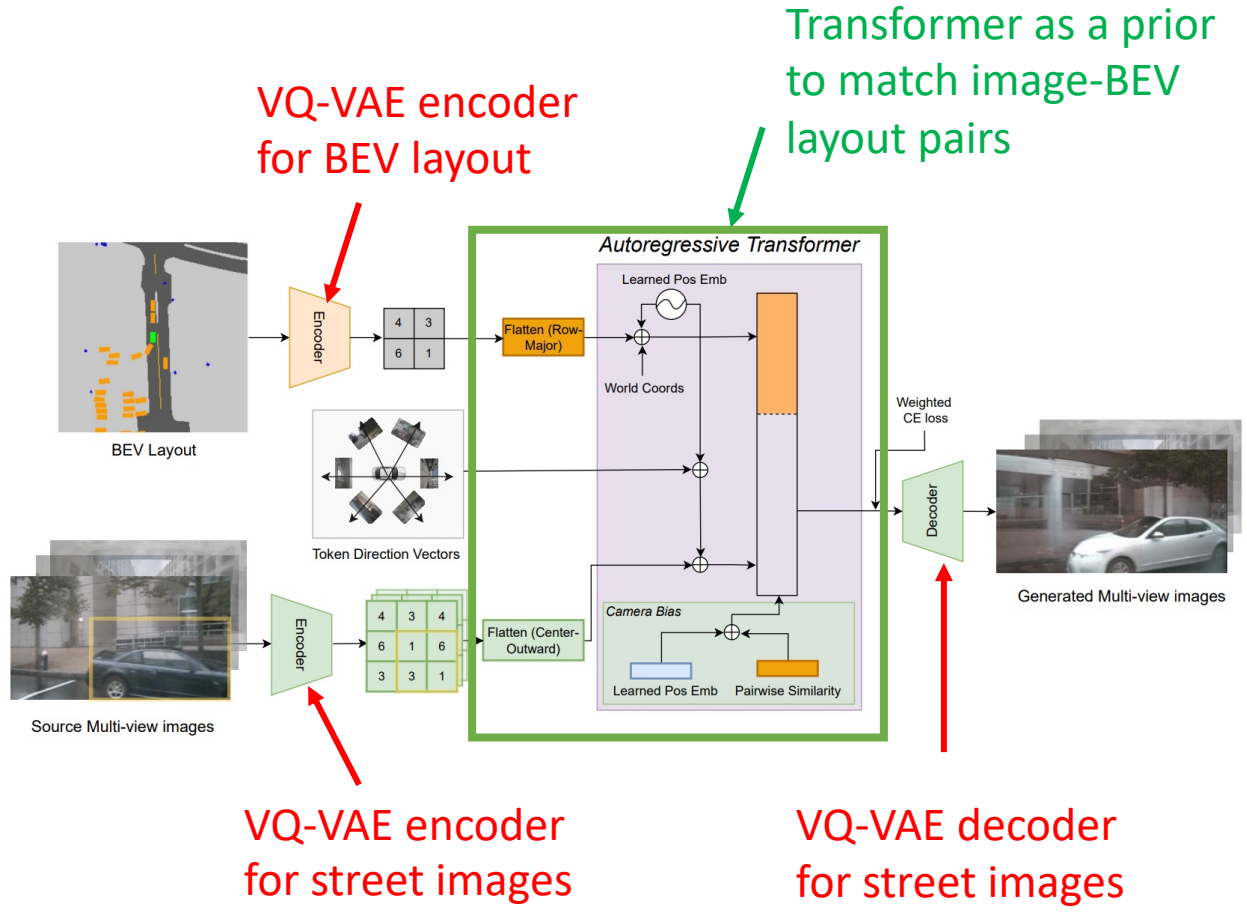
https://github.com/openai/DALL-E



https://arxiv.org/pdf/2102.12092.pdf

# Street-View Image Generation from a Bird's-Eye View Layout




https://arxiv.org/pdf/2301.04634.pdf

work done by former CS188 (prior to CS163) student Alex Swerdlow

# Generative Models So Far:

**Autoregressive Models** directly maximize likelihood of training data:

$$p_\theta(x) = \prod_{i=1}^{N} p_\theta(x_i | x_1, \ldots, x_{i-1})$$

# Generative Models So Far:

**Autoregressive Models** directly maximize likelihood of training data:

$$p_\theta(x) = \prod_{i=1}^{N} p_\theta(x_i | x_1, \ldots, x_{i-1})$$

**Variational Autoencoders** introduce a latent z, and maximize a lower bound:

$$p_\theta(x) = \int_Z p_\theta(x|z)p(z)dz \geq E_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}\left(q_\phi(z|x), p(z)\right)$$

# Generative Models So Far:

**Autoregressive Models** directly maximize likelihood of training data:

$$p_\theta(x) = \prod_{i=1}^{N} p_\theta(x_i | x_1, \ldots, x_{i-1})$$

**Variational Autoencoders** introduce a latent z, and maximize a lower bound:

$$p_\theta(x) = \int_Z p_\theta(x|z)p(z)dz \geq E_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}\left(q_\phi(z|x), p(z)\right)$$

**Generative Adversarial Networks** give up on modeling p(x), but allow us to draw samples from p(x)

# Generative Adversarial Networks

**Setup**: Assume we have data $x_i$ drawn from distribution $p_{data}(x)$. Want to sample from $p_{data}$.

Goodfellow et al, “Generative Adversarial Nets”, NeurIPS 2014

# Generative Adversarial Networks

**Setup**: Assume we have data $x_i$ drawn from distribution $p_{data}(x)$. Want to sample from $p_{data}$.

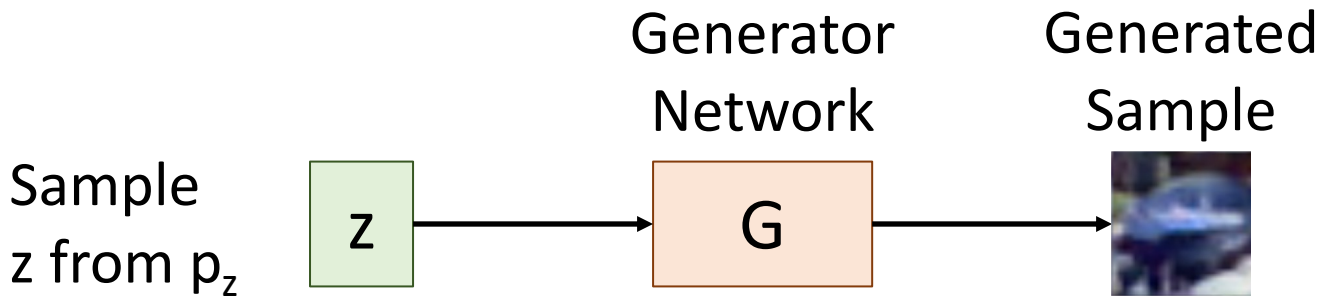
**Idea**: Introduce a latent variable z with simple prior p(z).
Sample $z \sim p(z)$ and pass to a **Generator Network** x = G(z)
Then x is a sample from the **Generator distribution** $p_G$. Want $p_G = p_{data}$!

Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

# Generative Adversarial Networks

**Setup**: Assume we have data $x_i$ drawn from distribution $p_{data}(x)$. Want to sample from $p_{data}$.

**Idea**: Introduce a latent variable z with simple prior p(z).
Sample $z \sim p(z)$ and pass to a **Generator Network** x = G(z)
Then x is a sample from the **Generator distribution** $p_G$. Want $p_G = p_{data}$!



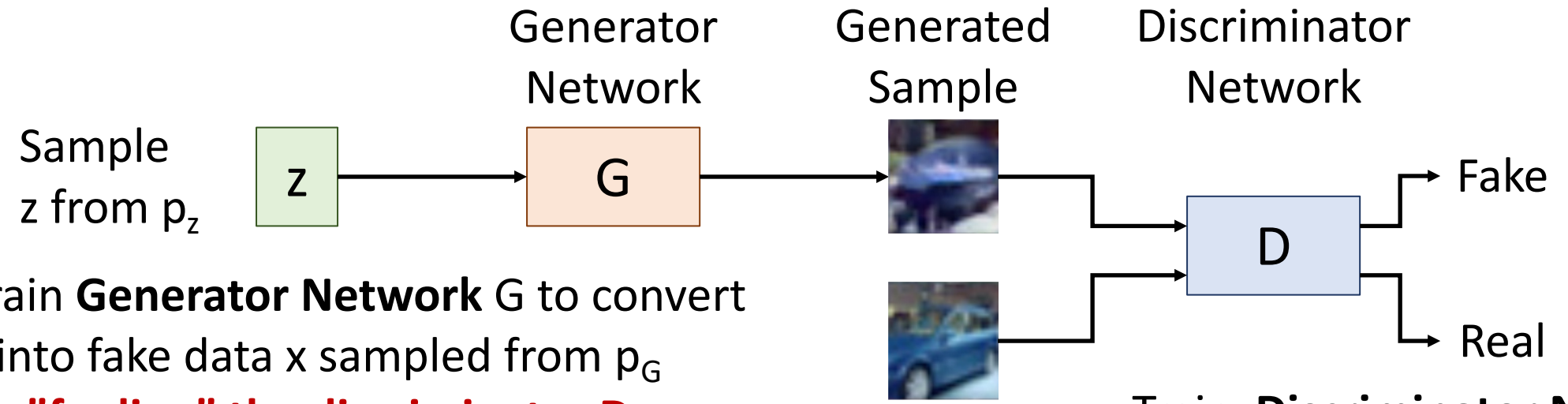

Train **Generator Network** G to convert z into fake data x sampled from $p_G$

Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

# Generative Adversarial Networks

**Setup**: Assume we have data $x_i$ drawn from distribution $p_{data}(x)$. Want to sample from $p_{data}$.

**Idea**: Introduce a latent variable z with simple prior p(z).
Sample $z \sim p(z)$ and pass to a **Generator Network** x = G(z)
Then x is a sample from the **Generator distribution** $p_G$. Want $p_G = p_{data}$!




Train **Generator Network** G to convert z into fake data x sampled from $p_G$
**by "fooling" the discriminator D**

Train **Discriminator Network** D to classify data as real or fake (1/0)

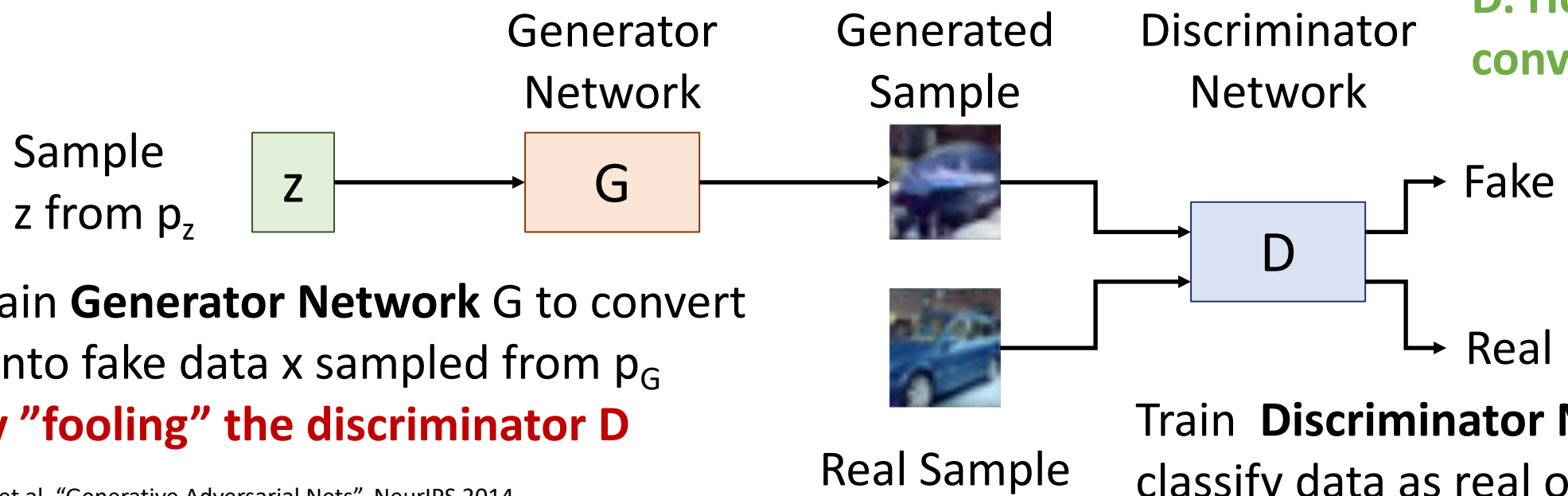
Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

# Generative Adversarial Networks

**Setup**: Assume we have data $x_i$ drawn from distribution $p_{data}(x)$. Want to sample from $p_{data}$.

**Idea**: Introduce a latent variable z with simple prior p(z).
Sample $z \sim p(z)$ and pass to a **Generator Network** x = G(z)
Then x is a sample from the **Generator distribution** $p_G$. Want $p_G = p_{data}$!

**Jointly train G and D. Hopefully $p_G$ converges to $p_{data}$!**




Train **Generator Network** G to convert z into fake data x sampled from $p_G$
**by "fooling" the discriminator D**

Train **Discriminator Network** D to classify data as real or fake (1/0)

Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

# Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

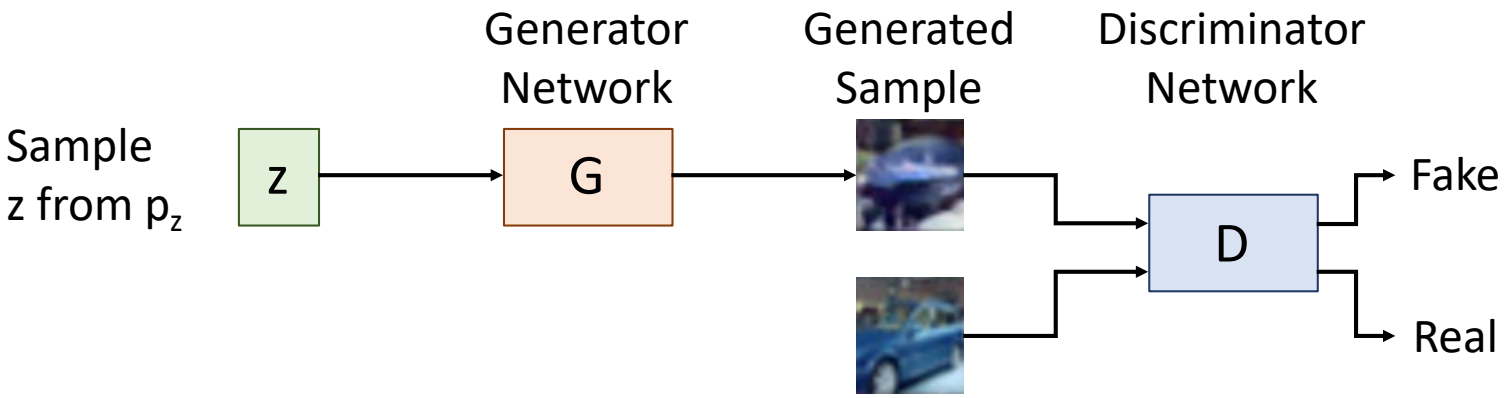
$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D\left(G(z)\right) \right) \right] \right)$$

Goodfellow et al, “Generative Adversarial Nets”, NeurIPS 2014

# Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

$$\min_{G} \max_{D} \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$



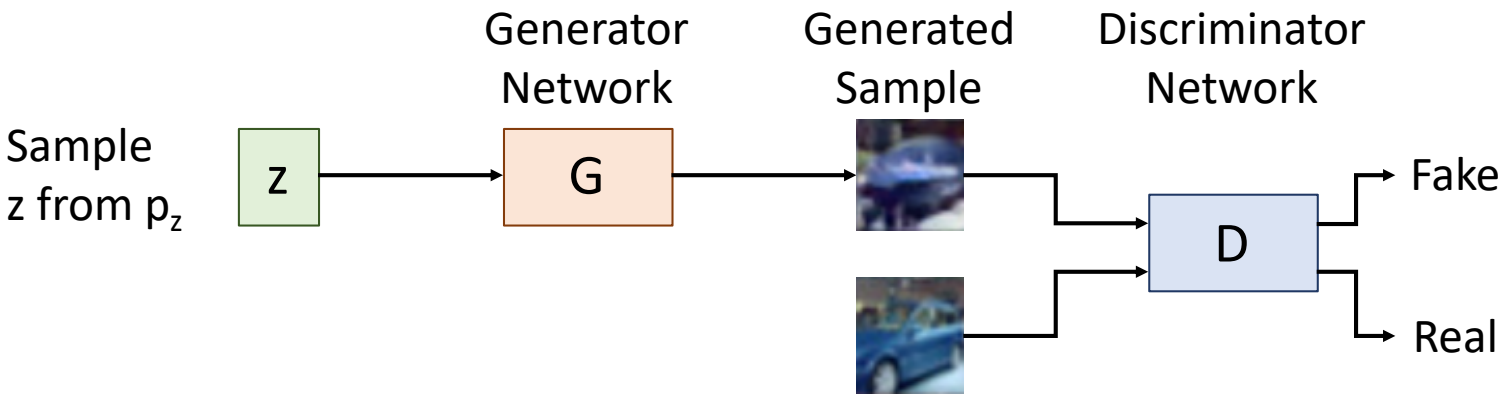

Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

# Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

**Discriminator wants**
**D(x) = 1 for real data**

$$\min_{G}\max_{D}\left(E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}\left[\log\left(1 - D(G(z))\right)\right]\right)$$




Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

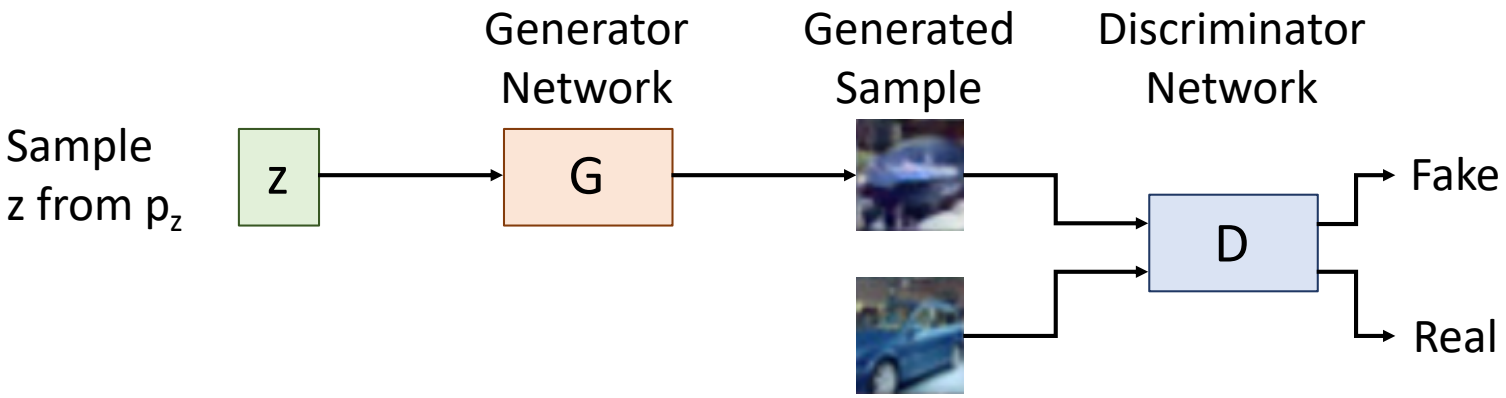
# Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

**Discriminator wants D(x) = 1 for real data**

**Discriminator wants D(x) = 0 for fake data**

$$\min_{G} \max_{D} \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$




Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

# Generative Adversarial Networks: Training Objective

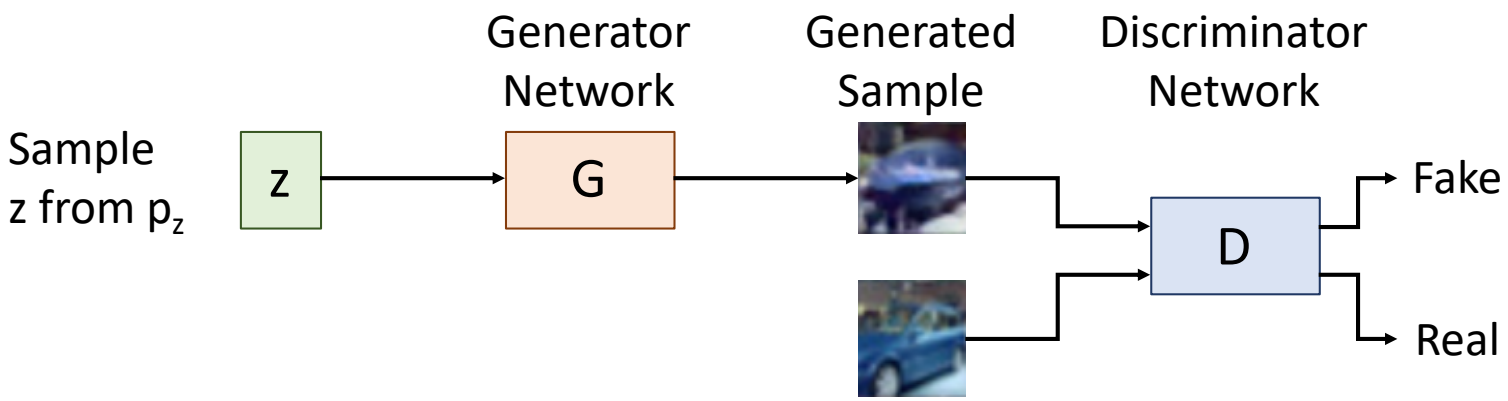
Jointly train generator G and discriminator D with a **minimax game**

**Discriminator wants D(x) = 1 for real data**

**Discriminator wants D(x) = 0 for fake data**

$$\min_{G} \max_{D} \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

**Generator wants D(x) = 1 for fake data**




Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

# Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

Train G and D using alternating gradient updates

$$\min_{G} \max_{D} \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

# Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

Train G and D using alternating gradient updates

$$\min_{\boldsymbol{G}} \max_{\boldsymbol{D}} \left( E_{x \sim p_{data}} [\log \boldsymbol{D}(x)] + E_{\boldsymbol{z} \sim p(\boldsymbol{z})} \left[ \log \left( 1 - \boldsymbol{D}(\boldsymbol{G}(\boldsymbol{z})) \right) \right] \right)$$

$$= \min_{\boldsymbol{G}} \max_{\boldsymbol{D}} \boldsymbol{V}(\boldsymbol{G}, \boldsymbol{D})$$

Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

# Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

Train G and D using alternating gradient updates

$$\min_{G} \max_{D} \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_{G} \max_{D} V(G, D)$$

For t in 1, … T:

1. (Update D) $D = D + \alpha_D \frac{\partial V}{\partial D}$
2. (Update G) $G = G - \alpha_G \frac{\partial V}{\partial G}$

Goodfellow et al, “Generative Adversarial Nets”, NeurIPS 2014

# Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

Train G and D using alternating gradient updates

$$\min_{G} \max_{D} \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_{G} \max_{D} V(G, D)$$

We are not minimizing any overall loss! No training curves to look at!

For t in 1, … T:

1. (Update D) $D = D + \alpha_D \frac{\partial V}{\partial D}$
2. (Update G) $G = G - \alpha_G \frac{\partial V}{\partial G}$

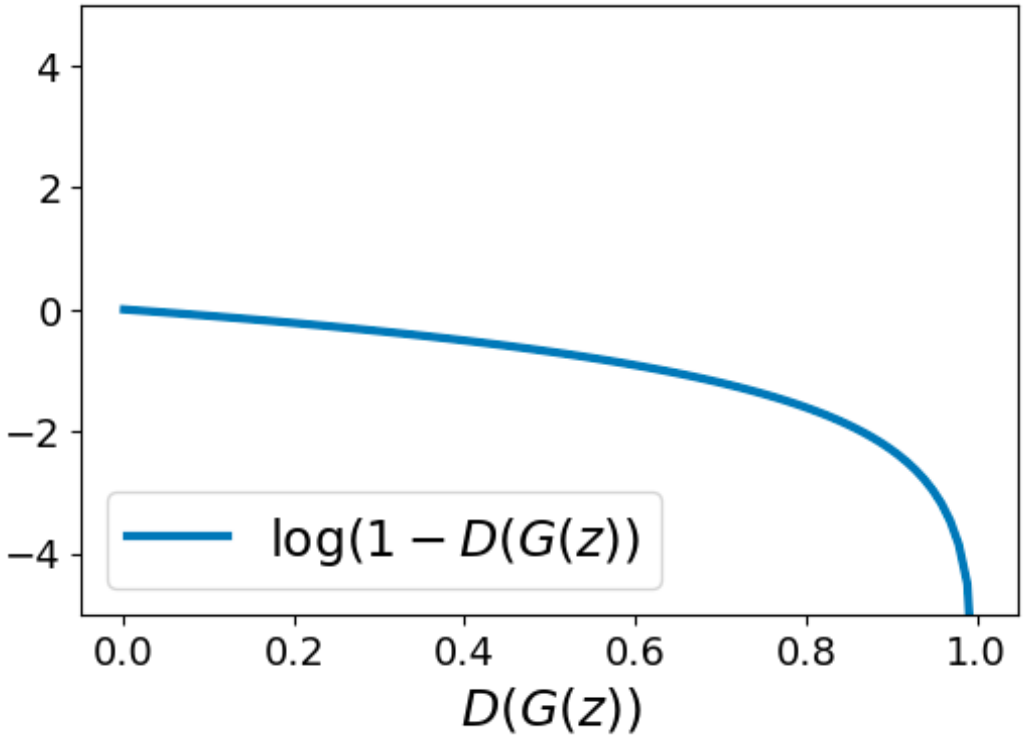
Goodfellow et al, “Generative Adversarial Nets”, NeurIPS 2014

# Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

$$\min_{G}\max_{D}\left(E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}\left[\log\left(1 - D(G(z))\right)\right]\right)$$

At start of training, generator is very bad and discriminator can easily tell apart real/fake, so D(G(z)) close to 0




Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

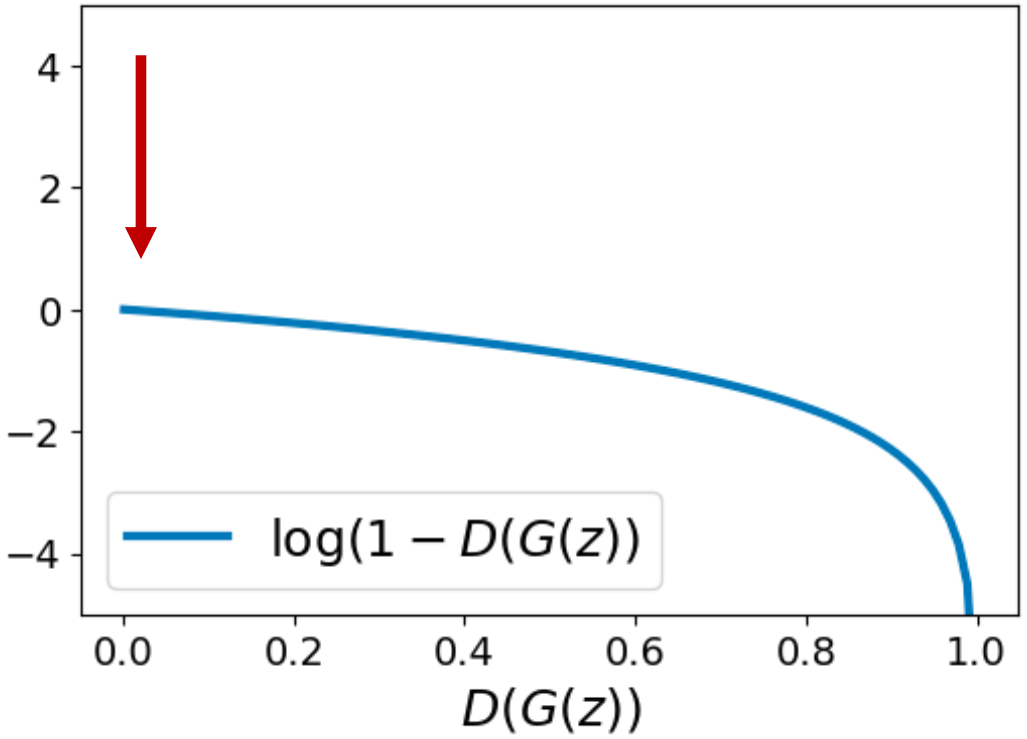
# Generative Adversarial Networks: Training Objective

Jointly train generator G and discriminator D with a **minimax game**

$$\min_{G}\max_{D}\left(E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}\left[\log\left(1 - D(G(z))\right)\right]\right)$$

At start of training, generator is very bad and discriminator can easily tell apart real/fake, so D(G(z)) close to 0

**Problem**: Vanishing gradients for G




Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

# Generative Adversarial Networks: Training Objective

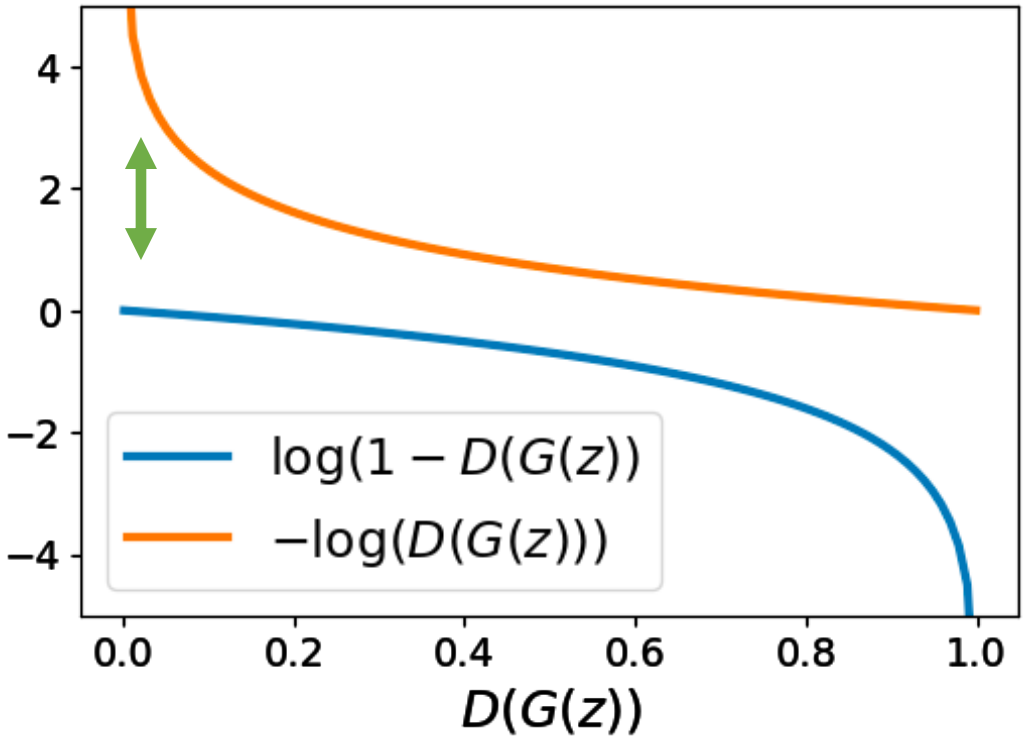
Jointly train generator G and discriminator D with a **minimax game**

$$\min_{G}\max_{D}\left(E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}\left[\log\left(1 - D(G(z))\right)\right]\right)$$

At start of training, generator is very bad and discriminator can easily tell apart real/fake, so D(G(z)) close to 0

**Problem**: Vanishing gradients for G

**Solution**: Right now G is trained to minimize log(1-D(G(z)). Instead, train G to minimize –log(D(G(z)). Then G gets strong gradients at start of training!




Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

# Generative Adversarial Networks: Optimality

Jointly train generator G and discriminator D with a **minimax game**

Why is this particular objective a good idea?

$$\min_{G} \max_{D} \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

This minimax game achieves its global minimum when $p_G = p_{data}$!

Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

# Generative Adversarial Networks: Optimality (derivation is in the appendix)

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_G (2 * JSD(p_{data}, p_G) - \log 4)$$

**Summary**: The global minimum of the minimax game happens when:

1. $D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$ (Optimal discriminator for any G)
2. $p_G(x) = p_{data}(x)$ (Optimal generator for optimal D)

**Caveats**:

1. G and D are neural nets with fixed architecture. We don't know whether they can actually represent the optimal D and G.
2. This tells us nothing about convergence to the optimal solution

$$\min_{\theta} \max_{\phi} V(G_\theta, D_\phi) = E_{\mathbf{x} \sim p_{\text{data}}}[\log D_\phi(\mathbf{x})] + E_{\mathbf{z} \sim p(\mathbf{z})}[\log(1 - D_\phi(G_\theta(\mathbf{z})))]$$

# GAN training algorithm

Repeat:

- Sample minibatch of $m$ training points $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(m)}$ from $\mathcal{D}$
- Sample minibatch of $m$ noise vectors $\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \ldots, \mathbf{z}^{(m)}$ from $p_z$
- Update the generator parameters $\theta$ by stochastic gradient **descent**

$$\nabla_\theta V(G_\theta, D_\phi) = \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} \log(1 - D_\phi(G_\theta(\mathbf{z}^{(i)})))$$

- Update the discriminator parameters $\phi$ by stochastic gradient **ascent**

$$\nabla_\phi V(G_\theta, D_\phi) = \frac{1}{m} \nabla_\phi \sum_{i=1}^{m} [\log D_\phi(\mathbf{x}^{(i)}) + \log(1 - D_\phi(G_\theta(\mathbf{z}^{(i)})))]$$

# Example code of training GAN

```
for epoch in range(opt.niter):
    for i, data in enumerate(dataloader, 0):
        ############################
        # (1) Update D network: maximize log(D(x)) + log(1 - D(G(z)))
        ###########################
        # train with real
        netD.zero_grad()
        real_cpu = data[0].to(device)
        batch_size = real_cpu.size(0)
        label = torch.full((batch_size,), real_label,
                           dtype=real_cpu.dtype, device=device)

        output = netD(real_cpu)
        errD_real = criterion(output, label)
        errD_real.backward()
        D_x = output.mean().item()

        # train with fake
        noise = torch.randn(batch_size, nz, 1, 1, device=device)
        fake = netG(noise)
        label.fill_(fake_label)
        output = netD(fake.detach())
        errD_fake = criterion(output, label)
        errD_fake.backward()
        D_G_z1 = output.mean().item()
        errD = errD_real + errD_fake
        optimizerD.step()
```

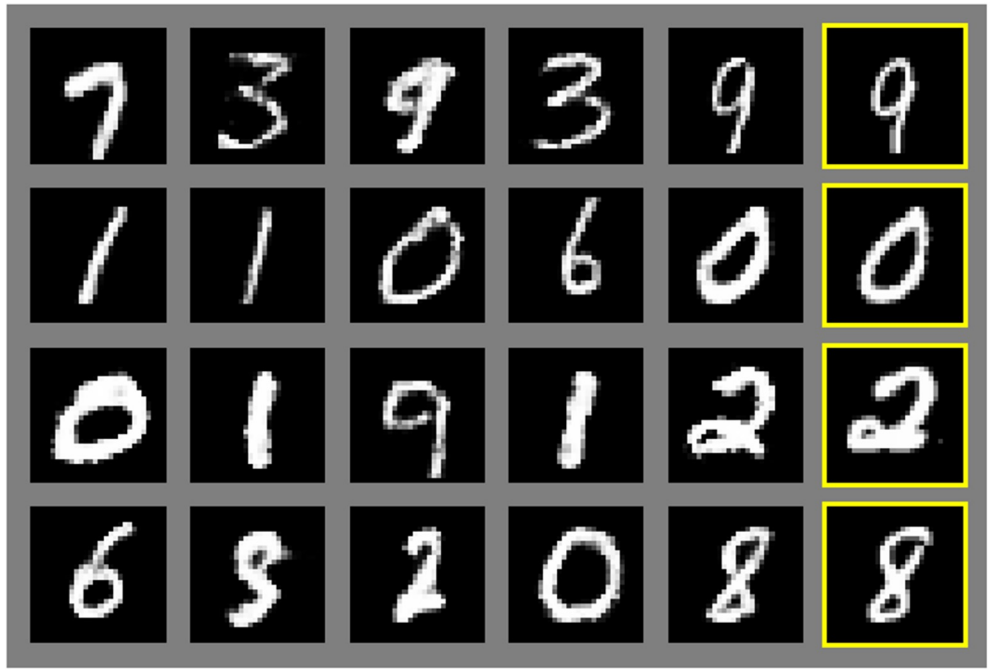
```
        ############################
        # (2) Update G network: maximize log(D(G(z)))
        ###########################
        netG.zero_grad()
        label.fill_(real_label)  # fake labels are real for generator cost
        output = netD(fake)
        errG = criterion(output, label)
        errG.backward()
        D_G_z2 = output.mean().item()
        optimizerG.step()
```

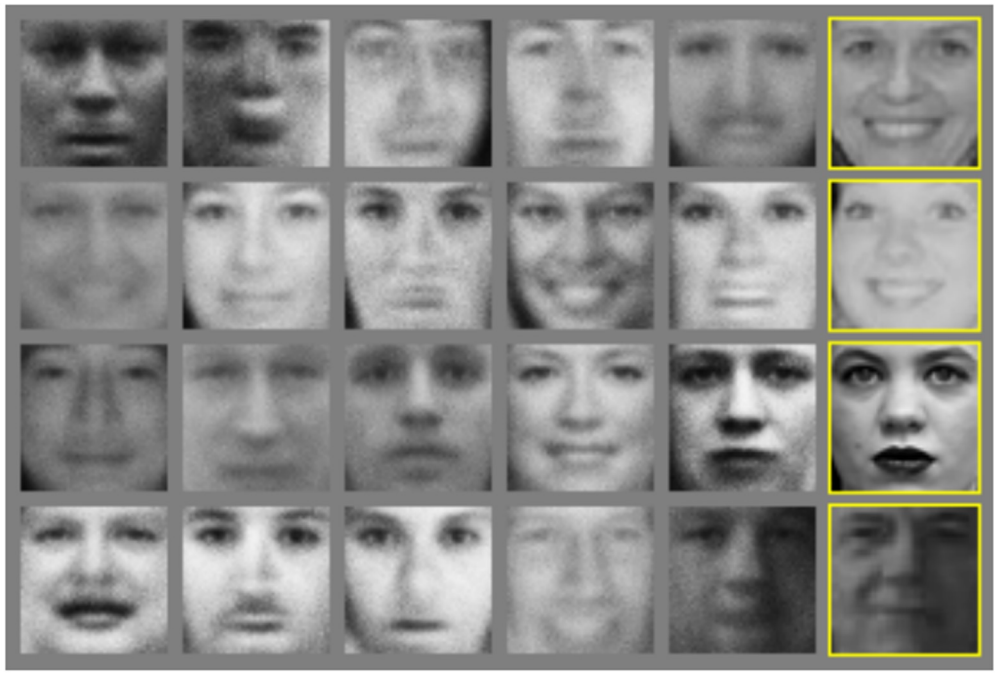
```
205   criterion = nn.BCELoss()
```

https://github.com/pytorch/examples/blob/master/dcgan/main.py

# Generative Adversarial Networks: Results

Generated samples

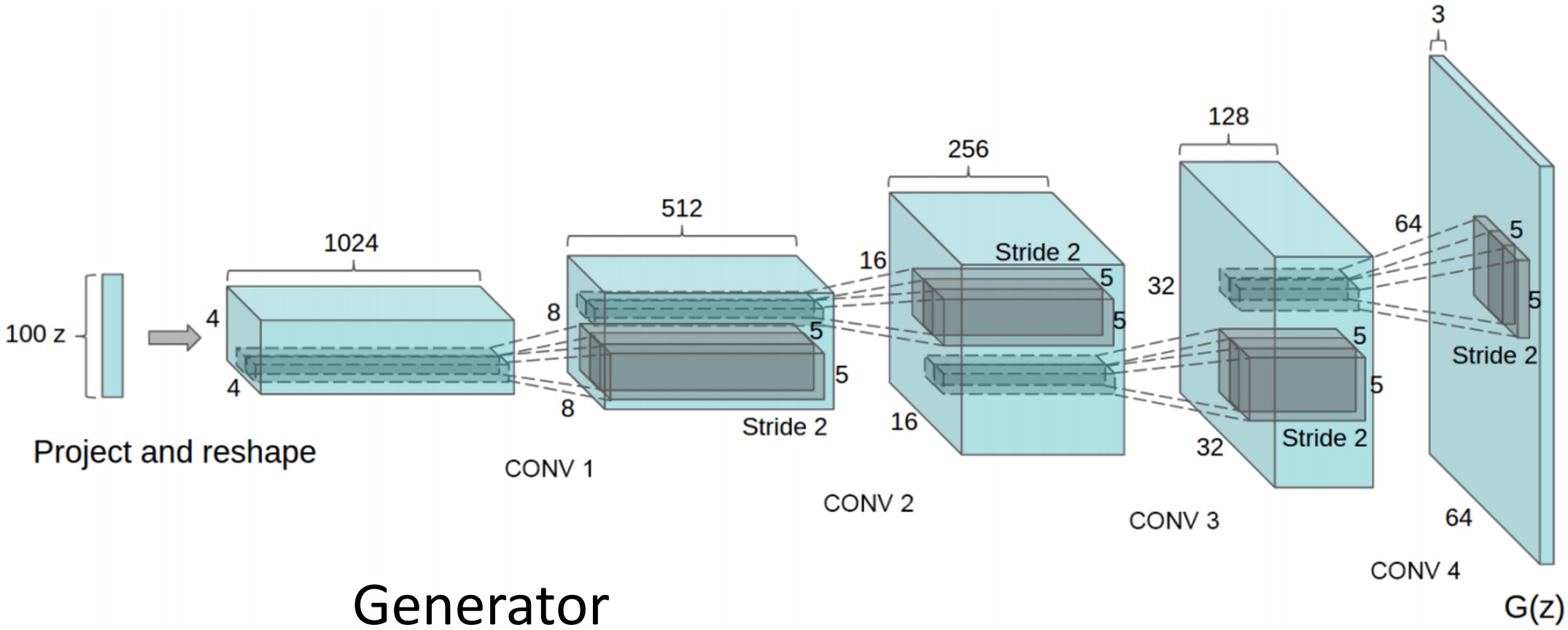




Nearest neighbor from training set

Goodfellow et al, “Generative Adversarial Nets”, NeurIPS 2014

# Generative Adversarial Networks: DC-GAN




Generator

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

# Generative Adversarial Networks: DC-GAN

Samples from the model look much better!

Radford et al, ICLR 2016

# Generative Adversarial Networks: Interpolation

Interpolating between points in latent z space

Radford et al, ICLR 2016

# Generative Adversarial Networks: Vector Math

Samples from the model

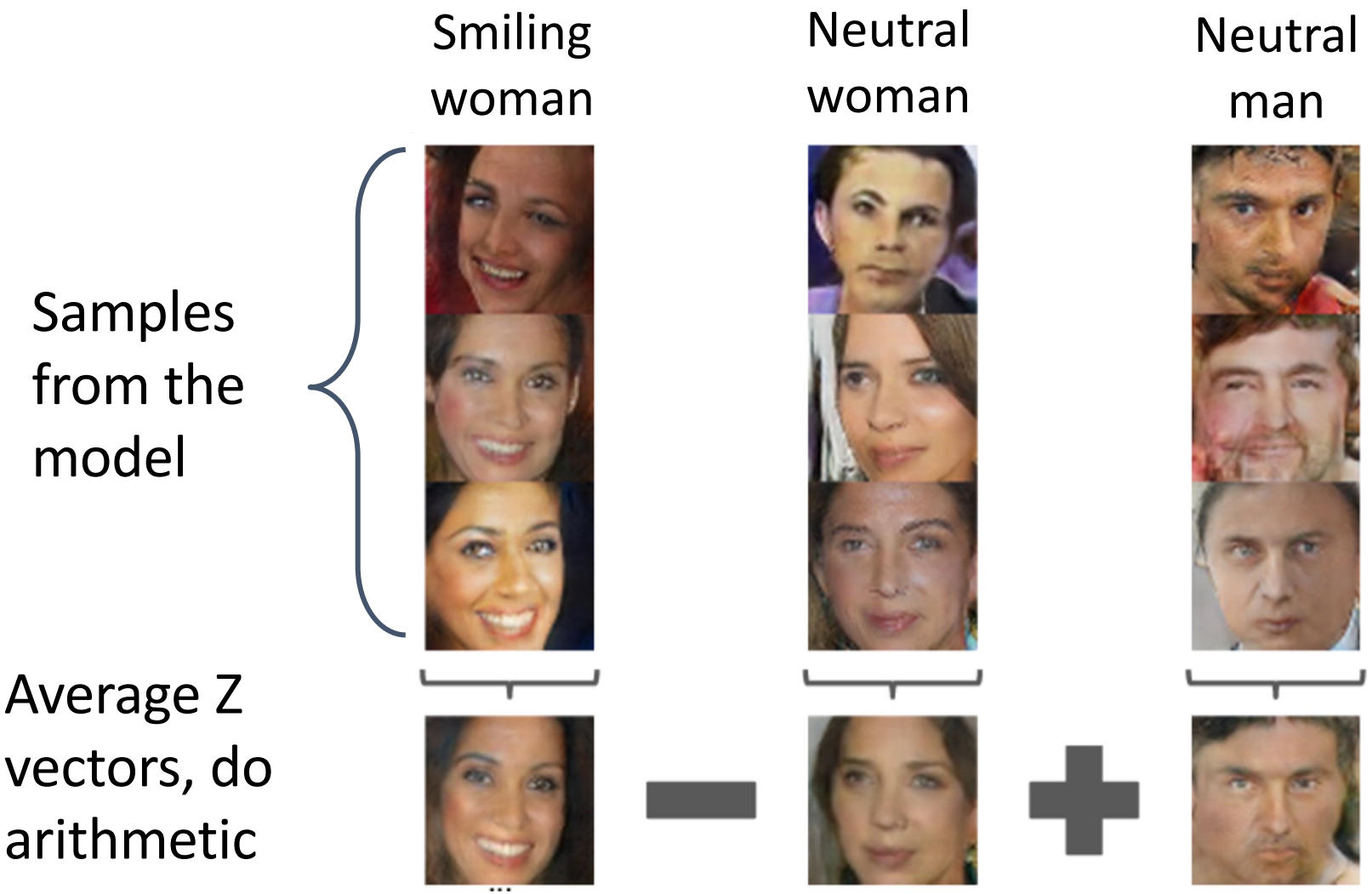
Smiling woman

Neutral woman

Neutral man

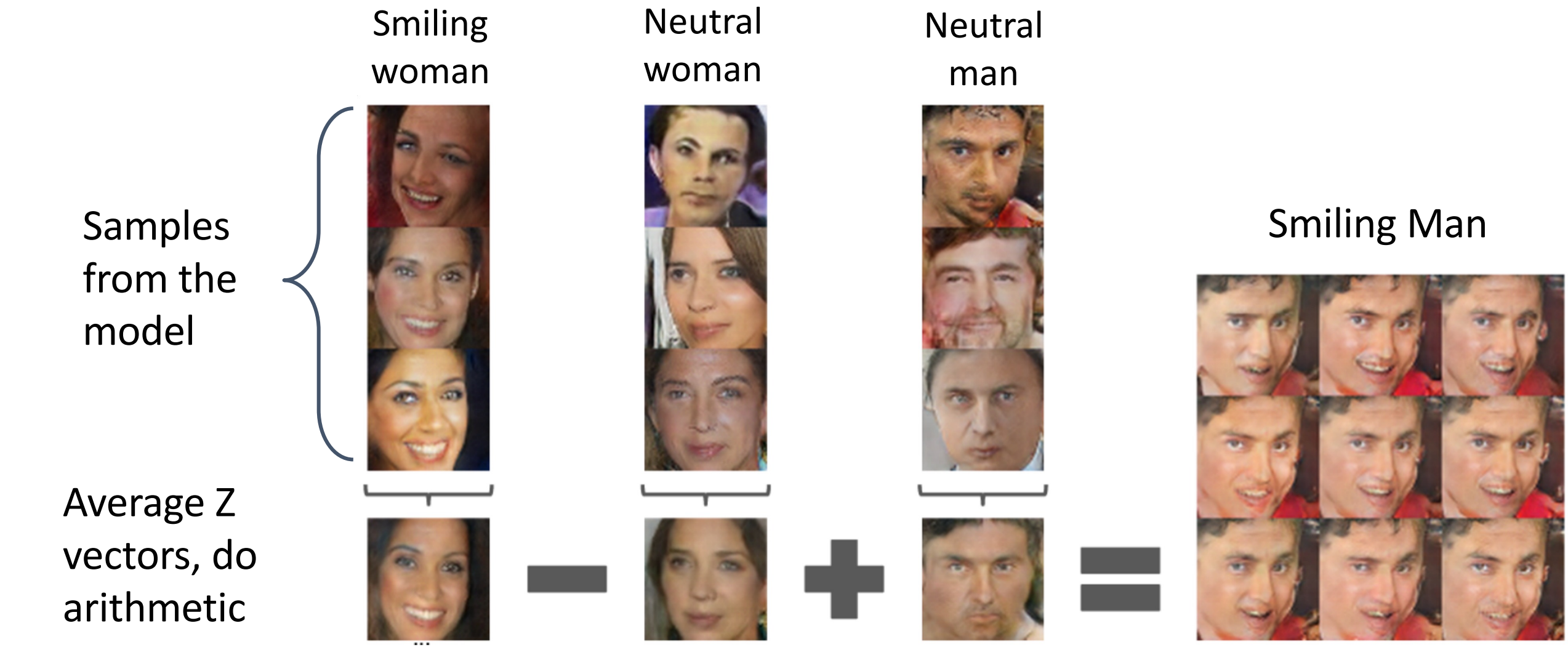
Radford et al, ICLR 2016

# Generative Adversarial Networks: Vector Math



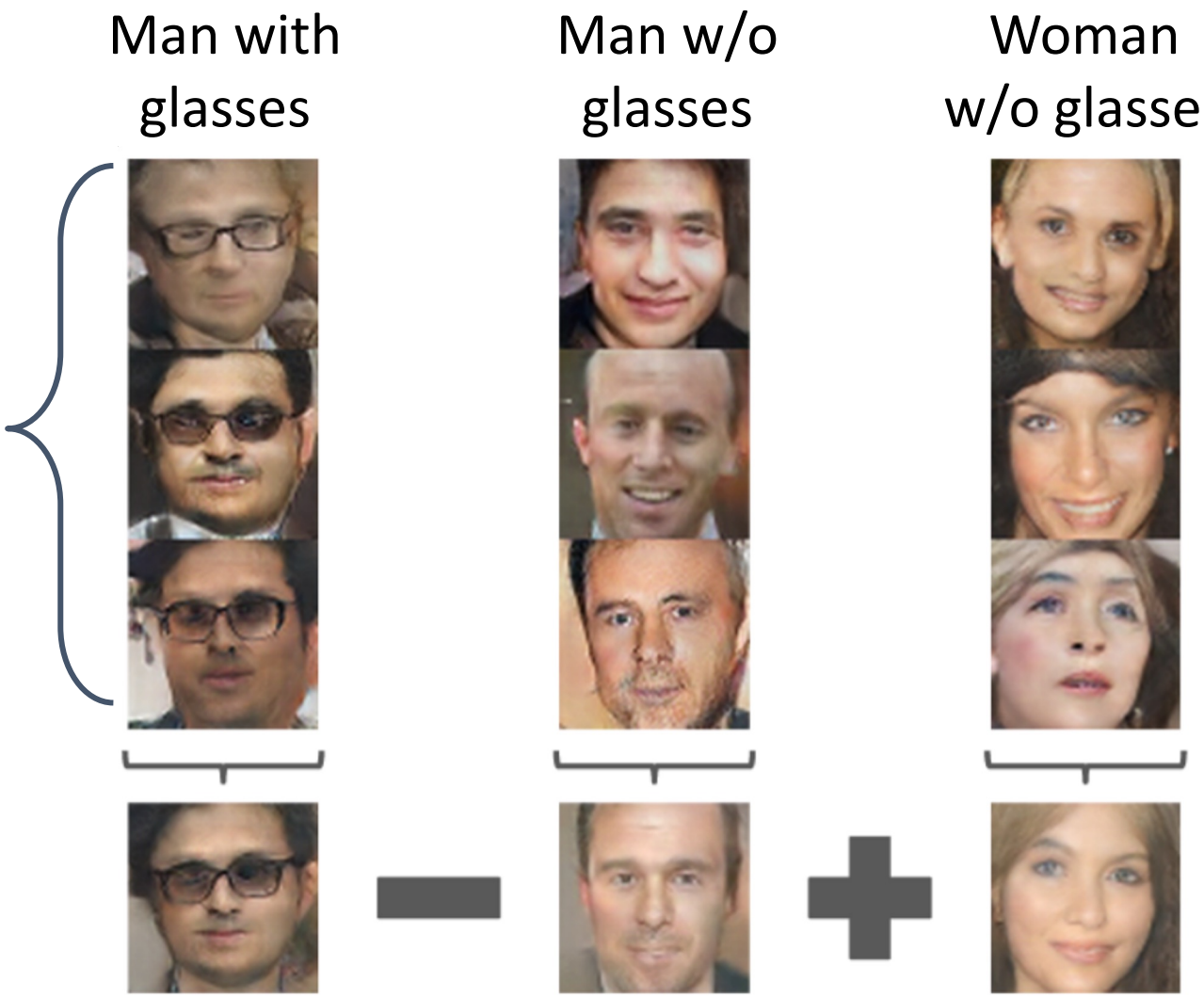

Radford et al, ICLR 2016

# Generative Adversarial Networks: Vector Math




Radford et al, ICLR 2016

# Generative Adversarial Networks: Vector Math




Radford et al, ICLR 2016

# Generative Adversarial Networks: Vector Math

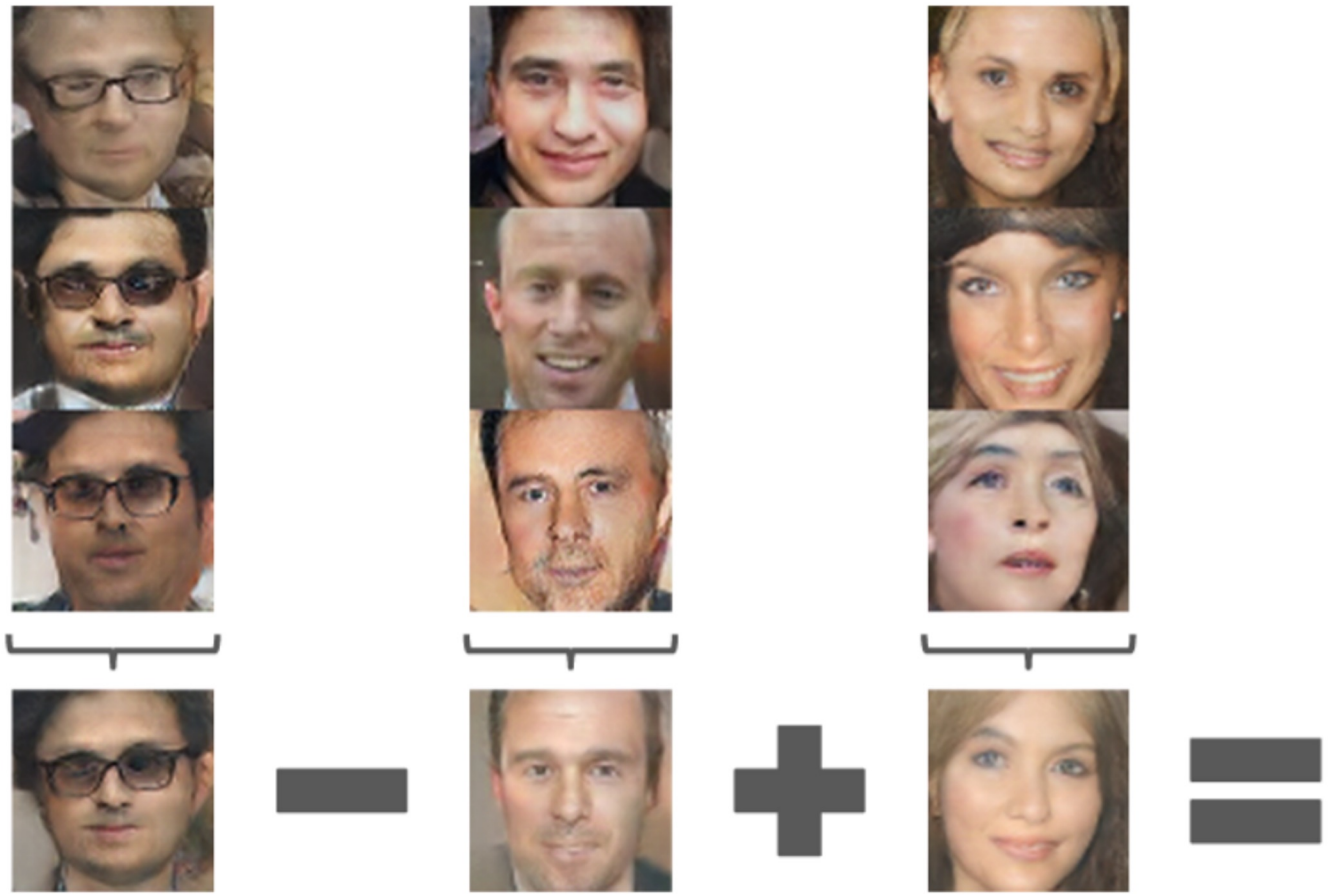
Man with glasses

Man w/o glasses

Woman w/o glasses

Samples from the model

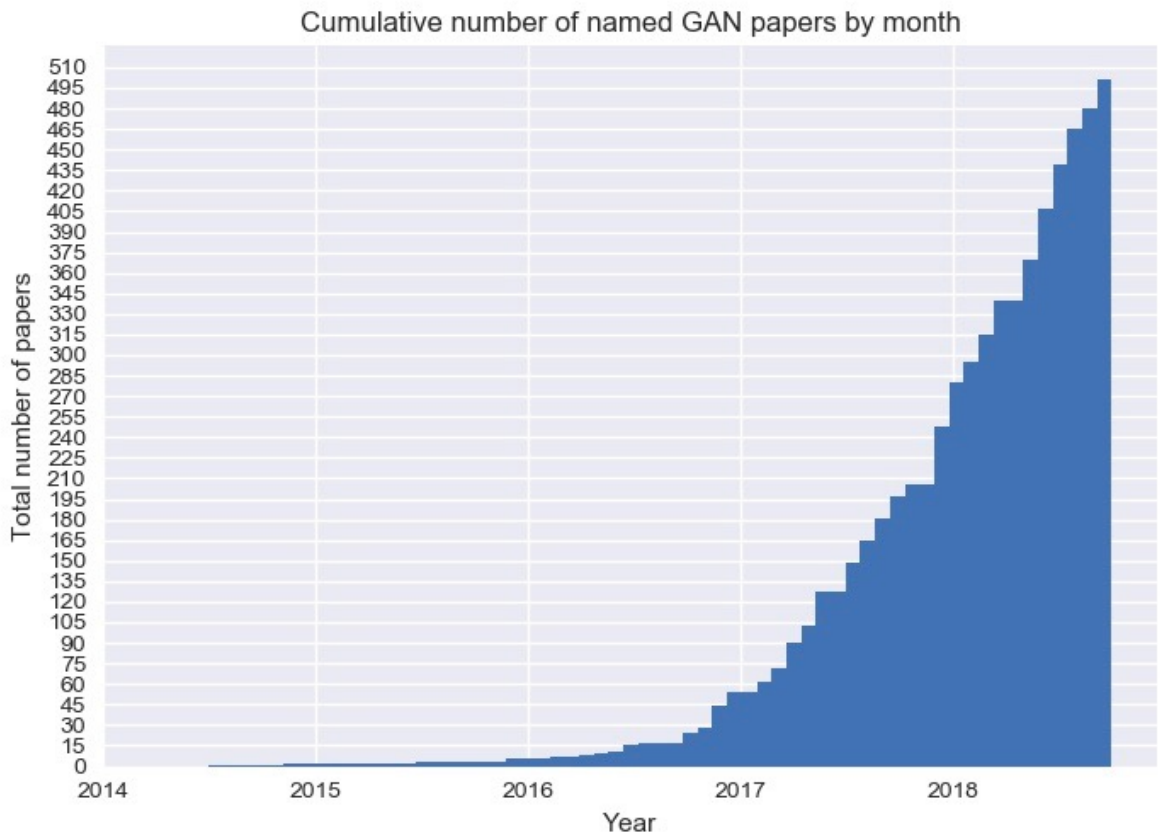
Average Z vectors, do arithmetic



Woman with glasses

Radford et al, ICLR 2016

# 2017 to 2022: Explosion of GANs




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

- 3D-ED-GAN - Shape Inpainting using 3D Generative Adversarial Network and Recurrent Convolutional Networks
- 3D-GAN - Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling (github)
- 3D-IWGAN - Improved Adversarial Systems for 3D Object Generation and Reconstruction (github)
- 3D-PhysNet - 3D-PhysNet: Learning the Intuitive Physics of Non-Rigid Object Deformations
- 3D-RecGAN - 3D Object Reconstruction from a Single Depth View with Adversarial Learning (github)
- ABC-GAN - ABC-GAN: Adaptive Blur and Control for improved training stability of Generative Adversarial Networks (github)
- ABC-GAN - GANs for LIFE: Generative Adversarial Networks for Likelihood Free Inference
- AC-GAN - Conditional Image Synthesis With Auxiliary Classifier GANs
- acGAN - Face Aging With Conditional Generative Adversarial Networks
- ACGAN - Coverless Information Hiding Based on Generative adversarial networks
- acGAN - On-line Adaptative Curriculum Learning for GANs
- ACtuAL - ACtuAL: Actor-Critic Under Adversarial Learning
- AdaGAN - AdaGAN: Boosting Generative Models
- Adaptive GAN - Customizing an Adversarial Example Generator with Class-Conditional GANs
- AdvEntuRe - AdvEntuRe: Adversarial Training for Textual Entailment with Knowledge-Guided Examples
- AdvGAN - Generating adversarial examples with adversarial networks
- AE-GAN - AE-GAN: adversarial eliminating with GAN
- AE-OT - Latent Space Optimal Transport for Generative Models
- AEGAN - Learning Inverse Mapping by Autoencoder based Generative Adversarial Nets
- AF-DCGAN - AF-DCGAN: Amplitude Feature Deep Convolutional GAN for Fingerprint Construction in Indoor Localization System
- AffGAN - Amortised MAP Inference for Image Super-resolution
- AIM - Generating Informative and Diverse Conversational Responses via Adversarial Information Maximization
- AL-CGAN - Learning to Generate Images of Outdoor Scenes from Attributes and Semantic Layouts
- ALI - Adversarially Learned Inference (github)
- AlignGAN - AlignGAN: Learning to Align Cross-Domain Images with Conditional Generative Adversarial Networks
- AlphaGAN - AlphaGAN: Generative adversarial networks for natural image matting
- AM-GAN - Activation Maximization Generative Adversarial Nets
- AmbientGAN - AmbientGAN: Generative models from lossy measurements (github)
- AMC-GAN - Video Prediction with Appearance and Motion Conditions
- AnoGAN - Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery
- APD - Adversarial Distillation of Bayesian Neural Network Posteriors
- APE-GAN - APE-GAN: Adversarial Perturbation Elimination with GAN
- ARAE - Adversarially Regularized Autoencoders for Generating Discrete Structures (github)
- ARDA - Adversarial Representation Learning for Domain Adaptation
- ARIGAN - ARIGAN: Synthetic Arabidopsis Plants using Generative Adversarial Network
- ArtGAN - ArtGAN: Artwork Synthesis with Conditional Categorial GANs
- ASDL-GAN - Automatic Steganographic Distortion Learning Using a Generative Adversarial Network
- ATA-GAN - Attention-Aware Generative Adversarial Networks (ATA-GANs)
- Attention-GAN - Attention-GAN for Object Transfiguration in Wild Images
- AttGAN - Arbitrary Facial Attribute Editing: Only Change What You Want (github)
- AttnGAN - AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks (github)
- AVID - AVID: Adversarial Visual Irregularity Detection
- B-DCGAN - B-DCGAN:Evaluation of Binarized DCGAN for FPGA
- b-GAN - Generative Adversarial Nets from a Density Ratio Estimation Perspective
- BAGAN - BAGAN: Data Augmentation with Balancing GAN
- Bayesian GAN - Deep and Hierarchical Implicit Models
- Bayesian GAN - Bayesian GAN (github)
- BCGAN - Bayesian Conditional Generative Adverserial Networks
- BCGAN - Bidirectional Conditional Generative Adversarial networks
- BEAM - Boltzmann Encoded Adversarial Machines
- BEGAN - BEGAN: Boundary Equilibrium Generative Adversarial Networks
- BEGAN-CS - Escaping from Collapsing Modes in a Constrained Space
- Bellman GAN - Distributional Multivariate Policy Evaluation and Exploration with the Bellman
- BGAN - Binary Generative Adversarial Networks for Image Retrieval (github)
- Bi-GAN - Autonomously and Simultaneously Refining Deep Neural Network Parameters by a Bi-Generative Adversarial Network Aided Genetic Algorithm
- BicycleGAN - Toward Multimodal Image-to-Image Translation (github)
- BiGAN - Adversarial Feature Learning
- BinGAN - BinGAN: Learning Compact Binary Descriptors with a Regularized GAN
- BourGAN - BourGAN: Generative Networks with Metric Embeddings
- BranchGAN - Branched Generative Adversarial Networks for Multi-Scale Image Manifold Learning
- BRE - Improving GAN Training via Binarized Representation Entropy (BRE) Regularization (github)
- BridgeGAN - Generative Adversarial Frontal View to Bird View Synthesis
- BS-GAN - Boundary-Seeking Generative Adversarial Networks
- BubGAN - BubGAN: Bubble Generative Adversarial Networks for Synthesizing Realistic Bubbly Flow Images
- BWGAN - Banach Wasserstein GAN
- C-GAN - Face Aging with Contextual Generative Adversarial Nets
- C-RNN-GAN - C-RNN-GAN: Continuous recurrent neural networks with adversarial training (github)
- CA-GAN - Composition-aided Sketch-realistic Portrait Generation
- CaloGAN - CaloGAN: Simulating 3D High Energy Particle Showers in Multi-Layer Electromagnetic Calorimeters with Generative Adversarial Networks (github)
- CAN - CAN: Creative Adversarial Networks, Generating Art by Learning About Styles and Deviating from Style Norms
- CapsGAN - CapsGAN: Using Dynamic Routing for Generative Adversarial Networks
- CapsuleGAN - CapsuleGAN: Generative Adversarial Capsule Network
- CatGAN - Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks
- CatGAN - CatGAN: Coupled Adversarial Transfer for Domain Generation
- CausalGAN - CausalGAN: Learning Causal Implicit Generative Models with Adversarial Training
- CC-GAN - Semi-Supervised Learning with Context-Conditional Generative Adversarial Networks (github)
- cd-GAN - Conditional Image-to-Image Translation
- CDcGAN - Simultaneously Color-Depth Super-Resolution with Conditional Generative Adversarial Network
- CE-GAN - Deep Learning for Imbalance Data Classification using Class Expert Generative Adversarial Network
- CFG-GAN - Composite Functional Gradient Learning of Generative Adversarial Models
- CGAN - Conditional Generative Adversarial Nets
- CGAN - Controllable Generative Adversarial Network
- Chekhov GAN - An Online Learning Approach to Generative Adversarial Networks
- ciGAN - Conditional Infilling GANs for Data Augmentation in Mammogram Classification
- CinCGAN - Unsupervised Image Super-Resolution using Cycle-in-Cycle Generative Adversarial Networks
- CipherGAN - Unsupervised Cipher Cracking Using Discrete GANs
- ClusterGAN - ClusterGAN : Latent Space Clustering in Generative Adversarial Networks
- CM-GAN - CM-GANs: Cross-modal Generative Adversarial Networks for Common Representation Learning
- CoAtt-GAN - Are You Talking to Me? Reasoned Visual Dialog Generation through Adversarial Learning
- CoGAN - Coupled Generative Adversarial Networks
- ComboGAN - ComboGAN: Unrestrained Scalability for Image Domain Translation (github)
- ConceptGAN - Learning Compositional Visual Concepts with Mutual Consistency
- Conditional cycleGAN - Conditional CycleGAN for Attribute Guided Face Image Generation
- constrast-GAN - Generative Semantic Manipulation with Contrasting GAN
- Context-RNN-GAN - Contextual RNN-GANs for Abstract Reasoning Diagram Generation
- CorrGAN - Correlated discrete data generation using adversarial training
- Coulomb GAN - Coulomb GANs: Provably Optimal Nash Equilibria via Potential Fields
- Cover-GAN - Generative Steganography with Kerckhoffs' Principle based on Generative Adversarial Networks
- cowboy - Defending Against Adversarial Attacks by Leveraging an Entire GAN
- CR-GAN - CR-GAN: Learning Complete Representations for Multi-view Generation
- Cramèr GAN - The Cramer Distance as a Solution to Biased Wasserstein Gradients
- Cross-GAN - Crossing Generative Adversarial Networks for Cross-View Person Re-identification
- crVAE-GAN - Channel-Recurrent Variational Autoencoders
- CS-GAN - Improving Neural Machine Translation with Conditional Sequence Generative Adversarial Nets
- CSG - Speech-Driven Expressive Talking Lips with Conditional Sequential Generative Adversarial Networks
- CT-GAN - CT-GAN: Conditional Transformation Generative Adversarial Network for Image Attribute Modification
- CVAE-GAN - CVAE-GAN: Fine-Grained Image Generation through Asymmetric Training
- CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks

# GAN Improvements: Improved Loss Functions

## Wasserstein GAN (WGAN)

Arjovsky, Chintala, and Bouttou, “Wasserstein GAN”, 2017

## WGAN with Gradient Penalty (WGAN-GP)

Gulrajani et al, “Improved Training of Wasserstein GANs”, NeurIPS 2017

# GAN Improvements: StyleGAN for Higher Resolution

256 x 256 bedrooms

1024 x 1024 faces

Karras et al, “Progressive Growing of GANs for Improved Quality, Stability, and Variation”, ICLR 2018

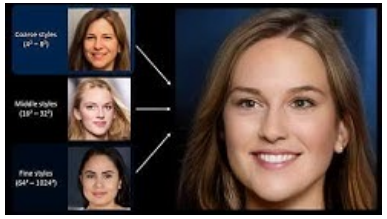
# GAN Improvements: StyleGAN for Higher Resolution

512 x 384 cars

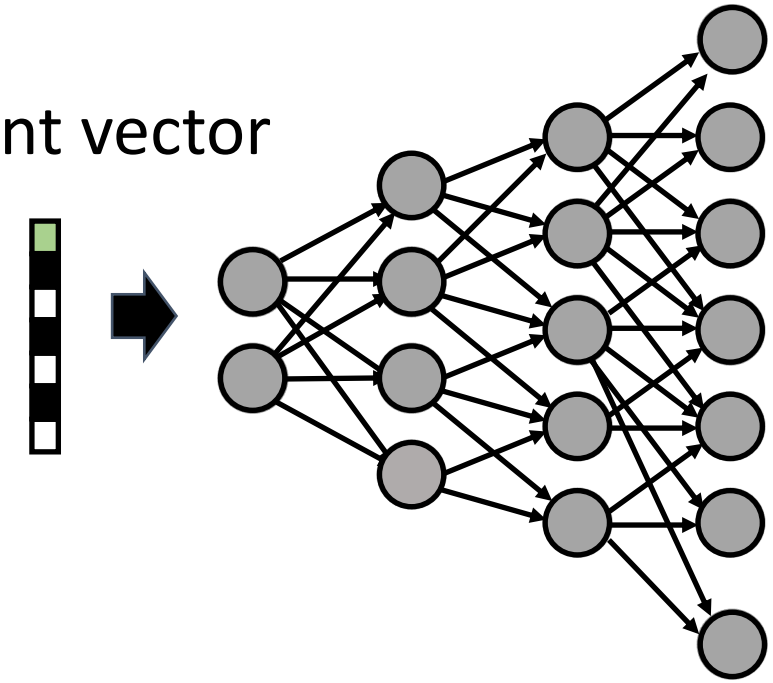
1024 x 1024 faces

Karras et al, "A Style-Based Generator Architecture for Generative Adversarial Networks", CVPR 2019

# Layer-wise Stochasity in StyleGAN



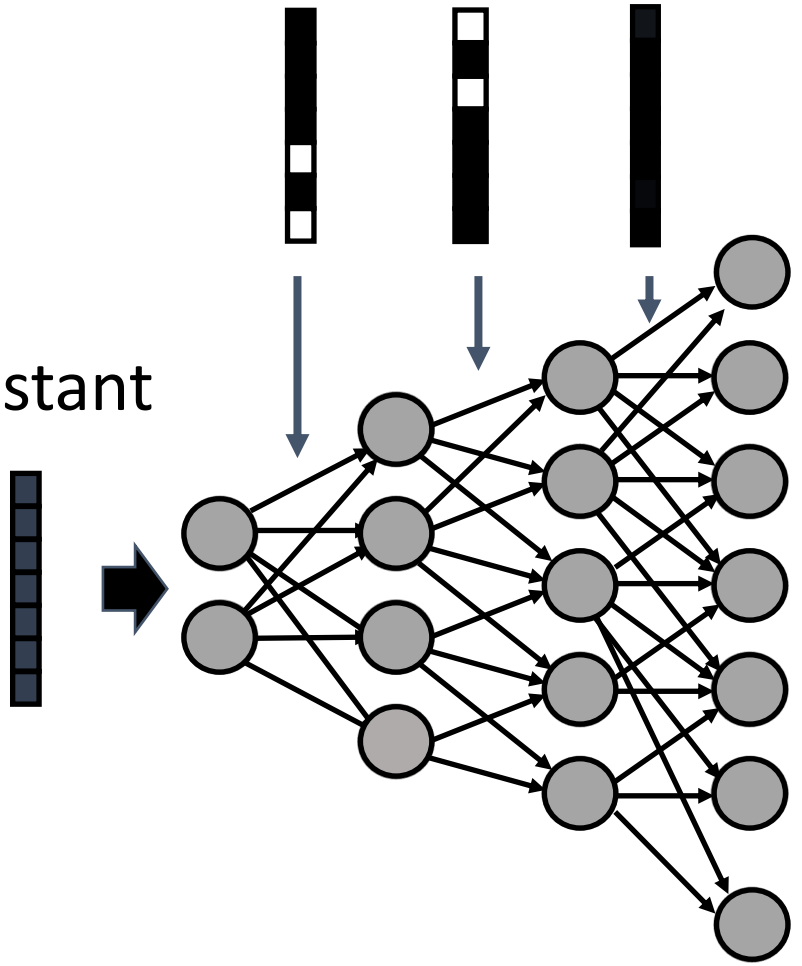
Latent vector



DC-GAN, PG-GAN

Layer-wise latent vectors

Constant



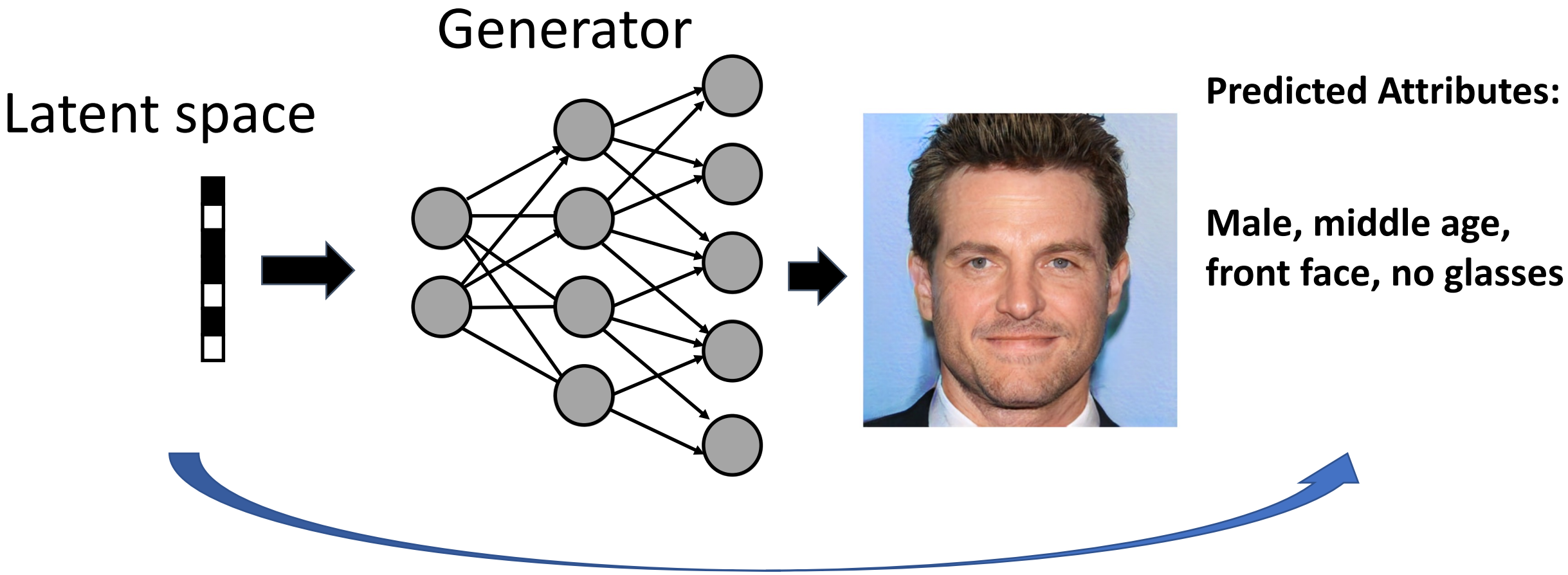
StyleGAN, StyleGANv2 [Karras et al]

# Random walk in the latent space

Karras et al,

Fd7kuaTkyeRJ86

# InterFaceGAN: Identifying Causal Relations in the Latent Space



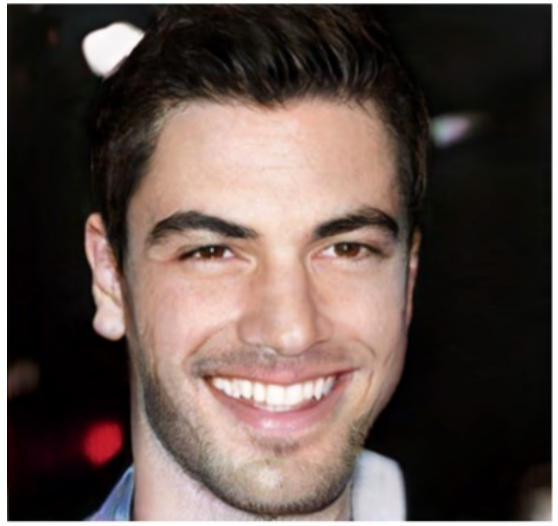

https://genforce.github.io/interfacegan/ , CVPR 2020, TPAMI 2020

Make me cooler

Make me younger

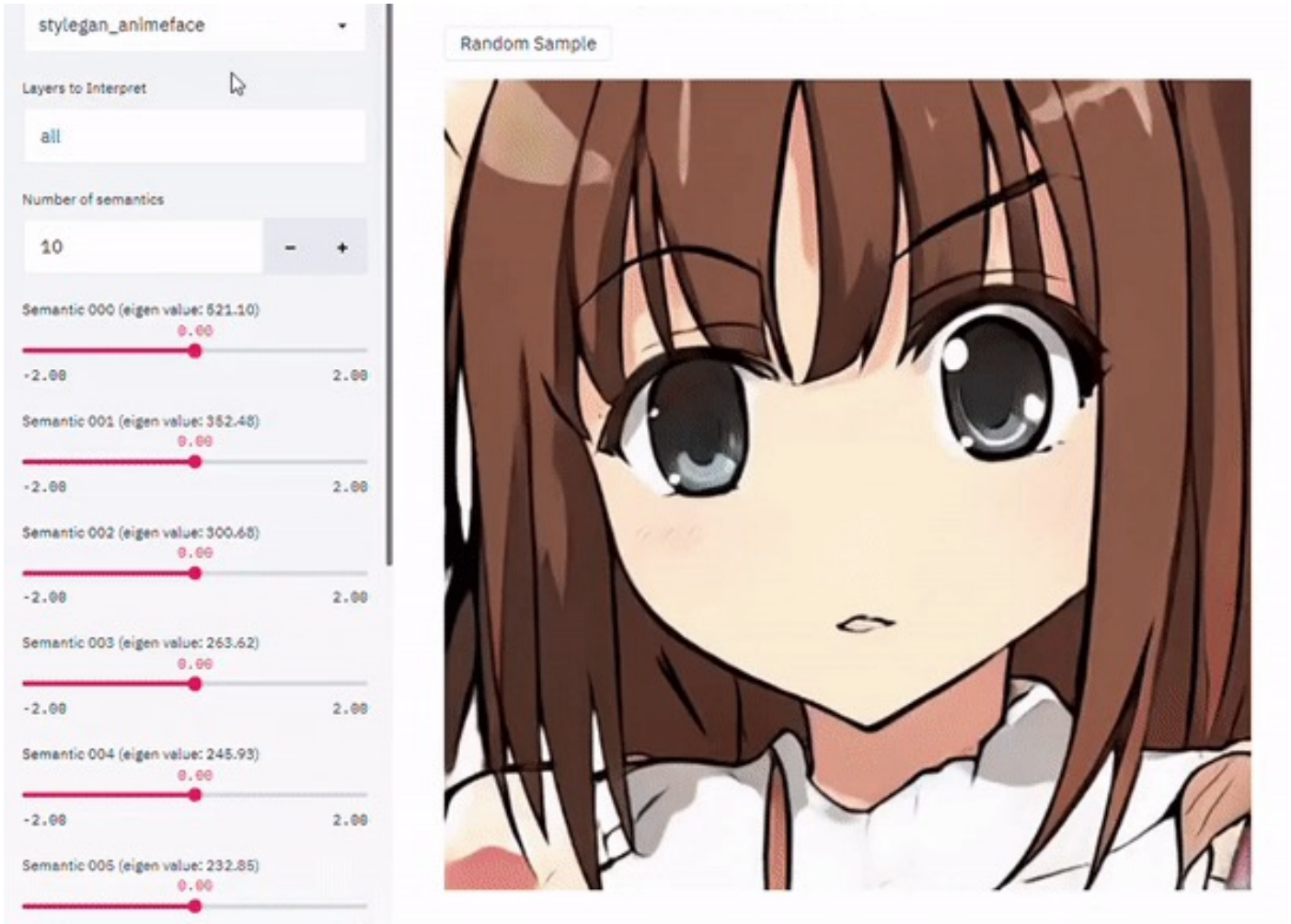
Make me front faced

Make me more man




Demo Video for
Semantic Face Editing
with InterFaceGAN

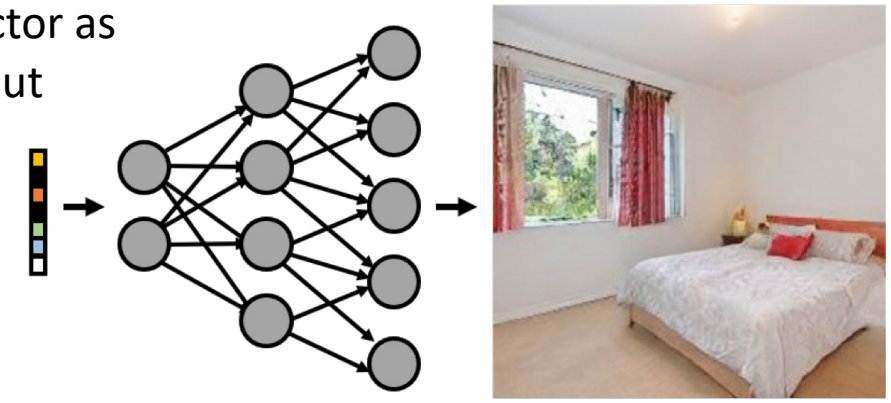
# Unsupervised Discovery of Steerable Factors

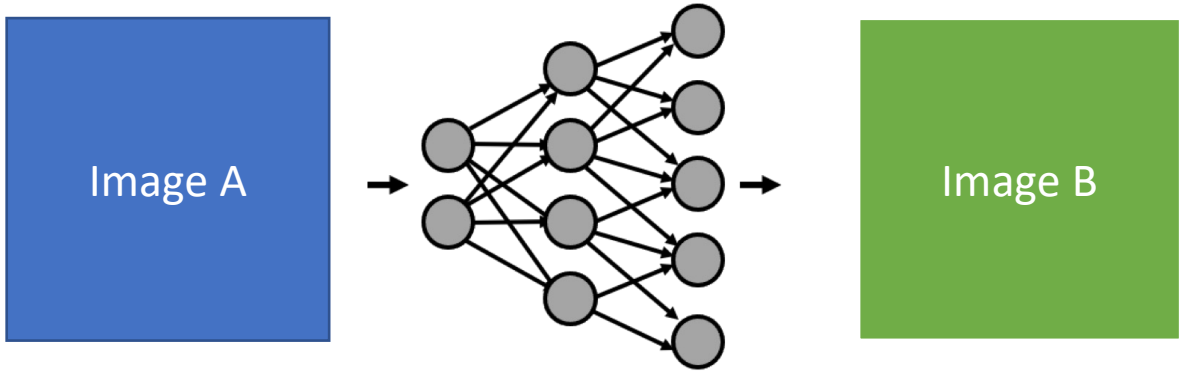



https://genforce.github.io/sefa/ (CVPR'21)

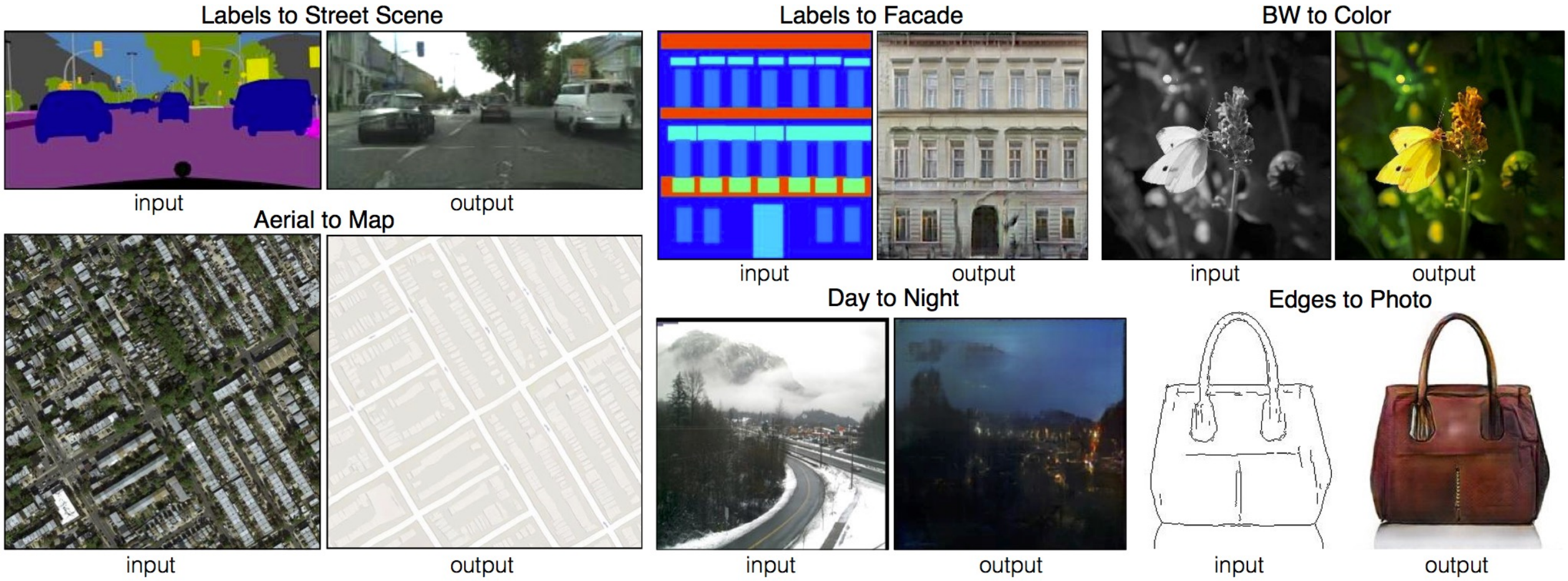
# Image-to-Image Translation: Pix2Pix

Random vector as input





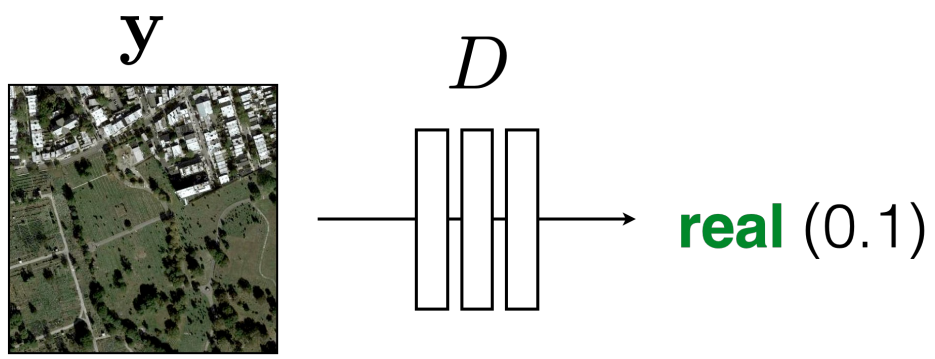
# Image-to-Image Translation: Pix2Pix




Isola et al, "Image-to-Image Translation with Conditional Adversarial Nets", CVPR 2017

# Image-to-Image Translation: Pix2Pix

Instead of input a random noise, we can input an image

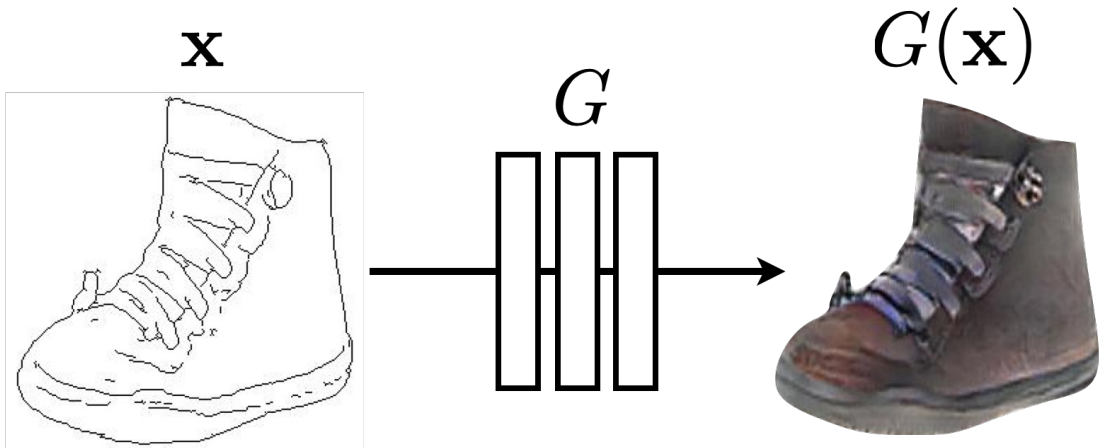

Philip Isola at MIT

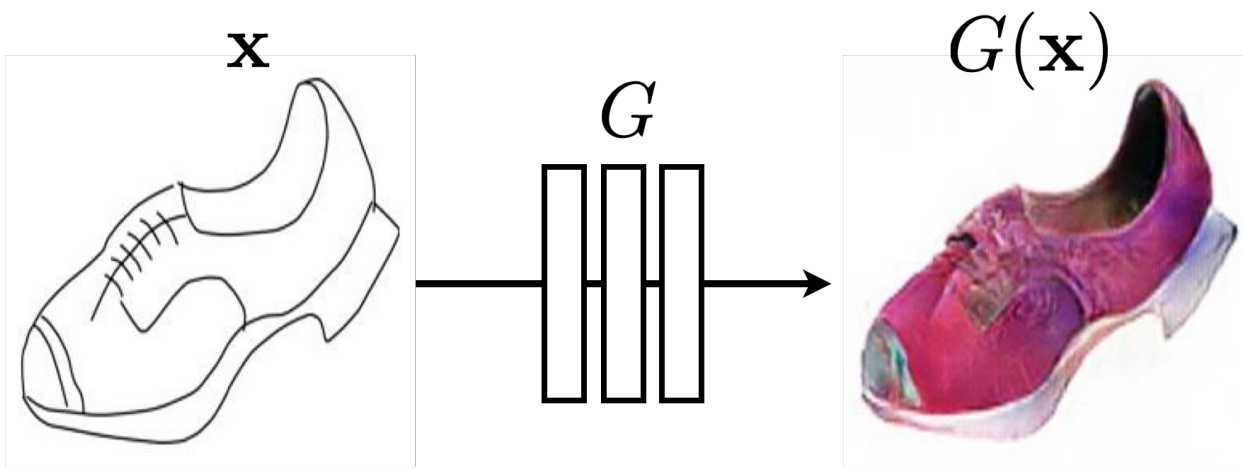
$$\arg\max_D \mathbb{E}_{\mathbf{x},\mathbf{y}}[\log D(G(\mathbf{x})) + \log(1 - D(\mathbf{y}))]$$

Isola et al, "Image-to-Image Translation with Conditional Adversarial Nets", CVPR 2017

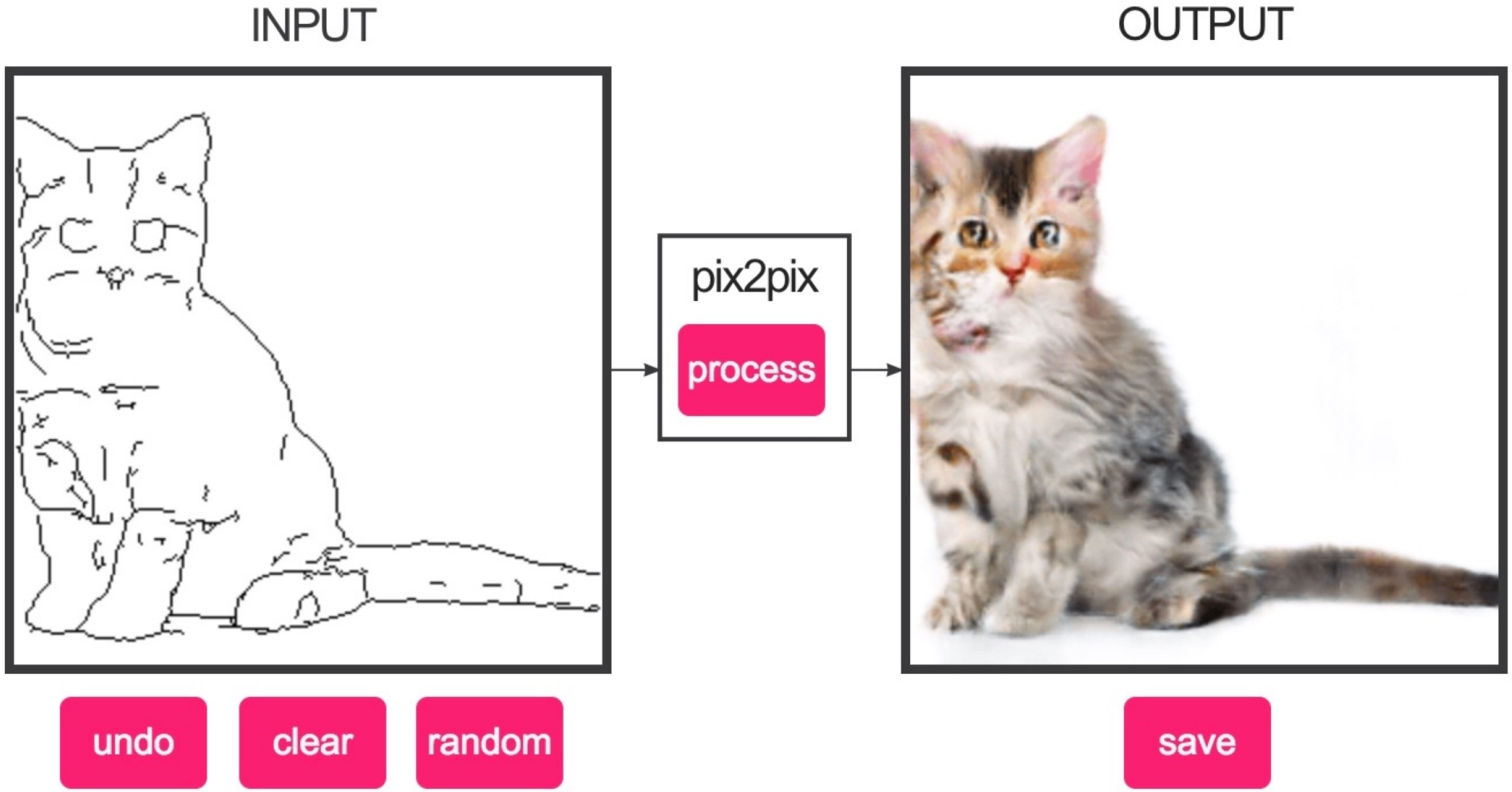
## *Training data*


[HED, Xie & Tu, 2015]

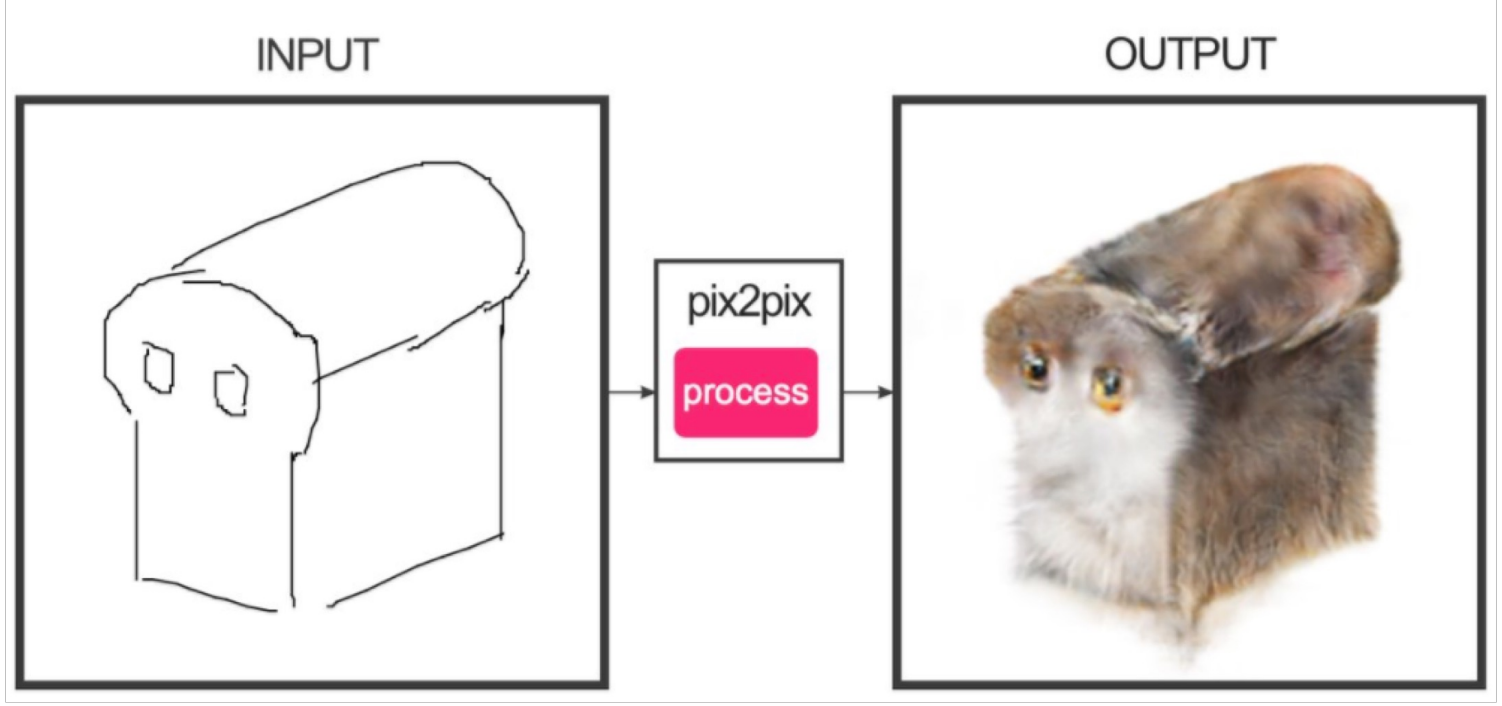


# Many creative applications of Pix2Pix




#edges2cats [Chris Hesse]

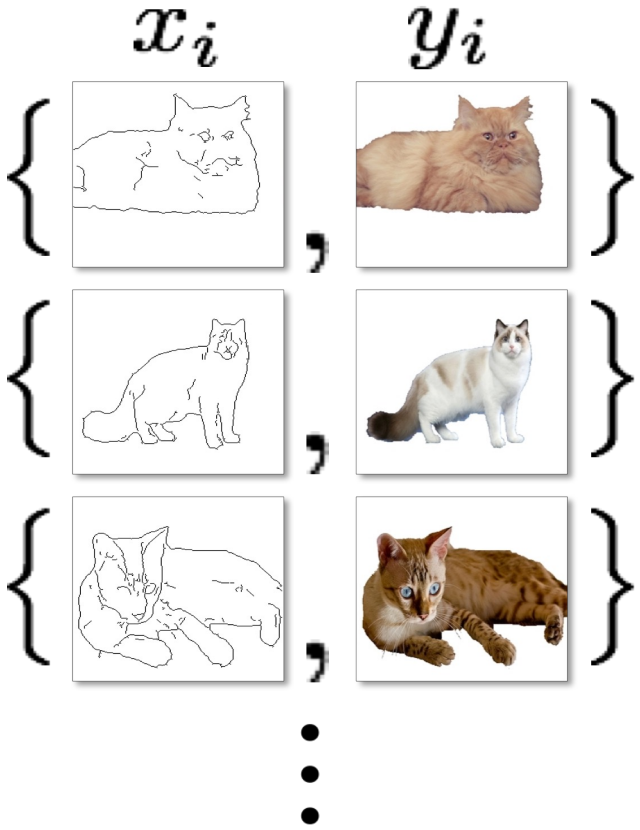



Ivy Tasi @ivymyt

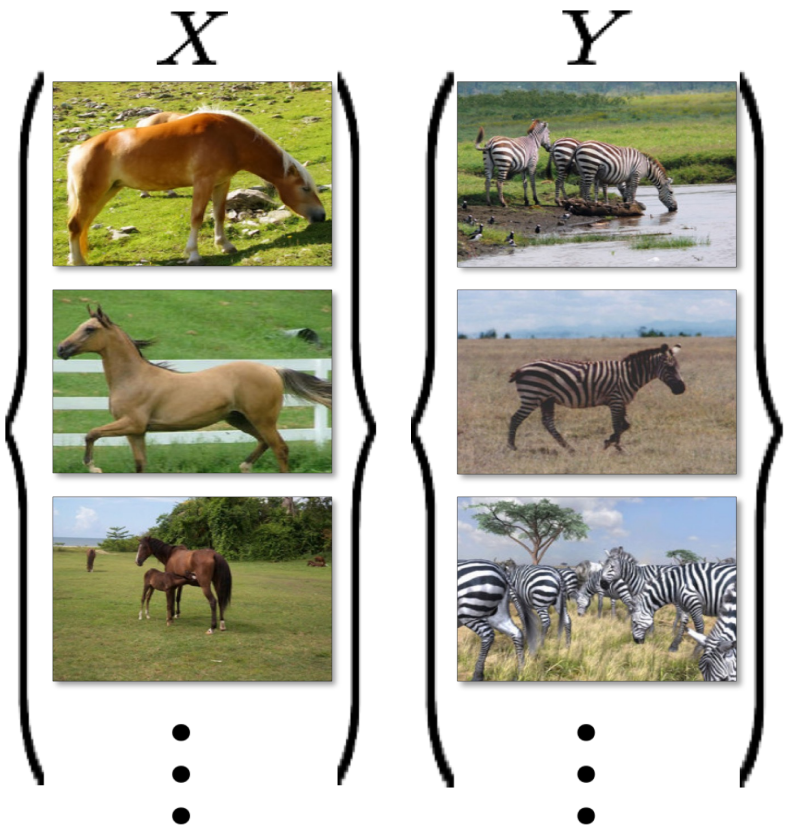
Vitaly Vidmirov @vvid

# How to handle unpaired data?

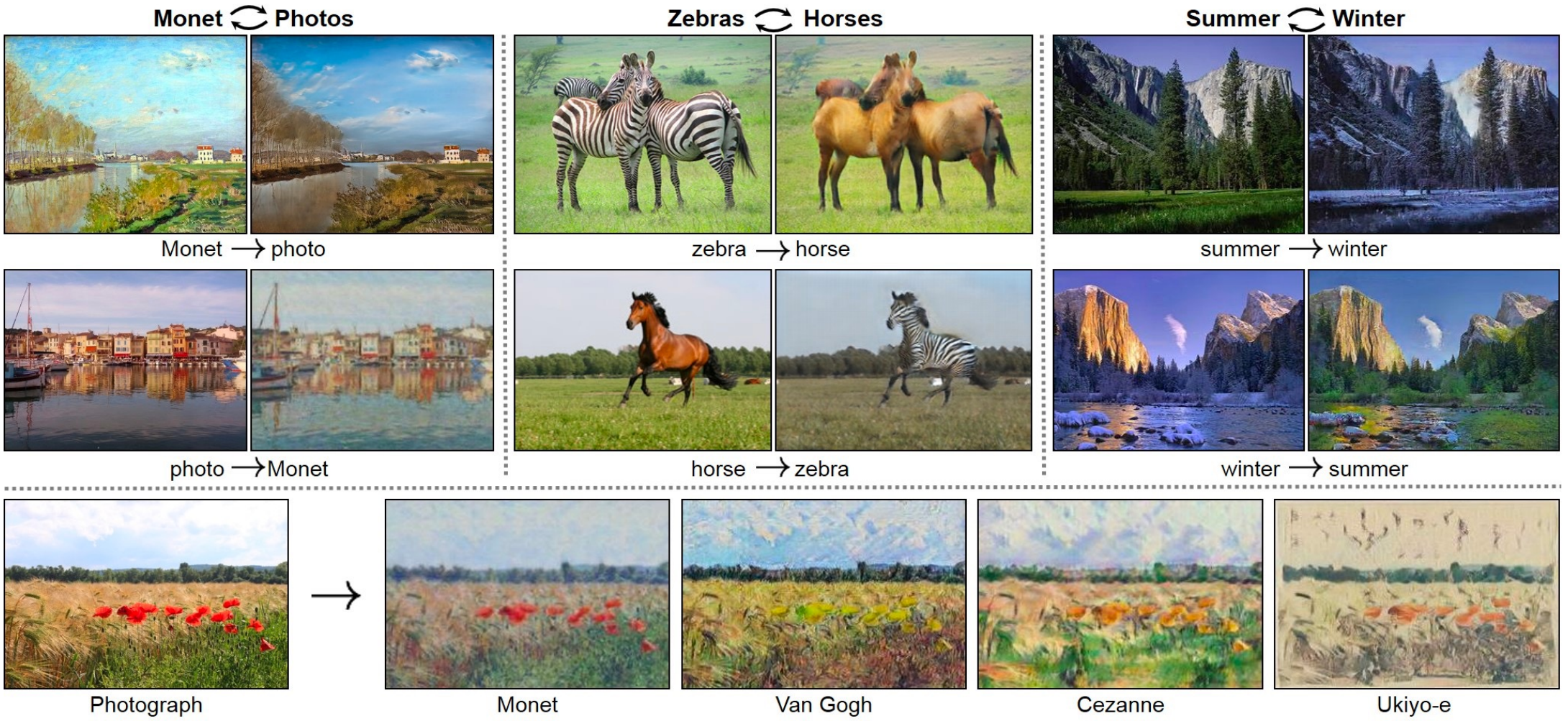
Paired data




Unpaired data



# Unpaired Image-to-Image Translation: CycleGAN




Zhu et al, "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks", ICCV 2017

# Unpaired Image-to-Image Translation: CycleGAN

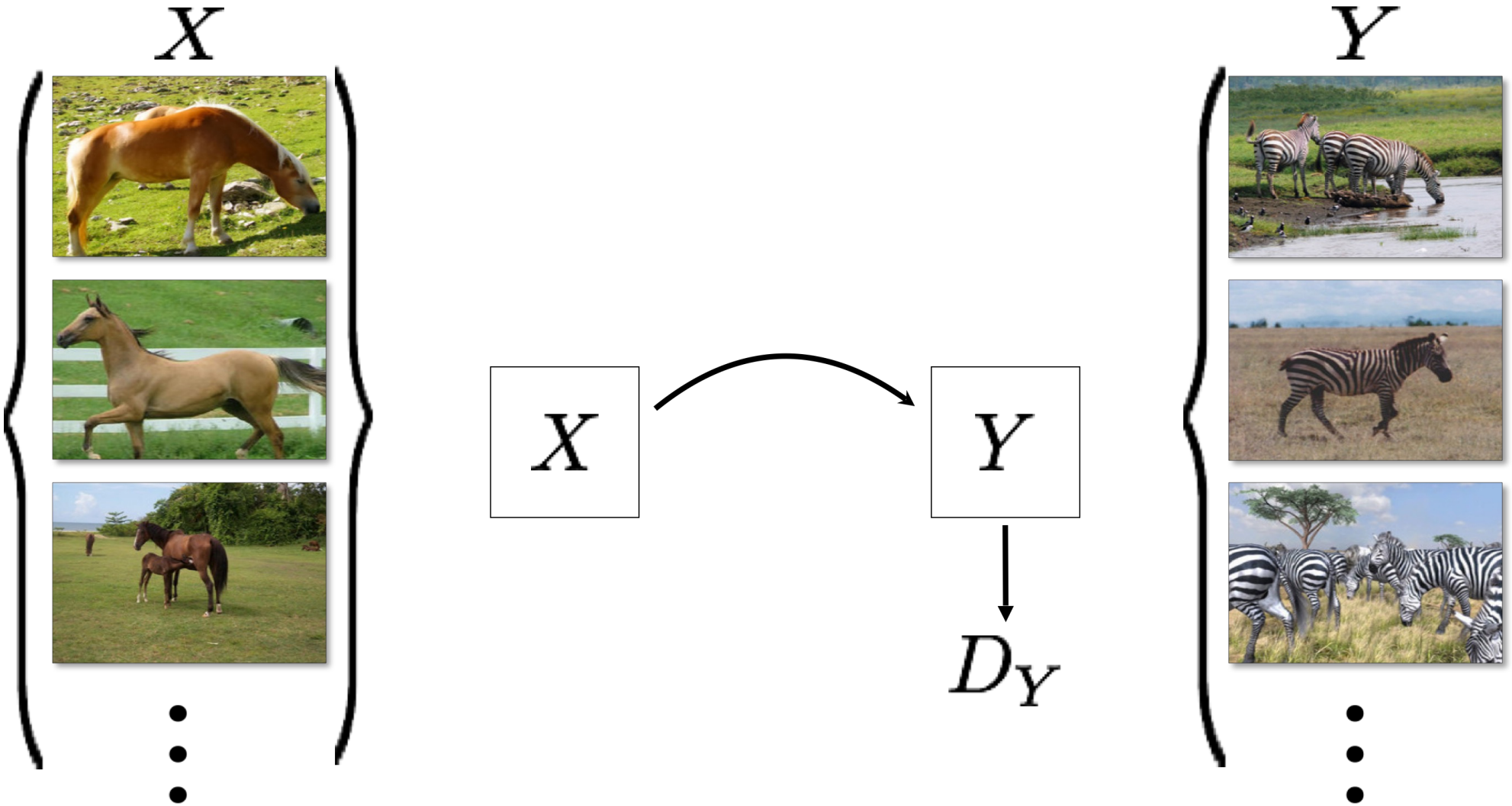
Input Video: Horse

Output Video: Zebra

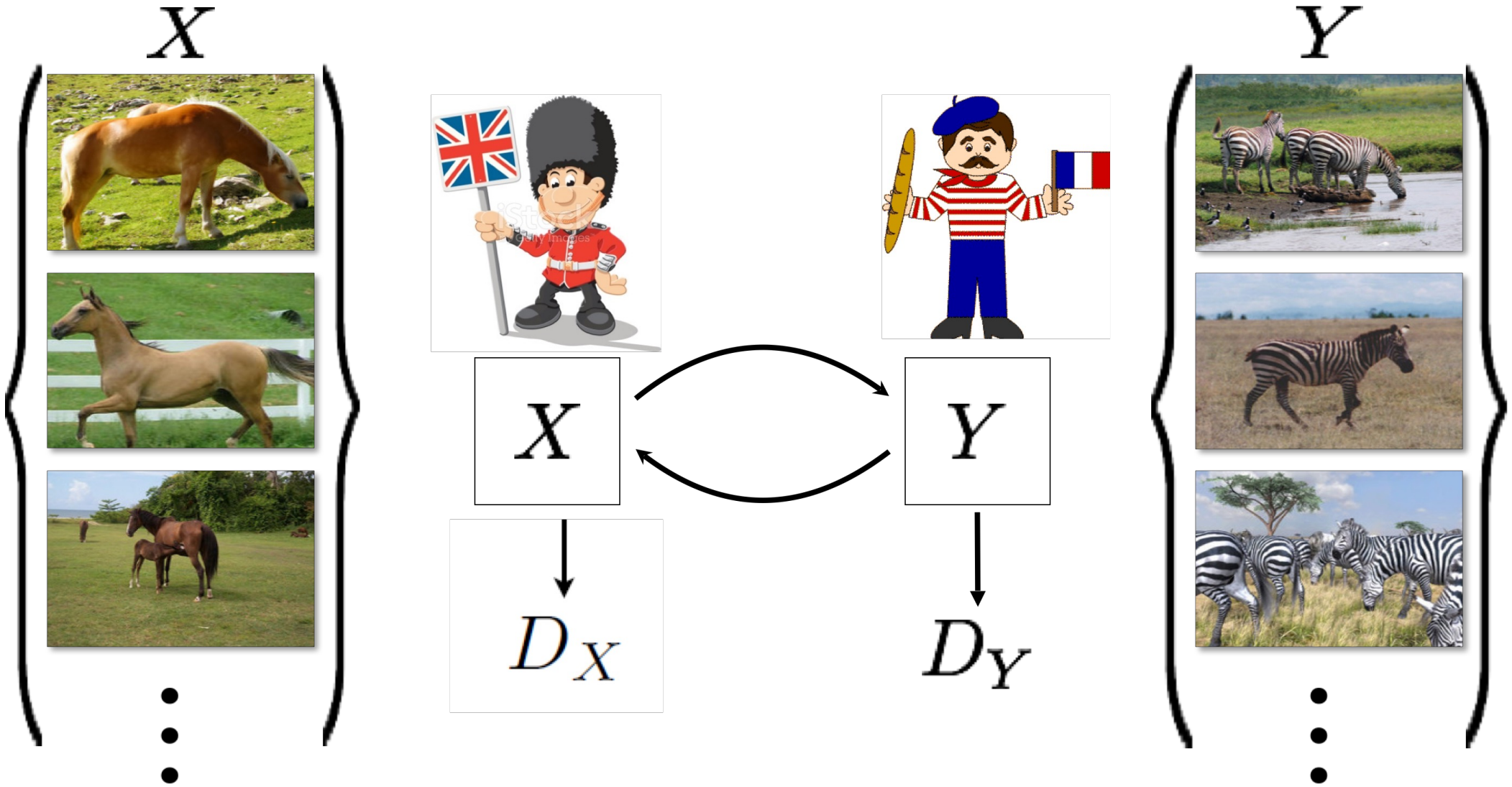
https://www.youtube.com/watch?v=9reHvktowLY

Zhu et al, “Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks”, ICCV 2017

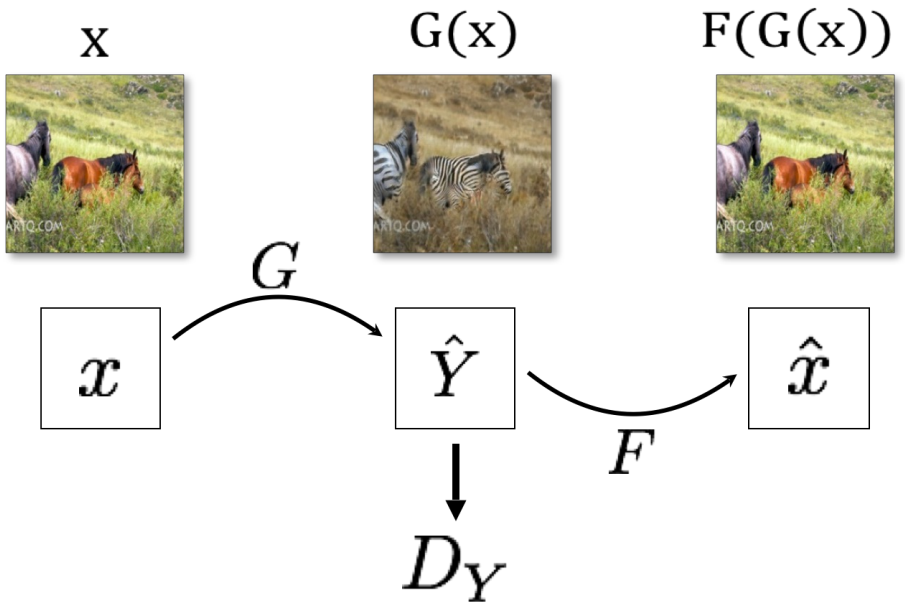
# CycleGAN, or there and back aGAN

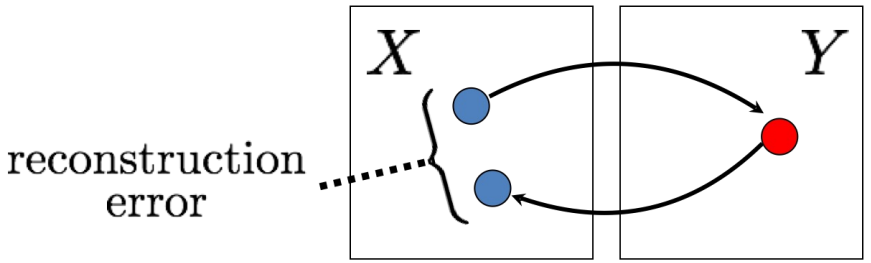


# CycleGAN, or there and back aGAN



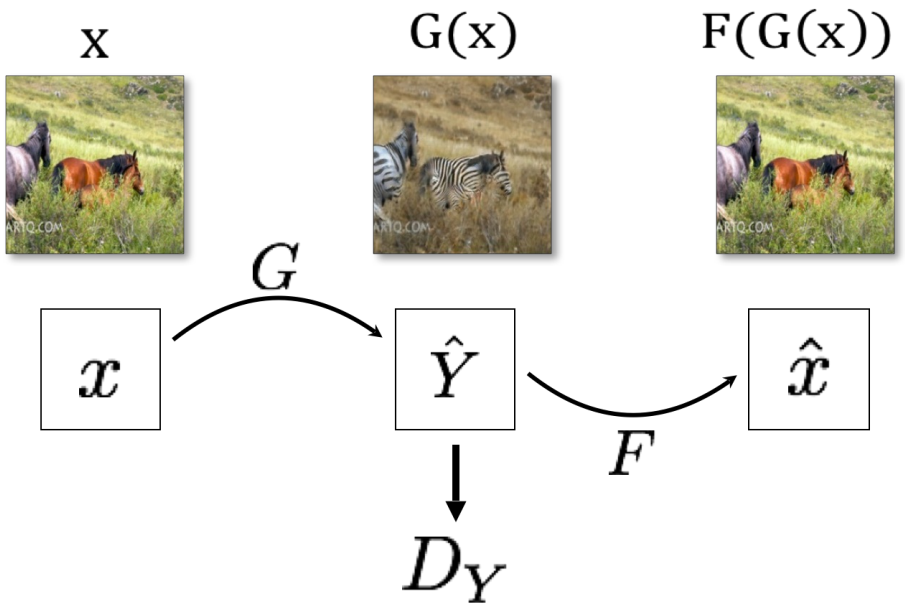
# Cycle Consistency Loss

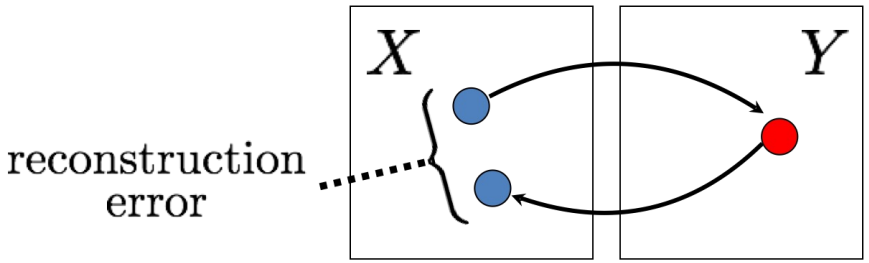


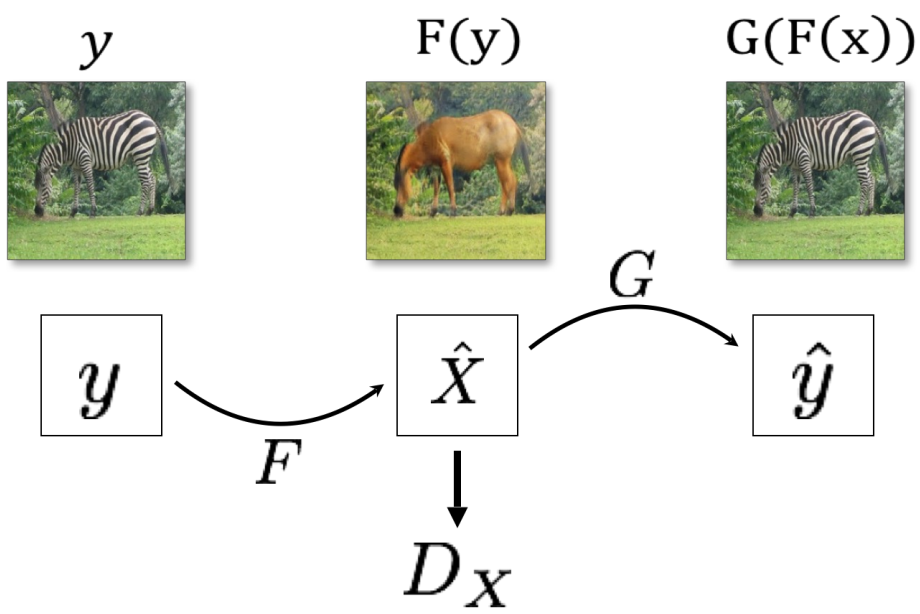

$$\|F(G(x)) - x\|_1$$

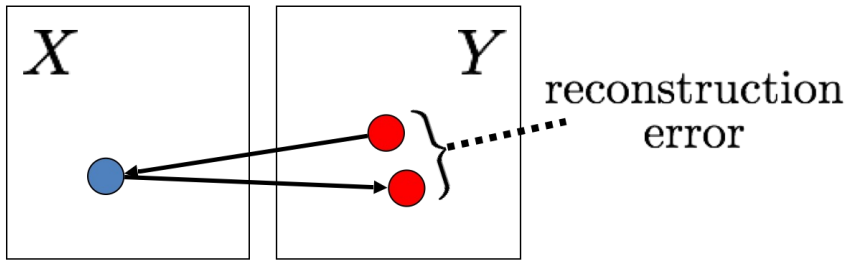
# Cycle Consistency Loss



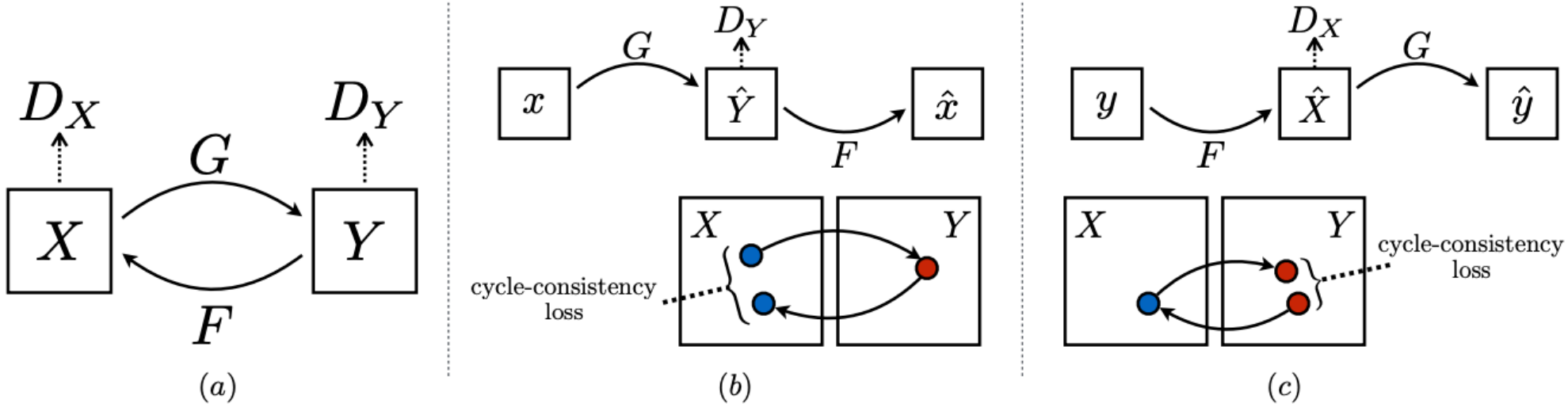

$$\|F(G(x)) - x\|_1$$




$$\|G(F(y)) - y\|_1$$

# Cycle Consistency Loss: full objective




$$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log(1 - D_Y(G(x))],$$

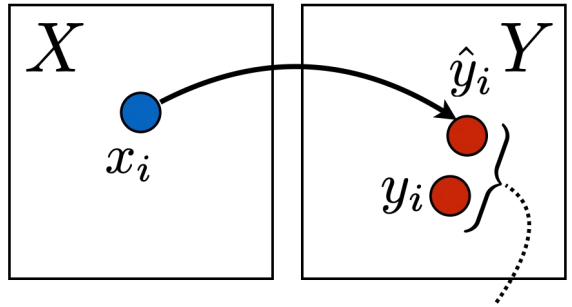
$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1].$$

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) + \mathcal{L}_{\text{GAN}}(F, D_X, Y, X) + \lambda \mathcal{L}_{\text{cyc}}(G, F),$$

## Paired translation

Training data


Objective




*regression error*

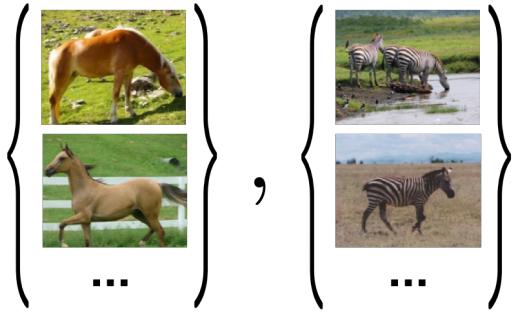
Input


Result

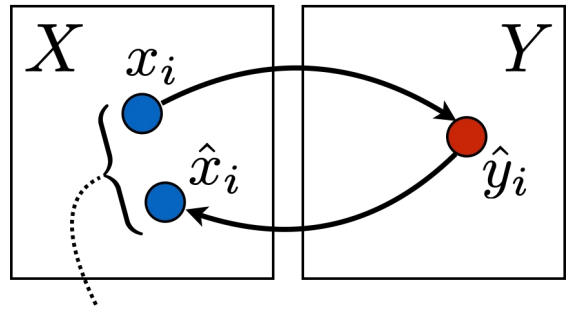

["pix2pix", Isola, Zhu, Zhou, Efros, 2017]

## Unpaired translation

Training data



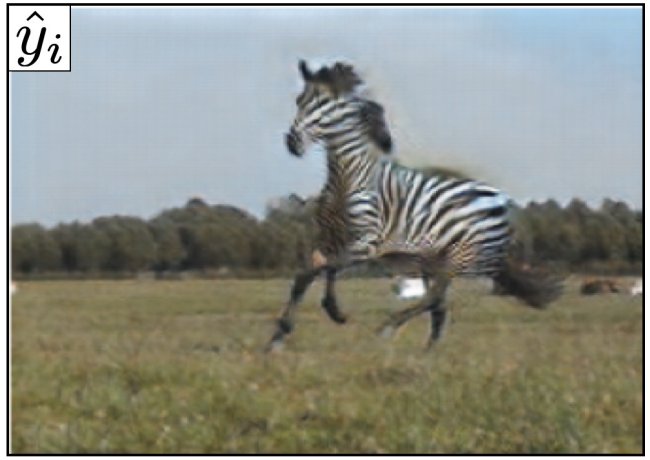

Objective




*cycle-consistency error*

Input


Result




["CycleGAN", Zhu*, Park*, Isola, Efros, 2017]

# Many many cool applications

## Portrait to Dollface

**Mario Klingemann** used our code to translate portraits into dollface. See how the characters in Game of Thrones look like in the doll world.

## Face ↔ Ramen

**Takuya Kato** performed a magical and hilarious Face ↔ Ramen translation with CycleGAN. Check out more results **here**

## Colorizing legacy photographs

**Mario Klingemann** trained our method to turn legacy black and white photos into color versions.

## Cats ↔ Dogs

**itok_msi** produced cats ↔ dogs CycleGAN results with a local+global discriminator and a smaller cycle loss.

## The Electronic Curator


Eran Hadas and Eyal Gruss used CycleGAN to convert human faces into vegetable portraits. They built a real-time art demo which allows users to interact with the model with their own faces.

## Turning Fortnite into PUBG


**Chintan Trivedi** used CycleGAN to translate between Fornite and PURB, two popular Battle Royale games with hundreds of millions of users. Now you can enjoy the gameplay of one game in the visuals of the other. Check out his **blog** for more cool demos.

https://junyanz.github.io/CycleGAN/

# Taxonomy of Generative Models

Model can compute p(x)

Model does not explicitly compute p(x), but can sample from p(x)

**Generative models**

Explicit density

Implicit density

Can compute approximation to p(x)

Tractable density

Approximate density

Markov Chain

Direct

Can compute p(x)
- Autoregressive
- NADE / MADE
- NICE / RealNVP
- Glow
- Ffjord

GSN

Generative Adversarial Networks (GANs)

Variational

Markov Chain

Variational Autoencoder

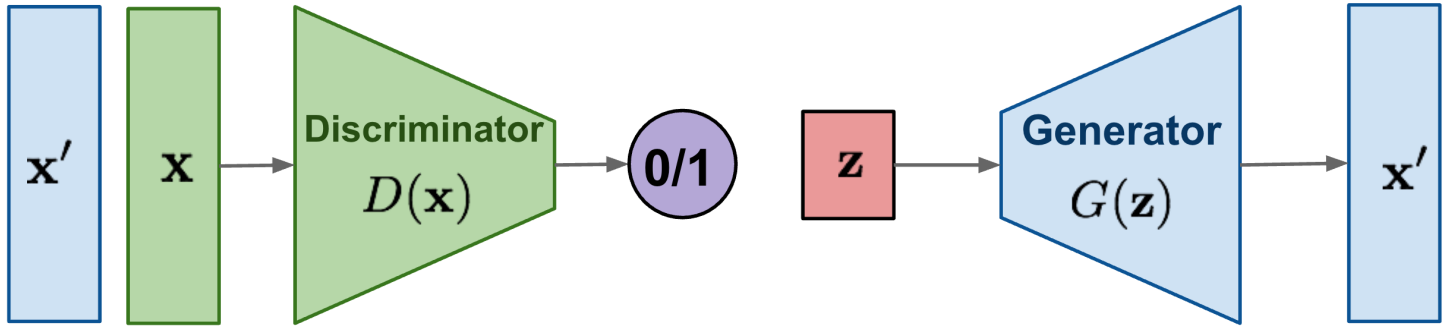
Boltzmann Machine

Diffusion Model

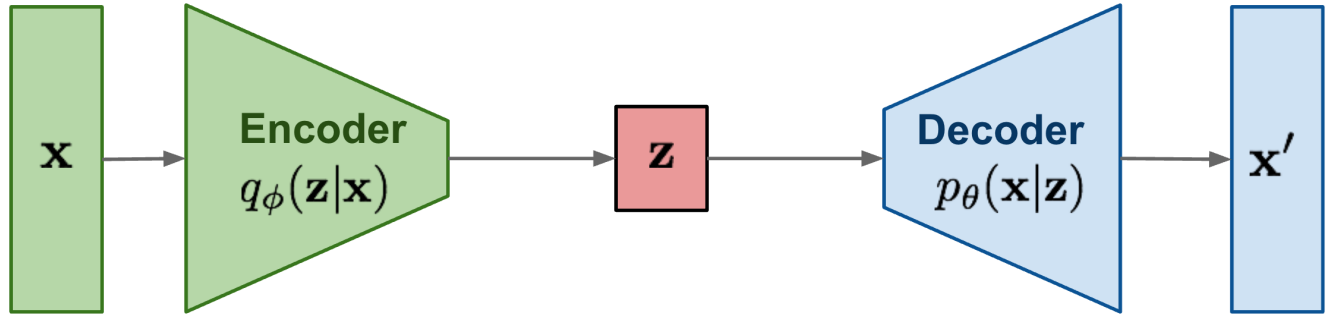
Nowadays…

# Diffusion Models: Overview

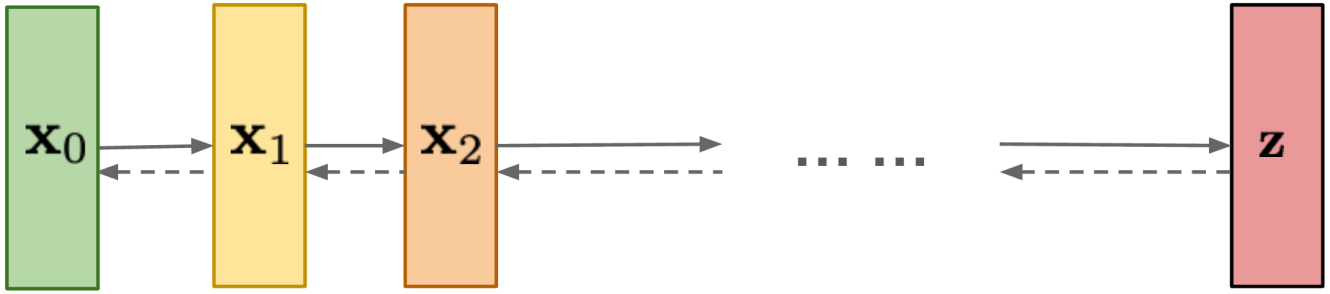
**GAN:** Adversarial training




**VAE:** maximize variational lower bound

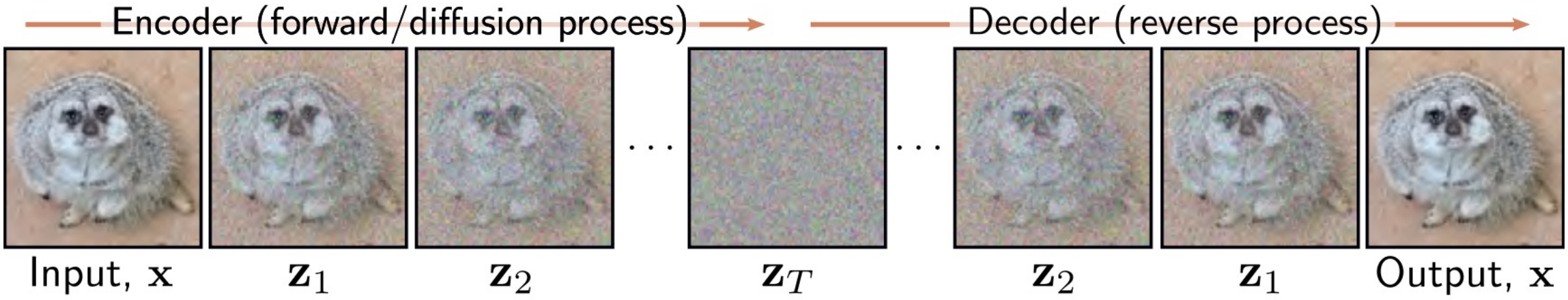



**Diffusion models:** Gradually add Gaussian noise and then reverse

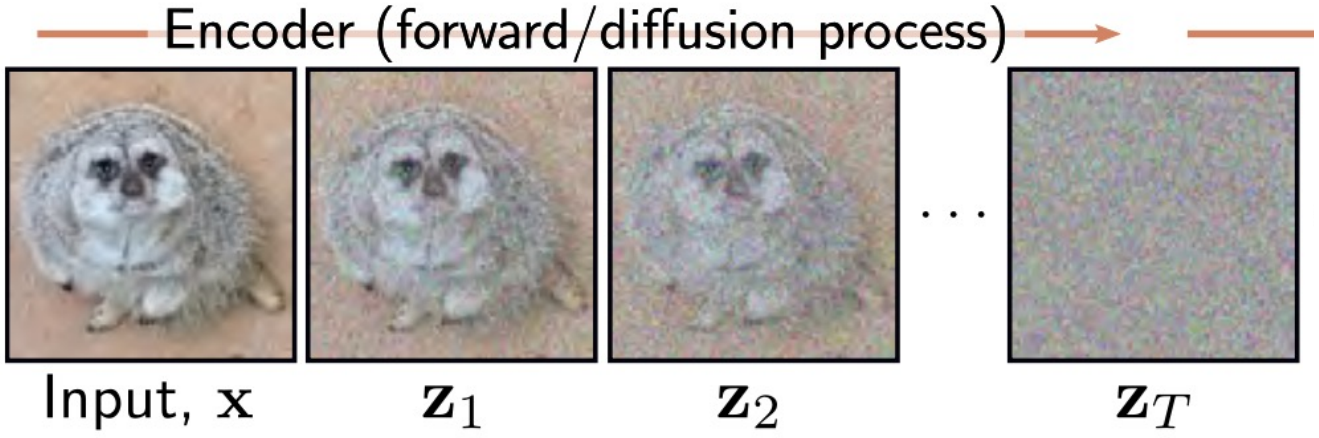



https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

# Diffusion Models: Diffusion process



# Diffusion Models: Forward/Diffusion process




$$\mathbf{z}_1 = \sqrt{1-\beta_1}\cdot\mathbf{x} + \sqrt{\beta_1}\cdot\boldsymbol{\epsilon}_1$$
$$\mathbf{z}_t = \sqrt{1-\beta_t}\cdot\mathbf{z}_{t-1} + \sqrt{\beta_t}\cdot\boldsymbol{\epsilon}_t \qquad \forall\, t \in 2,\ldots,T,$$

$\epsilon_t$ is noise drawn from a standard normal distribution, $\beta t \in [0, 1]$ determine how quickly the noise is blended and are collectively known as the noise schedule

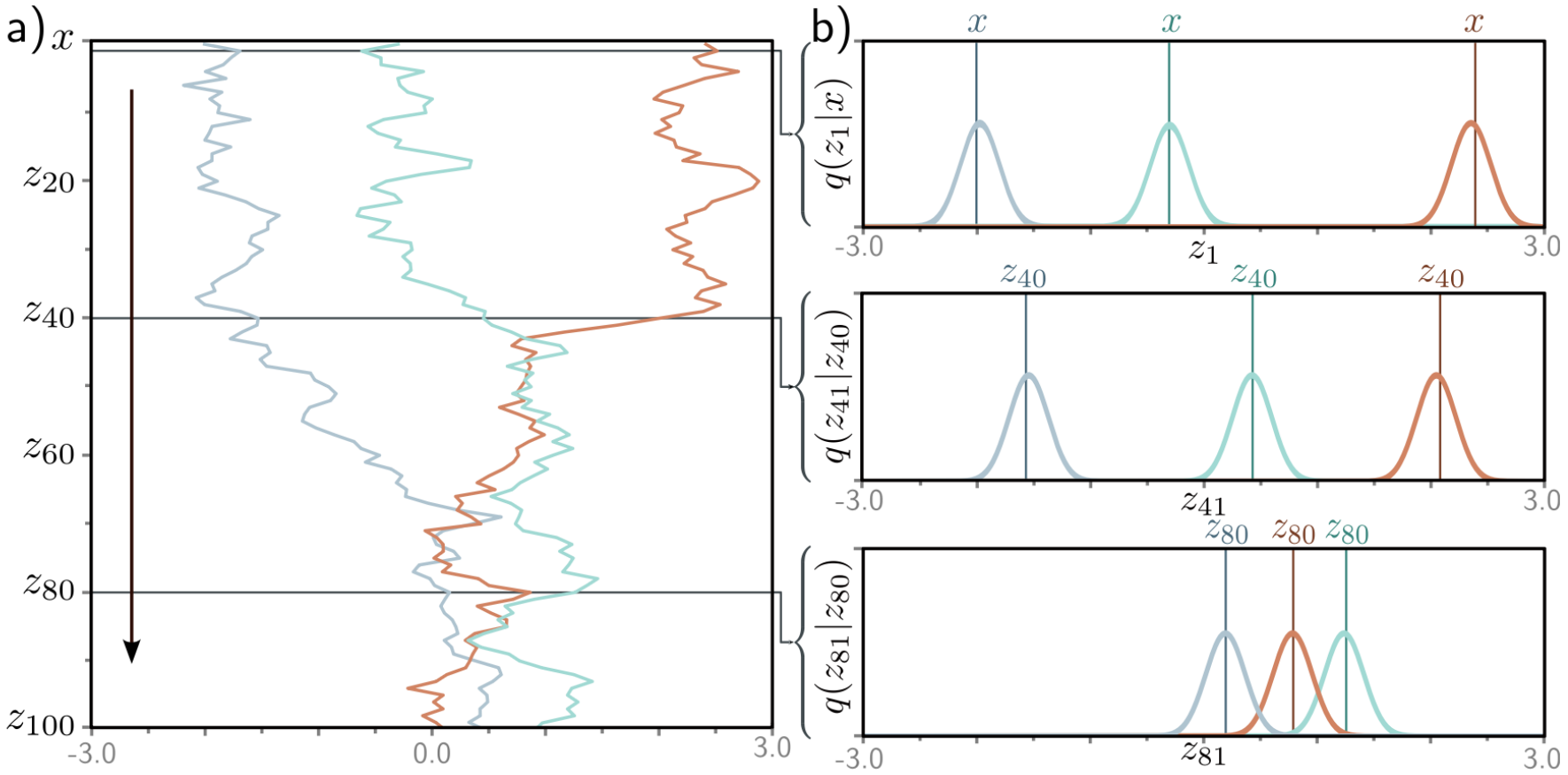
Also written like this:

$$q(\mathbf{z}_1|\mathbf{x}) = \text{Norm}_{\mathbf{z}_1}\left[\sqrt{1-\beta_1}\mathbf{x}, \beta_1\mathbf{I}\right]$$
$$q(\mathbf{z}_t|\mathbf{z}_{t-1}) = \text{Norm}_{\mathbf{z}_t}\left[\sqrt{1-\beta_t}\mathbf{z}_{t-1}, \beta_t\mathbf{I}\right]$$

Joint distribution:

$$q(\mathbf{z}_{1\ldots T}|\mathbf{x}) = q(\mathbf{z}_1|\mathbf{x})\prod_{t=2}^{T} q(\mathbf{z}_t|\mathbf{z}_{t-1}).$$

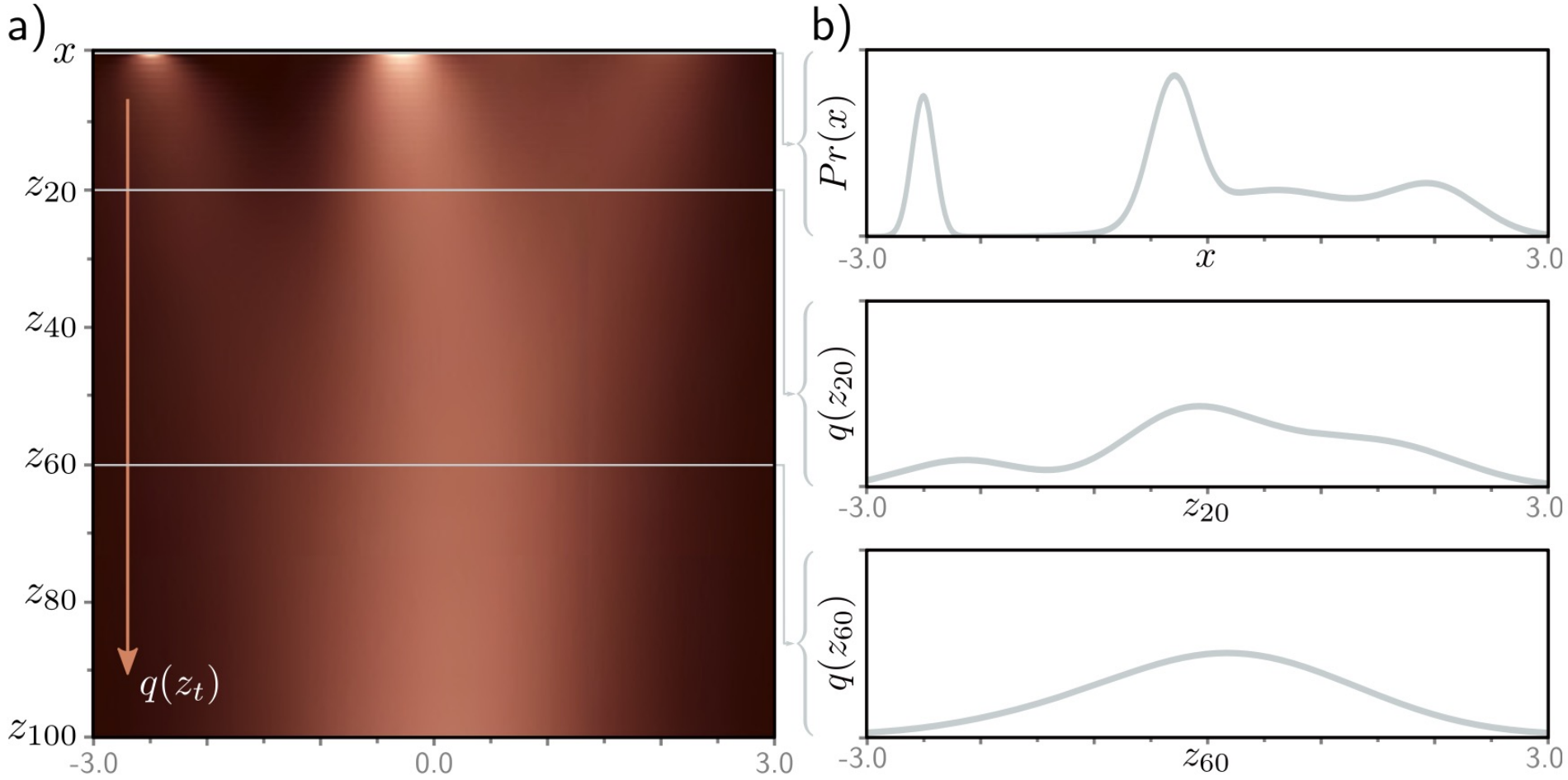
# Diffusion Models: Forward/Diffusion process




a) We consider one-dimensional data x with T = 100 latent variables $z_1, \ldots, z_{100}$ and $\beta = 0.03$ at all steps. Three values of x (gray, cyan, and orange) are initialized (top row).

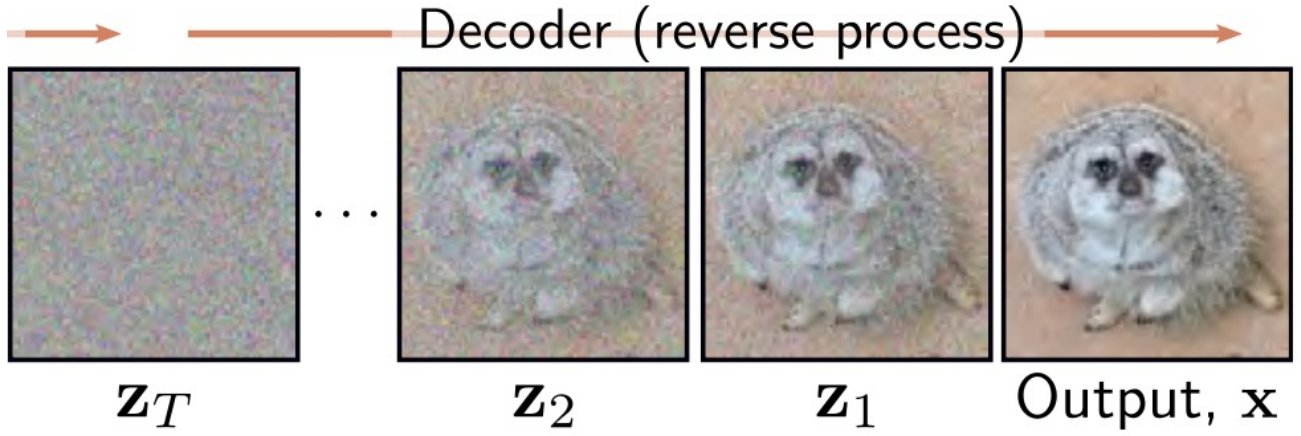
b) The conditional probabilities $Pr(z_1|x)$ and $Pr(z_t|z_{t-1})$ are normal distributions with a mean that is slightly closer to zero than the current point and a fixed variance $\beta_t$

# Diffusion Models: Forward/Diffusion process




Marginal distributions. a) Given an initial density Pr(x) (top row), the diffusion process gradually blurs the distribution as it passes through the latent variables $z_t$ and moves it toward a standard normal distribution.

# Diffusion Models: Reverse/Denoising process




When we learn a diffusion model, we learn the reverse process. In other words, we learn a series of probabilistic mappings back from latent variable $z_T$ to $z_{T-1}$, from $z_{T-1}$ to $z_{T-2}$, and so on, until we reach the data x.

$$
\begin{aligned}
Pr(\mathbf{z}_T) &= \text{Norm}_{\mathbf{z}_T}[\mathbf{0}, \mathbf{I}] \\
Pr(\mathbf{z}_{t-1}|\mathbf{z}_t, \boldsymbol{\phi}_t) &= \text{Norm}_{\mathbf{z}_{t-1}}\Big[\mathbf{f}_t[\mathbf{z}_t, \boldsymbol{\phi}_t], \sigma_t^2\mathbf{I}\Big] \\
Pr(\mathbf{x}|\mathbf{z}_1, \boldsymbol{\phi}_1) &= \text{Norm}_{\mathbf{x}}\Big[\mathbf{f}_1[\mathbf{z}_1, \boldsymbol{\phi}_1], \sigma_1^2\mathbf{I}\Big],
\end{aligned}
$$

$f_t[z_t, \phi_t]$ is a neural network that computes the mean of the normal distribution in the estimated mapping from $z_t$ to the preceding latent variable $z_{t-1}$. The terms $\{\sigma_{t2}\}$ are predetermined.

# Diffusion Models: Training

$$Pr(\mathbf{x}, \mathbf{z}_{1...T}|\boldsymbol{\phi}_{1...T}) = Pr(\mathbf{x}|\mathbf{z}_1, \boldsymbol{\phi}_1) \prod_{t=2}^{T} Pr(\mathbf{z}_{t-1}|\mathbf{z}_t, \boldsymbol{\phi}_t) \cdot Pr(\mathbf{z}_T).$$

$$Pr(\mathbf{x}|\boldsymbol{\phi}_{1...T}) = \int Pr(\mathbf{x}, \mathbf{z}_{1...T}|\boldsymbol{\phi}_{1...T}) d\mathbf{z}_{1...T}.$$

Objective: maximize the log-likelihood of the training data $\{x_i\}$ with respect to the parameters $\phi$:

$$\hat{\boldsymbol{\phi}}_{1...T} = \underset{\boldsymbol{\phi}_{1...T}}{\operatorname{argmax}} \left[ \sum_{i=1}^{I} \log \left[ Pr(\mathbf{x}_i|\boldsymbol{\phi}_{1...T}) \right] \right].$$

We can't maximize this directly because the marginalization is intractable, maximize Evidence lower bound (ELBO) instead, like VAE

# Diffusion Models: Training

After some simplification

$$\text{ELBO}[\boldsymbol{\phi}_{1\ldots T}] = \mathbb{E}_{q(\mathbf{z}_1|\mathbf{x})}\Big[\log\left[Pr(\mathbf{x}|\mathbf{z}_1,\boldsymbol{\phi}_1)\right]\Big] - \sum_{t=2}^{T}\mathbb{E}_{q(\mathbf{z}_t|\mathbf{x})}\Big[\mathrm{D}_{KL}\Big[q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})\Big|\Big|Pr(\mathbf{z}_{t-1}|\mathbf{z}_t,\boldsymbol{\phi}_t)\Big]\Big],$$

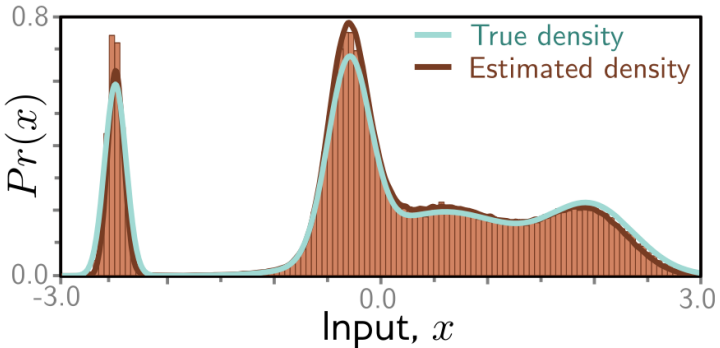
Reconstruction term in VAE

KL divergence between the two normal distributions

$$\begin{aligned} Pr(\mathbf{z}_{t-1}|\mathbf{z}_t,\boldsymbol{\phi}_t) &= \text{Norm}_{\mathbf{z}_{t-1}}\Big[\mathbf{f}_t[\mathbf{z}_t,\boldsymbol{\phi}_t],\sigma_t^2\mathbf{I}\Big] \qquad (18.27)\\ q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x}) &= \text{Norm}_{\mathbf{z}_{t-1}}\left[\frac{(1-\alpha_{t-1})}{1-\alpha_t}\sqrt{1-\beta_t}\mathbf{z}_t + \frac{\sqrt{\alpha_{t-1}}\beta_t}{1-\alpha_t}\mathbf{x}, \frac{\beta_t(1-\alpha_{t-1})}{1-\alpha_t}\mathbf{I}\right]. \end{aligned}$$

Diffusion loss function:

$$\begin{aligned} L[\boldsymbol{\phi}_{1\ldots T}] = \sum_{i=1}^{I}\Bigg( & \overbrace{-\log\Big[\text{Norm}_{\mathbf{x}_i}\left[\mathbf{f}_1[\mathbf{z}_{i1},\boldsymbol{\phi}_1],\sigma_1^2\mathbf{I}\right]\Big]}^{\text{reconstruction term}} \qquad (18.29)\\ & + \sum_{t=2}^{T}\frac{1}{2\sigma_t^2}\Bigg\|\underbrace{\frac{1-\alpha_{t-1}}{1-\alpha_t}\sqrt{1-\beta_t}\mathbf{z}_{it} + \frac{\sqrt{\alpha_{t-1}}\beta_t}{1-\alpha_t}\mathbf{x}_i}_{\text{target, mean of } q(\mathbf{z}_{t-1}|\mathbf{z}_t,\mathbf{x})} - \underbrace{\mathbf{f}_t[\mathbf{z}_{it},\boldsymbol{\phi}_t]}_{\text{predicted } \mathbf{z}_{t-1}}\Bigg\|^2\Bigg), \end{aligned}$$



# Diffusion Models: Training

Diffusion models have been found to work better with a different parameterization: See Understanding Deep Learning, Chapter 18.5

**Reparameterization of target**

$$L[\boldsymbol{\phi}_{1\ldots T}] = \sum_{i=1}^{I} -\log\Big[\text{Norm}_{\mathbf{x}_i}\left[\mathbf{f}_1[\mathbf{z}_{i1}, \boldsymbol{\phi}_1], \sigma_1^2\mathbf{I}\right]\Big] \tag{18.34}$$
$$+\sum_{t=2}^{T} \frac{1}{2\sigma_t^2}\left\|\left(\frac{1}{\sqrt{1-\beta_t}}\mathbf{z}_{it} - \frac{\beta_t}{\sqrt{1-\alpha_t}\sqrt{1-\beta_t}}\boldsymbol{\epsilon}_{it}\right) - \mathbf{f}_t[\mathbf{z}_{it}, \boldsymbol{\phi}_t]\right\|^2.$$

**Reparameterization of network**

Now we replace the model $\hat{\mathbf{z}}_{t-1} = \mathbf{f}_t[\mathbf{z}_t, \boldsymbol{\phi}_t]$ with a new model $\hat{\boldsymbol{\epsilon}} = \mathbf{g}_t[\mathbf{z}_t, \boldsymbol{\phi}_t]$, which predicts the noise $\boldsymbol{\epsilon}$ that was mixed with $\mathbf{x}$ to create $\mathbf{z}_t$:

$$\mathbf{f}_t[\mathbf{z}_t, \boldsymbol{\phi}_t] = \frac{1}{\sqrt{1-\beta_t}}\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}\sqrt{1-\beta_t}}\mathbf{g}_t[\mathbf{z}_t, \boldsymbol{\phi}_t]. \tag{18.35}$$

# Diffusion Models: Training

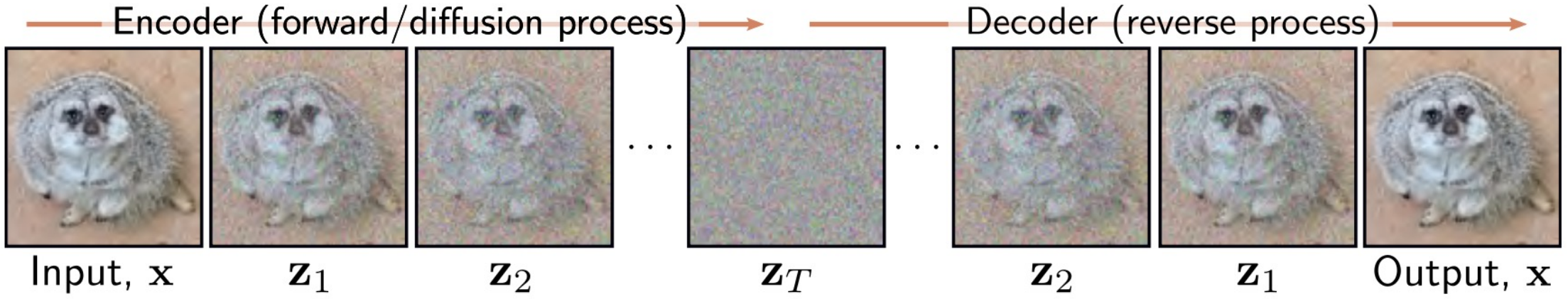
**Algorithm 18.1:** Diffusion model training

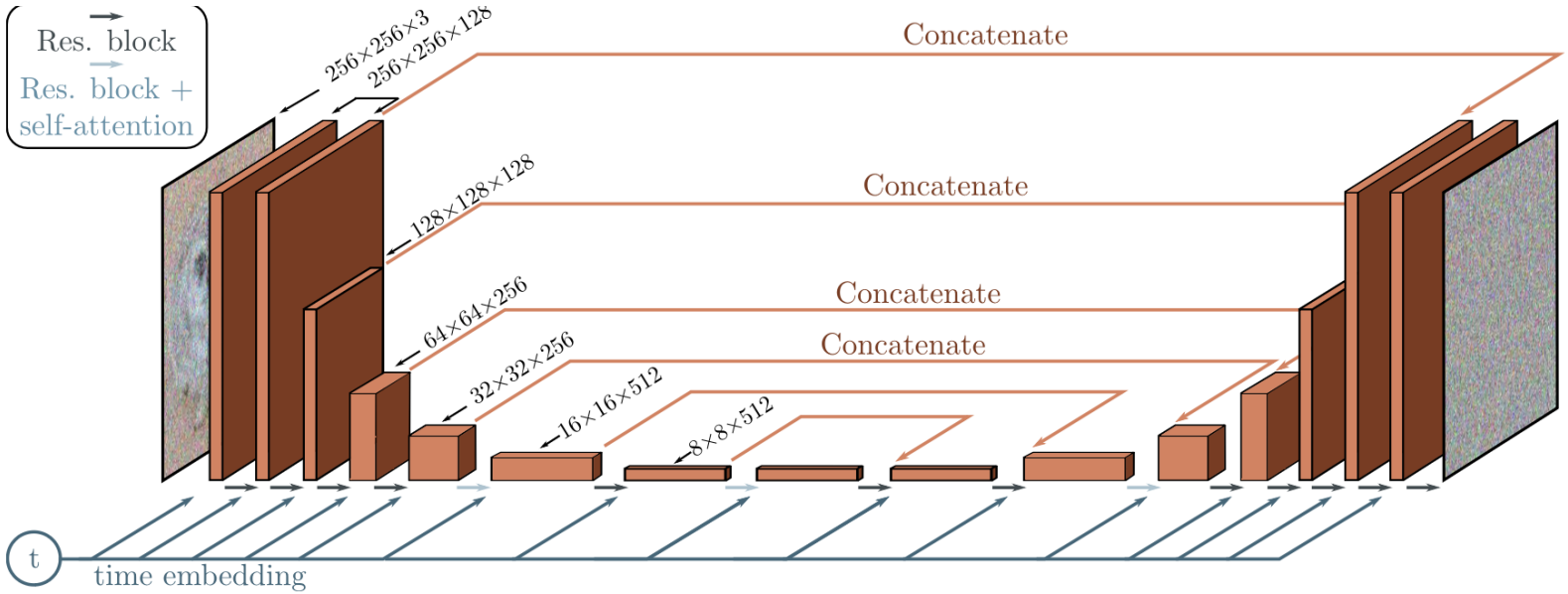
**Input:** Training data $\mathbf{x}$
**Output:** Model parameters $\boldsymbol{\phi}_t$
**repeat**
  **for** $i \in \mathcal{B}$ **do** // For every training example index in batch
    $t \sim \text{Uniform}[1, \ldots T]$ // Sample random timestep
    $\boldsymbol{\epsilon} \sim \text{Norm}[\mathbf{0}, \mathbf{I}]$ // Sample noise
    $\ell_i = \left\| \mathbf{g}_t\left[\sqrt{\alpha_t}\mathbf{x}_i + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}, \boldsymbol{\phi}_t\right] - \boldsymbol{\epsilon} \right\|^2$ // Compute individual loss
  Accumulate losses for batch and take gradient step
**until** converged

**Algorithm 18.2:** Sampling

**Input:** Model, $\mathbf{g}_t[\bullet, \boldsymbol{\phi}_t]$
**Output:** Sample, $\mathbf{x}$
$\mathbf{z}_T \sim \text{Norm}_{\mathbf{z}}[\mathbf{0}, \mathbf{I}]$ // Sample last latent variable
**for** $t = T \ldots 2$ **do**
  $\hat{\mathbf{z}}_{t-1} = \frac{1}{\sqrt{1-\beta_t}}\mathbf{z}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}\sqrt{1-\beta_t}}\mathbf{g}_t[\mathbf{z}_t, \boldsymbol{\phi}_t]$ // Predict previous latent variable
  $\boldsymbol{\epsilon} \sim \text{Norm}_{\boldsymbol{\epsilon}}[\mathbf{0}, \mathbf{I}]$ // Draw new noise vector
  $\mathbf{z}_{t-1} = \hat{\mathbf{z}}_{t-1} + \sigma_t\boldsymbol{\epsilon}$ // Add noise to previous latent variable
$\mathbf{x} = \frac{1}{\sqrt{1-\beta_1}}\mathbf{z}_1 - \frac{\beta_1}{\sqrt{1-\alpha_1}\sqrt{1-\beta_1}}\mathbf{g}_1[\mathbf{z}_1, \boldsymbol{\phi}_1]$ // Generate sample from $\mathbf{z}_1$ without noise

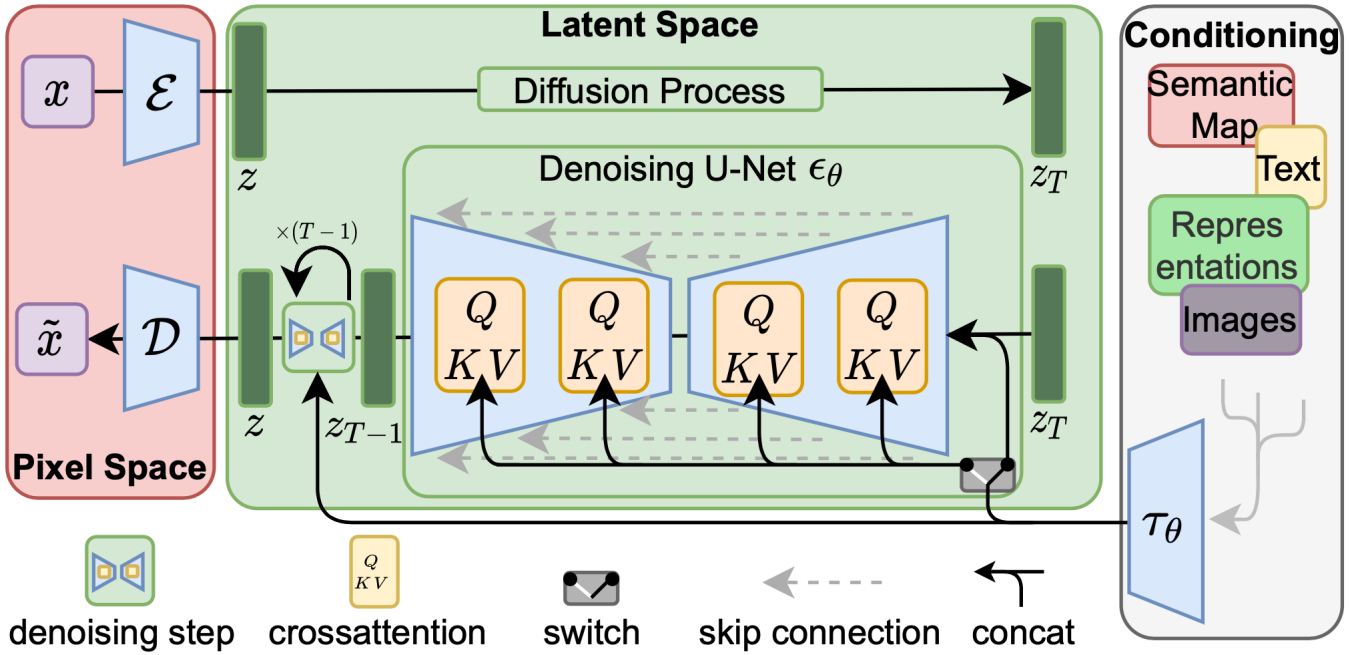
# Diffusion Models: U-Net used in image generation



# Latent Diffusion Model

Run diffusion process in the latent space instead of pixel space, making training cost lower and inference speed faster

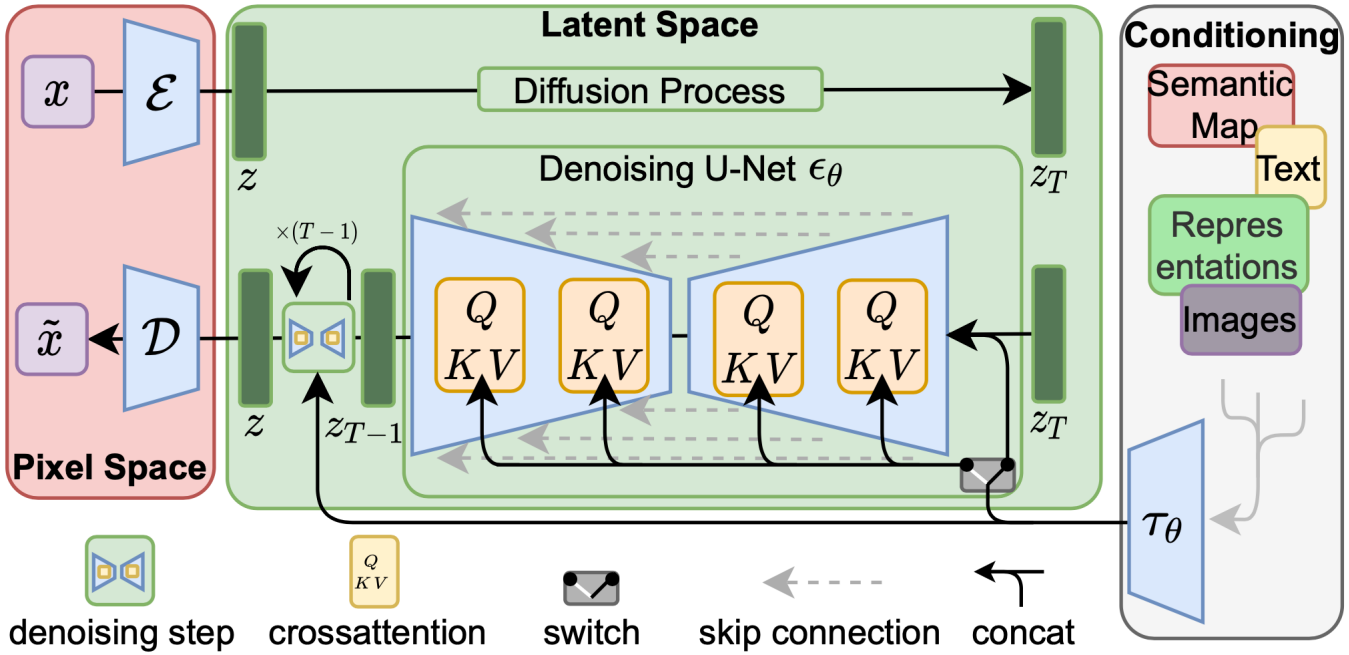



augmenting their underlying U-Net backbone with the cross-attention mechanism

https://arxiv.org/pdf/2112.10752.pdf

# Latent Diffusion Model

Run diffusion process in the latent space instead of pixel space, making training cost lower and inference speed faster

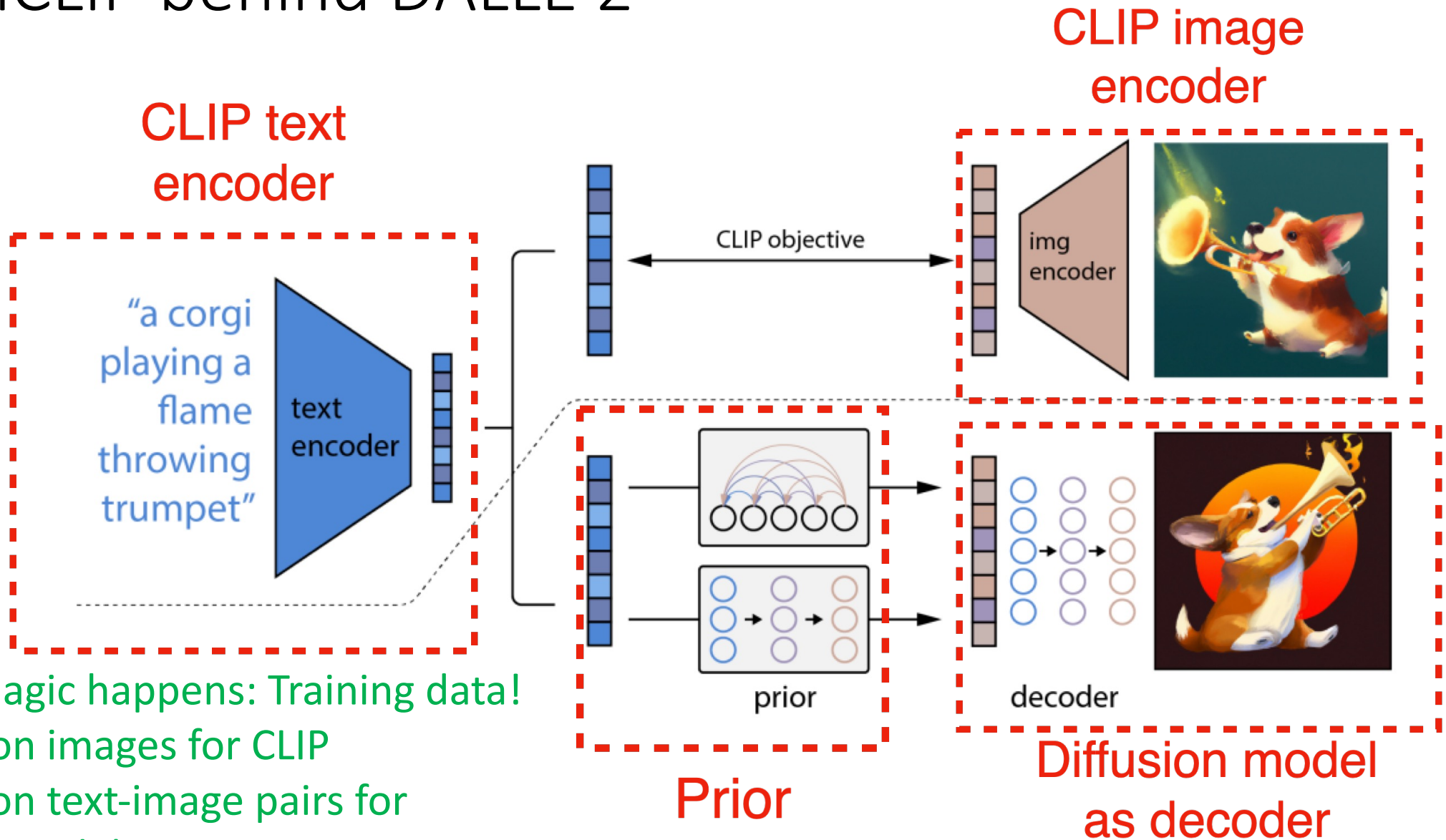



$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right) \cdot V, \text{ with}$$

$$Q = W_Q^{(i)} \cdot \varphi_i(z_t),\ K = W_K^{(i)} \cdot \tau_\theta(y),\ V = W_V^{(i)} \cdot \tau_\theta(y).$$

A domain specific encoder τ_θ projects y to an intermediate representation τ_θ(y), which is then mapped to the intermediate layers of the UNet via a cross-attention layer

# unCLIP behind DALLE 2

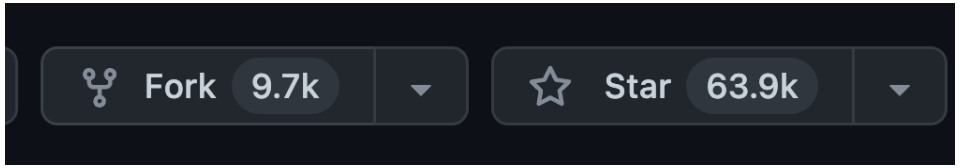



Where magic happens: Training data!
650 million images for CLIP
250 million text-image pairs for diffusion model



Paper: https://arxiv.org/pdf/2204.06125v1.pdf

# Stable Diffusion: Open-source Large Image Generation




*Stable Diffusion was made possible thanks to a collaboration with Stability AI and Runway and builds upon our previous work:*

**High-Resolution Image Synthesis with Latent Diffusion Models**
Robin Rombach*, Andreas Blattmann*, Dominik Lorenz, Patrick Esser, Björn Ommer
*CVPR '22 Oral* | *GitHub* | *arXiv* | *Project page*

Stable Diffusion is a latent text-to-image diffusion model. Thanks to a generous compute donation from Stability AI and support from LAION, we were able to train a Latent Diffusion Model on 512x512 images from a subset of the LAION-5B database. Similar to Google's Imagen, this model uses a frozen CLIP ViT-L/14 text encoder to condition the model on text prompts. With its 860M UNet and 123M text encoder, the model is relatively lightweight and runs on a GPU with at least 10GB VRAM. See this section below and the model card.

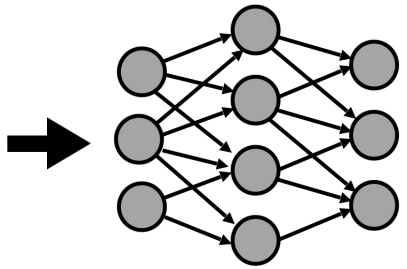
https://github.com/CompVis/stable-diffusion

https://stability.ai/news/stable-diffusion-3

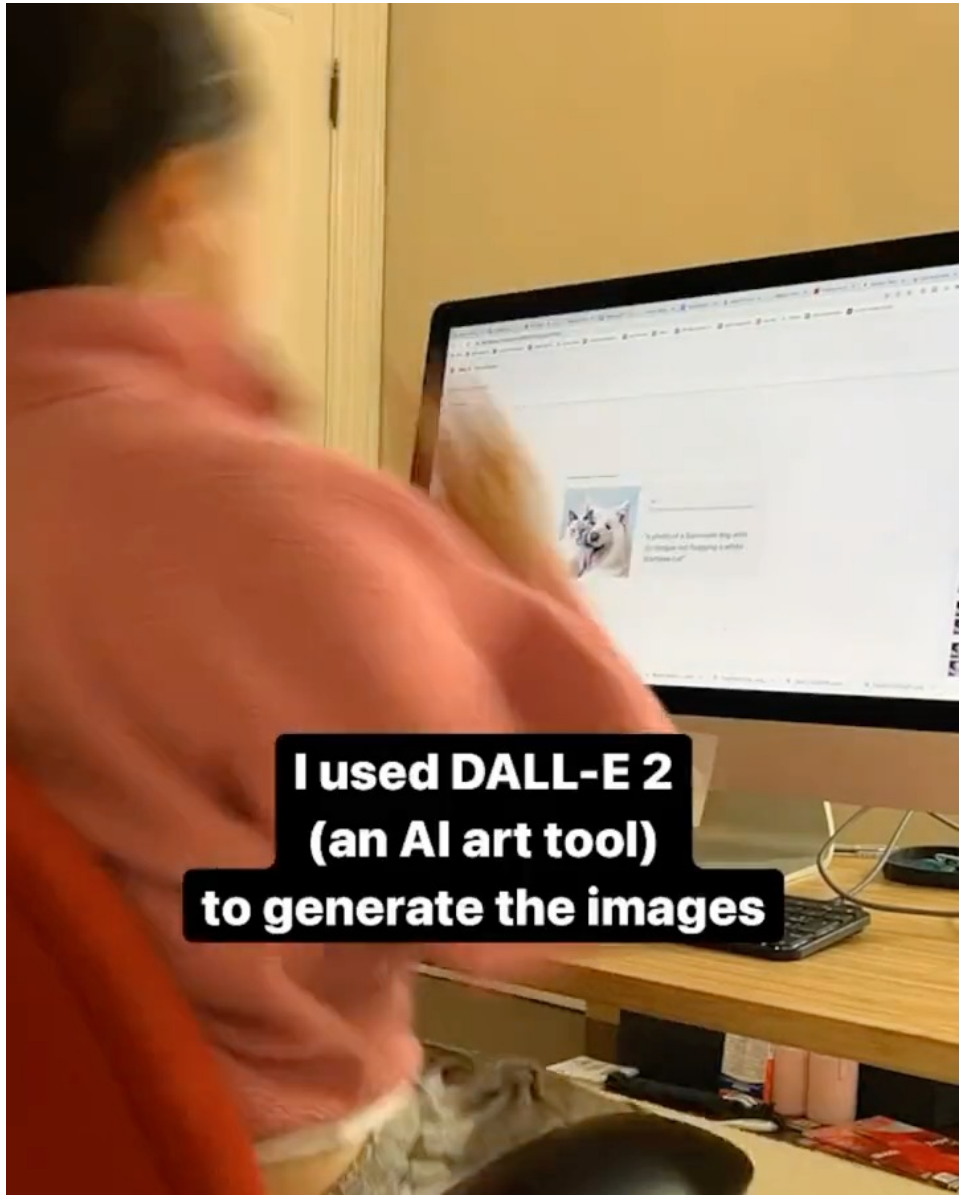
Announced Feb.23, “While the model is not yet broadly available, today, we are opening the waitlist for an early preview. This preview phase, as with previous models, is crucial for gathering insights to improve its performance and safety ahead of an open release.”

# Adding Control to Text-to-Image Diffusion

An astronaut playing basketball with cats in space in a watercolor style



But text description is a very loose way to convey what a human want



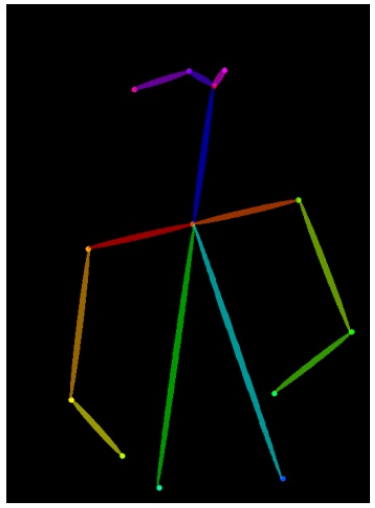
# Adding Control to Text-to-Image Diffusion: ControlNet

Input Canny edge

Default

“masterpiece of fairy tale, giant deer, golden antlers”

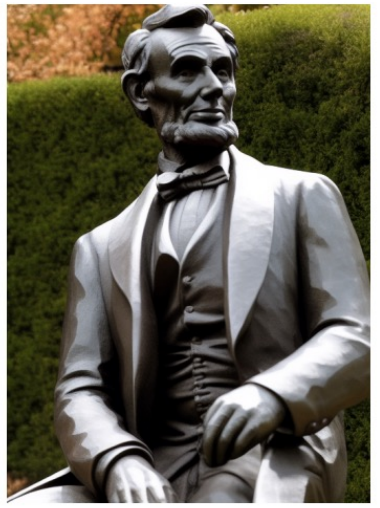
“..., quaint city Galic”



Input human pose

Default

“chef in kitchen”

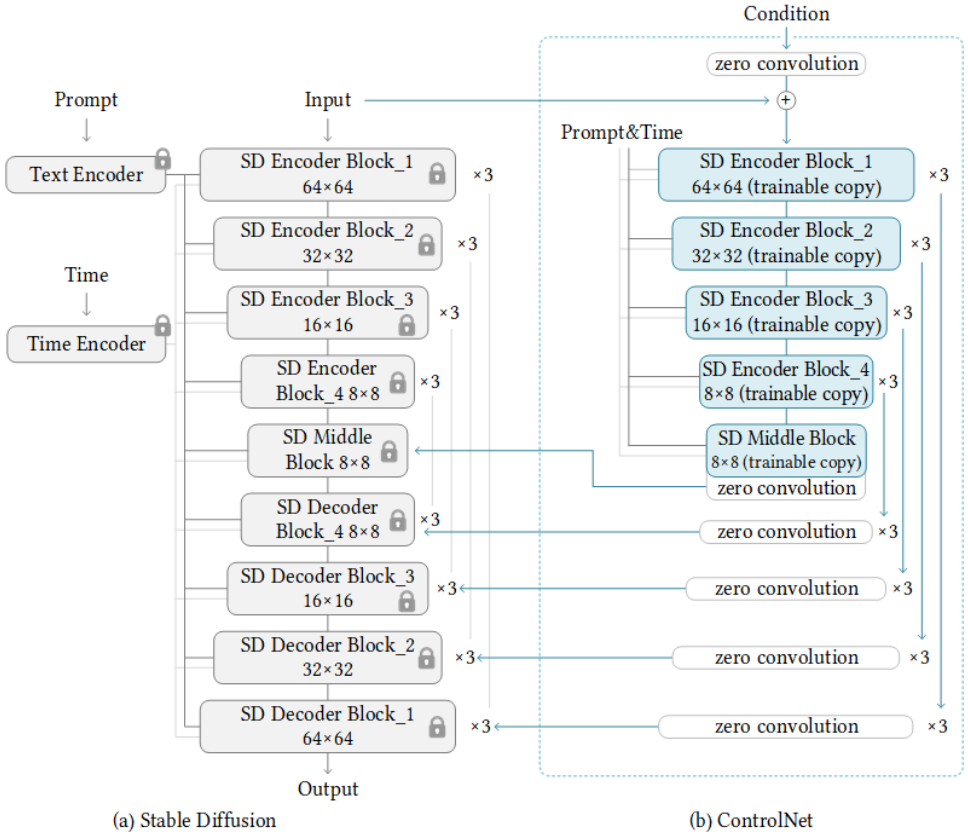


“Lincoln statue”

*Zhang et al, “Adding Conditional Control to Text-to-Image Diffusion Models”, ICCV 2023*
*Best paper award*

# Adding Control to Text-to-Image Diffusion: ControlNet

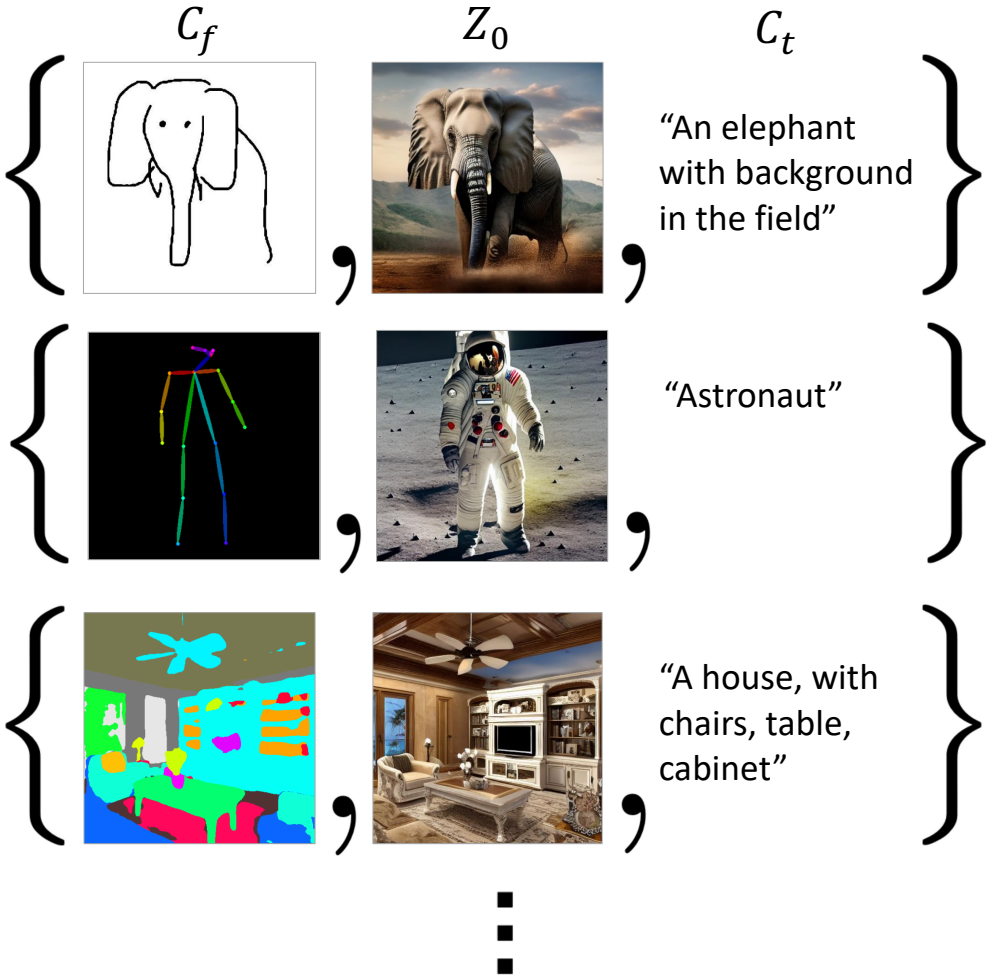
Input condition image and text prompt together




$$\mathcal{L} = \mathbb{E}_{\boldsymbol{z}_0, \boldsymbol{t}, \boldsymbol{c}_t, \boldsymbol{c}_\mathrm{f}, \epsilon \sim \mathcal{N}(0,1)} \left[ \| \epsilon - \epsilon_\theta(\boldsymbol{z}_t, \boldsymbol{t}, \boldsymbol{c}_t, \boldsymbol{c}_\mathrm{f}) \|_2^2 \right]$$

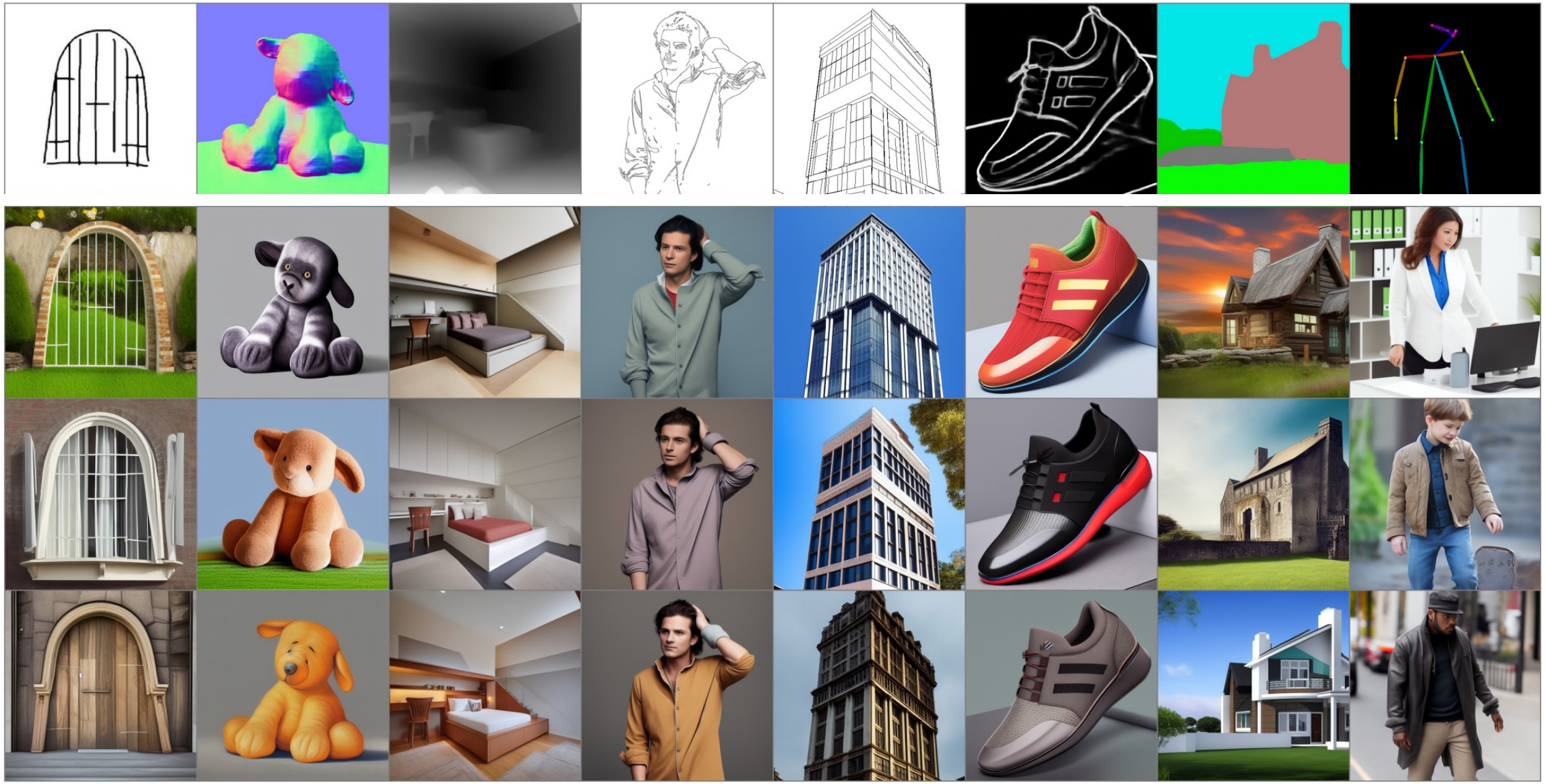
Training data: 0.5 - 3M Pairs

Training time: 100 – 600 A100 hours




*Zhang et al, "Adding Conditional Control to Text-to-Image Diffusion Models", ICCV 2023*

https://github.com/lllyasviel/ControlNet

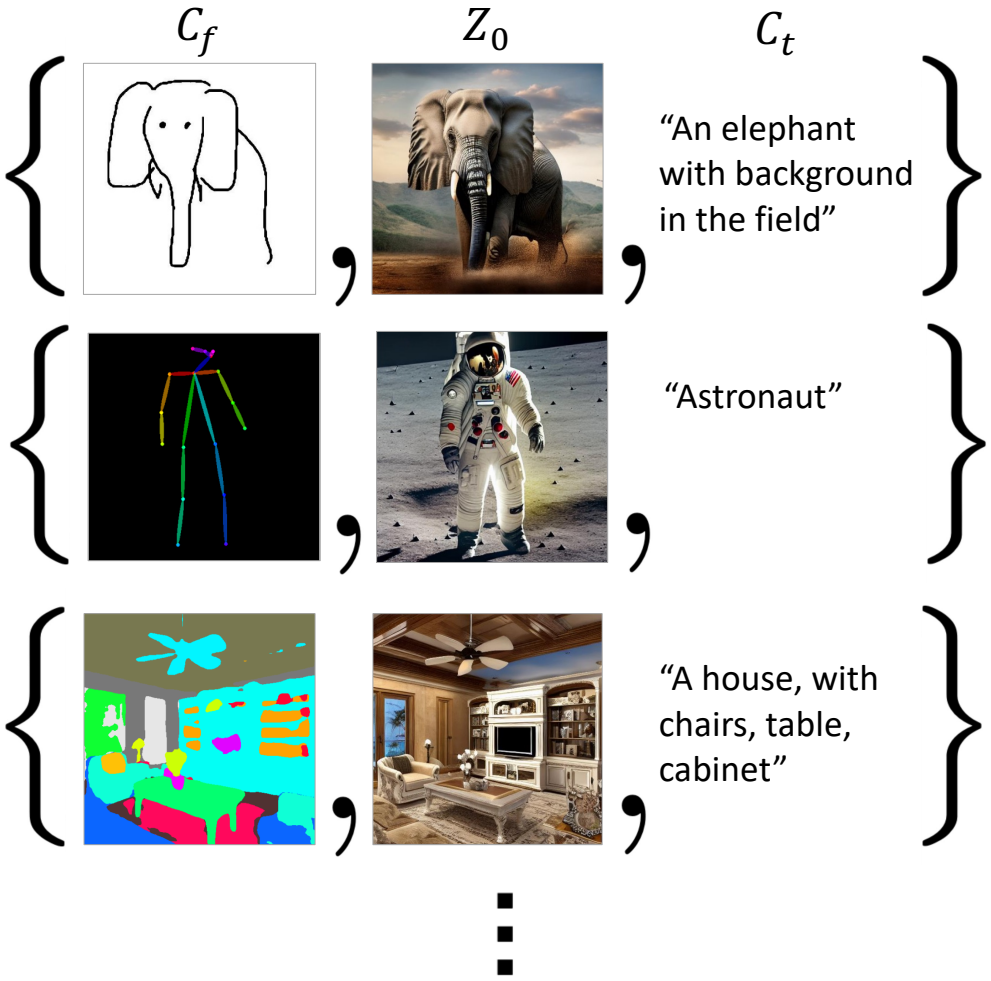



*Zhang et al, "Adding Conditional Control to Text-to-Image Diffusion Models", ICCV 2023*

# Can we have a training-free approach?

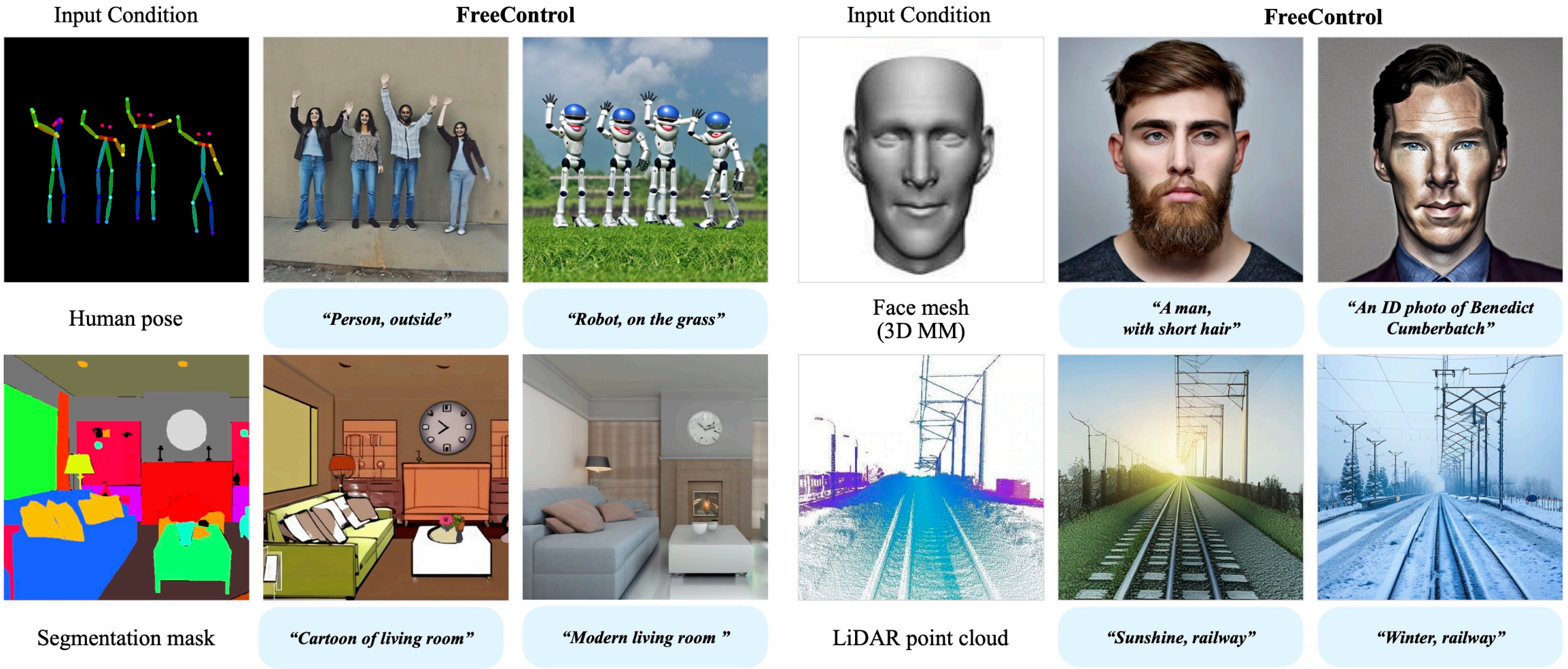
Limitations of ControlNet:

- Training data: 0.5 - 3M Pairs
- Training time: 100 – 600 A100 hours




*Zhang et al, "Adding Conditional Control to Text-to-Image Diffusion Models", ICCV 2023*

# FreeControl: Training-Free Control of T2I Diffusion Models with Any Condition

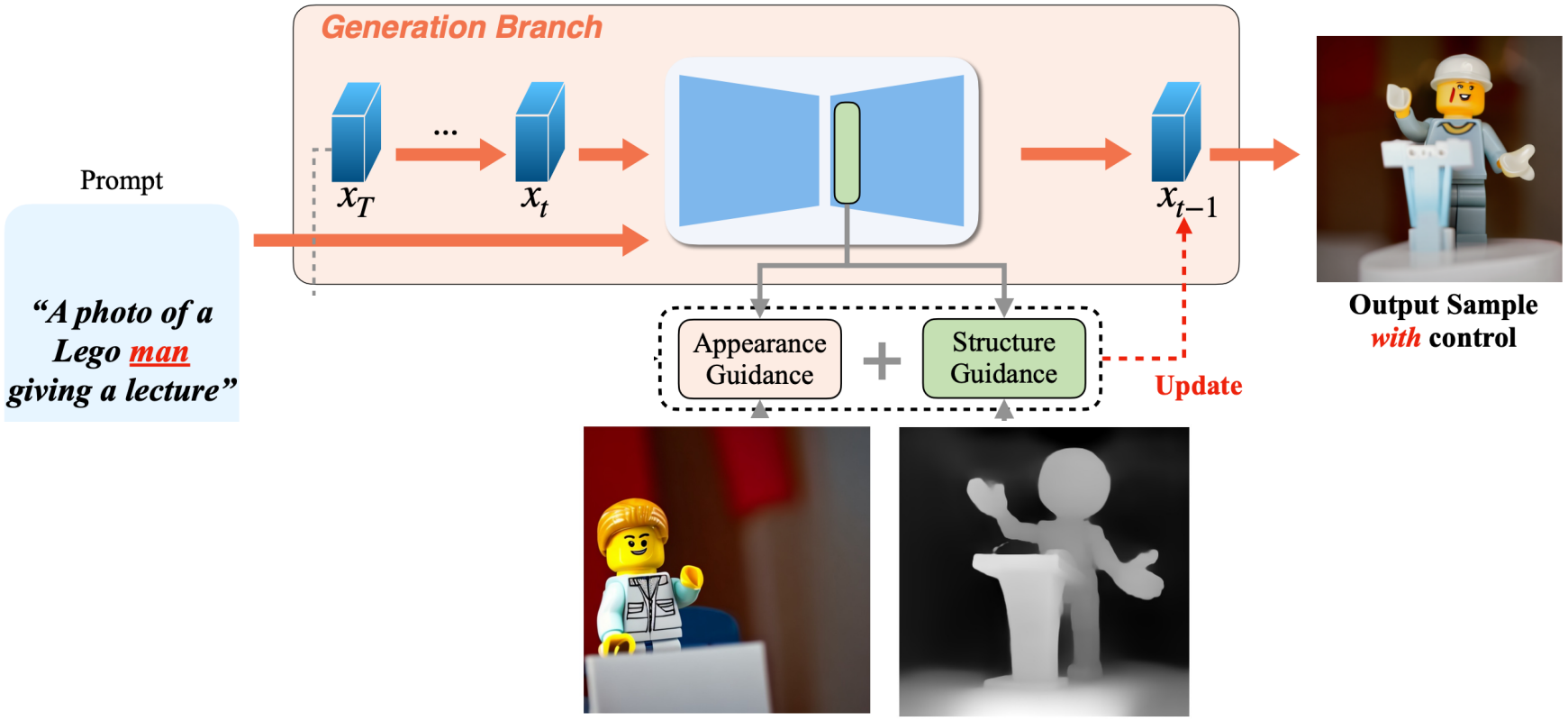



Former CS188 student Jordan Lin contributed to this project, Mo et al. CVPR'24

https://genforce.github.io/freecontrol/

# FreeControl: Training-Free Control of T2I Diffusion Models with Any Condition

**Approach:**




**Guidance:** $$\hat{\epsilon}_\theta(\mathbf{x}_t; t, \mathbf{c}) = \epsilon_\theta(\mathbf{x}_t; t, \mathbf{c}) - s\, g(\mathbf{x_t}; t, y)$$

*Mo et al, "FreeControl: Training-Free Spatial Control of Any Text-to-Image Diffusion Model with Any Condition", CVPR 2025*

https://genforce.github.io/freecontrol/

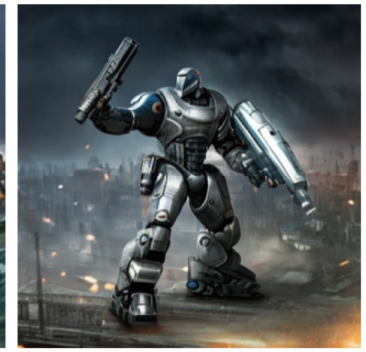
Body mesh
(AMASS)

*"A man, in a suit, on the beach"*

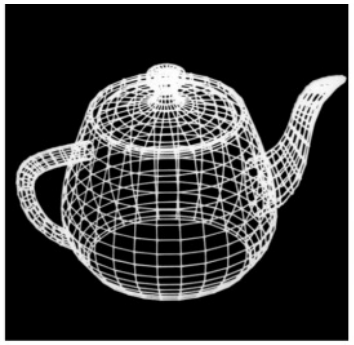
*"A monkey, in the snow"*

Humanoid

*"Man, with sword and shield, in the river"*



*"Large robot, with weapon, over a city"*

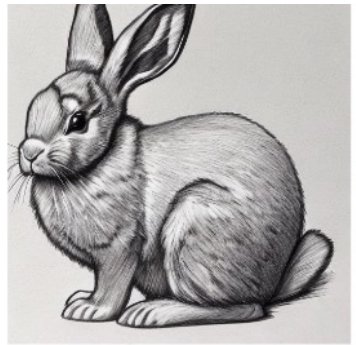


Wirefame

*"A teapot, from China"*

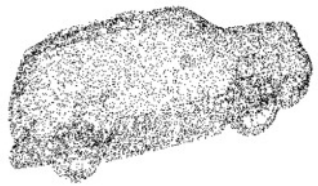
*"An uncolored syderolife teapot"*

Trianglar
mesh



*"A sketch of a bunny"*

*"A cartoon bunny, on the grass"*



Point cloud

*"A SUV car, on the road"*

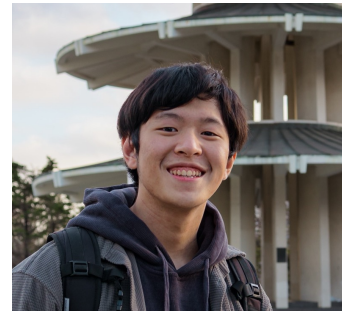
*"A small wooden SUV car"*

Point cloud
(w/ part seg.)

*"A plane, in the sky"*

*"A plane on the runway, landing"*

# NeurIPS'24 work
# Ctrl-X

Student who has taken CS188 in prior year

Jordan Lin*, Sicheng Mo*, et al. NeurIPS'24. https://genforce.github.io/ctrl-x/

# Generative Models Summary

**Autoregressive Models** directly maximize likelihood of training data:

$$p_\theta(x) = \prod_{i=1}^{N} p_\theta(x_i | x_1, \ldots, x_{i-1})$$

Good image quality, can evaluate with perplexity. Slow to generate data, needs tricks to scale up.

**Variational Autoencoders** introduce a latent z, and maximize a lower bound:

$$p_\theta(x) = \int_Z p_\theta(x|z)p(z)dz \geq E_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}\left(q_\phi(z|x), p(z)\right)$$

Latent z allows for powerful interpolation and editing applications.

**Generative Adversarial Networks** give up on modeling p(x), but allow us to draw samples from p(x). Difficult to evaluate, but best qualitative results today

**Diffusion models** rely on a long Markov chain of diffusion steps to model p(x), with flexible models to fit arbitrary structures in data, but evaluating, training, or sampling from these models is usually expensive.

# Reading and practice

D2L textbook: Chapter 17 Generative Adversarial Networks

Understanding Deep Learning: Chapter 18: Diffusion Models
https://udlbook.github.io/udlbook/

Math derivation of diffusion process: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

# Derivation of GAN Optimality in Appendix

# Generative Adversarial Networks: Optimality

$$\min_{G} \max_{D} \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

(Our objective so far)

# Generative Adversarial Networks: Optimality

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{x \sim p_G} [\log (1 - D(x))] \right)$$

(Change of variables on second term)

# Generative Adversarial Networks: Optimality

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{x \sim p_G} [\log (1 - D(x))] \right)$$

$$= \min_G \max_D \int_X \left( p_{data}(x) \log D(x) + p_G(x) \log (1 - D(x)) \right) dx$$

(Definition of expectation)

# Generative Adversarial Networks: Optimality

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{x \sim p_G} [\log (1 - D(x))] \right)$$

$$= \min_G \int_X \max_D \left( p_{data}(x) \log D(x) + p_G(x) \log (1 - D(x)) \right) dx$$

(Push $\max_D$ inside integral)

# Generative Adversarial Networks: Optimality

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{x \sim p_G} [\log (1 - D(x))] \right)$$

$$= \min_G \int_X \max_D \left( p_{data}(x) \log D(x) + p_G(x) \log (1 - D(x)) \right) dx$$

$$f(y) = a \log y + b \log (1 - y)$$

(Side computation to compute max)

# Generative Adversarial Networks: Optimality

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{x \sim p_G} [\log (1 - D(x))] \right)$$

$$= \min_G \int_X \max_D \left( p_{data}(x) \log D(x) + p_G(x) \log (1 - D(x)) \right) dx$$

$$f(y) = a \log y + b \log (1 - y)$$

$$f'(y) = \frac{a}{y} - \frac{b}{1 - y}$$

# Generative Adversarial Networks: Optimality

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{x \sim p_G} [\log (1 - D(x))] \right)$$

$$= \min_G \int_X \max_D \left( p_{data}(x) \log D(x) + p_G(x) \log (1 - D(x)) \right) dx$$

$f(y) = a \log y + b \log(1 - y)$ $\quad$ $f'(y) = 0 \iff y = \frac{a}{a+b}$ (local max)

$$f'(y) = \frac{a}{y} - \frac{b}{1 - y}$$

# Generative Adversarial Networks: Optimality

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{x \sim p_G} [\log (1 - D(x))] \right)$$

$$= \min_G \int_X \max_D \left( p_{data}(x) \log D(x) + p_G(x) \log (1 - D(x)) \right) dx$$

$f(y) = a \log y + b \log(1 - y)$ $\quad f'(y) = 0 \iff y = \frac{a}{a+b}$ (local max)

$f'(y) = \frac{a}{y} - \frac{b}{1-y}$ **Optimal Discriminator:** $D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$

# Generative Adversarial Networks: Optimality

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{x \sim p_G} [\log (1 - D(x))] \right)$$

$$= \min_G \int_X \max_D \left( p_{data}(x) \log D(x) + p_G(x) \log (1 - D(x)) \right) dx$$

**Optimal Discriminator:** $D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$

# Generative Adversarial Networks: Optimality

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{x \sim p_G} [\log (1 - D(x))] \right)$$

$$= \min_G \int_X \left( p_{data}(x) \log D_G^*(x) + p_G(x) \log (1 - D_G^*(x)) \right) dx$$

**Optimal Discriminator:** $D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$

# Generative Adversarial Networks: Optimality

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{x \sim p_G} [\log (1 - D(x))] \right)$$

$$= \min_G \int_X \left( p_{data}(x) \log D_G^*(x) + p_G(x) \log (1 - D_G^*(x)) \right) dx$$

$$= \min_G \int_X \left( p_{data}(x) \log \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} + p_G(x) \log \frac{p_G(x)}{p_{data}(x) + p_G(x)} \right) dx$$

**Optimal Discriminator:** $D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$

# Generative Adversarial Networks: Optimality

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_G \int_X \left( p_{data}(x) \log \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} + p_G(x) \log \frac{p_G(x)}{p_{data}(x) + p_G(x)} \right) dx$$

# Generative Adversarial Networks: Optimality

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_G \int_X \left( p_{data}(x) \log \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} + p_G(x) \log \frac{p_G(x)}{p_{data}(x) + p_G(x)} \right) dx$$

$$= \min_G \left( E_{x \sim p_{data}} \left[ \log \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} \right] + E_{x \sim p_G} \left[ \log \frac{p_G(x)}{p_{data}(x) + p_G(x)} \right] \right)$$

(Definition of expectation)

# Generative Adversarial Networks: Optimality

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_G \int_X \left( p_{data}(x) \log \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} + p_G(x) \log \frac{p_G(x)}{p_{data}(x) + p_G(x)} \right) dx$$

$$= \min_G \left( E_{x \sim p_{data}} \left[ \log \frac{2}{2} \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} \right] + E_{x \sim p_G} \left[ \log \frac{2}{2} \frac{p_G(x)}{p_{data}(x) + p_G(x)} \right] \right)$$

(Multiply by a constant)

# Generative Adversarial Networks: Optimality

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_G \int_X \left( p_{data}(x) \log \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} + p_G(x) \log \frac{p_G(x)}{p_{data}(x) + p_G(x)} \right) dx$$

$$= \min_G \left( E_{x \sim p_{data}} \left[ \log \frac{2}{2} \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} \right] + E_{x \sim p_G} \left[ \log \frac{2}{2} \frac{p_G(x)}{p_{data}(x) + p_G(x)} \right] \right)$$

$$= \min_G \left( E_{x \sim p_{data}} \left[ \log \frac{2 * p_{data}(x)}{p_{data}(x) + p_G(x)} \right] + E_{x \sim p_G} \left[ \log \frac{2 * p_G(x)}{p_{data}(x) + p_G(x)} \right] - \log 4 \right)$$

# Generative Adversarial Networks: Optimality

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_G \left( E_{x \sim p_{data}} \left[ \log \frac{2 * p_{data}(x)}{p_{data}(x) + p_G(x)} \right] + E_{x \sim p_G} \left[ \log \frac{2 * p_G(x)}{p_{data}(x) + p_G(x)} \right] - \log 4 \right)$$

# Generative Adversarial Networks: Optimality

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_G \left( E_{x \sim p_{data}} \left[ \log \frac{2 * p_{data}(x)}{p_{data}(x) + p_G(x)} \right] + E_{x \sim p_G} \left[ \log \frac{2 * p_G(x)}{p_{data}(x) + p_G(x)} \right] - \log 4 \right)$$

**Kullback-Leibler Divergence:**

$$KL(p, q) = E_{x \sim p} \left[ \log \frac{p(x)}{q(x)} \right]$$

# Generative Adversarial Networks: Optimality

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_G \left( E_{x \sim p_{data}} \left[ \log \frac{2 * p_{data}(x)}{p_{data}(x) + p_G(x)} \right] + E_{x \sim p_G} \left[ \log \frac{2 * p_G(x)}{p_{data}(x) + p_G(x)} \right] - \log 4 \right)$$

**Kullback-Leibler Divergence:**

$$KL(p, q) = E_{x \sim p} \left[ \log \frac{p(x)}{q(x)} \right]$$

# Generative Adversarial Networks: Optimality

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_G \left( E_{x \sim p_{data}} \left[ \log \frac{2 * p_{data}(x)}{p_{data}(x) + p_G(x)} \right] + E_{x \sim p_G} \left[ \log \frac{2 * p_G(x)}{p_{data}(x) + p_G(x)} \right] - \log 4 \right)$$

$$= \min_G \left( KL \left( p_{data}, \frac{p_{data} + p_G}{2} \right) + KL \left( p_G, \frac{p_{data} + p_G}{2} \right) - \log 4 \right)$$

**Kullback-Leibler Divergence:**

$$KL(p, q) = E_{x \sim p} \left[ \log \frac{p(x)}{q(x)} \right]$$

# Generative Adversarial Networks: Optimality

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_G \left( E_{x \sim p_{data}} \left[ \log \frac{2 * p_{data}(x)}{p_{data}(x) + p_G(x)} \right] + E_{x \sim p_G} \left[ \log \frac{2 * p_G(x)}{p_{data}(x) + p_G(x)} \right] - \log 4 \right)$$

$$= \min_G \left( KL \left( p_{data}, \frac{p_{data} + p_G}{2} \right) + KL \left( p_G, \frac{p_{data} + p_G}{2} \right) - \log 4 \right)$$

# Generative Adversarial Networks: Optimality

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_G \left( E_{x \sim p_{data}} \left[ \log \frac{2 * p_{data}(x)}{p_{data}(x) + p_G(x)} \right] + E_{x \sim p_G} \left[ \log \frac{2 * p_G(x)}{p_{data}(x) + p_G(x)} \right] - \log 4 \right)$$

$$= \min_G \left( KL \left( p_{data}, \frac{p_{data} + p_G}{2} \right) + KL \left( p_G, \frac{p_{data} + p_G}{2} \right) - \log 4 \right)$$

**Jensen-Shannon Divergence:**

$$JSD(p, q) = \frac{1}{2} KL \left( p, \frac{p + q}{2} \right) + \frac{1}{2} KL \left( q, \frac{p + q}{2} \right)$$

# Generative Adversarial Networks: Optimality

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_G \left( E_{x \sim p_{data}} \left[ \log \frac{2 * p_{data}(x)}{p_{data}(x) + p_G(x)} \right] + E_{x \sim p_G} \left[ \log \frac{2 * p_G(x)}{p_{data}(x) + p_G(x)} \right] - \log 4 \right)$$

$$= \min_G \left( KL \left( p_{data}, \frac{p_{data} + p_G}{2} \right) + KL \left( p_G, \frac{p_{data} + p_G}{2} \right) - \log 4 \right)$$

**Jensen-Shannon Divergence:**

$$JSD(p, q) = \frac{1}{2} KL \left( p, \frac{p + q}{2} \right) + \frac{1}{2} KL \left( q, \frac{p + q}{2} \right)$$

# Generative Adversarial Networks: Optimality

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_G \left( E_{x \sim p_{data}} \left[ \log \frac{2 * p_{data}(x)}{p_{data}(x) + p_G(x)} \right] + E_{x \sim p_G} \left[ \log \frac{2 * p_G(x)}{p_{data}(x) + p_G(x)} \right] - \log 4 \right)$$

$$= \min_G \left( KL \left( p_{data}, \frac{p_{data} + p_G}{2} \right) + KL \left( p_G, \frac{p_{data} + p_G}{2} \right) - \log 4 \right)$$

$$= \min_G (2 * JSD(p_{data}, p_G) - \log 4)$$

**Jensen-Shannon Divergence:**

$$JSD(p, q) = \frac{1}{2} KL \left( p, \frac{p + q}{2} \right) + \frac{1}{2} KL \left( q, \frac{p + q}{2} \right)$$

# Generative Adversarial Networks: Optimality

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_G \left( E_{x \sim p_{data}} \left[ \log \frac{2 * p_{data}(x)}{p_{data}(x) + p_G(x)} \right] + E_{x \sim p_G} \left[ \log \frac{2 * p_G(x)}{p_{data}(x) + p_G(x)} \right] - \log 4 \right)$$

$$= \min_G \left( KL \left( p_{data}, \frac{p_{data} + p_G}{2} \right) + KL \left( p_G, \frac{p_{data} + p_G}{2} \right) - \log 4 \right)$$

$$= \min_G (2 * JSD(p_{data}, p_G) - \log 4)$$

JSD is always nonnegative, and zero only when the two distributions are equal!
Thus $p_{data} = p_G$ is the global min, QED

**Jensen-Shannon Divergence:**

$$JSD(p, q) = \frac{1}{2} KL \left( p, \frac{p + q}{2} \right) + \frac{1}{2} KL \left( q, \frac{p + q}{2} \right)$$

# Generative Adversarial Networks: Optimality

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_G (2 * JSD(p_{data}, p_G) - \log 4)$$

# Generative Adversarial Networks: Optimality

$$\min_G \max_D \left( E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[ \log \left( 1 - D(G(z)) \right) \right] \right)$$

$$= \min_G (2 * JSD(p_{data}, p_G) - \log 4)$$

**Summary**: The global minimum of the minimax game happens when:

1. $D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$ (Optimal discriminator for any G)
2. $p_G(x) = p_{data}(x)$ (Optimal generator for optimal D)