Lecture 20:
Generative Models, Part 2

Admin: A4

A4 due yesterday, many people still working

Admin: A5

A5 Released last night

Recurrent networks, image captioning, Transformers

Due Tuesday April 12th at 11:59pm ET

Admin: Project Proposal

If you want to propose your own project:

Need to submit a project proposal by tomorrow, 4/1 on Piazza

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!

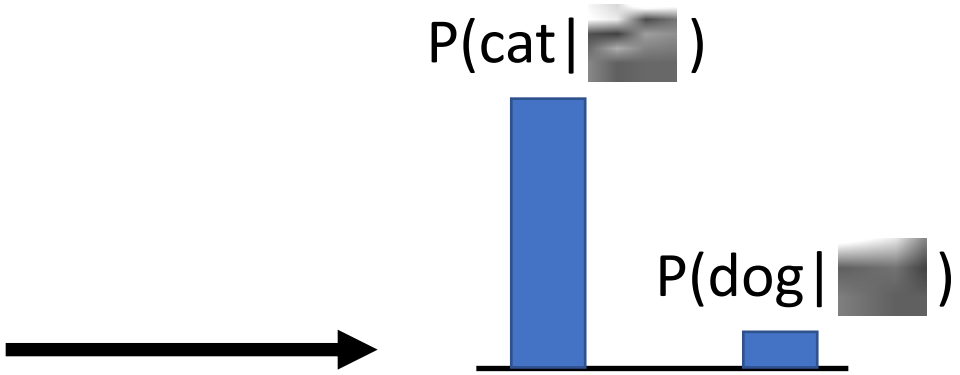
Goal: Learn some underlying hidden *structure* of the data

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)

Conditional Generative Model: Learn p(x|y)

Density Function

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

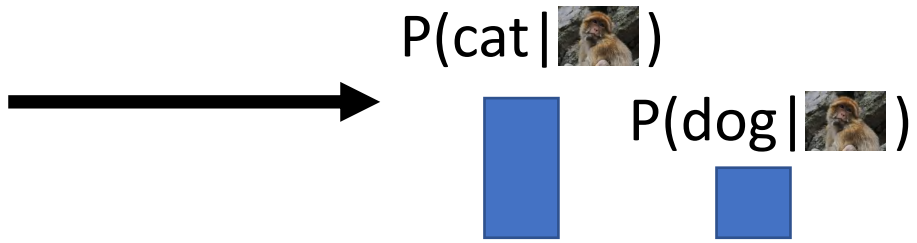
Density functions are **normalized**:

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

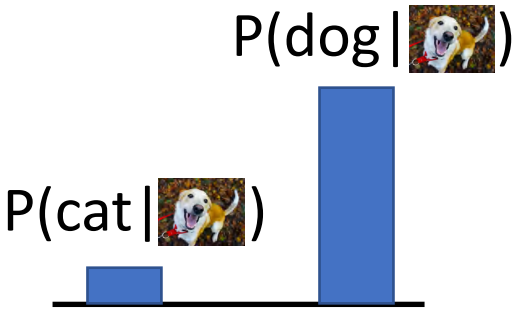
Different values of x **compete** for density

Last Time: Discriminative vs Generative Models

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



Generative Model: Learn a probability distribution p(x)



Conditional Generative Model: Learn p(x|y)

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

Monkey image is CC0 Public Domain

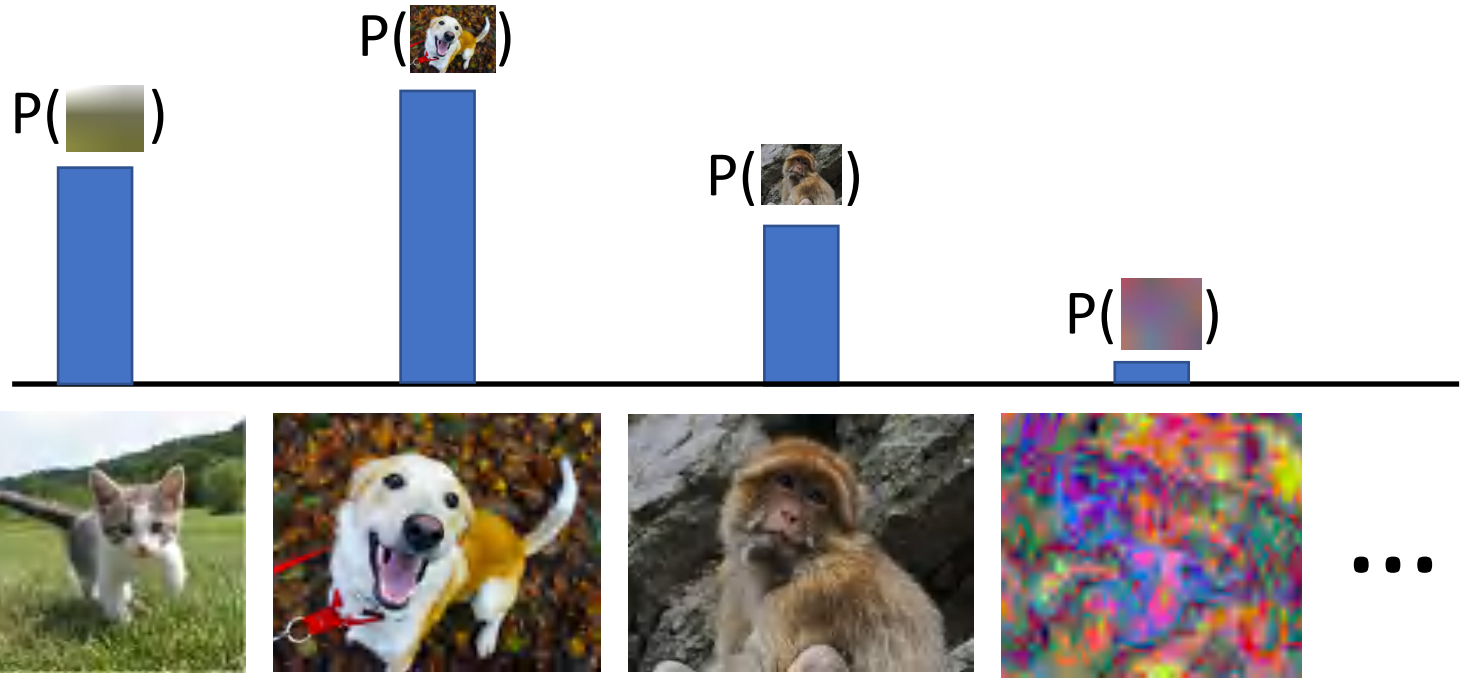
Cat image is CC0 public domain
Dog image is CC0 Public Domain
Monkey image is CC0 Public Domain
Abstract image is free to use under the Pixabay license

Last Time: Discriminative vs Generative Models

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

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)



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

Requires deep image understanding! Is a dog more likely to sit or stand? How about 3-legged dog vs 3-armed monkey?

Cat image is CC0 public domain
Dog image is CC0 Public Domain
Monkey image is CC0 Public Domain
Abstract image is free to use under the Pixabay license

Last Time: Discriminative vs Generative Models

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

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)

Recall **Bayes' Rule:**

$$P(x \mid y) = \frac{P(y \mid x)}{P(y)} P(x)$$

Conditional Generative Model

Discriminative Model

Prior over labels

(Unconditional) Generative Model

We can build a conditional generative model from other components!

Last Time: Taxonomy of Generative Models

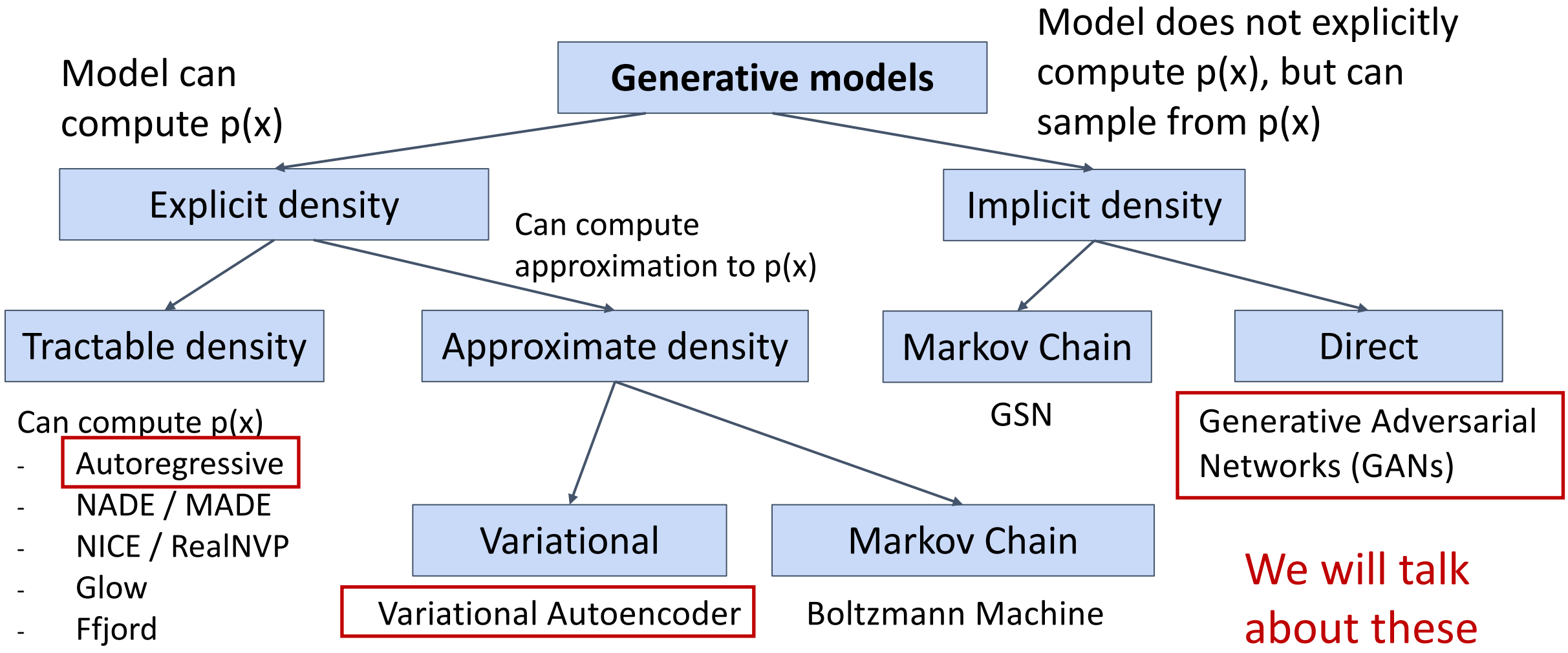


Figure adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

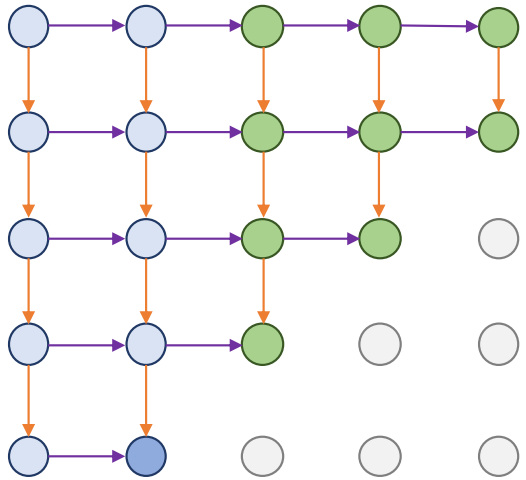
Last Time: Autoregressive Models

Explicit Density Function

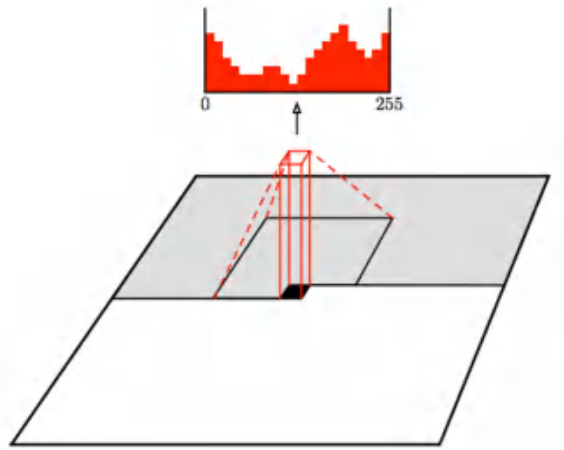
$$p(x) = p(x_1, x_2, x_3, \ldots, x_T)$$
$$= p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \ldots$$
$$= \prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1})$$

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

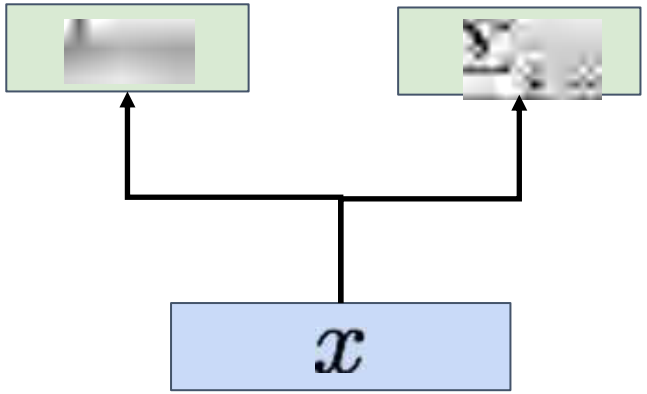
Last Time: Variational Autoencoders

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

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

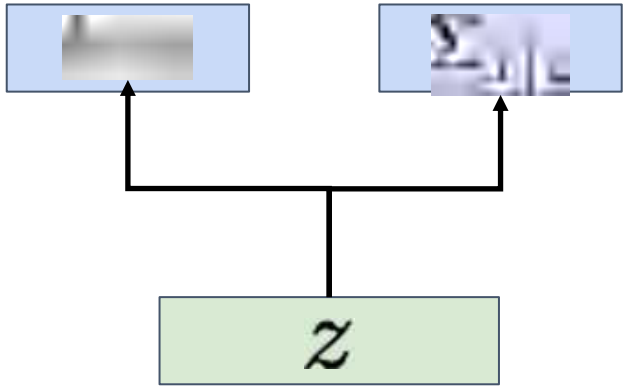
Encoder Network

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



Decoder Network

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



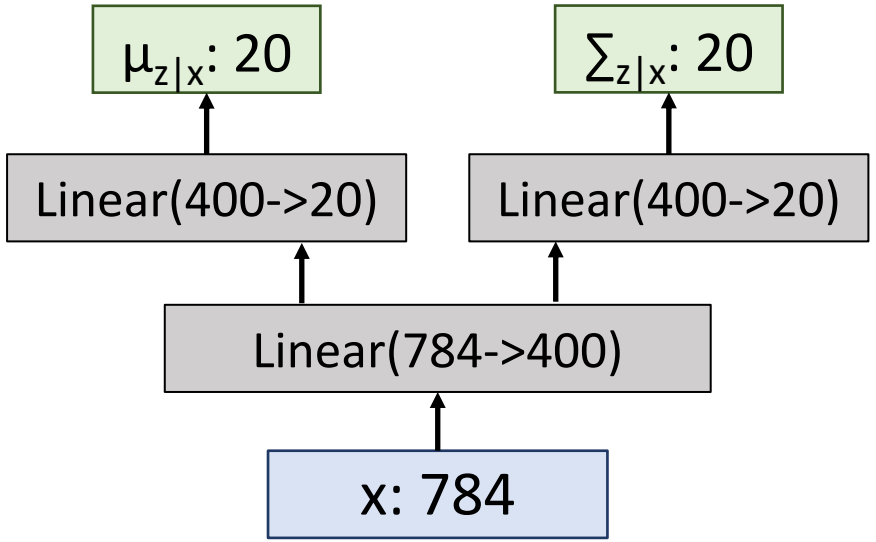
Example: Fully-Connected VAE

x: 28x28 image, flattened to 784-dim vector

z: 20-dim vector

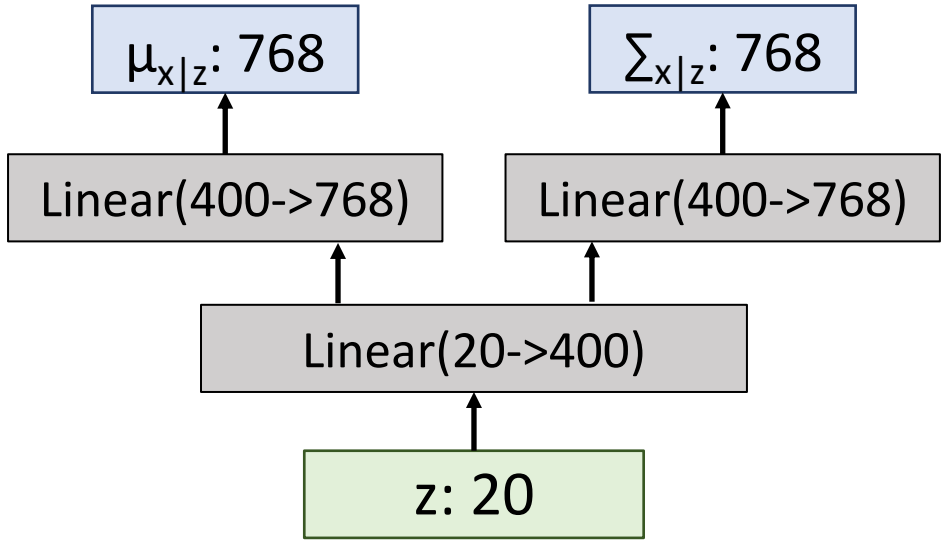
Encoder Network

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



Decoder Network

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

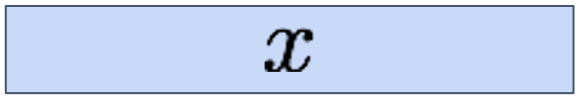


Variational Autoencoders

Train by maximizing the
variational lower bound

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

Input Data

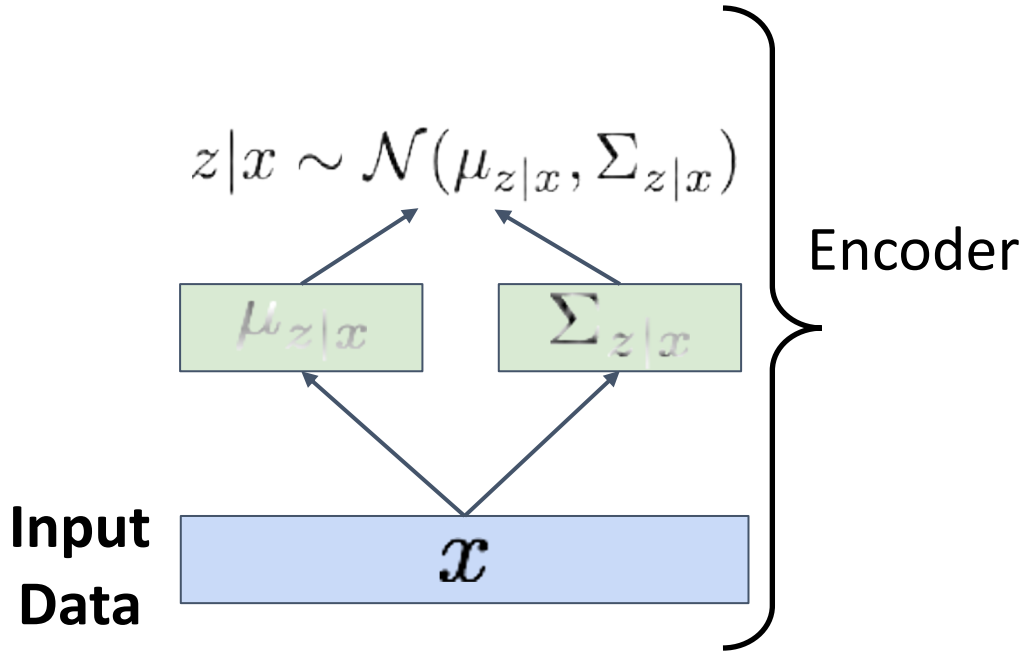


Variational Autoencoders

Train by maximizing the **variational lower bound**

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

1. Run input data through **encoder** to get a distribution over latent codes

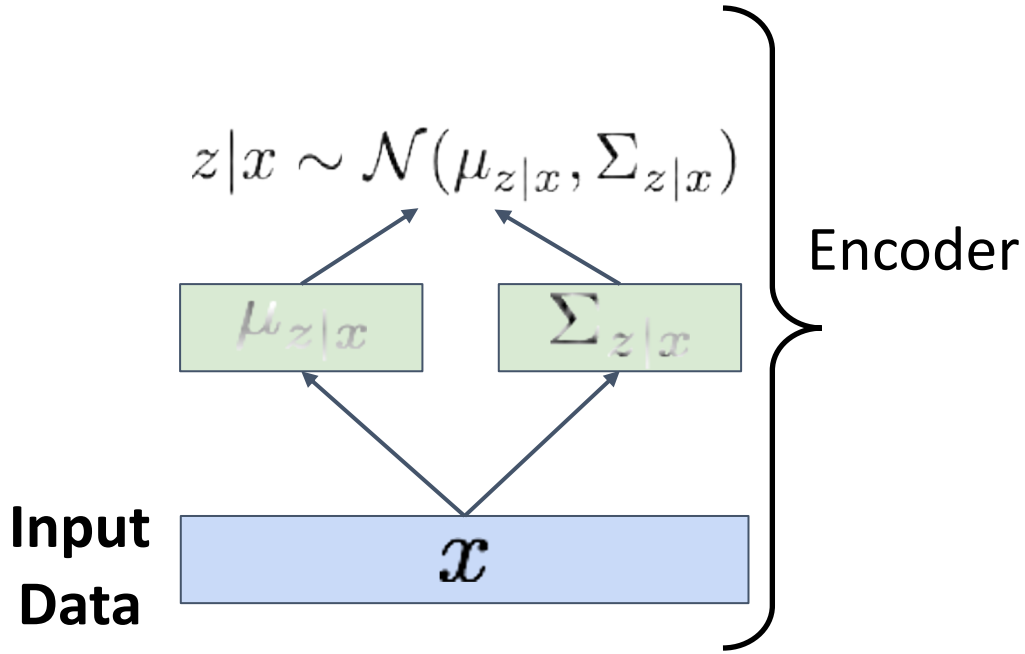


Variational Autoencoders

Train by maximizing the
variational lower bound

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

1. Run input data through **encoder** to get a distribution over latent codes
2. **Encoder output should match the prior p(z)!**



Variational Autoencoders

Train by maximizing the **variational lower bound**

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

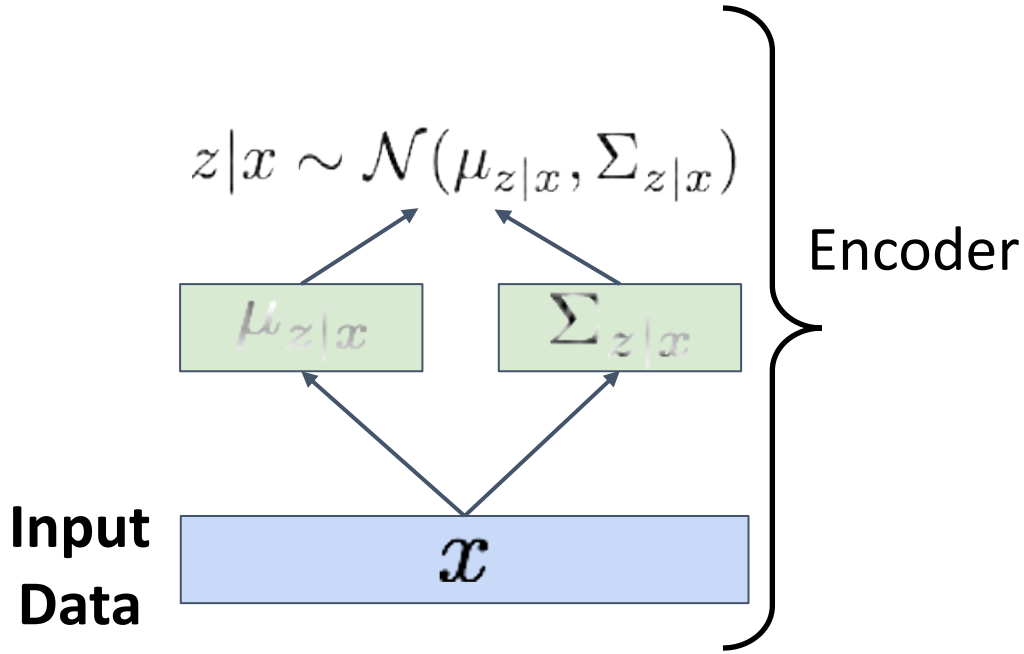
Closed form solution when q_ϕ is diagonal Gaussian and p is unit Gaussian! (Assume z has dimension J)

1. Run input data through **encoder** to get a distribution over latent codes
2. **Encoder output should match the prior p(z)!**

$$-D_{KL}\left(q_\phi(z|x), p(z)\right) = \int_Z q_\phi(z|x) \log \frac{p(z)}{q_\phi(z|x)} dz$$

$$= \int_Z N\left(z; \mu_{z|x}, \Sigma_{z|x}\right) \log \frac{N(z; 0, I)}{N\left(z; \mu_{z|x}, \Sigma_{z|x}\right)} dz$$

$$= \frac{1}{2}\sum_{j=1}^{J}\left(1 + \log\left(\left(\Sigma_{z|x}\right)_j^2\right) - \left(\mu_{z|x}\right)_j^2 - \left(\Sigma_{z|x}\right)_j^2\right)$$

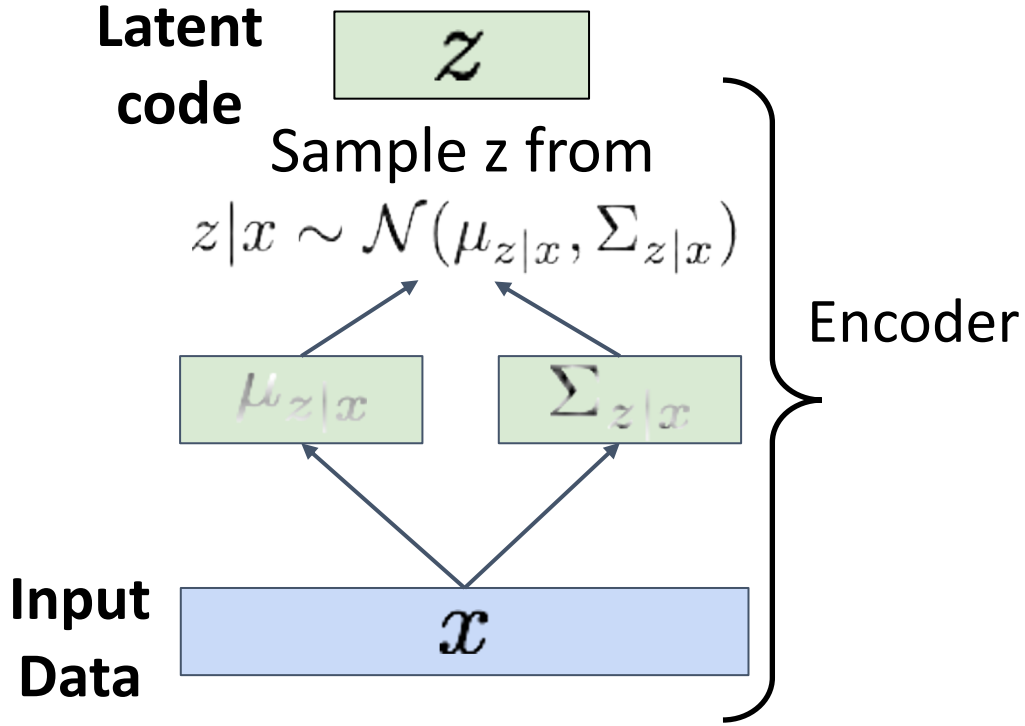


Variational Autoencoders

Train by maximizing the
variational lower bound

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

1. Run input data through **encoder** to get a distribution over latent codes
2. **Encoder output should match the prior p(z)!**
3. Sample code z from encoder output

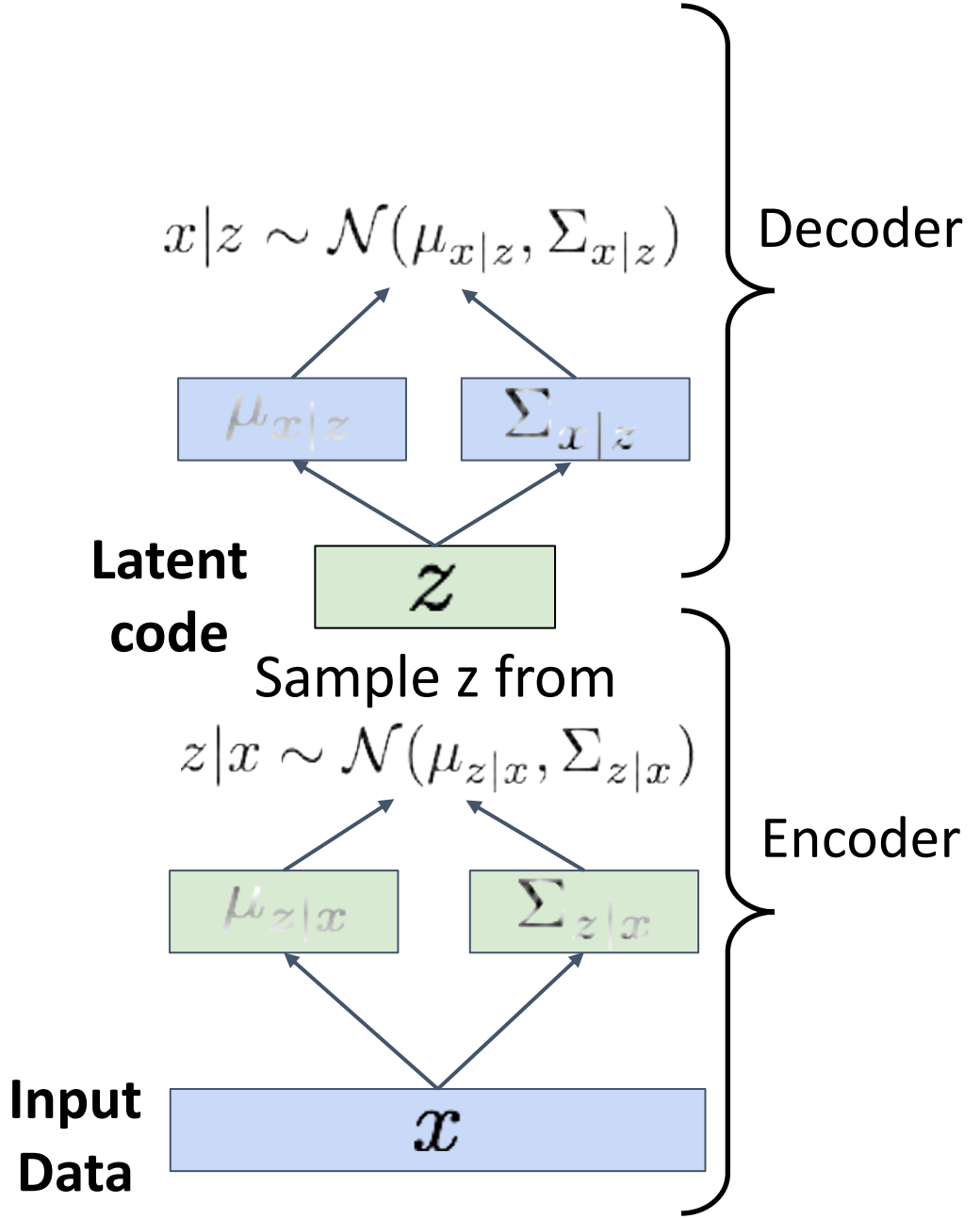


Variational Autoencoders

Train by maximizing the **variational lower bound**

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

1. Run input data through **encoder** to get a distribution over latent codes
2. **Encoder output should match the prior p(z)!**
3. Sample code z from encoder output
4. Run sampled code through **decoder** to get a distribution over data samples

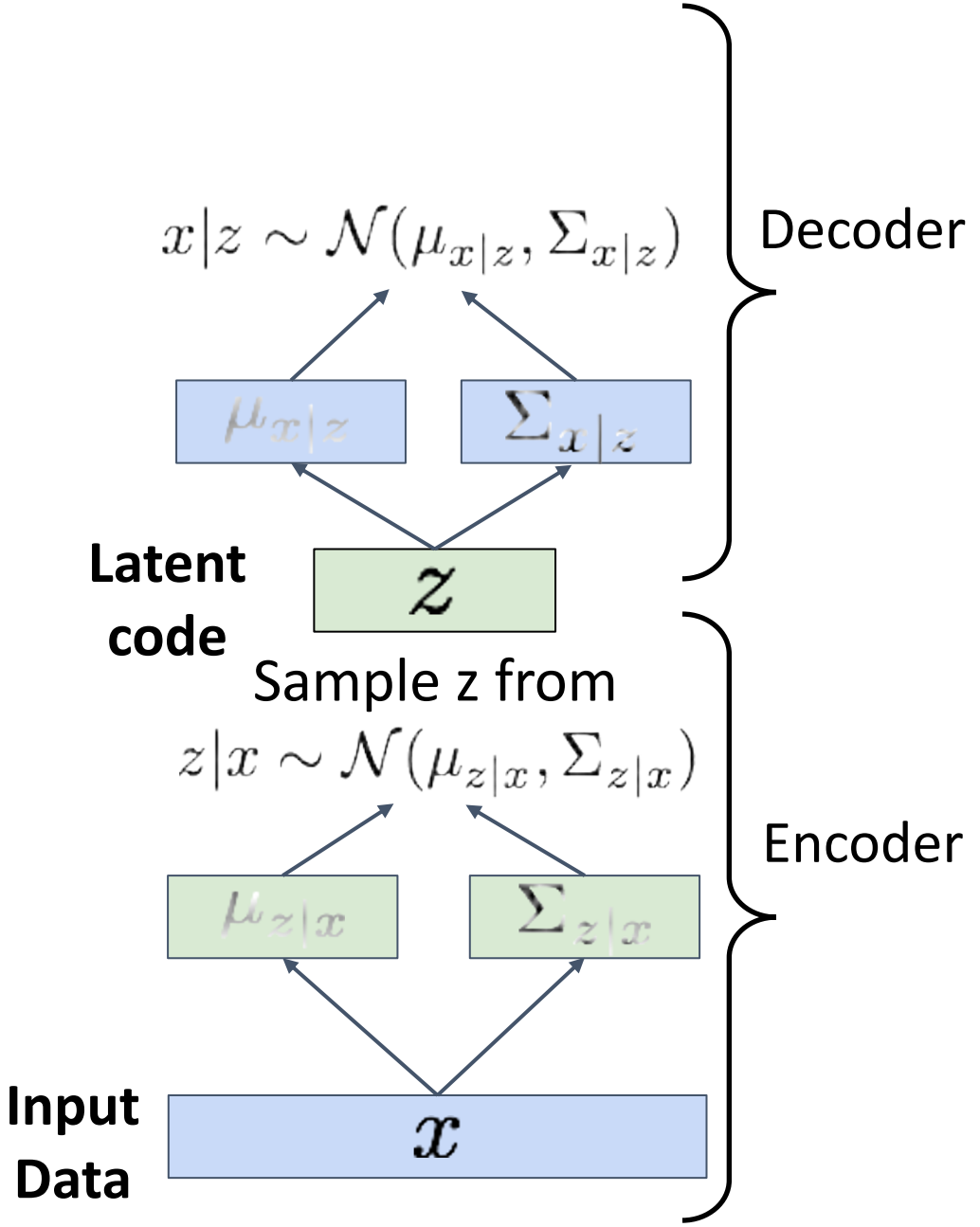


Variational Autoencoders

Train by maximizing the **variational lower bound**

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

1. Run input data through **encoder** to get a distribution over latent codes
2. **Encoder output should match the prior p(z)!**
3. Sample code z from encoder output
4. Run sampled code through **decoder** to get a distribution over data samples
5. **Original input data should be likely under the distribution output from (4)!**

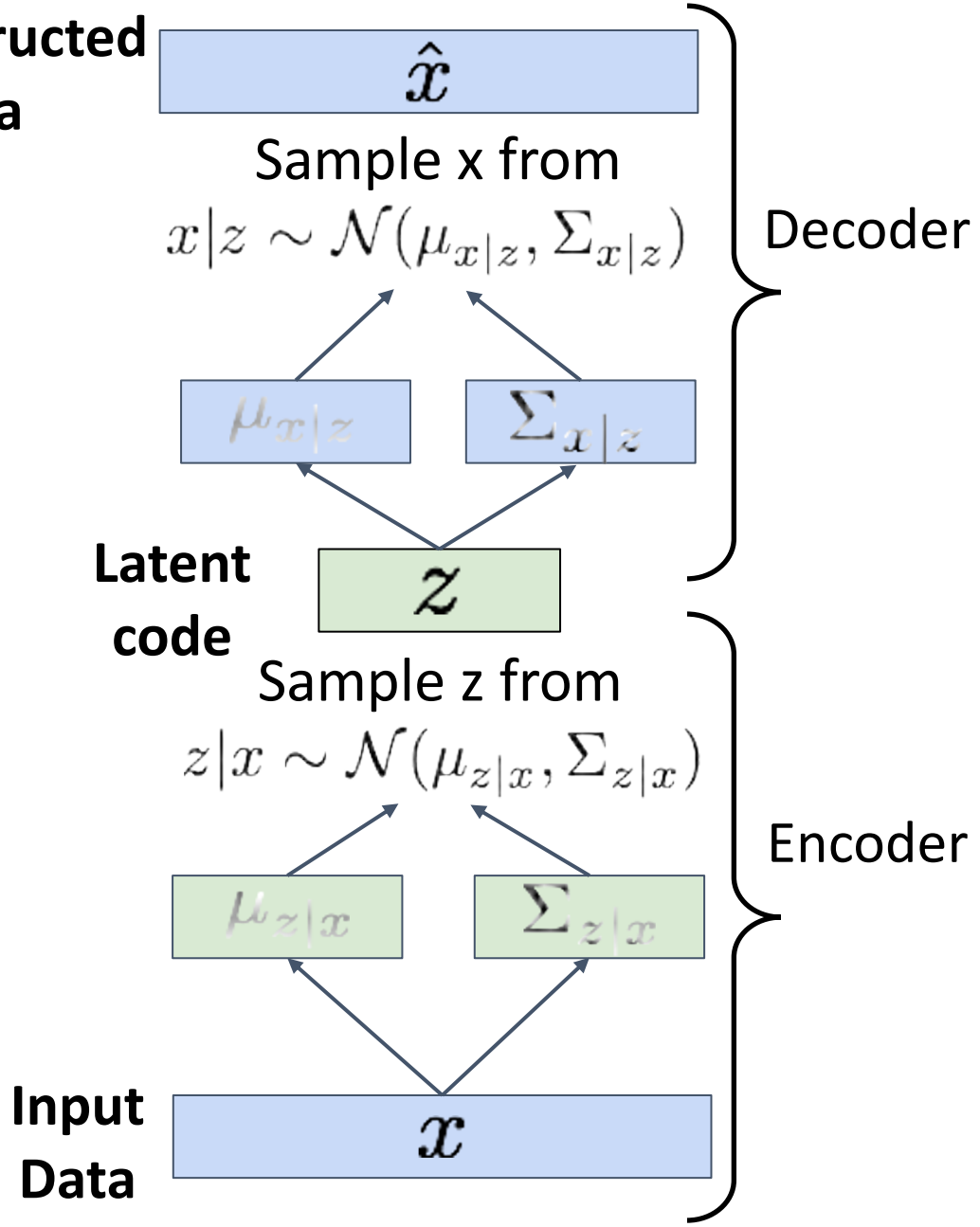


Variational Autoencoders

Train by maximizing the **variational lower bound**

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

1. Run input data through **encoder** to get a distribution over latent codes
2. **Encoder output should match the prior p(z)!**
3. Sample code z from encoder output
4. Run sampled code through **decoder** to get a distribution over data samples
5. **Original input data should be likely under the distribution output from (4)!**
6. Can sample a reconstruction from (4)

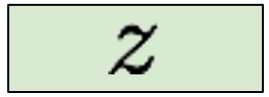


Variational Autoencoders: Generating Data

After training we can generate new data!

1. Sample z from prior p(z)

Latent code

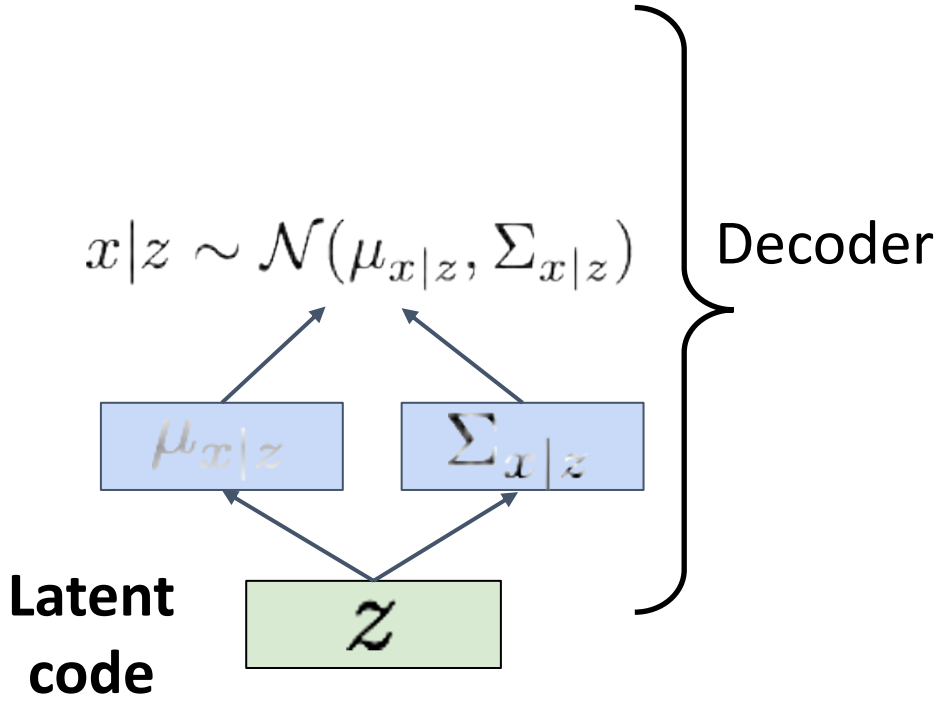


Sample z from prior p(z)

Variational Autoencoders: Generating Data

After training we can generate new data!

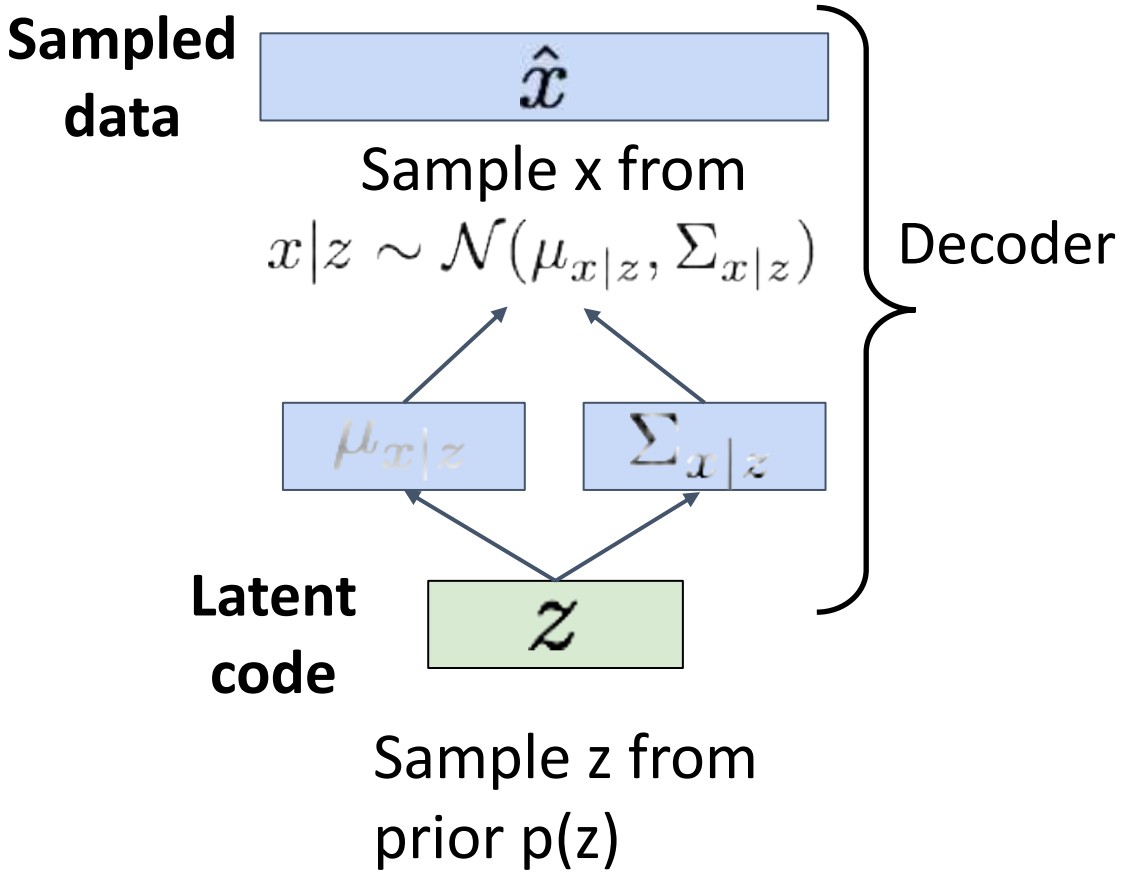
1. Sample z from prior p(z)
2. Run sampled z through decoder to get distribution over data x



Variational Autoencoders: Generating Data

After training we can generate new data!

1. Sample z from prior p(z)
2. Run sampled z through decoder to get distribution over data x
3. Sample from distribution in (2) to generate data



Variational Autoencoders: Generating Data

32x32 CIFAR-10

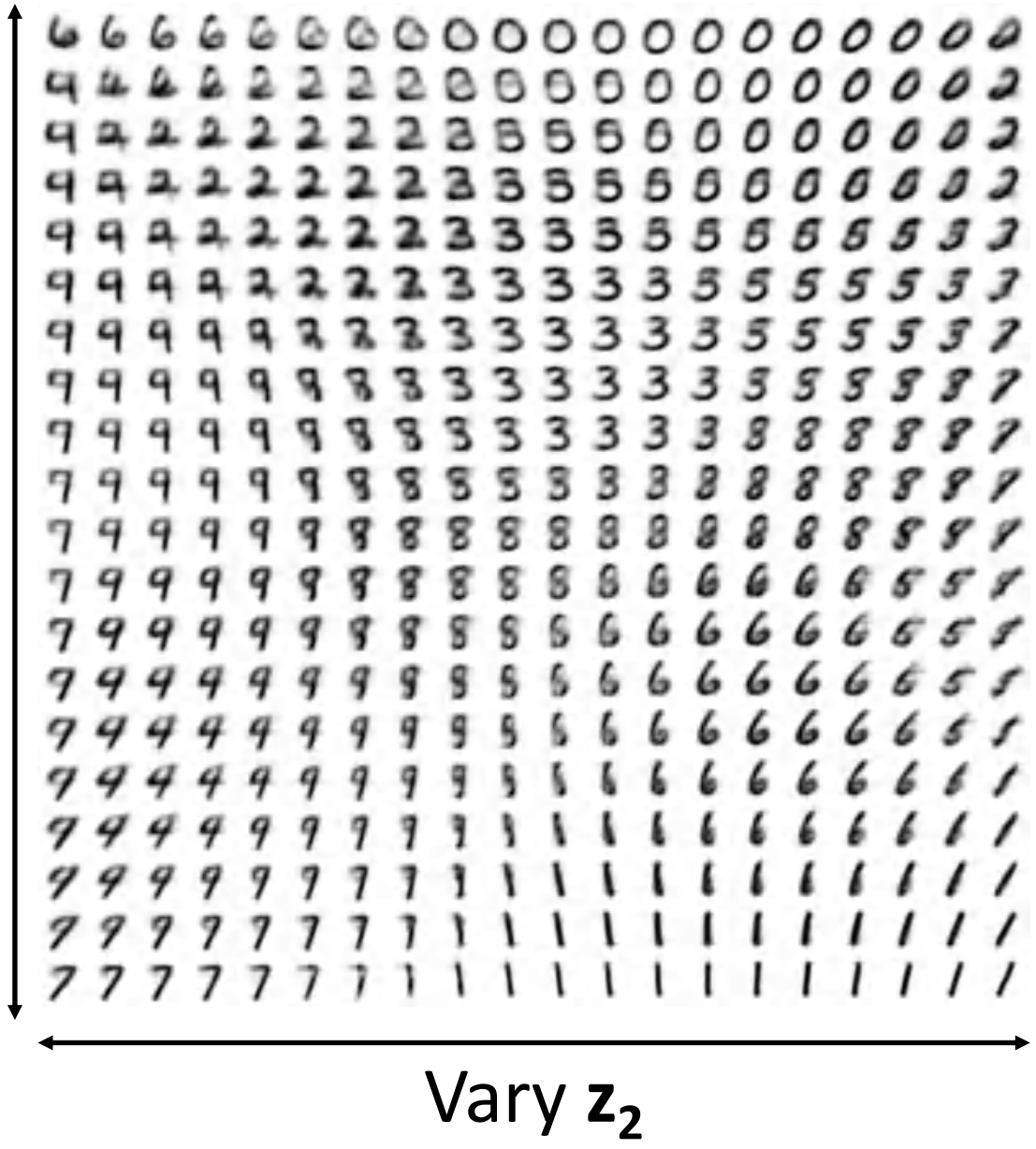
Labeled Faces in the Wild

Figures from (L) Dirk Kingma et al. 2016; (R) Anders Larsen et al. 2017.

Variational Autoencoders

The diagonal prior on p(z) causes dimensions of z to be independent

“Disentangling factors of variation”

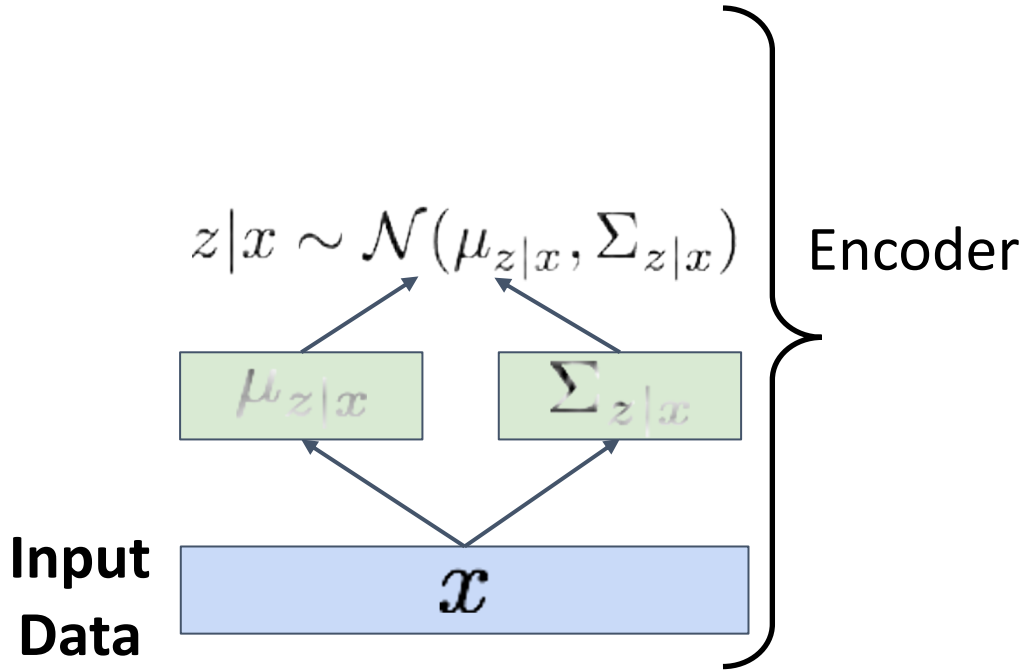


Kingma and Welling, Auto-Encoding Variational Beyes, ICLR 2014

Variational Autoencoders

After training we can **edit images**

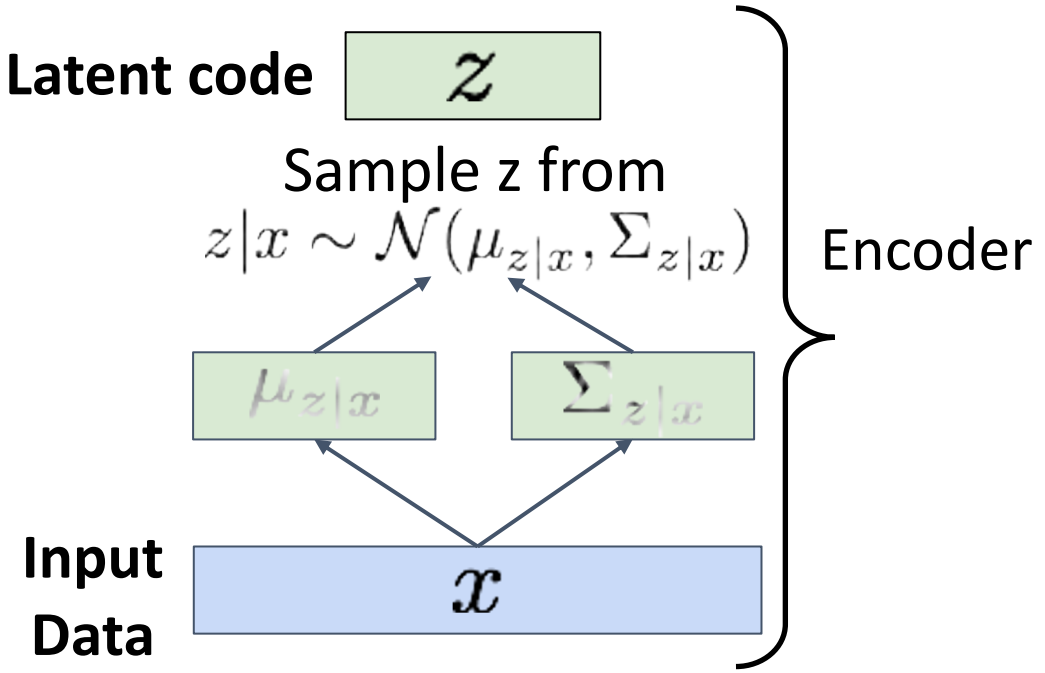
1. Run input data through **encoder** to get a distribution over latent codes



Variational Autoencoders

After training we can **edit images**

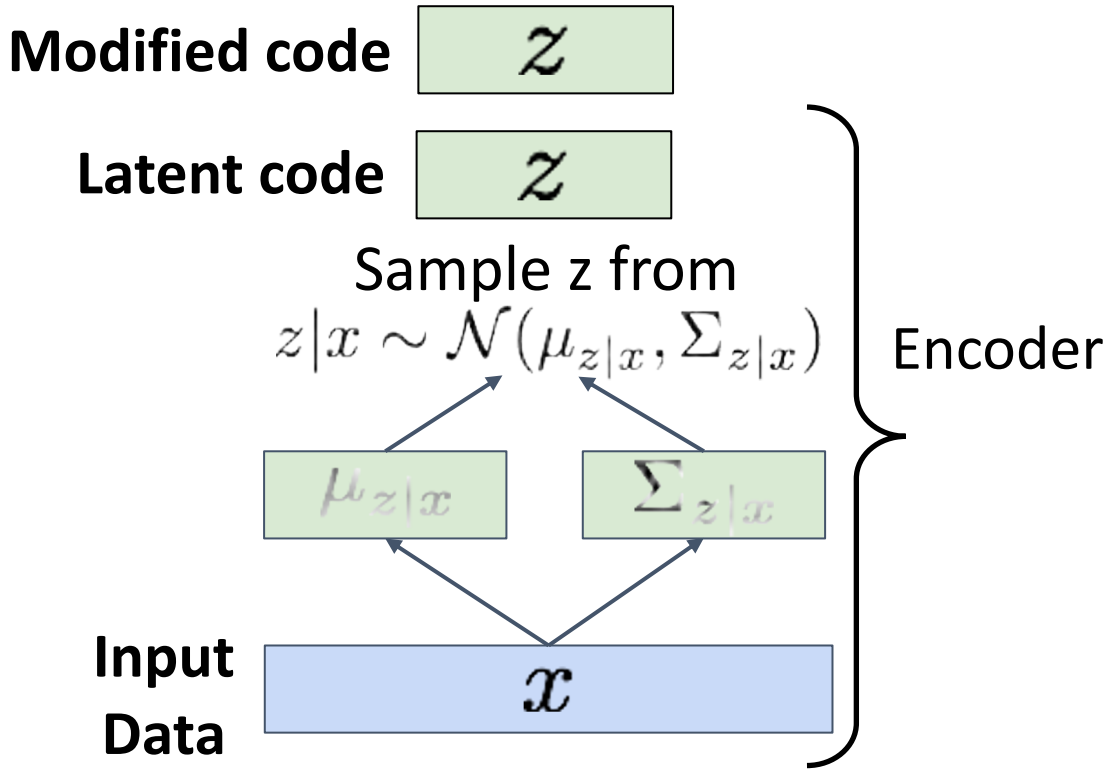
1. Run input data through **encoder** to get a distribution over latent codes
2. Sample code z from encoder output



Variational Autoencoders

After training we can **edit images**

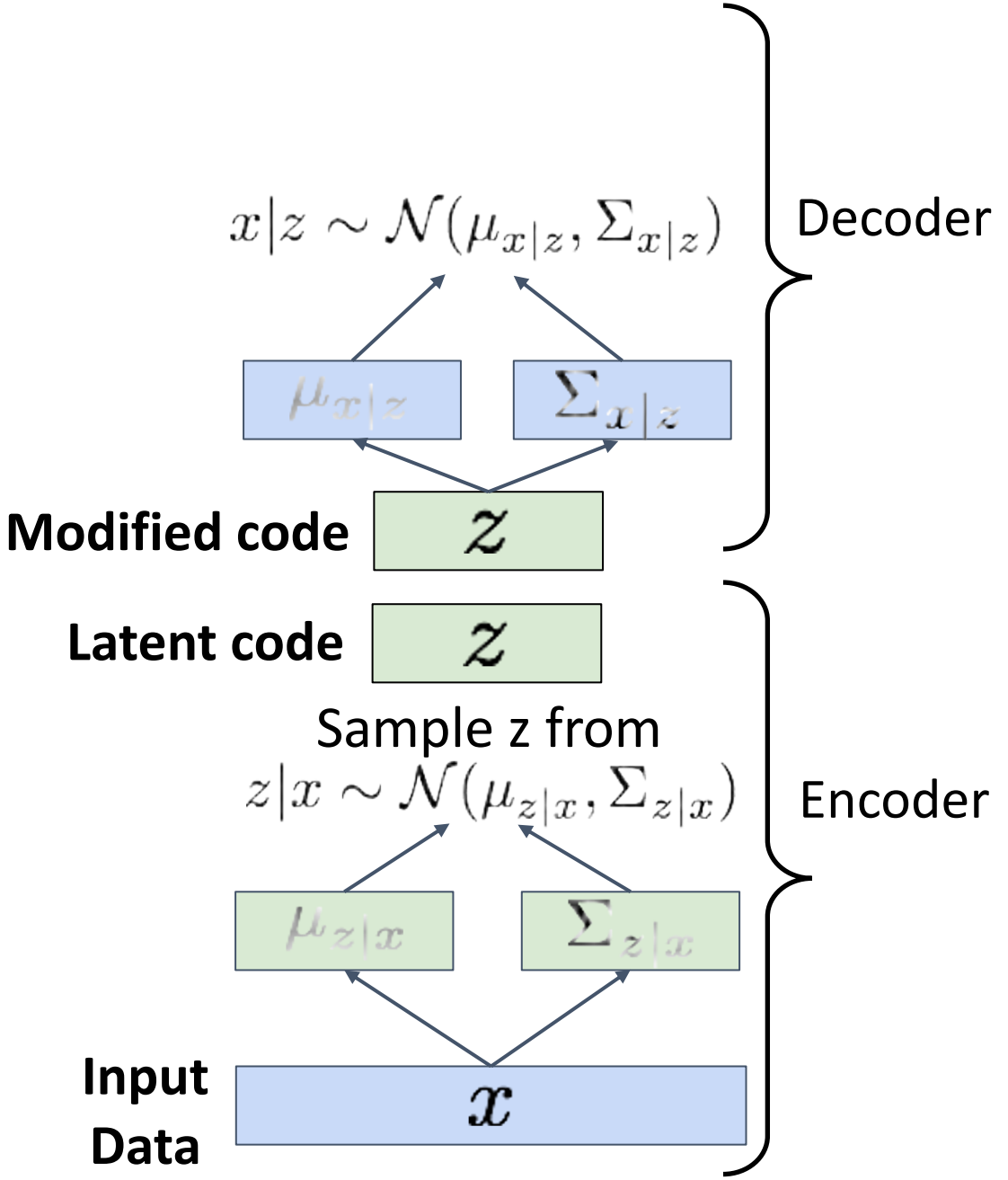
1. Run input data through **encoder** to get a distribution over latent codes
2. Sample code z from encoder output
3. Modify some dimensions of sampled code



Variational Autoencoders

After training we can **edit images**

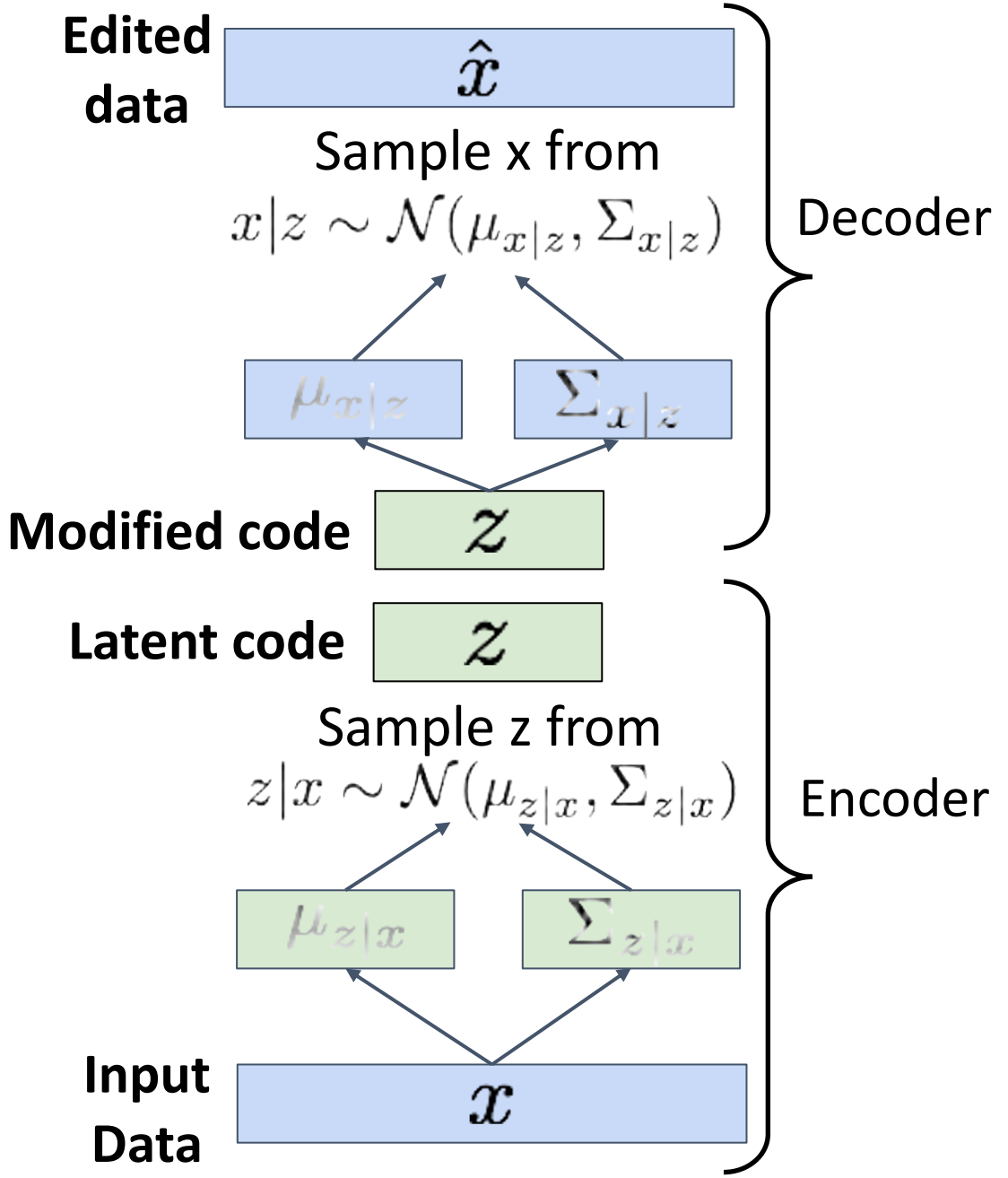
1. Run input data through **encoder** to get a distribution over latent codes
2. Sample code z from encoder output
3. Modify some dimensions of sampled code
4. Run modified z through **decoder** to get a distribution over data sample



Variational Autoencoders

After training we can **edit images**

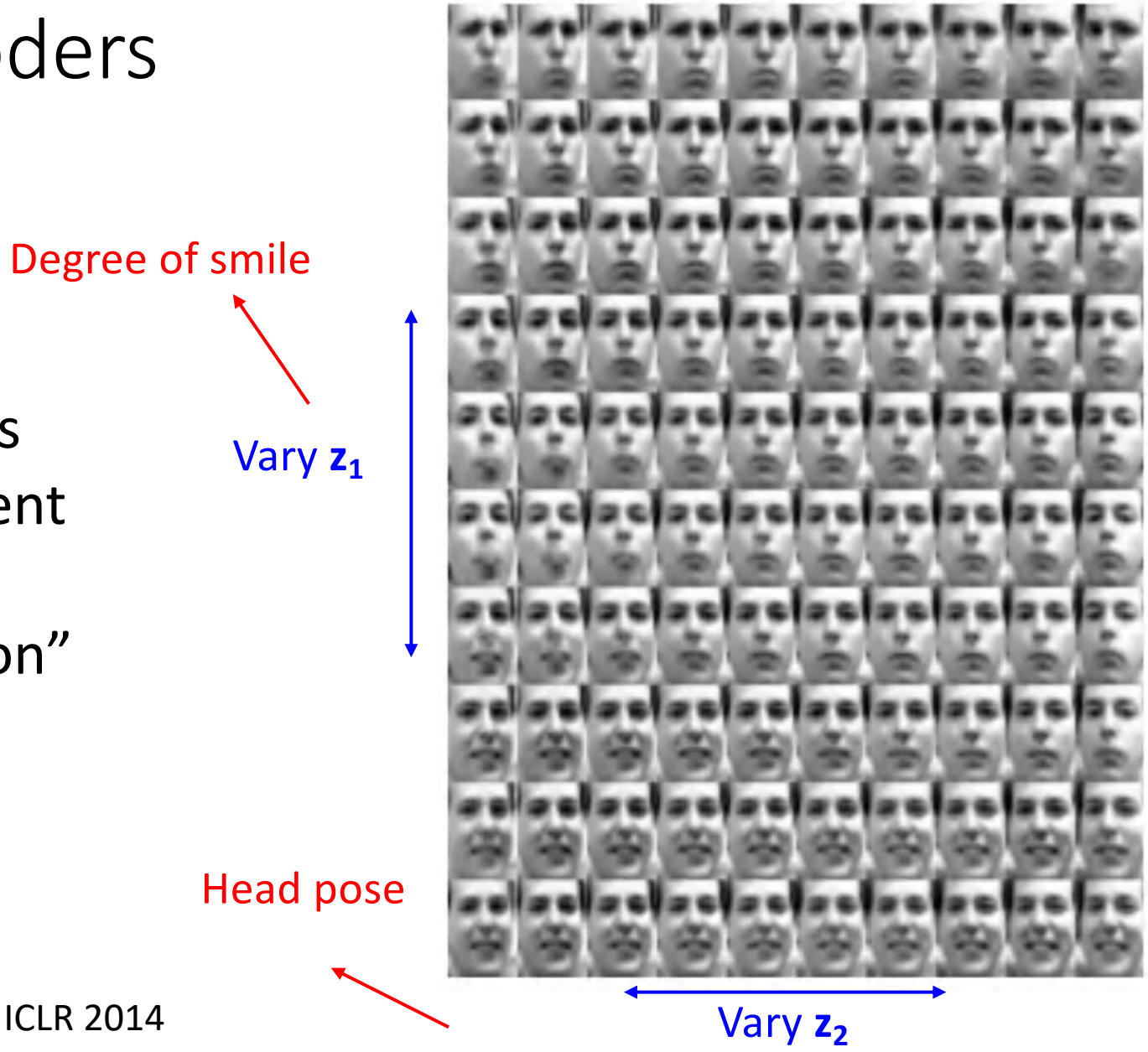
1. Run input data through **encoder** to get a distribution over latent codes
2. Sample code z from encoder output
3. Modify some dimensions of sampled code
4. Run modified z through **decoder** to get a distribution over data samples
5. Sample new data from (4)



Variational Autoencoders

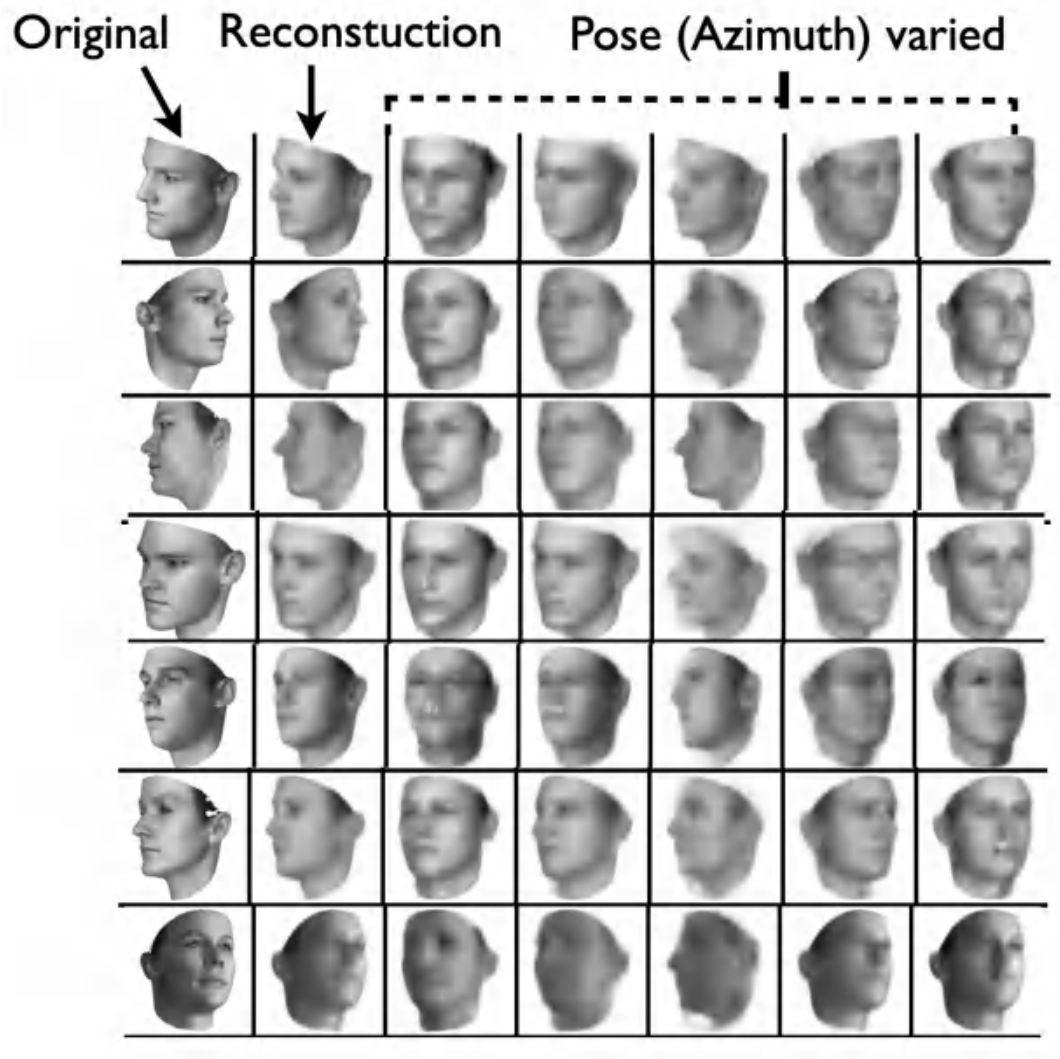
The diagonal prior on p(z) causes dimensions of z to be independent

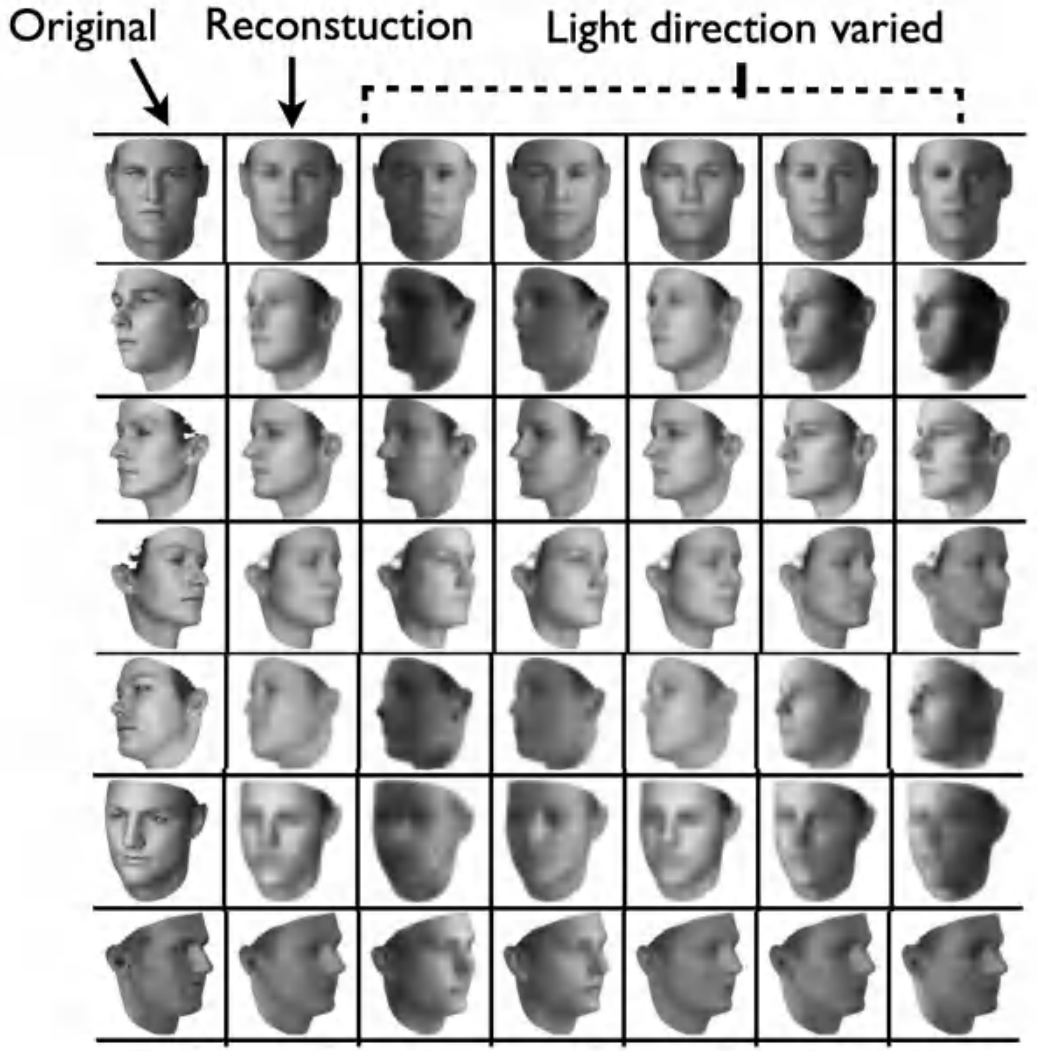
"Disentangling factors of variation"



Kingma and Welling, Auto-Encoding Variational Beyes, ICLR 2014

Variational Autoencoders: Image Editing





Kulkarni et al, "Deep Convolutional Inverse Graphics Networks", NeurIPS 2014

Variational Autoencoder: Summary

Probabilistic spin to traditional autoencoders => allows generating data
Defines an intractable density => derive and optimize a (variational) lower bound

Pros:

- Principled approach to generative models
- Allows inference of $q(z|x)$, can be useful feature representation for other tasks

Cons:

- Maximizes lower bound of likelihood: okay, but not as good evaluation as PixelRNN/PixelCNN
- Samples blurrier and lower quality compared to state-of-the-art (GANs)

Active areas of research:

- More flexible approximations, e.g. richer approximate posterior instead of diagonal Gaussian, e.g., Gaussian Mixture Models (GMMs)
- Incorporating structure in latent variables, e.g., Categorical Distributions

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

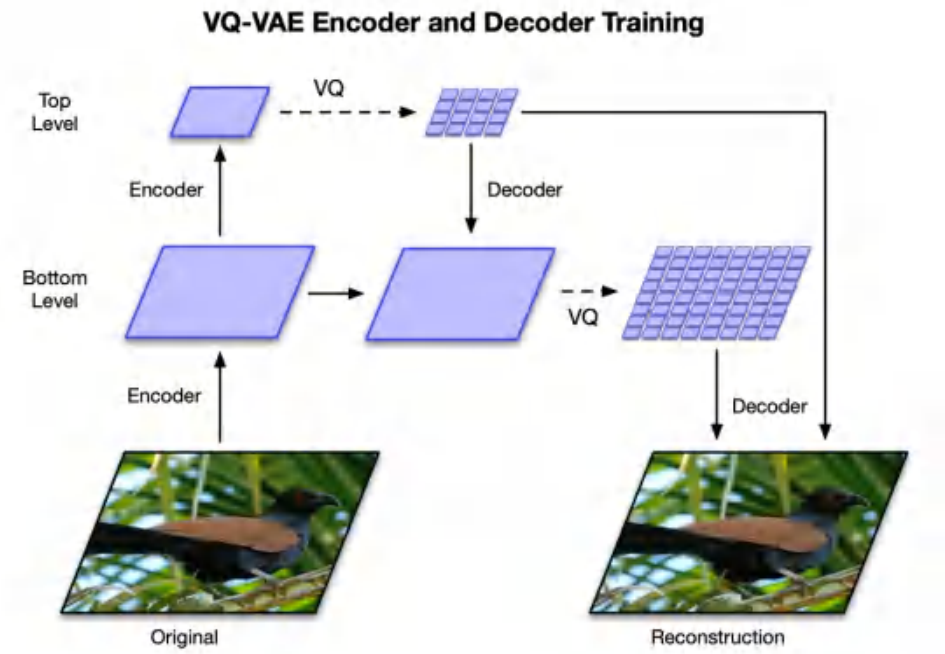
Variational models

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

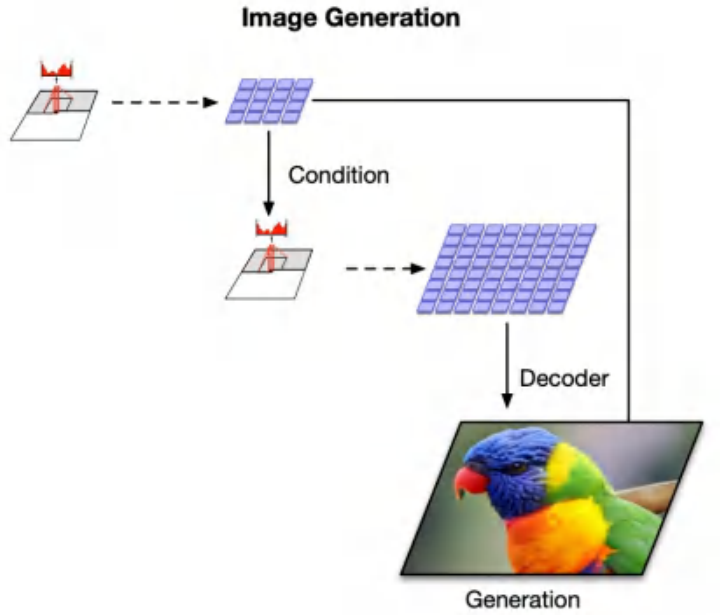
Can we combine them and get the best of both worlds?

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

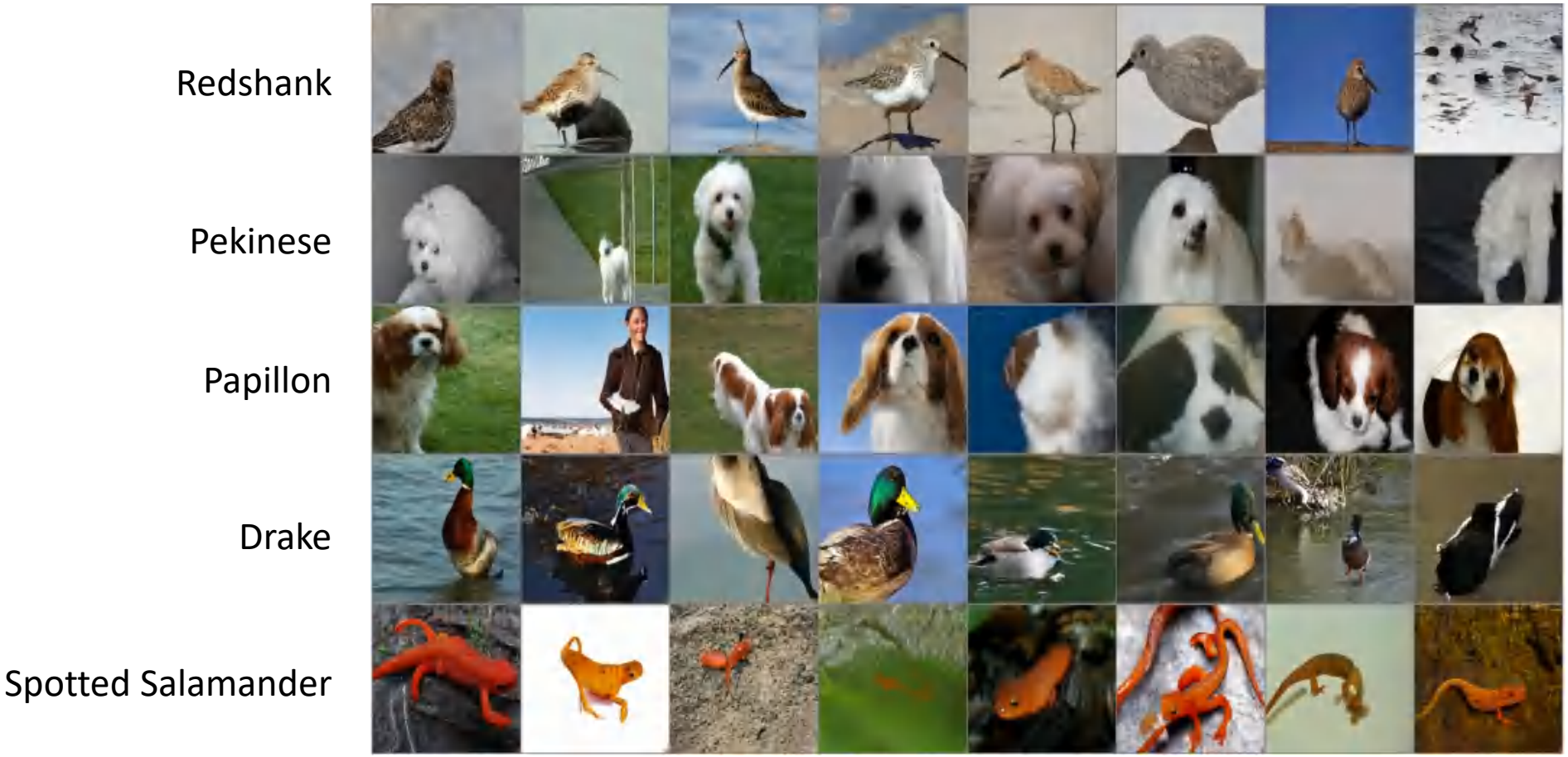
Combining VAE + Autoregressive: VQ-VAE2

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

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}.

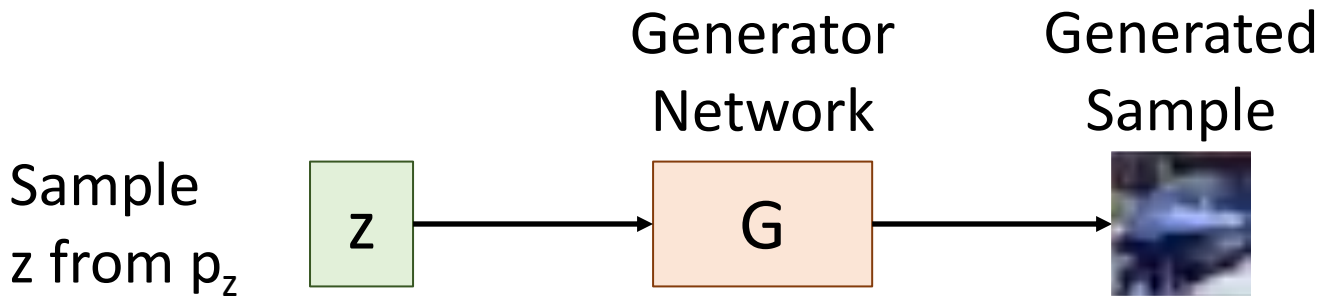
Idea: Introduce a latent variable z with simple prior p(z).
Sample $z \sim p(z)$ and pass to a **Generator Network** x = G(z)
Then x is a sample from the **Generator distribution** p_G. Want $p_G = p_{data}$!

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 $p_G = p_{data}$!



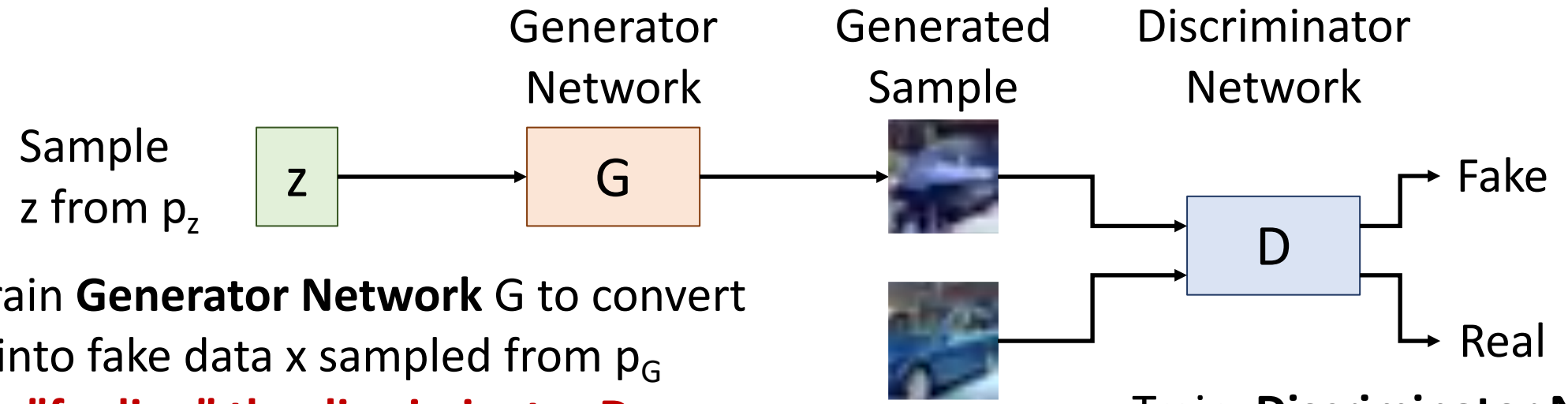
Train **Generator Network** G to convert z into fake data x sampled from p_G

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)

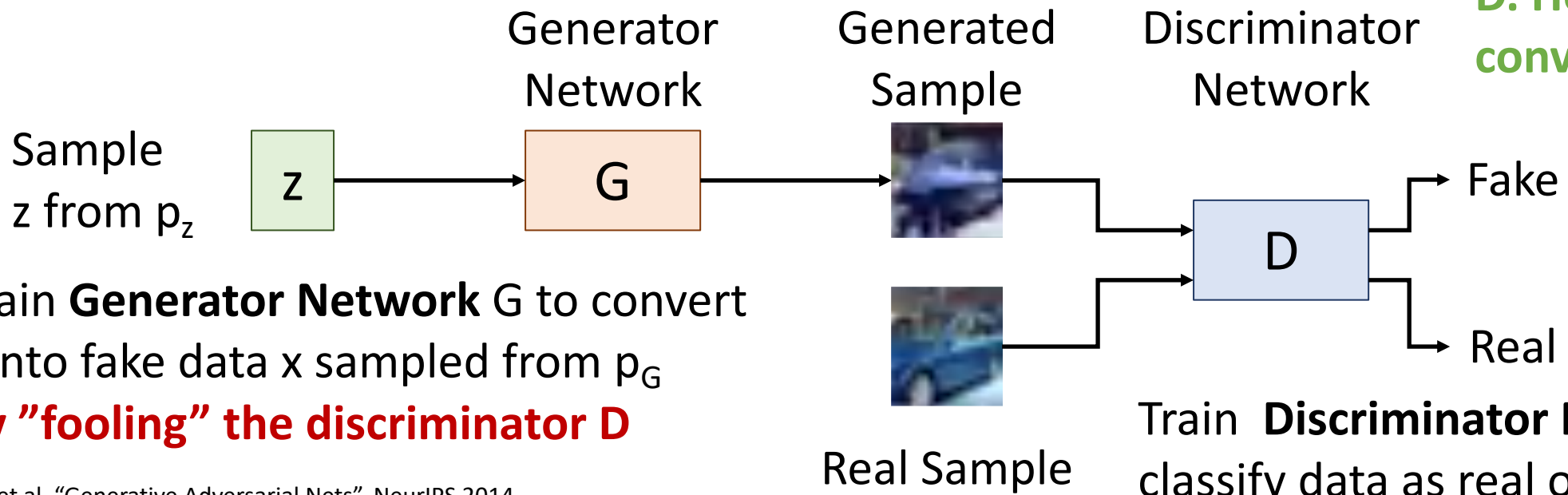
Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

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**

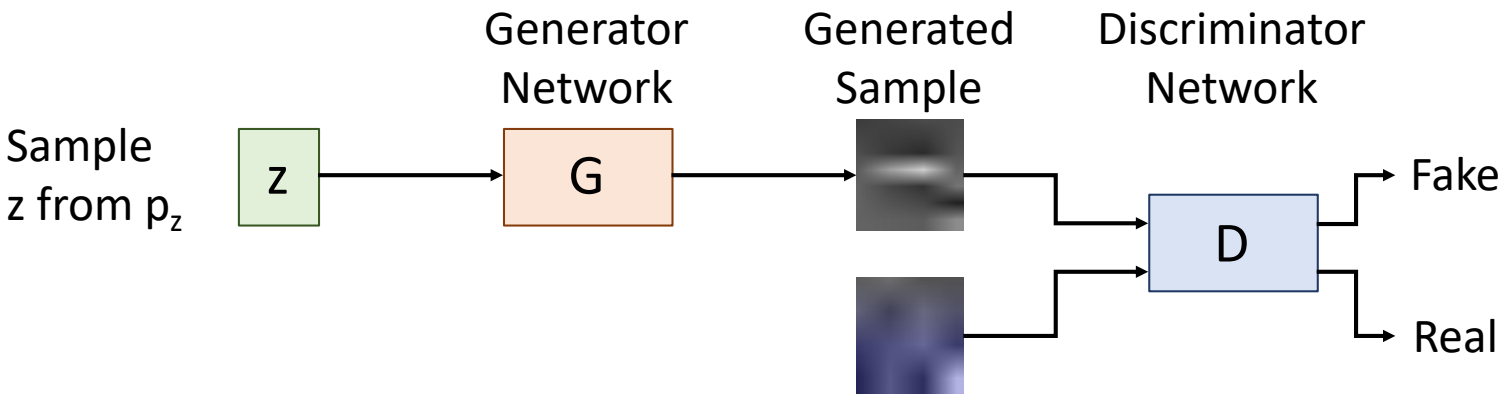
$$\min_G \max_D \left(E_{x \sim p_{data}} [\log D(x)] + E_{z \sim p(z)} \left[\log \left(1 - D\left(G(z)\right) \right) \right] \right)$$

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

Generative Adversarial Networks: Training Objective

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

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



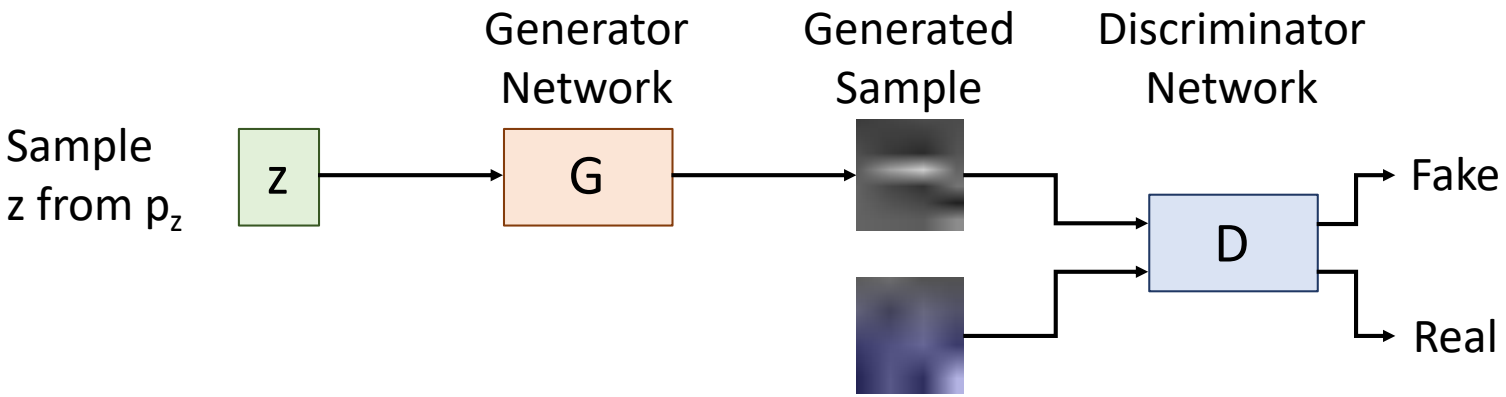
Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

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

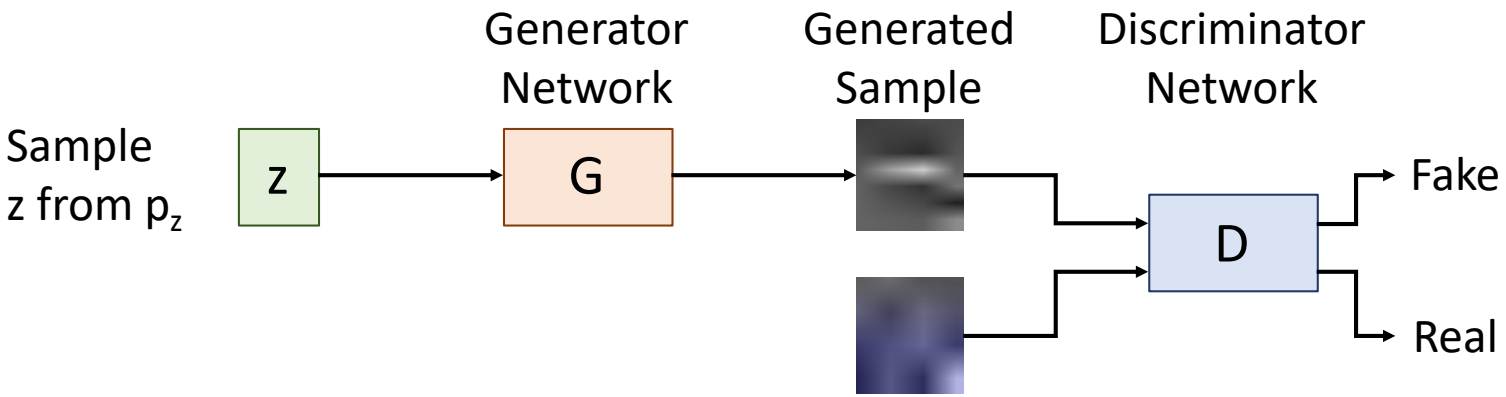
Generative Adversarial Networks: Training Objective

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

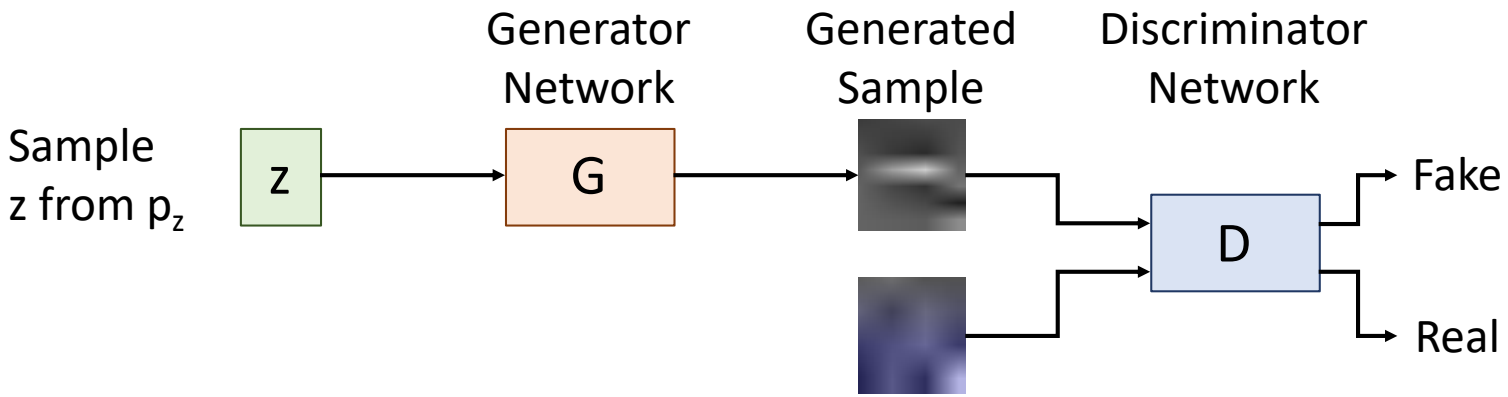
Jointly train generator G and discriminator D with a **minimax game**

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

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

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

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) $G = G - \alpha_G \frac{\partial V}{\partial G}$

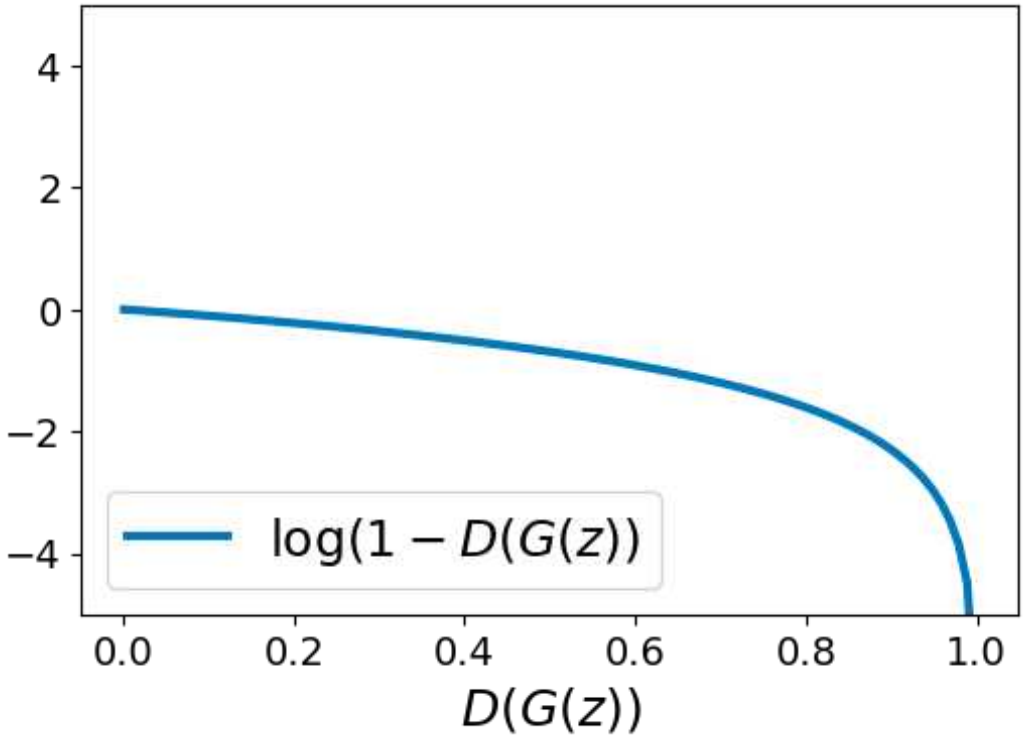
Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

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

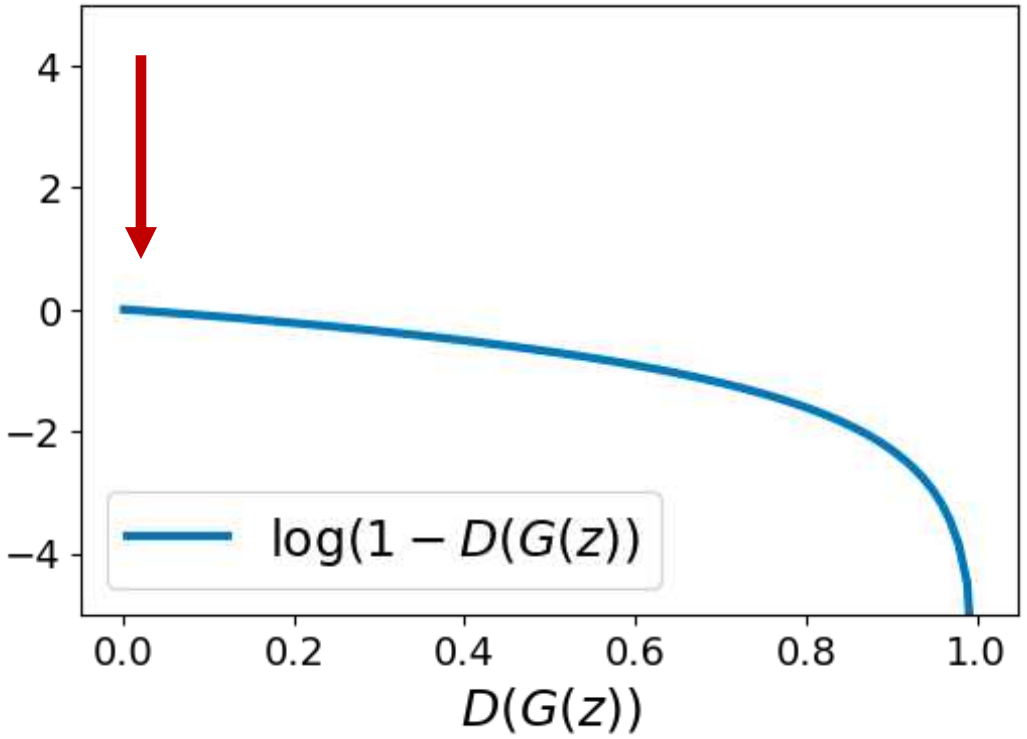
Generative Adversarial Networks: Training Objective

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

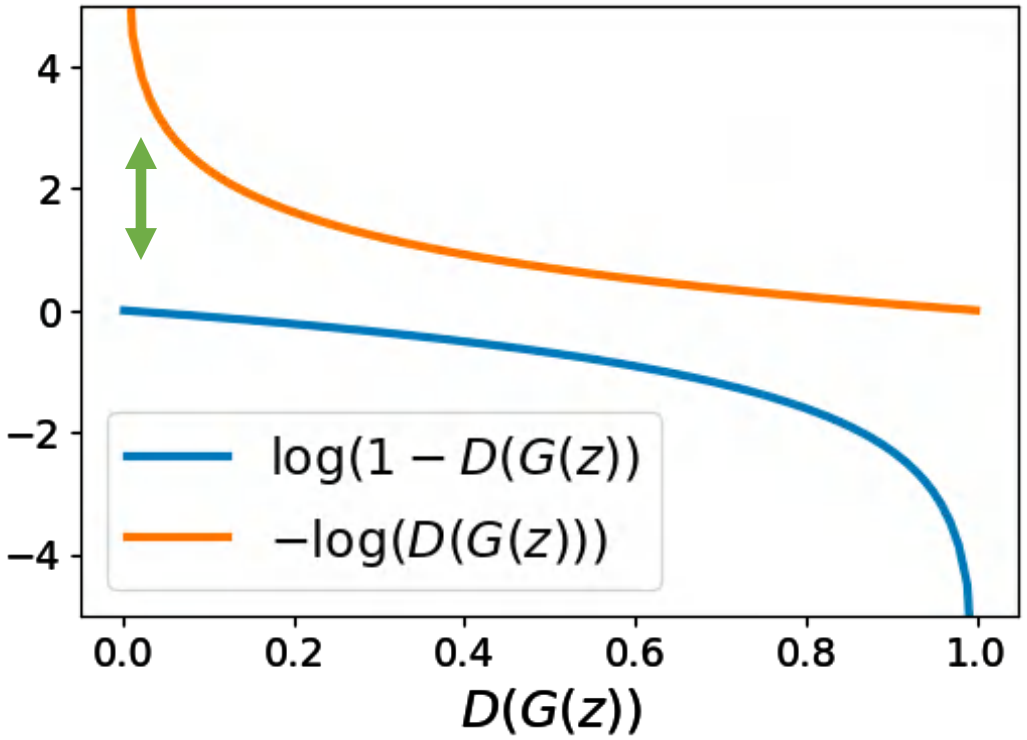
Jointly train generator G and discriminator D with a **minimax game**

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

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

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

(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

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

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

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

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

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

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)

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)

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

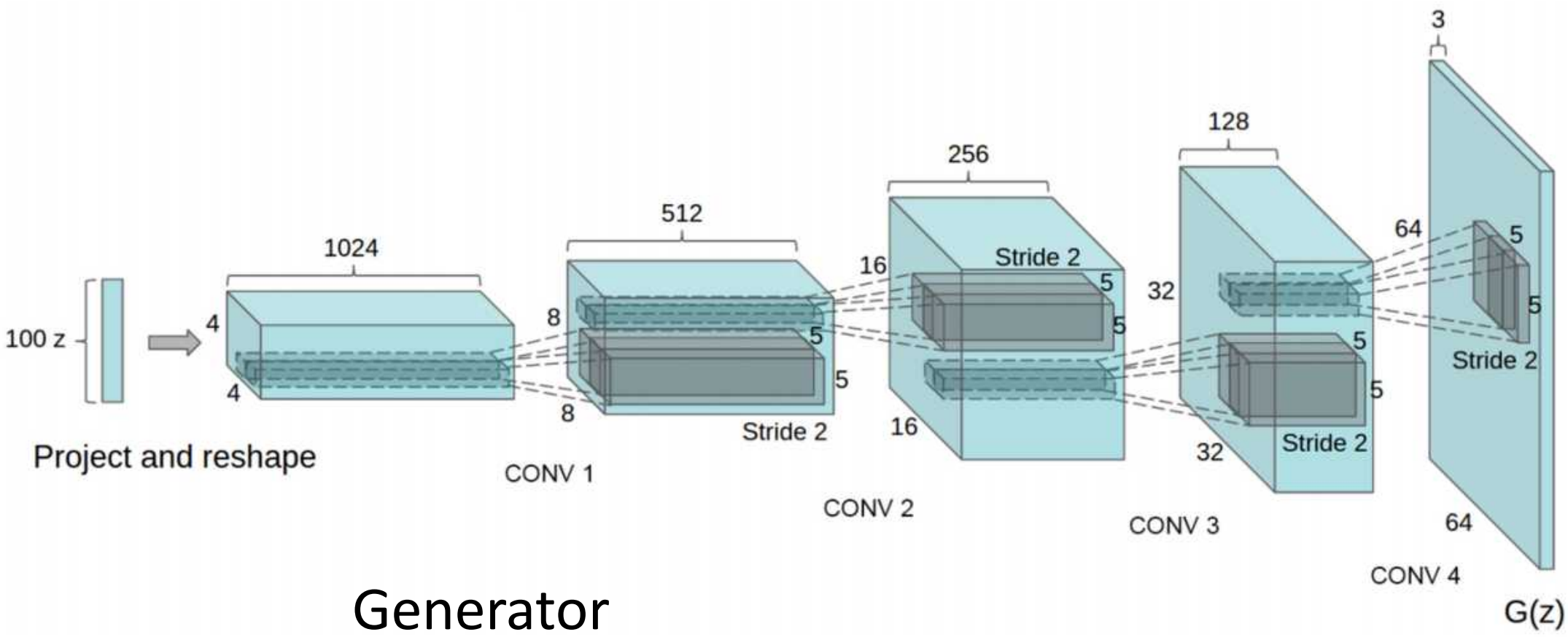
Generative Adversarial Networks: Results

Generated samples

Nearest neighbor from training set

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

Generative Adversarial Networks: DC-GAN

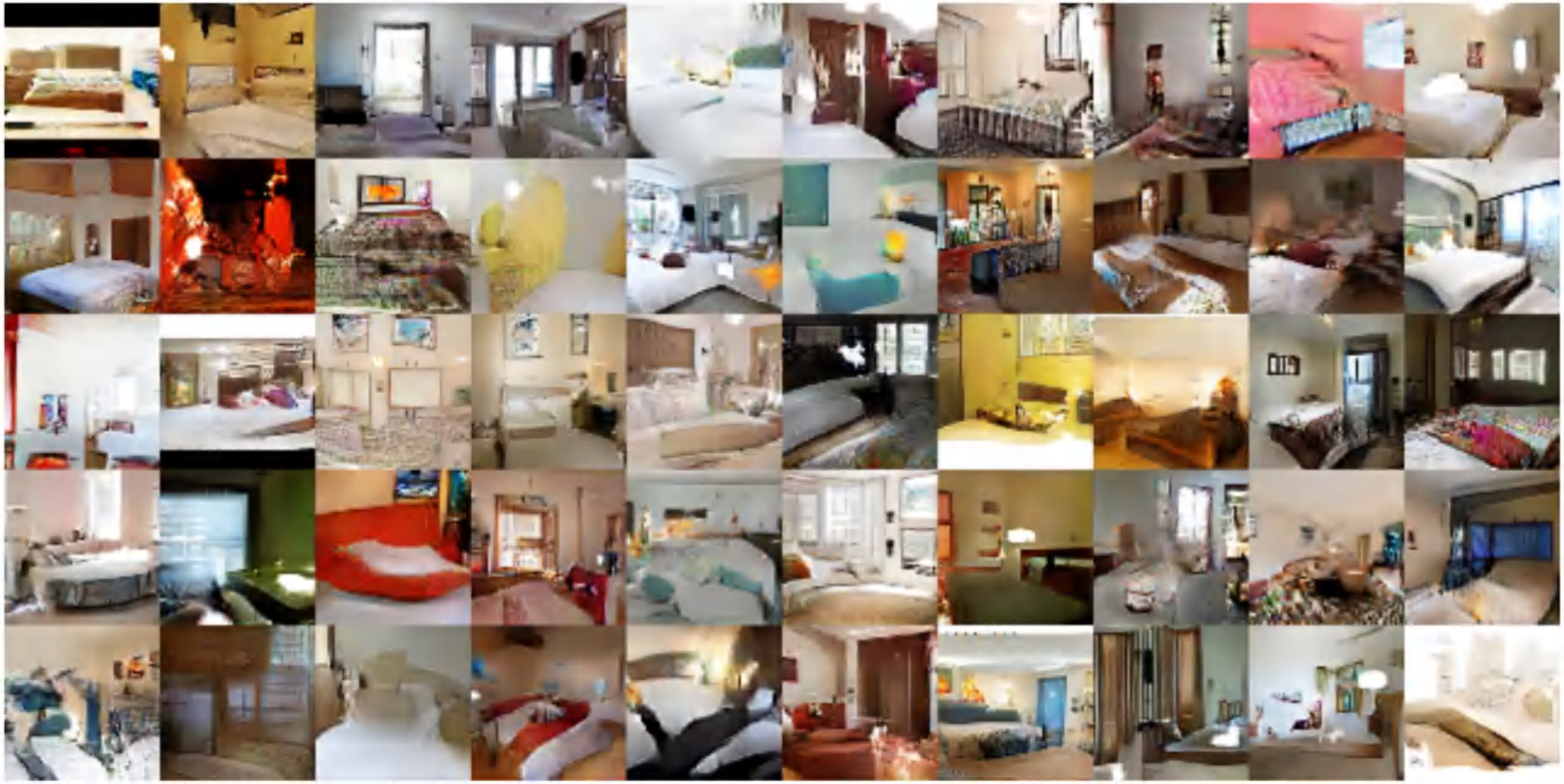


Generator

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

Smiling woman

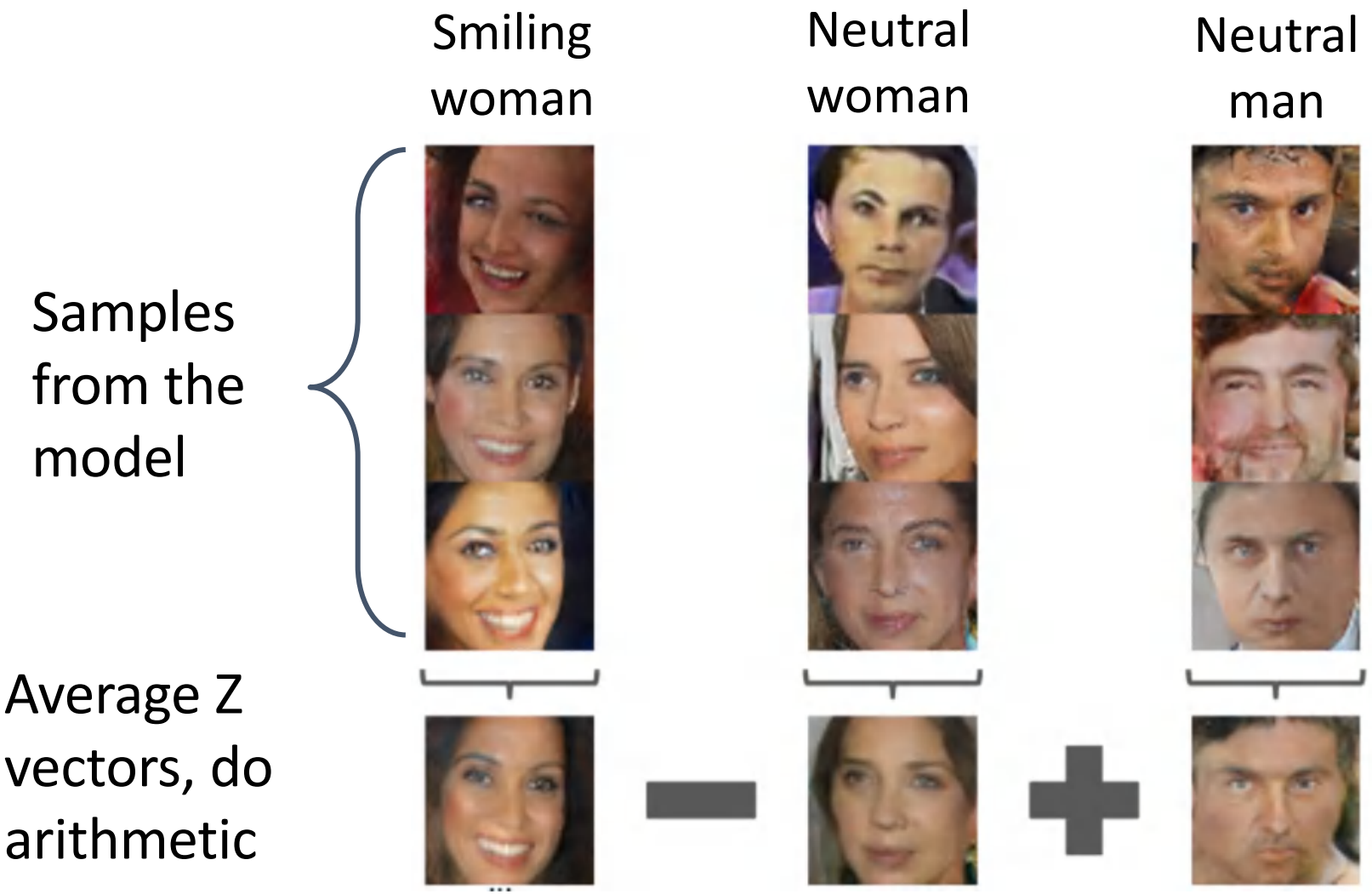
Neutral woman

Neutral man

Samples from the model

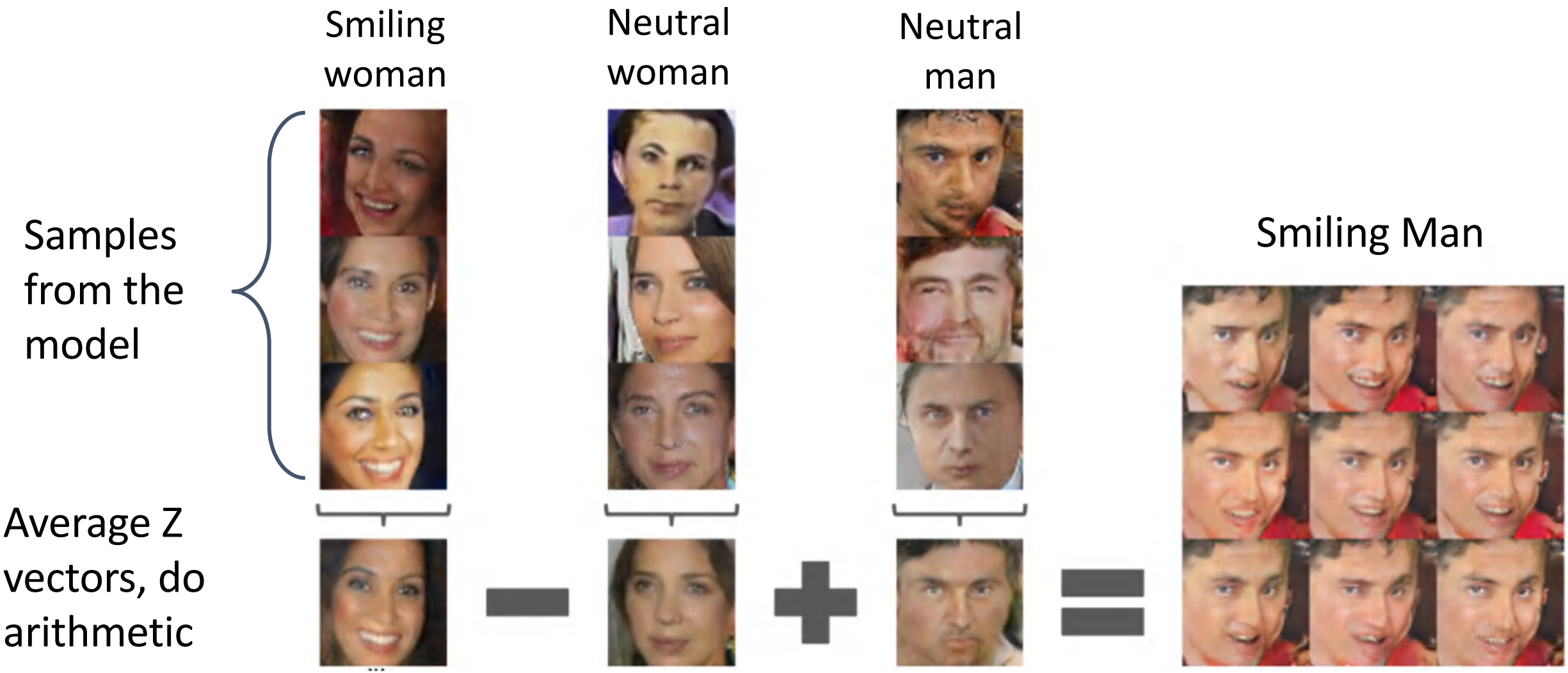
Radford et al, ICLR 2016

Generative Adversarial Networks: Vector Math



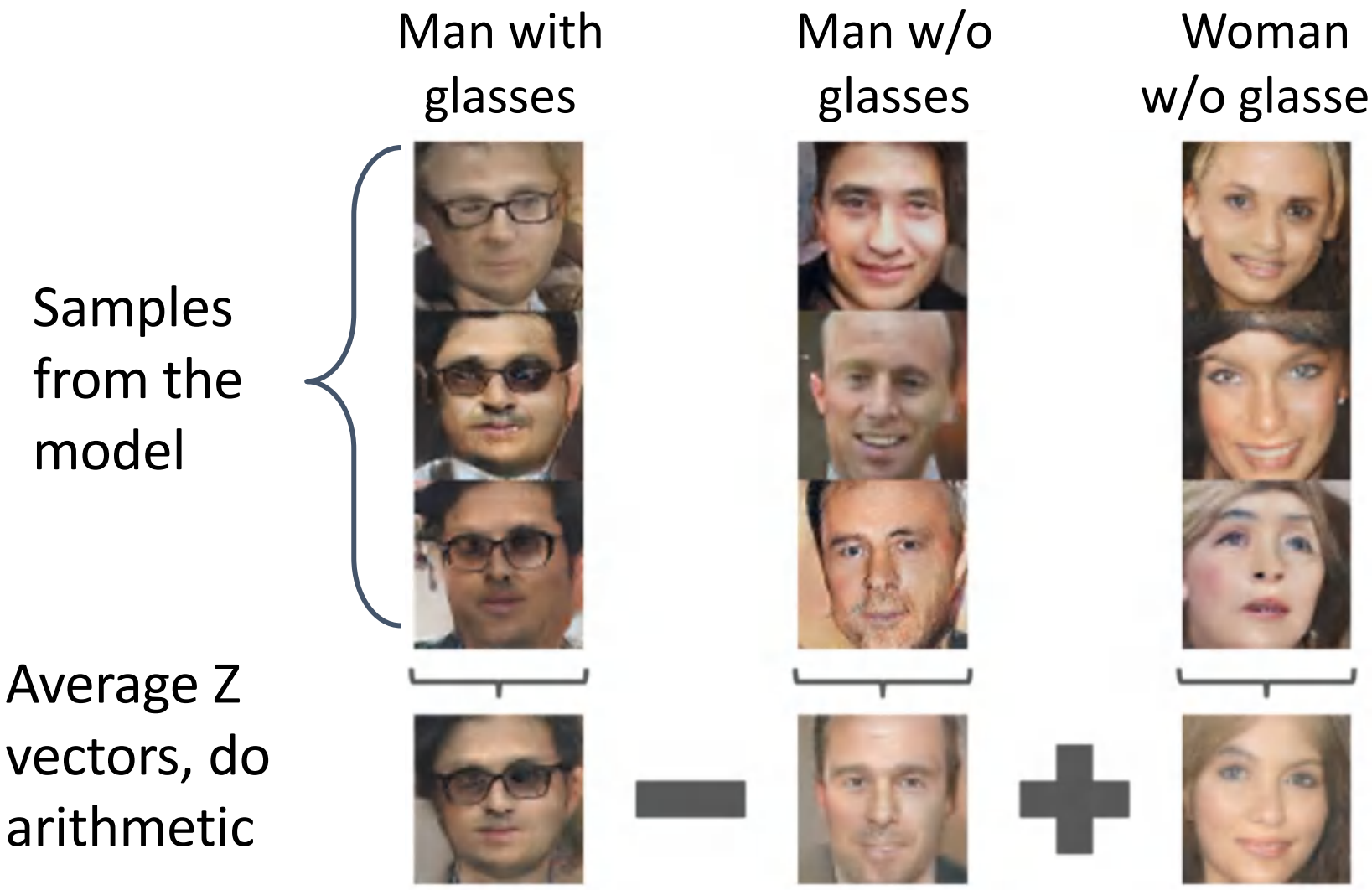
Radford et al, ICLR 2016

Generative Adversarial Networks: Vector Math



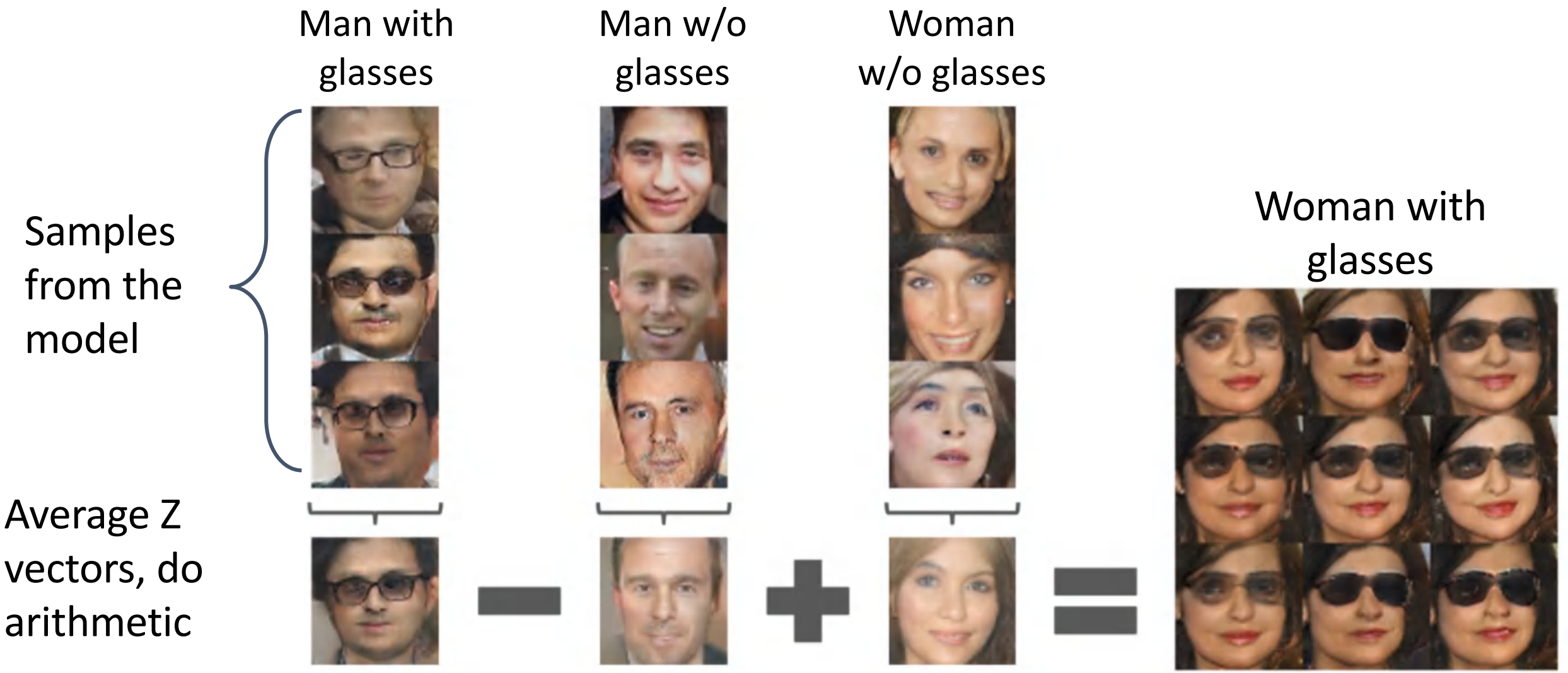
Radford et al, ICLR 2016

Generative Adversarial Networks: Vector Math



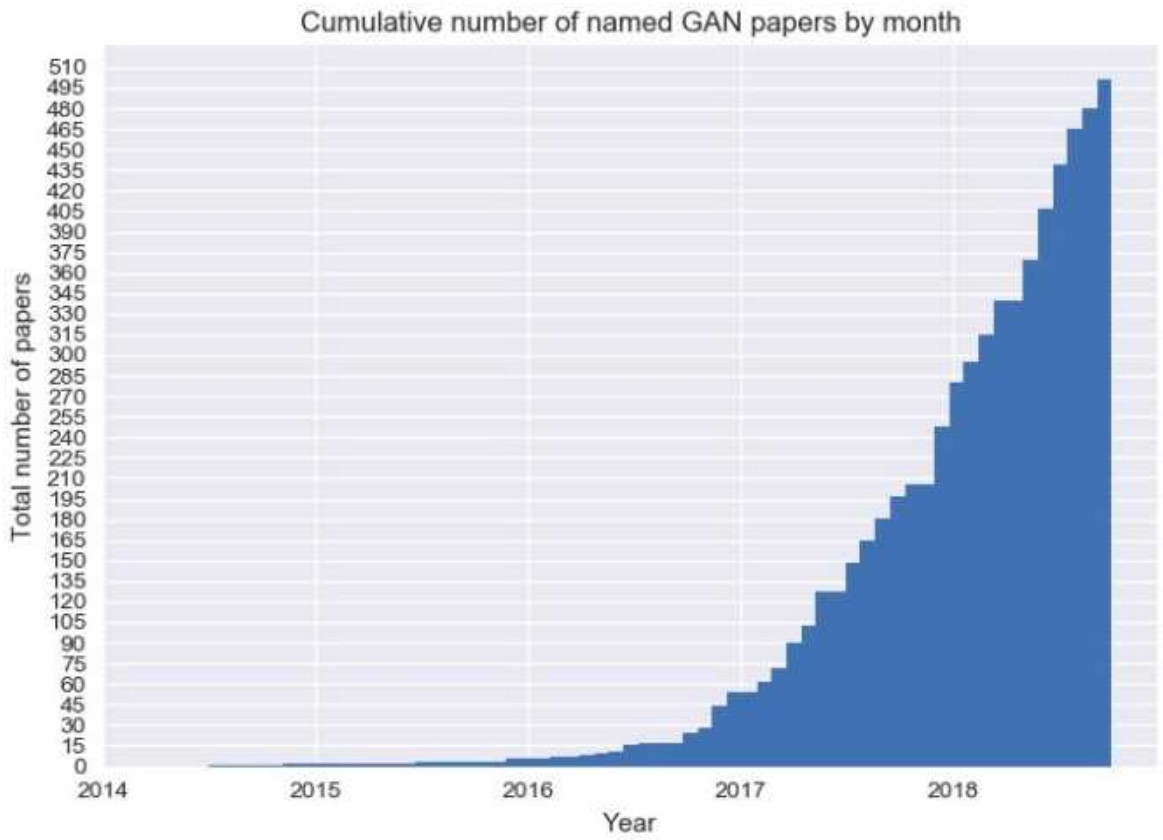
Radford et al, ICLR 2016

Generative Adversarial Networks: Vector Math



Radford et al, ICLR 2016

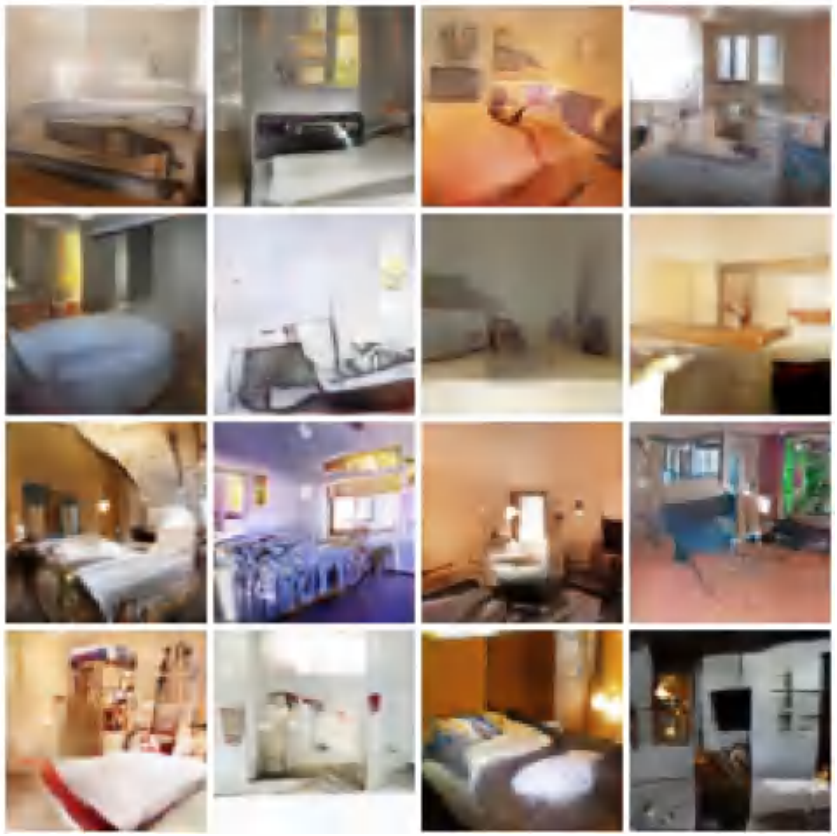
2017 to present: Explosion of GANs



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

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: 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

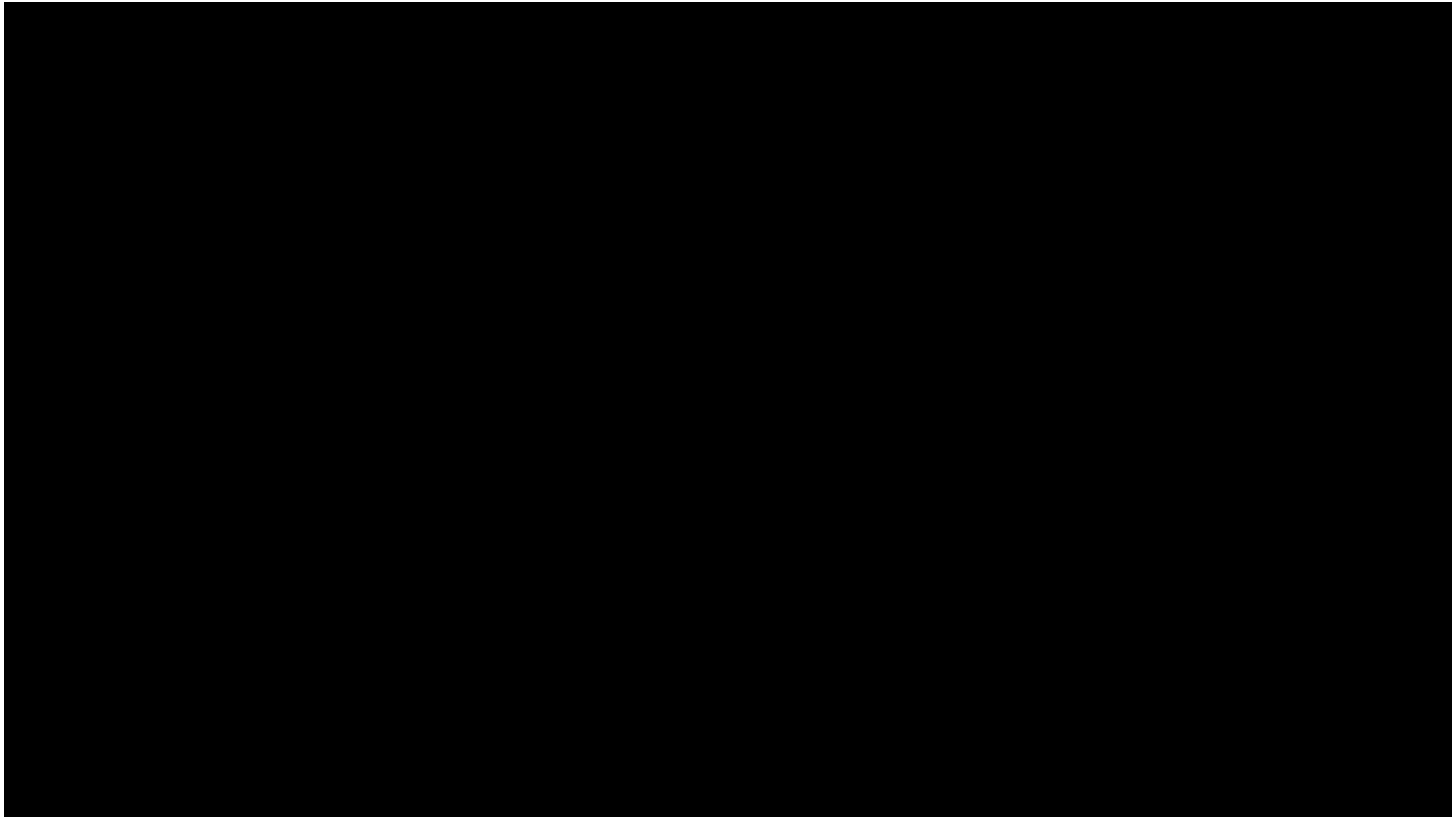
GAN Improvements: Higher Resolution

512 x 384 cars

1024 x 1024 faces

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

Images are licensed under CC BY-NC 4.0



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

Video is licensed under CC BY-NC 4.0.
Source: https://drive.google.com/drive/folders/1NFO7_vH0t98J13ckJYFd7kuaTkyeRJ86

StyleGAN2

Karras et al, “Analyzing and Improving the Image Quality of StyleGAN”, CVPR 2020

Conditional GANs

Recall: Conditional Generative Models learn p(x|y) instead of p(x)
Make generator and discriminator both take label y as an additional input!

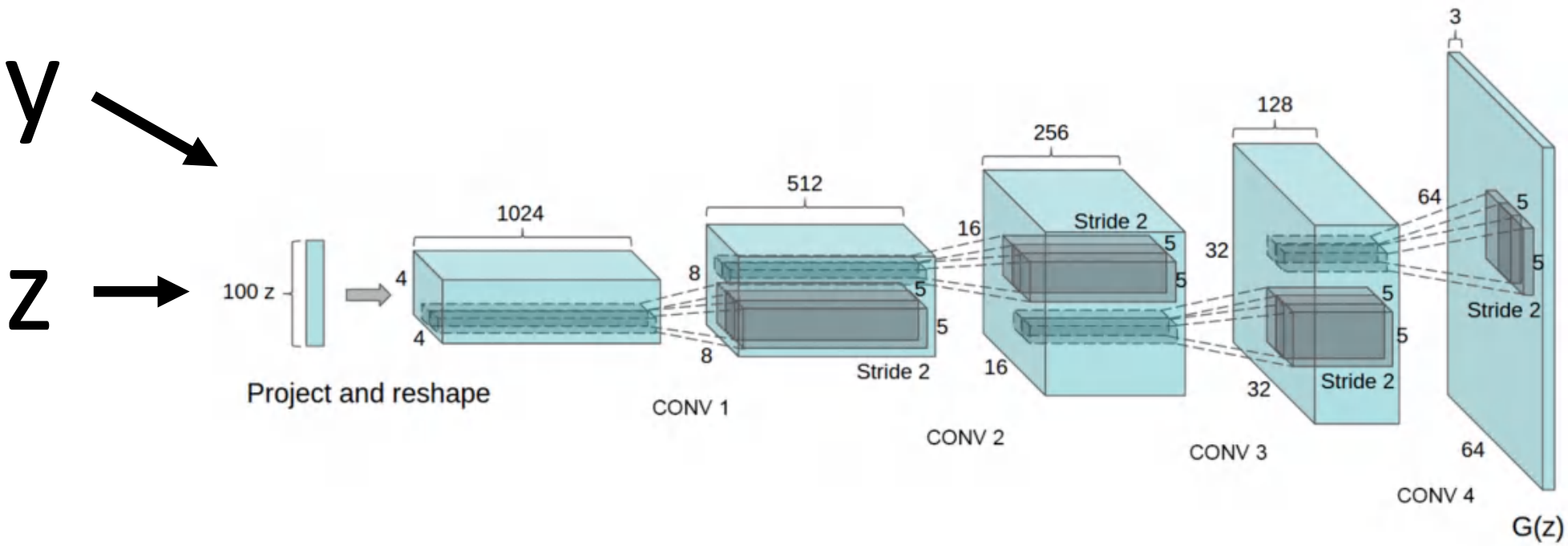


Figure credit: Radford et al, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", ICLR 2016

Conditional GANs: Conditional Batch Normalization

Batch Normalization

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

Learn a separate scale and shift for each different label y

Conditional Batch Normalization

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

$$y_{i,j} = \boldsymbol{\gamma_j^y} \hat{x}_{i,j} + \boldsymbol{\beta_j^y}$$

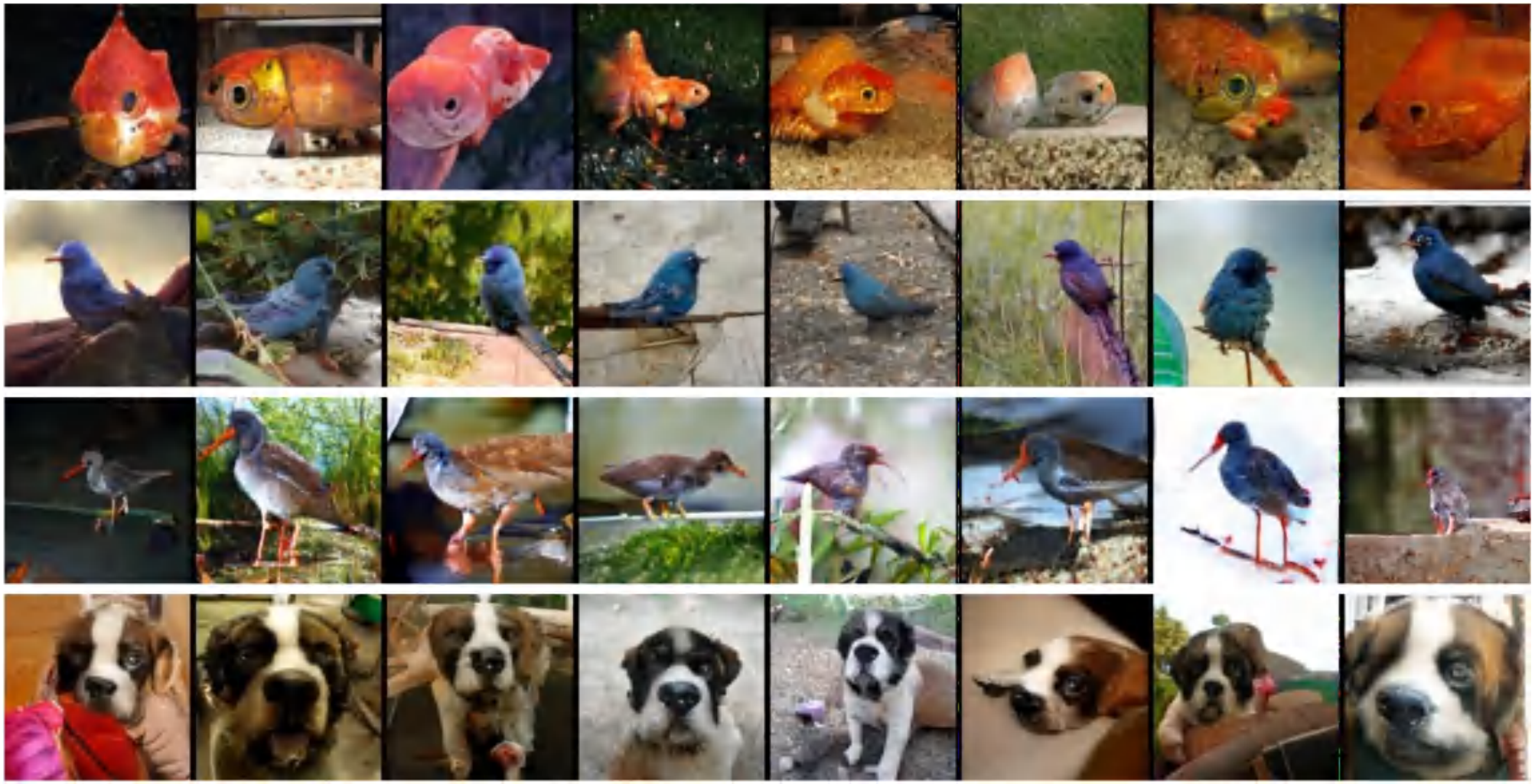
Dumoulin et al, "A learned representation for artistic style", ICLR 2017

Conditional GANs: Spectral Normalization

Miyato et al, "Spectral Normalization for Generative Adversarial Networks", ICLR 2018

128x128 images on ImageNet

Conditional GANs: Self-Attention



Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2019

128x128 images on ImageNet

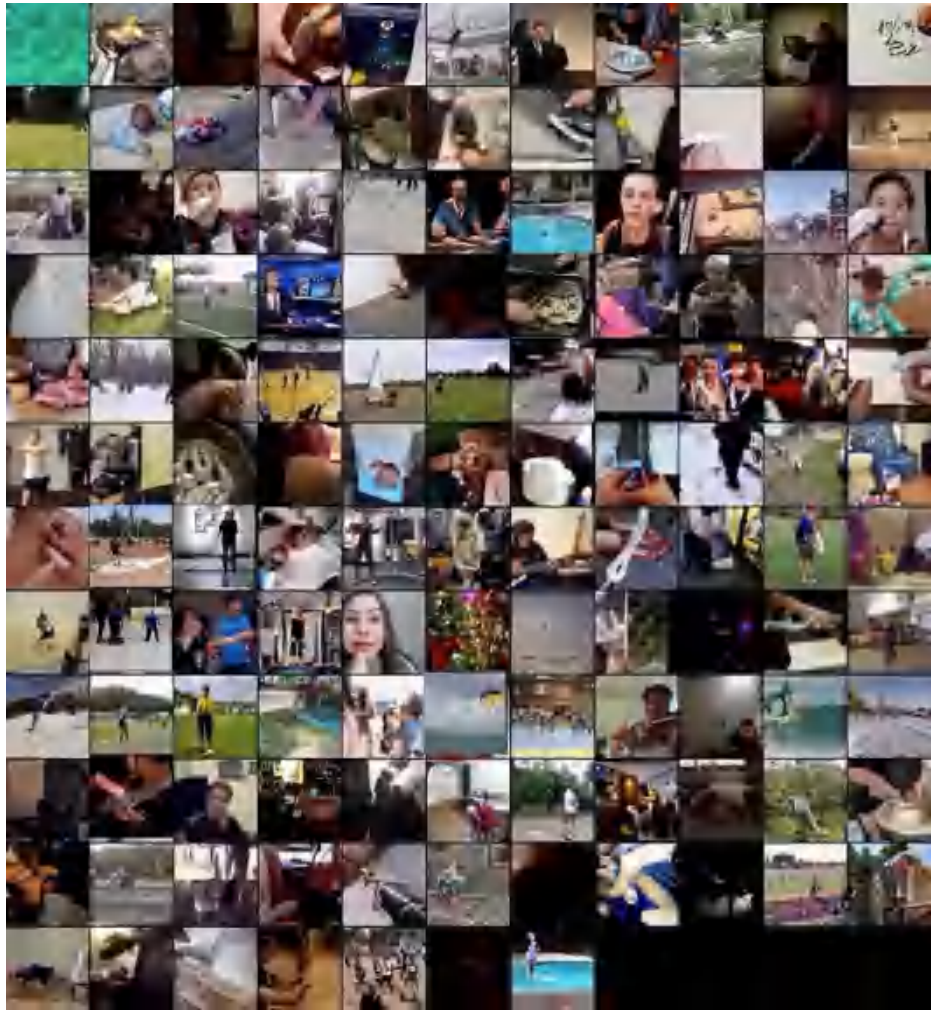
Conditional GANs: BigGAN

Brock et al, “Large Scale GAN Training for High Fidelity Natural Image Synthesis”, ICLR 2019

512x512 images on ImageNet

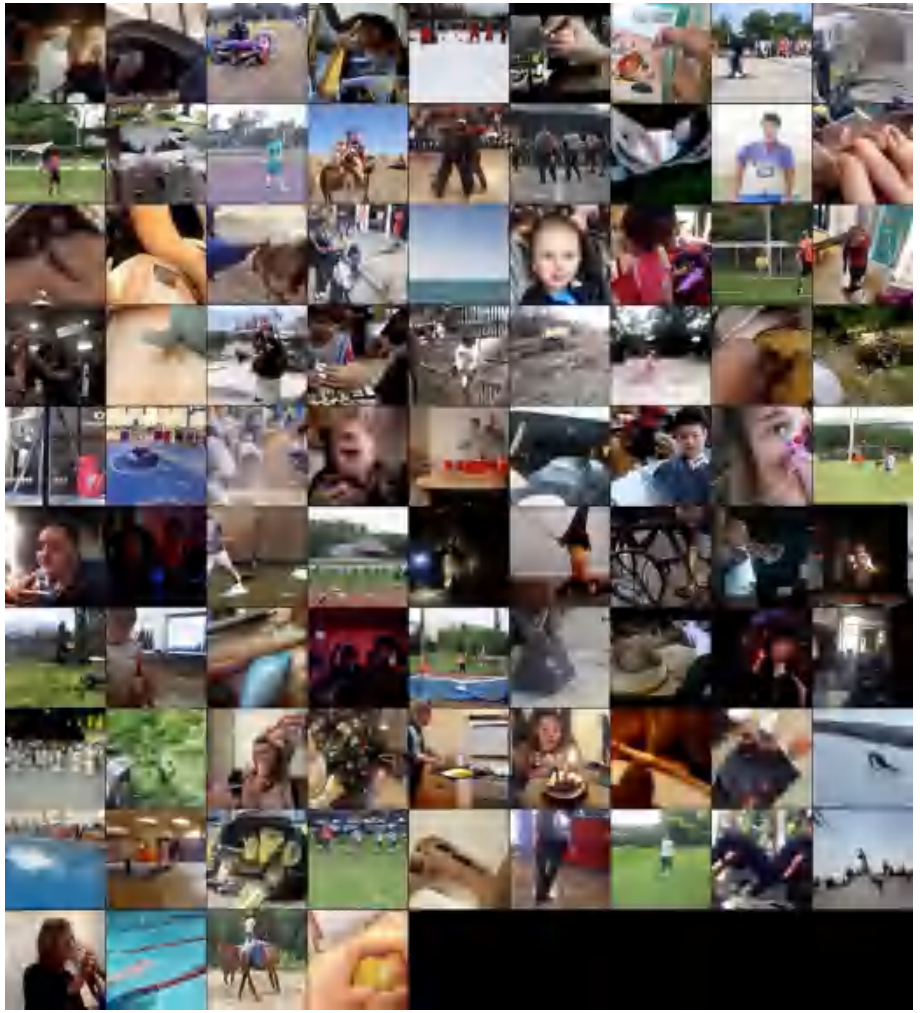
Generating Videos with GANs

Clark et al, "Adversarial Video Generation on Complex Datasets", arXiv 2019



64x64 images, 48 frames

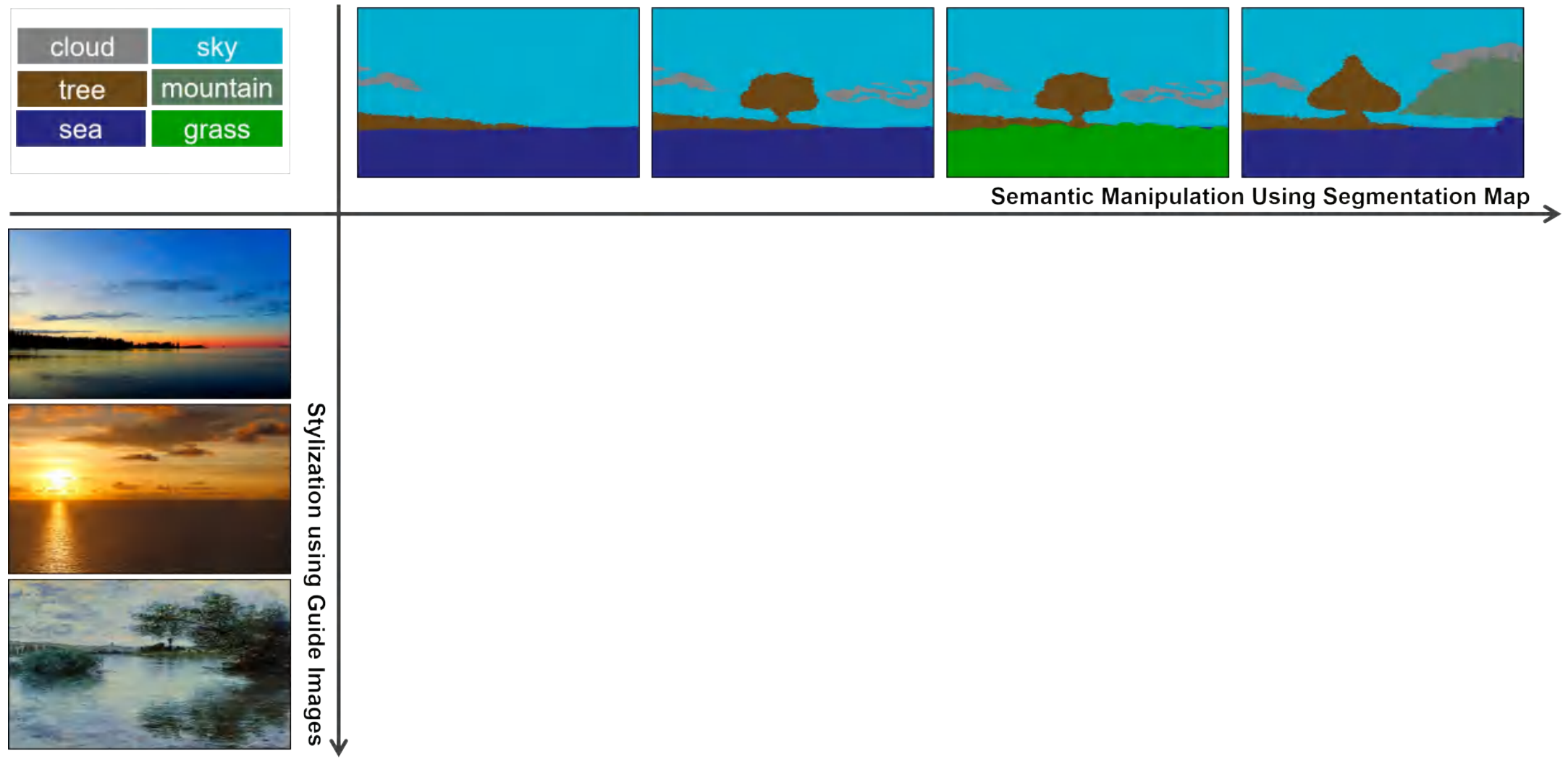
https://drive.google.com/file/d/1FjOQYdUuxPXvS8yeOhXdPQMapUQaklLi/view



128x128 images, 12 frames

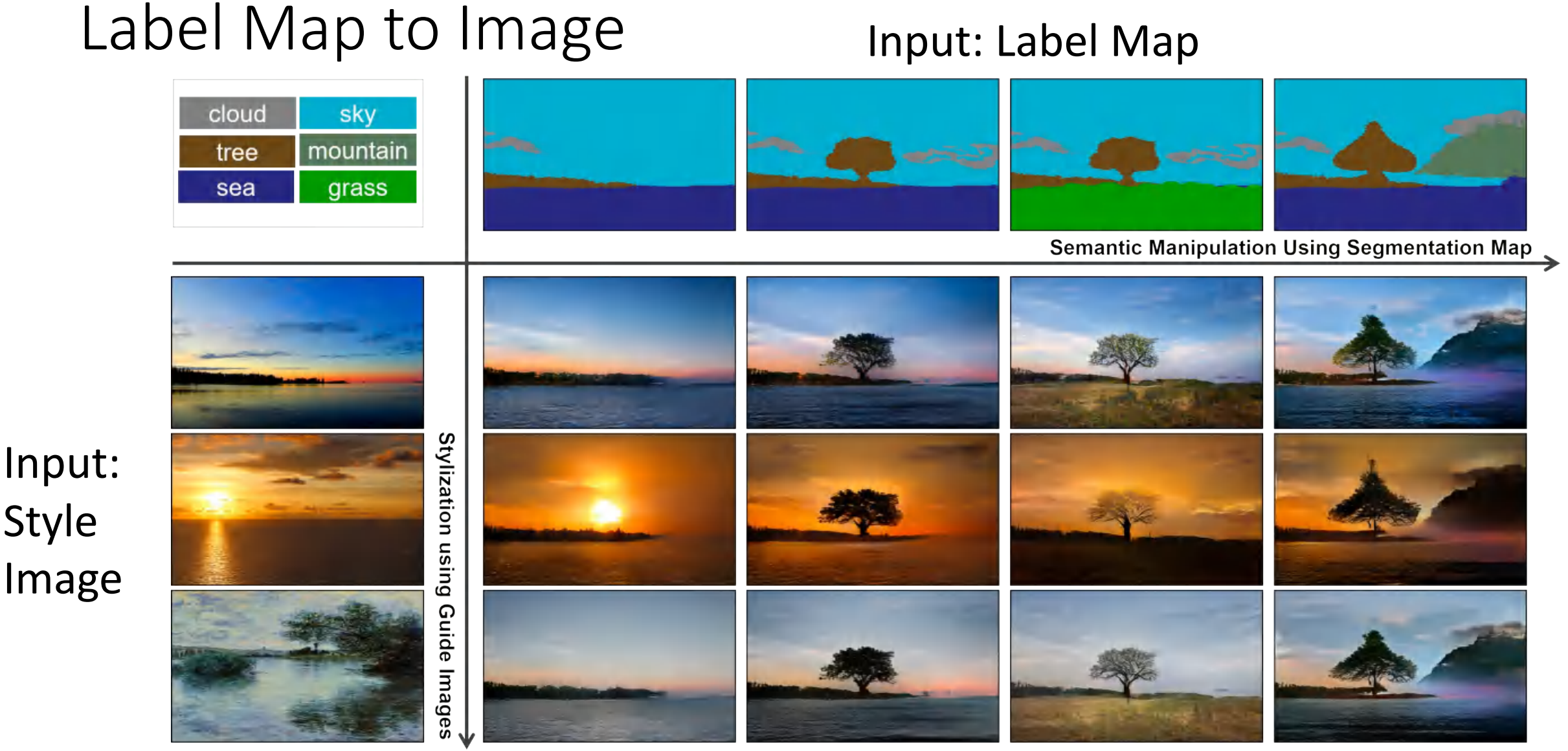
https://drive.google.com/file/d/165Yxuvvu3viOy-39LhhSDGtczbWphj_i/view

Label Map to Image



Park et al, "Semantic Image Synthesis with Spatially-Adaptive Normalization", CVPR 2019

Label Map to Image



Park et al, "Semantic Image Synthesis with Spatially-Adaptive Normalization", CVPR 2019

Conditioning on more than labels! Text to Image

Zhang et al, "StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks.", TPAMI 2018
Zhang et al, "StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks.", ICCV 2017
Reed et al, "Generative Adversarial Text-to-Image Synthesis", ICML 2016

Text to Image: DALL-E

Step 1: Train VQ-VAE (discrete grid of latent codes)

Step 2: Train autoregressive Transformer model to predict sequence of latent codes (Giant model on 250M image/text pairs)

Step 3: Given text prompt, sample new image codes; pass through VQ-VAE decoder to generate images

Ramesh et al, “Zero-Shot Text-to-Image Generation”, ICML 2021

Text to Image: DALL-E

Step 1: Train VQ-VAE (discrete grid of latent codes)

Step 2: Train autoregressive Transformer model to predict sequence of latent codes (Giant model on 250M image/text pairs)

Step 3: Given text prompt, sample new image codes; pass through VQ-VAE decoder to generate images

an illustration of a baby hedgehog in a christmas sweater walking a dog

Ramesh et al, "Zero-Shot Text-to-Image Generation", ICML 2021

Text to Image: DALL-E

Step 1: Train VQ-VAE (discrete grid of latent codes)

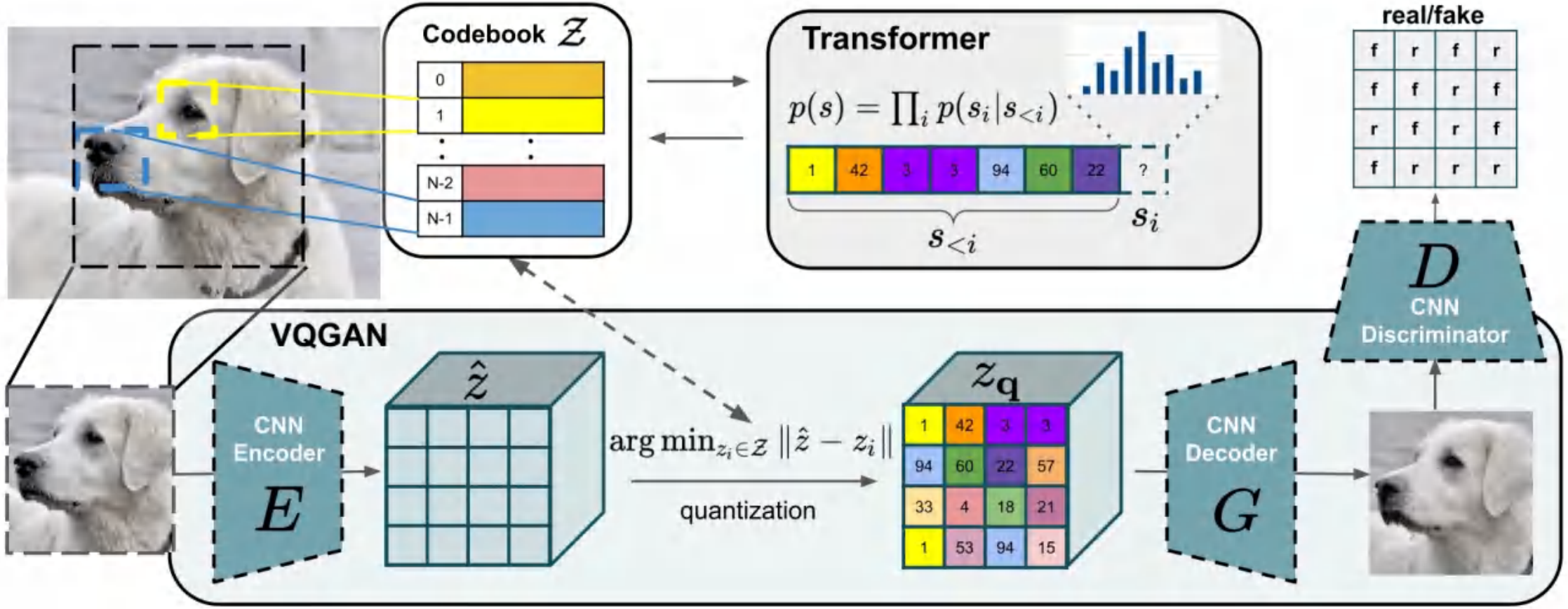
Step 2: Train autoregressive Transformer model to predict sequence of latent codes (Giant model on 250M image/text pairs)

Step 3: Given text prompt, sample new image codes; pass through VQ-VAE decoder to generate images

a neon sign that reads "backprop". a neon sign that reads "backprop". backprop neon sign

Ramesh et al, "Zero-Shot Text-to-Image Generation", ICML 2021

VQ-GAN



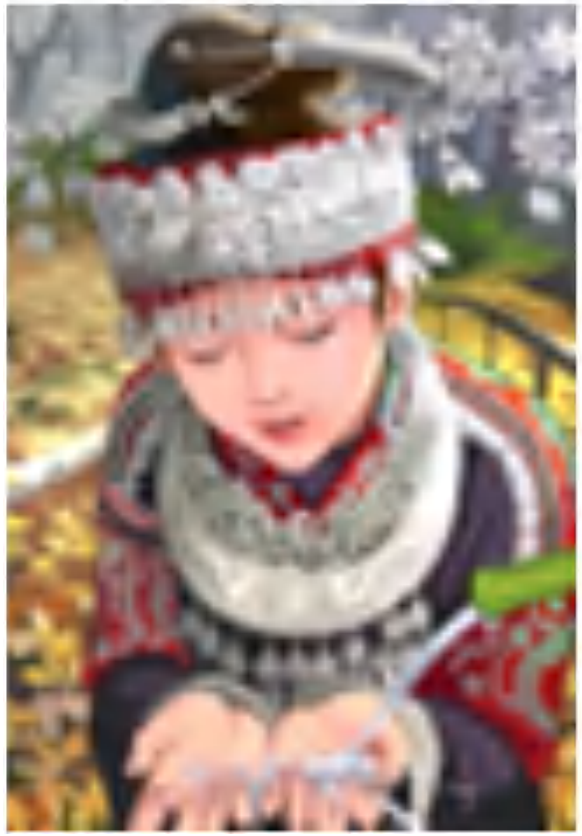
Esser et al, "Taming Transformers for High-Resolution Image Synthesis", CVPR 2021

VQ-GAN (Semantic Segmentation to Image)

Esser et al, “Taming Transformers for High-Resolution Image Synthesis”, CVPR 2021

Image Super-Resolution: Low-Res to High-Res

bicubic
(21.59dB/0.6423)



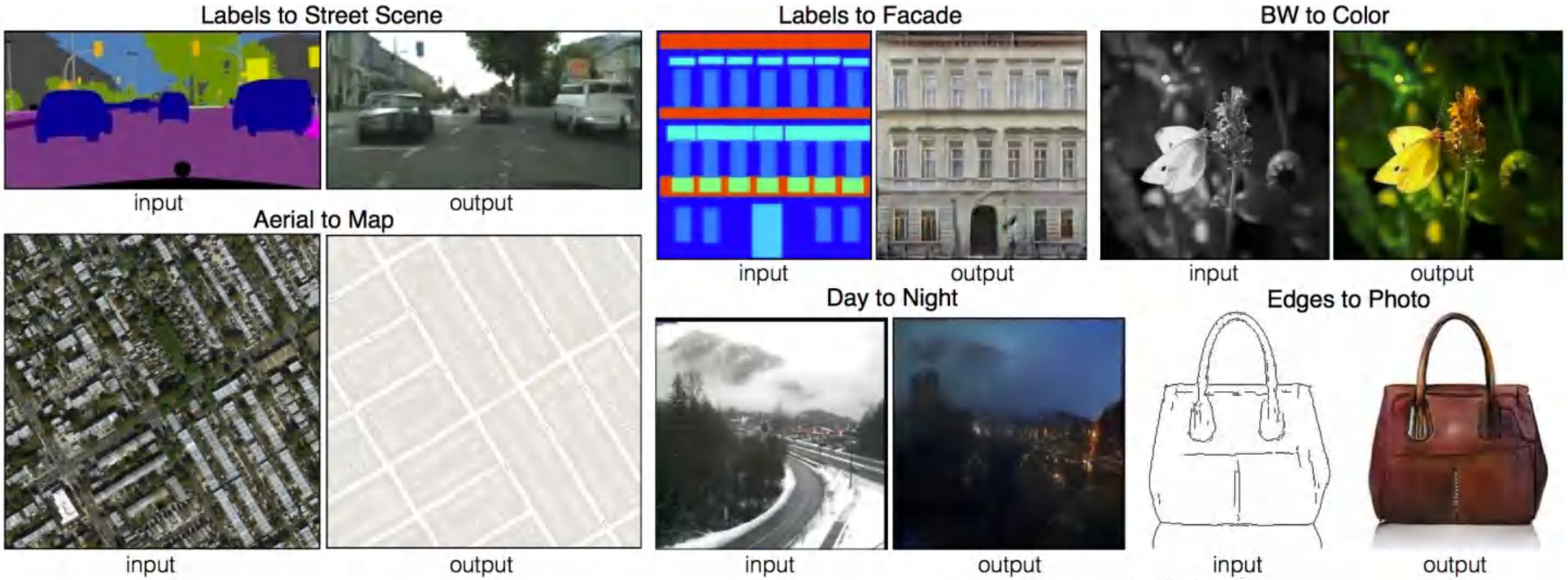
SRResNet
(23.53dB/0.7832)

SRGAN
(21.15dB/0.6868)

original

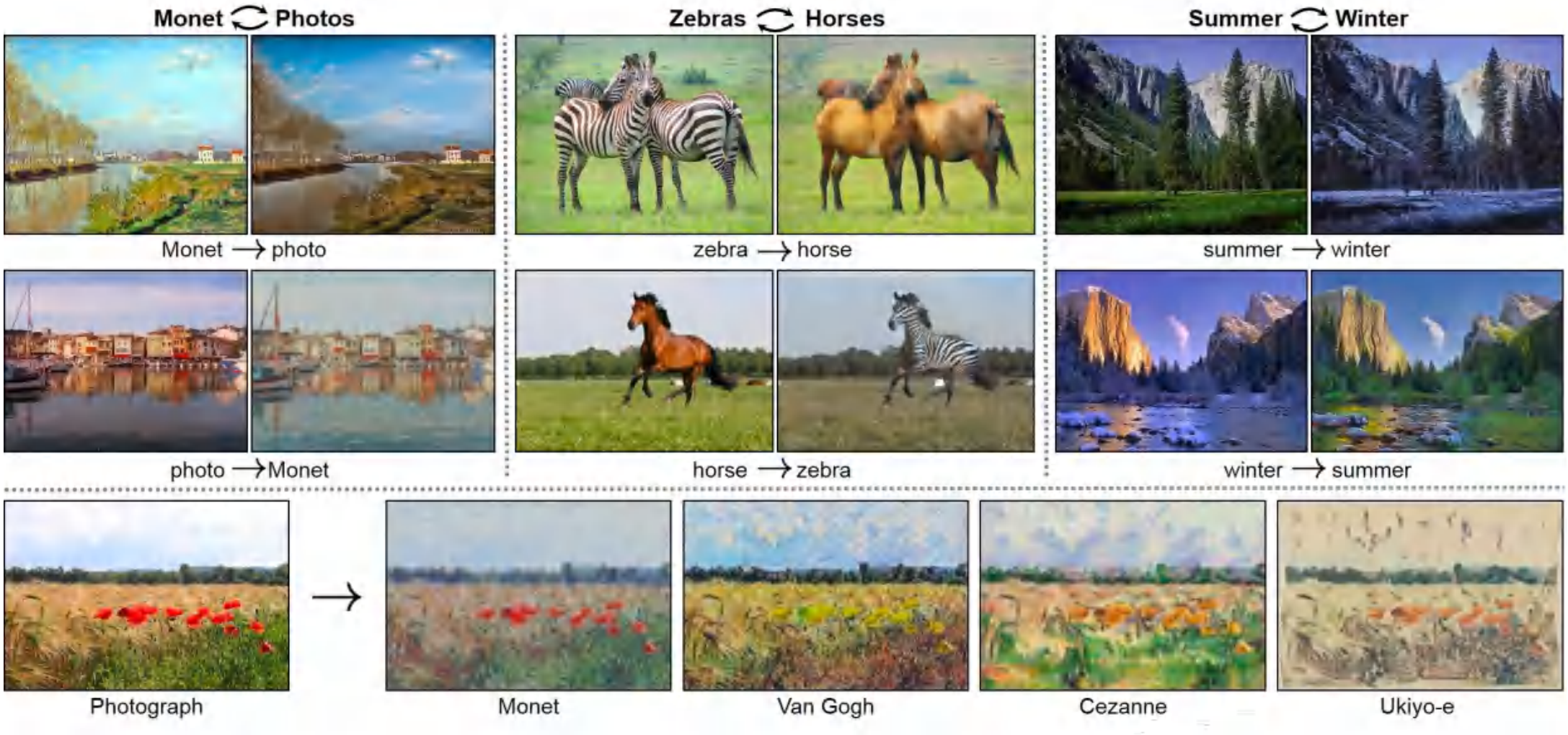
Ledig et al, “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network”, CVPR 2017

Image-to-Image Translation: Pix2Pix



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

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

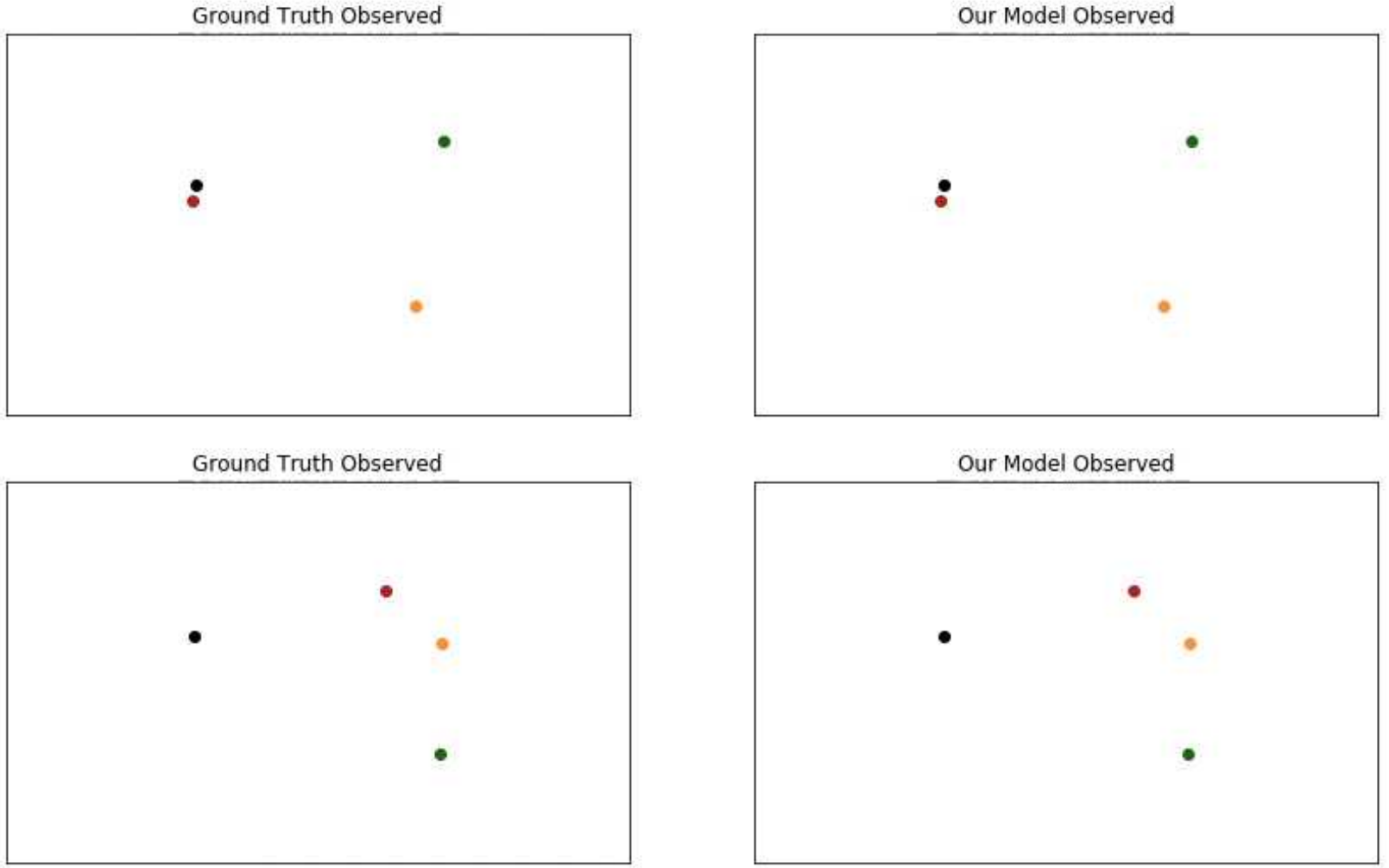
Input Video: Horse

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

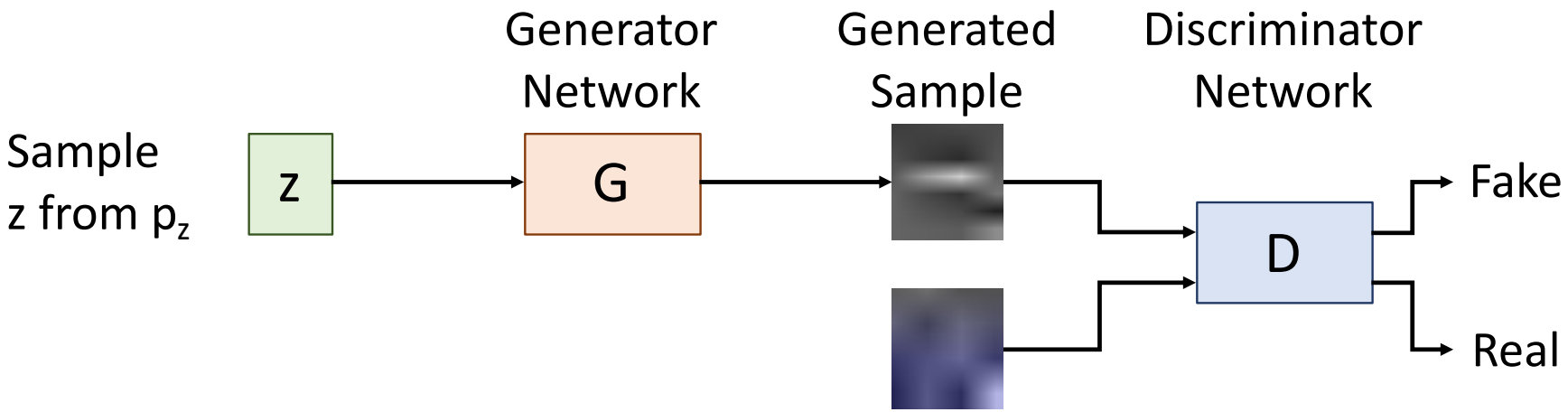
GANs: Not just for images! Trajectory Prediction



Gupta, **Johnson**, Li, Savarese, Alahi, "Social GAN: Socially Acceptable Trajectories with Generative Adversarial Networks", CVPR 2018

GAN Summary

Jointly train two networks:
Discriminator: Classify data as real or fake
Generator: Generate data that fools the discriminator



Under some assumptions, generator converges to true data distribution
Many applications! Very active area of research!

Taxonomy of Generative Models

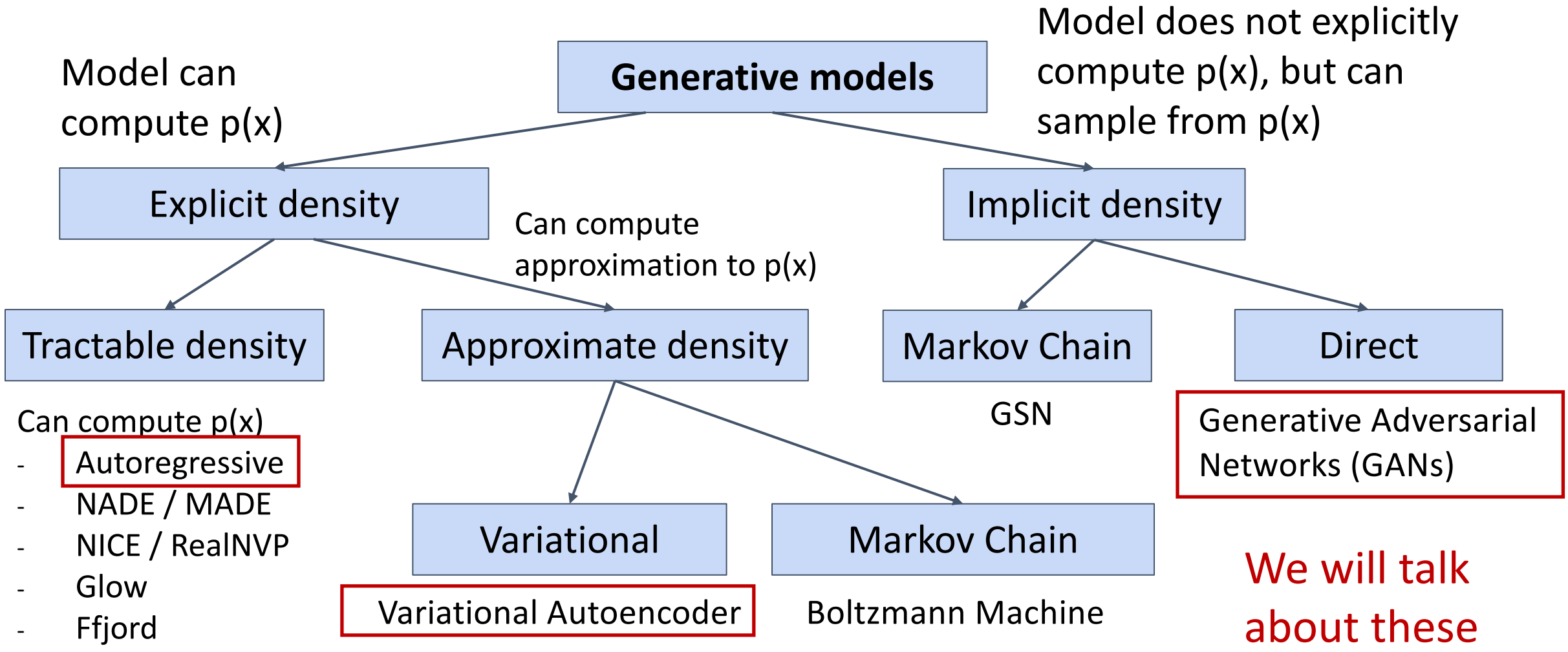


Figure adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

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

Next Time: Visualizing Models and Generating Images