22. Generative Models

STA3142 Statistical Machine Learning

Kibok Lee

Assistant Professor of

Applied Statistics / Statistics and Data Science

Jun 4, 2024

** Slides adapted from EECS498/598 @ Univ. of Michigan by Justin Johnson*

연세대학교
YONSEI UNIVERSITY

Rest of the Course Schedule

- **6/2 Tue**: 22. Generative Models
- **6/6 Thu**: 23. Recurrent Neural Networks (We have a class!)
- **6/9 Tue**: 24. Transformers & 25. Reinforcement Learning
- **6/13 Thu**: 26. ML Advice
- **6/14 Fri**: Final Assignment Deadline

Assignment 5 (Final Exam Replacement)

- Due **Friday 6/14, 11:59pm**
- Topic: Convolutional Neural Networks
  - Derive gradients for NN layers
  - Implement layers for CNNs
  - Train a CNN classifier for MNIST digit recognition
- Please read the instruction carefully!
  - Submit one pdf and one zip file separately
  - Write your code only in the designated spaces
  - Do not import additional libraries
  - …
- If you feel difficult, consider to take option 2.

Overview

- What Are Generative Models?
- Variational Autoencoders
- Diffusion Models
- Generative Adversarial Networks

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, image captioning, etc.

Classification:
Predict discrete categories

Cat

Supervised vs. Unsupervised Learning

Supervised Learning

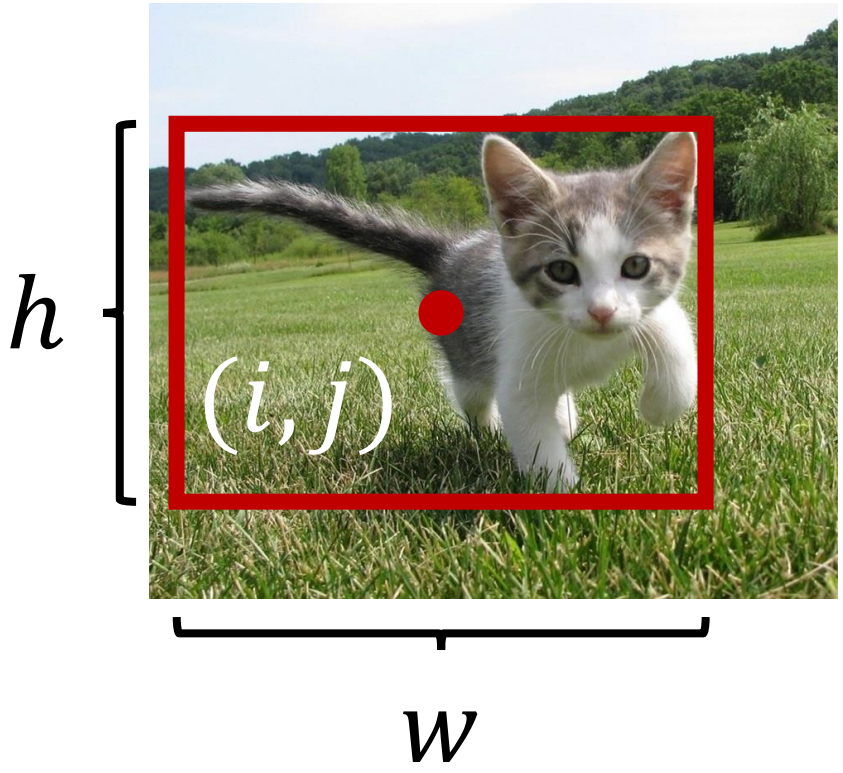
Data: (x, y)

x is data, y is label

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

Examples: Classification, regression, image captioning, etc.

Regression:
Predict continuous values



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, image captioning, etc.

Image captioning:
Predict a sequence of words

A cat sitting on a suitcase on the floor

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, image captioning, etc.

Unsupervised Learning

Data: x

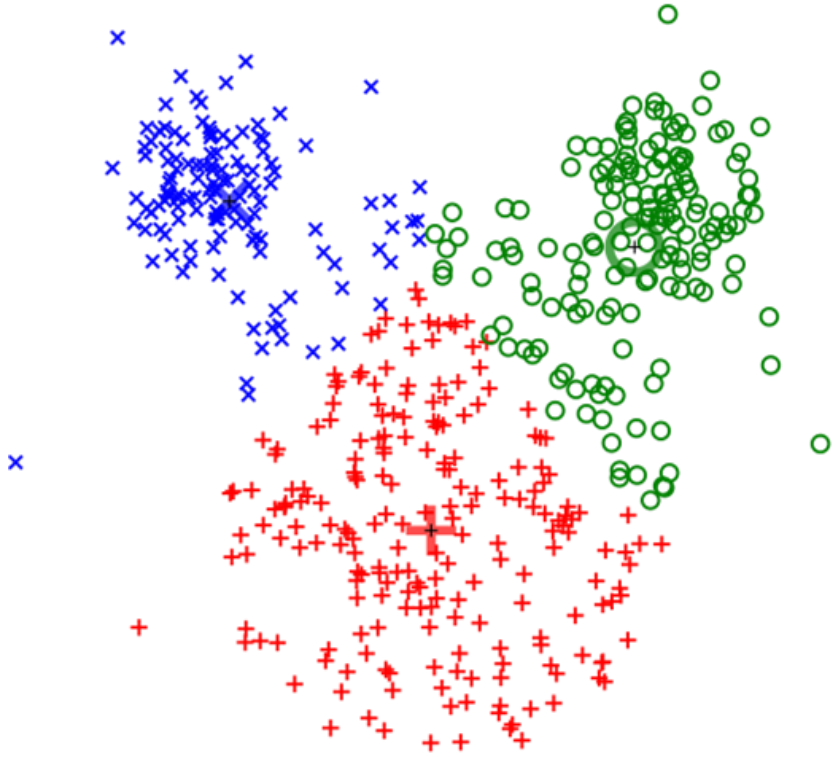
Just data, no labels!

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

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

Supervised vs. Unsupervised Learning

Clustering
(e.g., K-Means)



Unsupervised Learning

Data: x

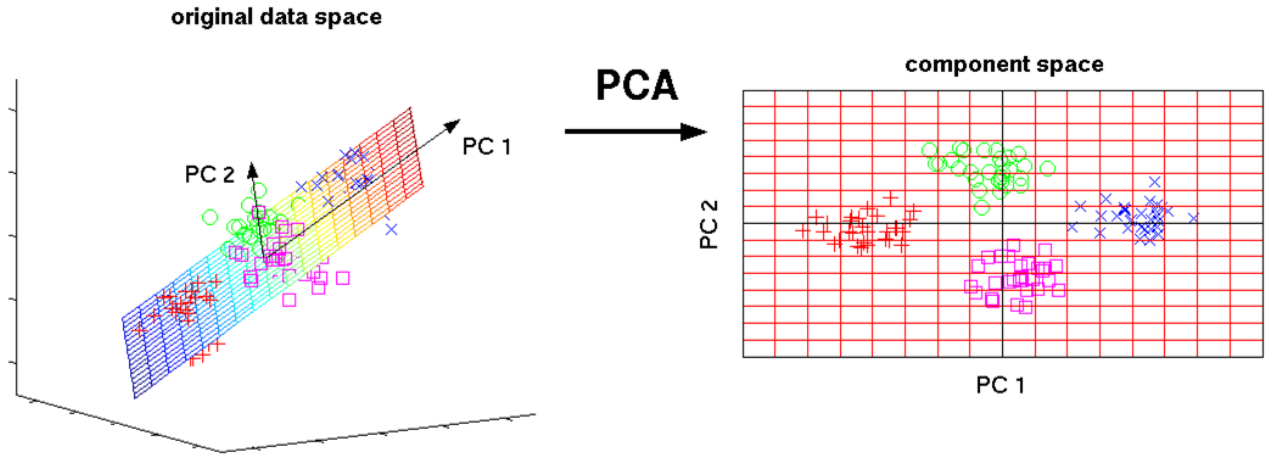
Just data, no labels!

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

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

Supervised vs. Unsupervised Learning

Dimensionality Reduction
(e.g., Principal Components Analysis)



3D → 2D

Unsupervised Learning

Data: x

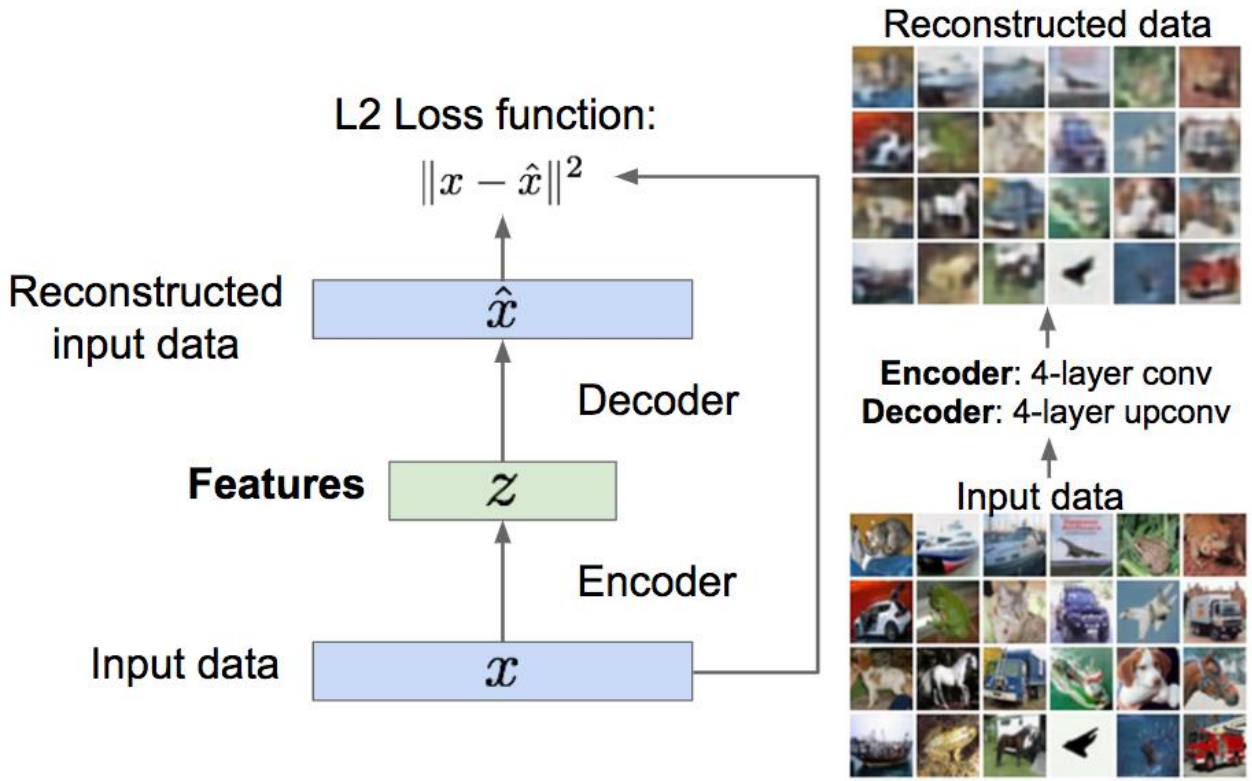
Just data, no labels!

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

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

Supervised vs. Unsupervised Learning

Feature Learning (e.g., autoencoders)



Unsupervised Learning

Data: x

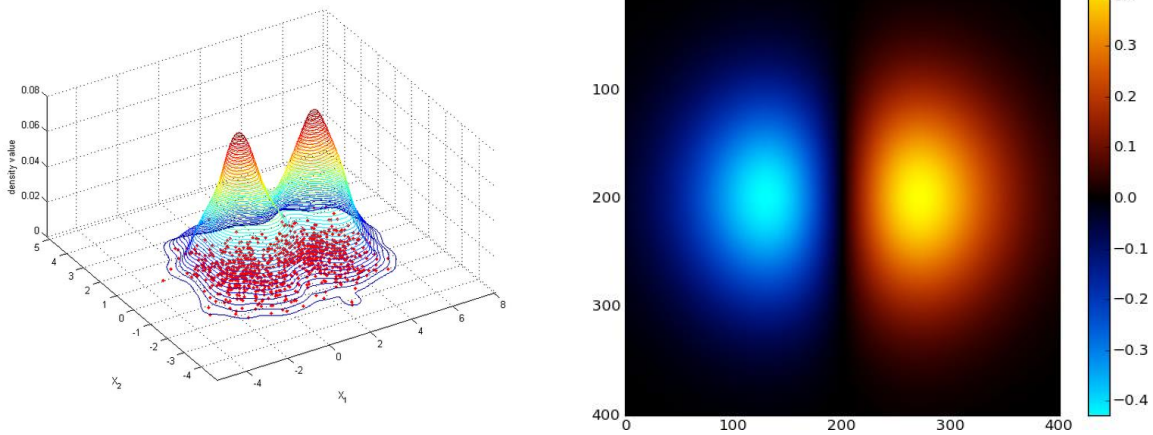
Just data, no labels!

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

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

Supervised vs. Unsupervised Learning

Density Estimation



Images left and right are CC0 public domain

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.

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, 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.

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)

Data: x

Label: y

Cat

Probability Recap:

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

Density functions are **normalized**:

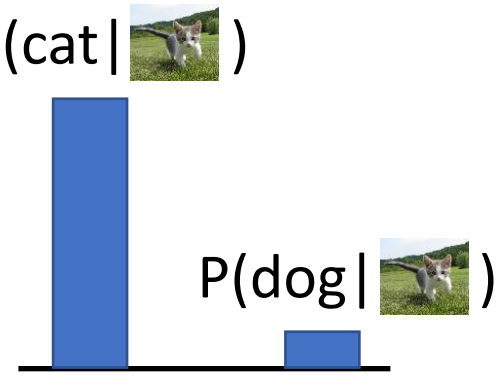
$$\int_X p(x)dx = 1$$

Different values of x **compete** for density

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

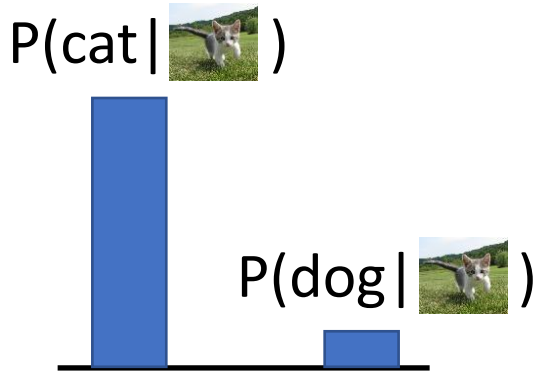
Density functions are **normalized**:

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

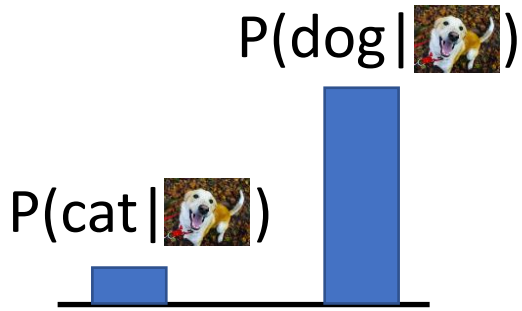
Different values of x **compete** for density

Discriminative vs. Generative Models

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



Generative Model: Learn a probability distribution p(x)

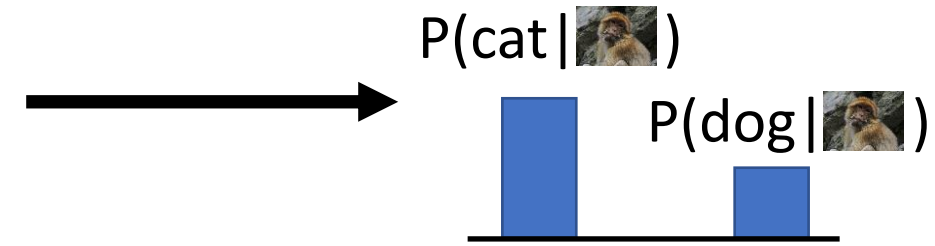


Conditional Generative Model: Learn p(x|y)

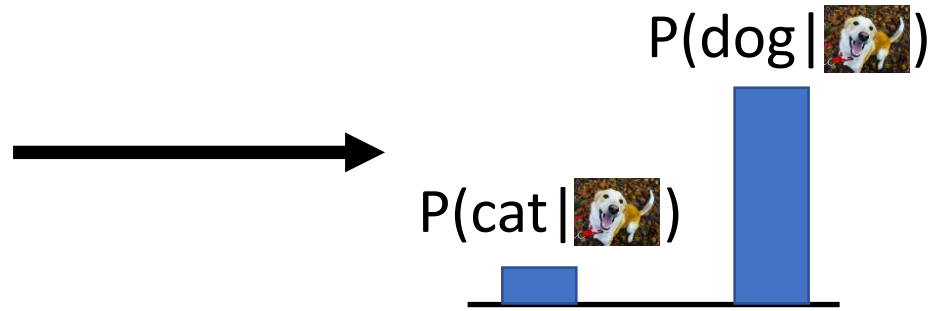
Discriminative model: the possible labels for each input *compete* for probability mass. But no competition between images

Discriminative vs. Generative Models

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



Generative Model: Learn a probability distribution p(x)



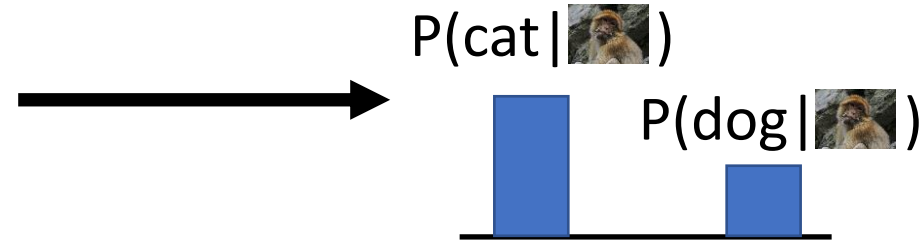
Conditional Generative Model: Learn p(x|y)

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

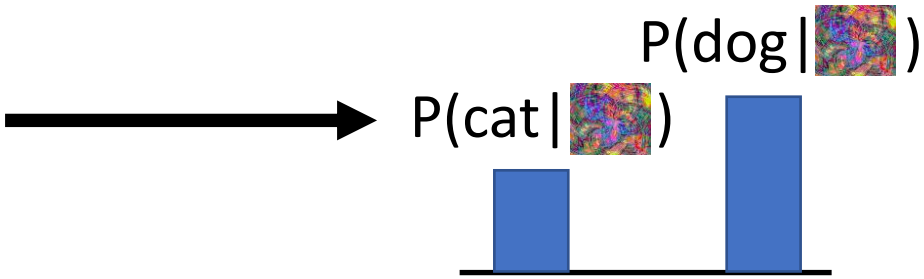
Monkey image is CC0 Public Domain

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)

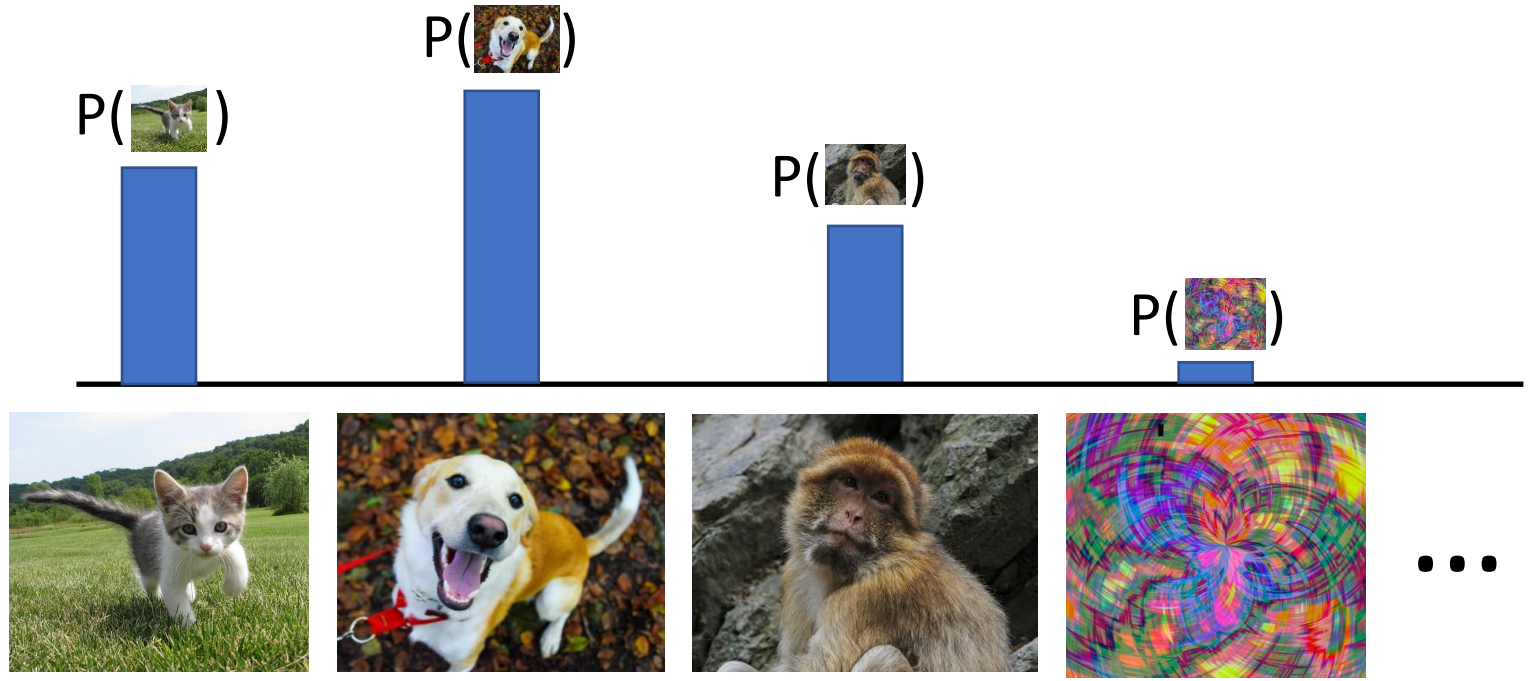
Discriminative model: No way for the model to handle unreasonable inputs; it must give label distributions for all images

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

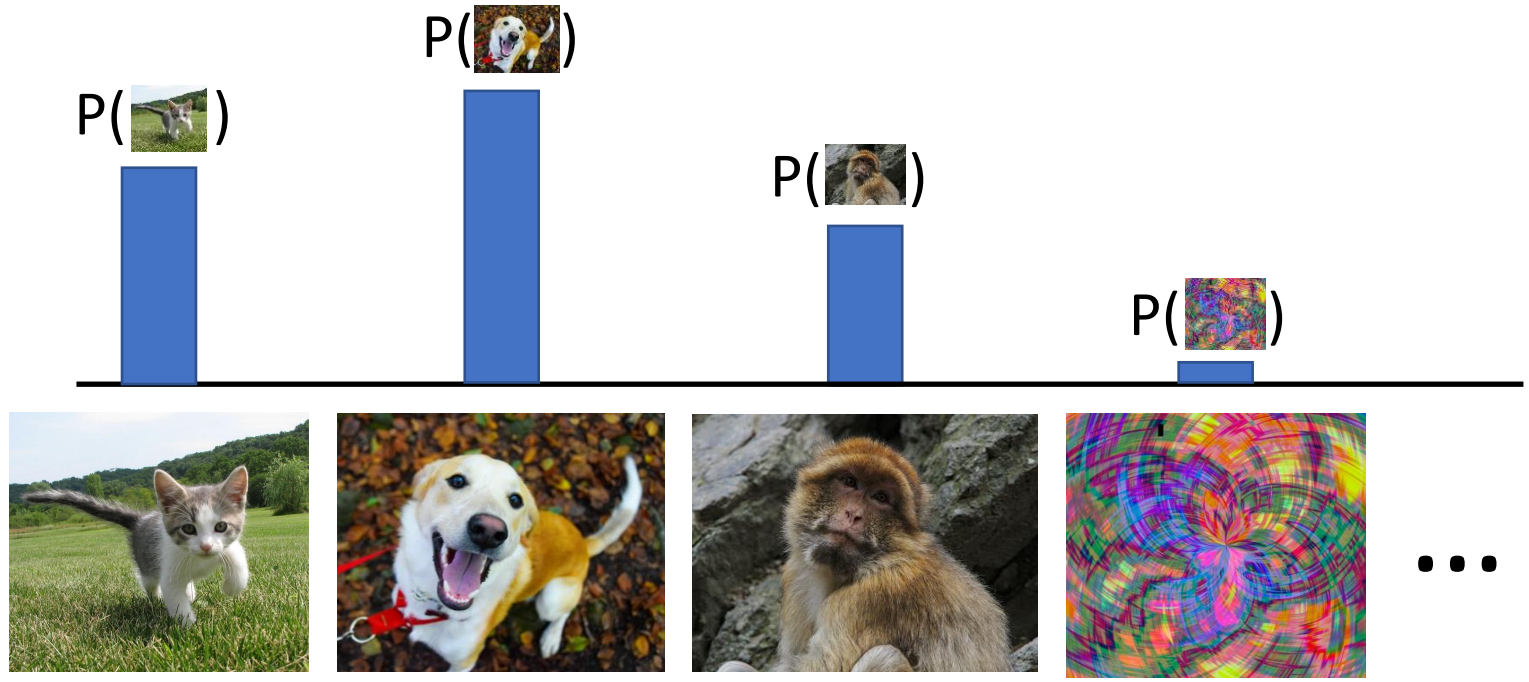
Requires deep image understanding! Is a dog more likely to sit or stand? How about 3-legged dog vs. 3-armed monkey?

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

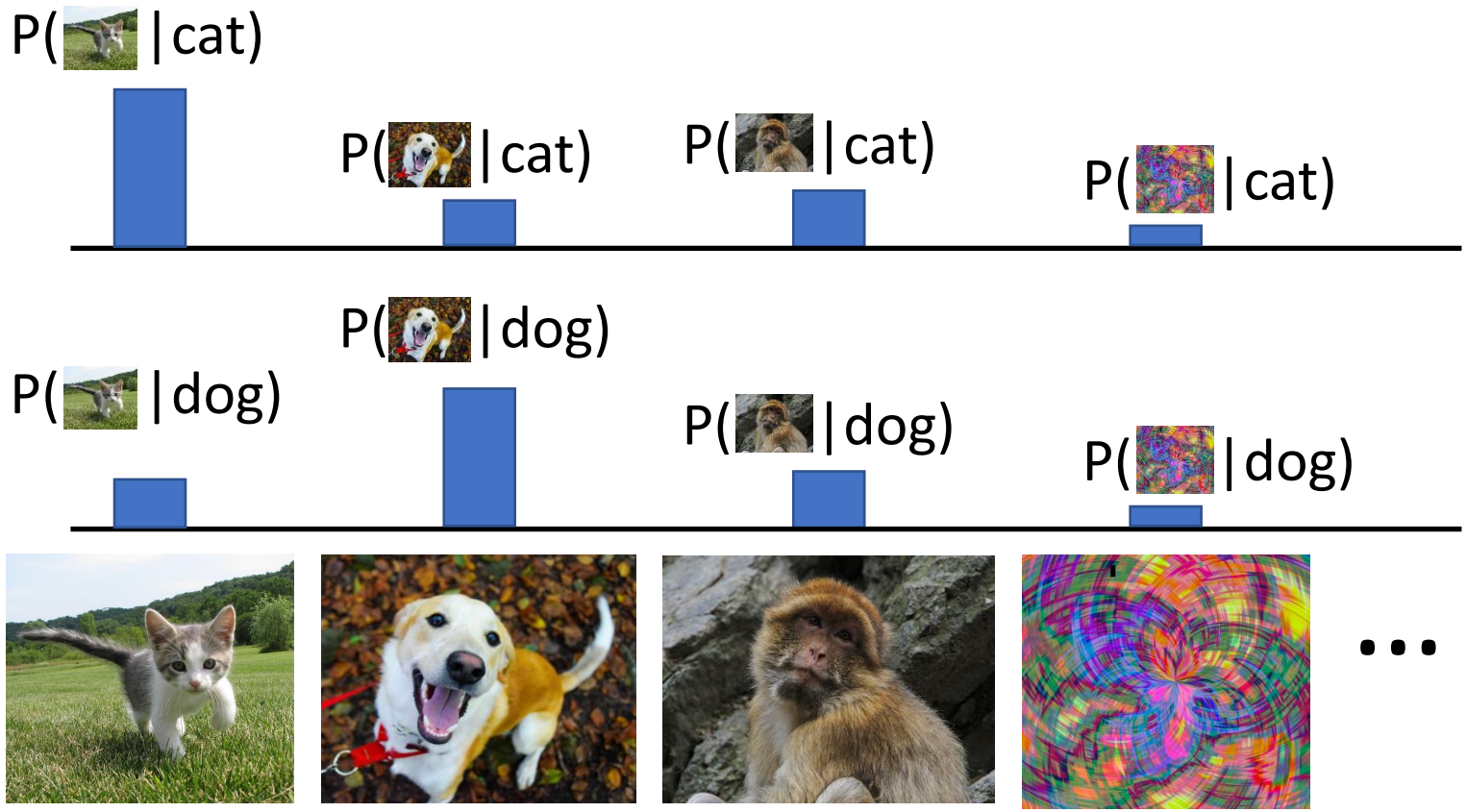
Model can *reject* unreasonable inputs by assigning them small values

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)



Conditional Generative Model: Each possible label induces a competition among all images

What can we do with a discriminative model?

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

Assign labels to data
Feature learning (with labels)

Generative Model:
Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)

What can we do with a generative model?

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

→ Assign labels to data
Feature learning (with labels)

Generative Model:
Learn a probability distribution p(x)

→ Detect outliers
Feature learning (without labels)
Sample to **generate** new data

Conditional Generative Model: Learn p(x|y)

→ Assign labels, while rejecting outliers!
Generate new data conditioned on input labels

Deep Generative Models

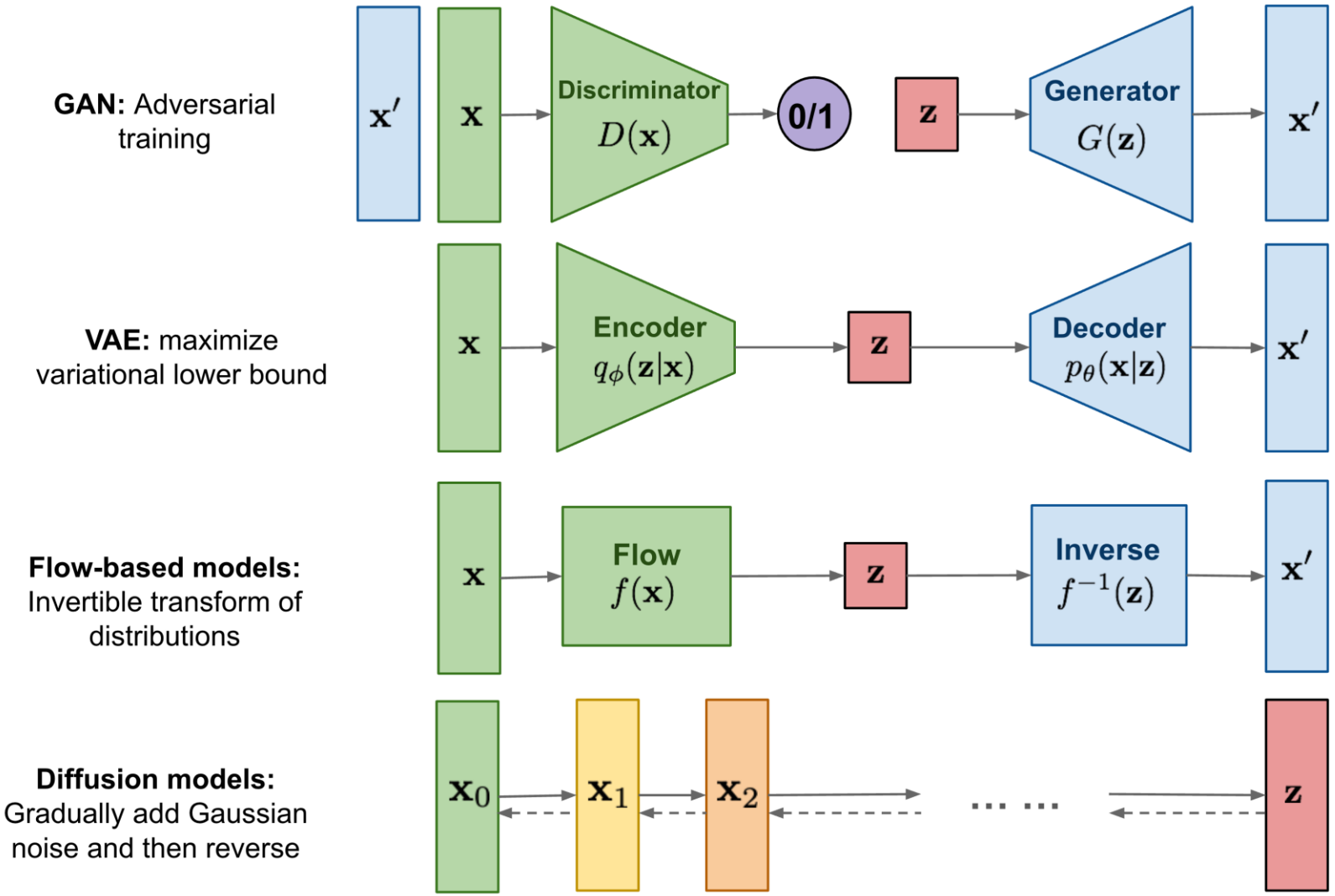


Image source: Lilian Weng, "What are Diffusion Models?", 2021

Taxonomy of Generative Models

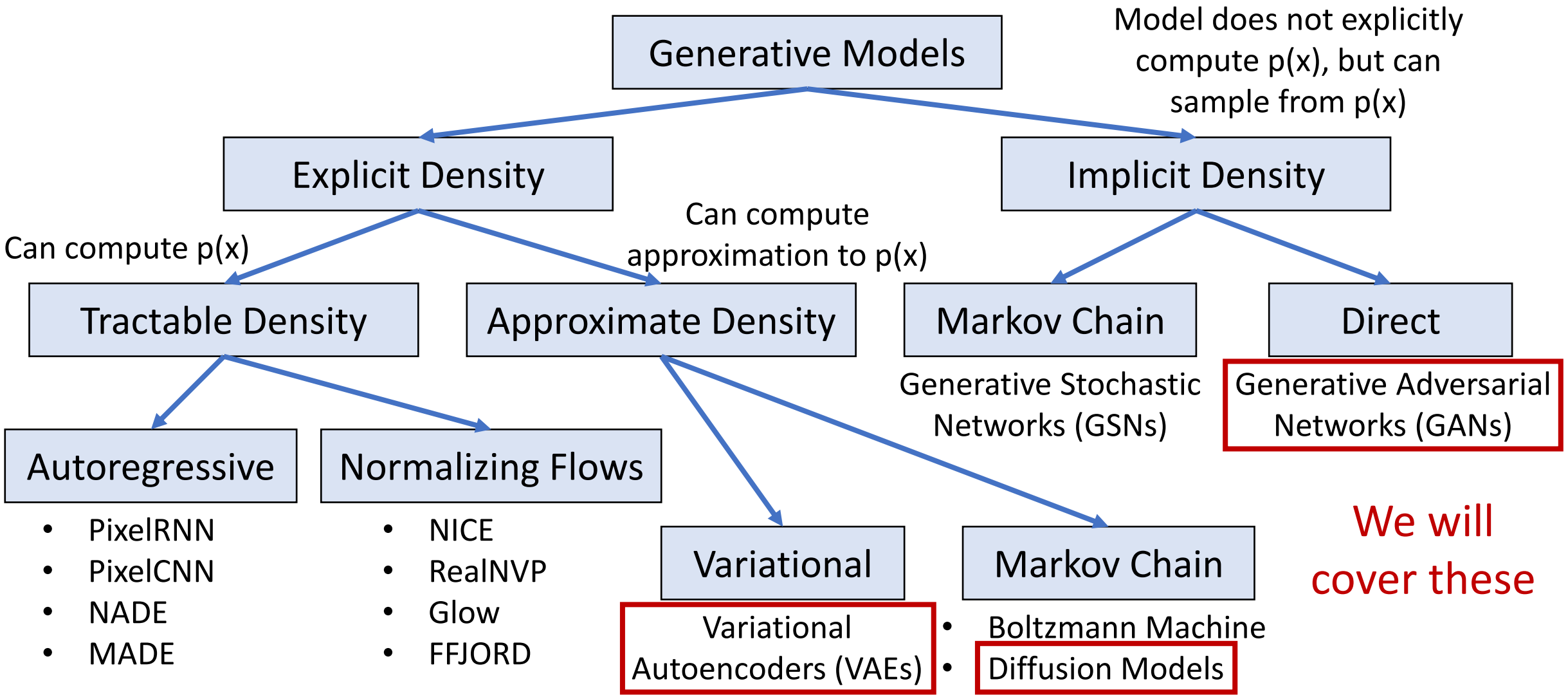


Figure adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

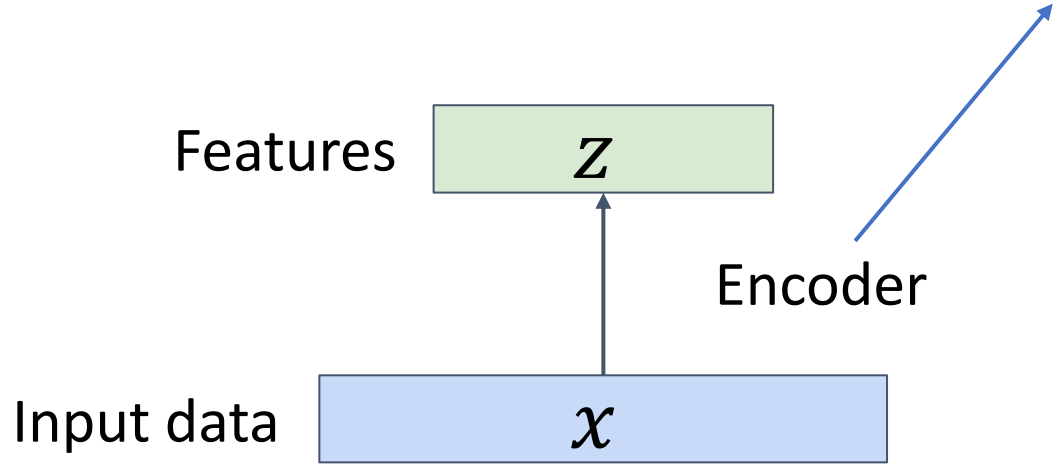
Variational Autoencoders

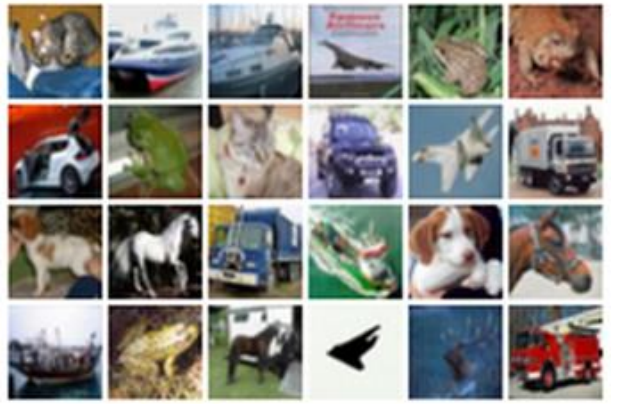
(Regular, non-variational) Autoencoders

Unsupervised method for learning feature vectors from raw data x, without any labels

Features should extract useful information (maybe object identities, properties, scene type, etc.) that we can use for downstream tasks

Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN





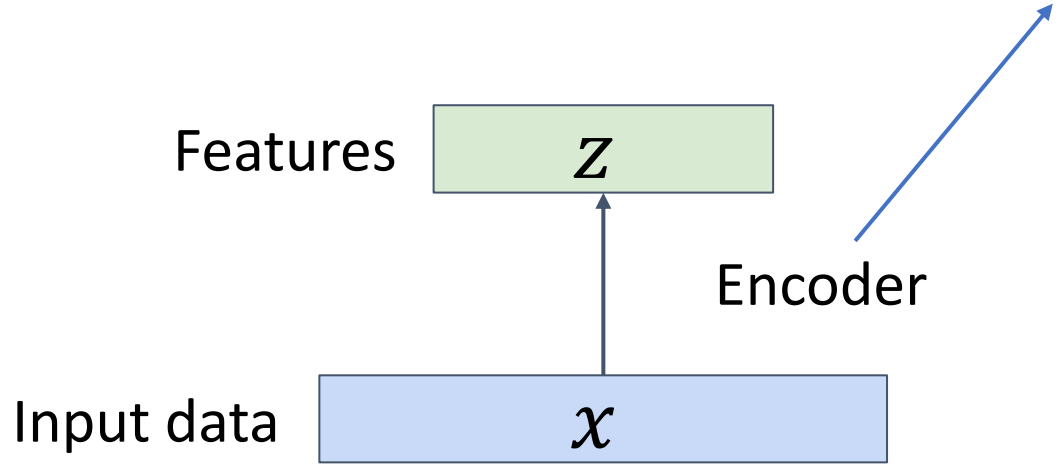
Input Data

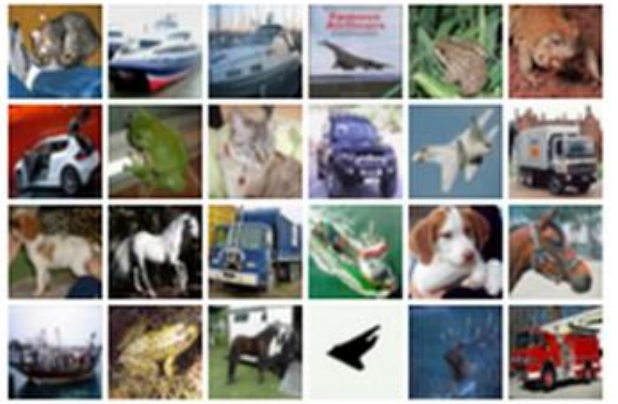
(Regular, non-variational) Autoencoders

Problem: How can we learn this feature transform from raw data?

Features should extract useful information (maybe object identities, properties, scene type, etc.) that we can use for downstream tasks
But we can't observe features!

Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN





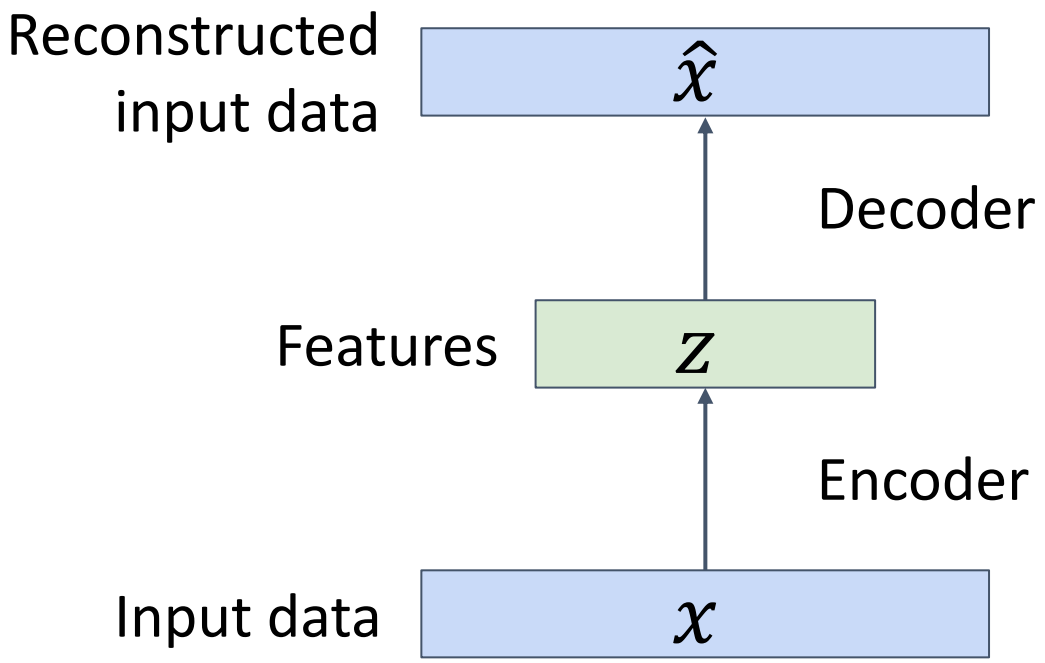
Input Data

(Regular, non-variational) Autoencoders

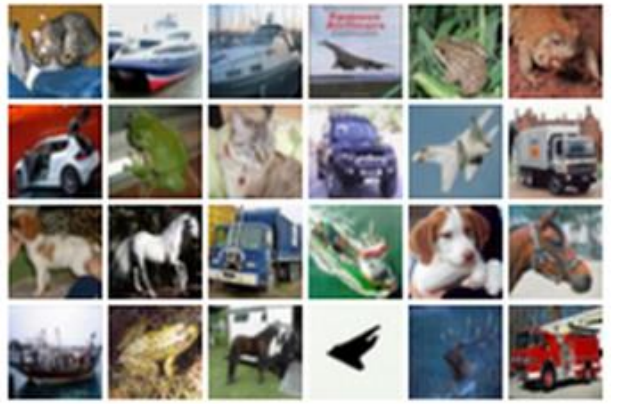
Problem: How can we learn this feature transform from raw data?

Idea: Use the features to reconstruct the input data with a **decoder**

"Autoencoding" = encoding itself



Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN (upconv)



Input Data

(Regular, non-variational) Autoencoders

Loss: L2 distance between input and reconstructed data.

Does not use any labels! Just raw data!

Loss Function

$$\|\hat{x} - x\|_2^2$$

Reconstructed input data

$\hat{x}$

Decoder

Features need to be **lower dimensional** than the data

Features

z

Encoder

Input data

x

Reconstructed data

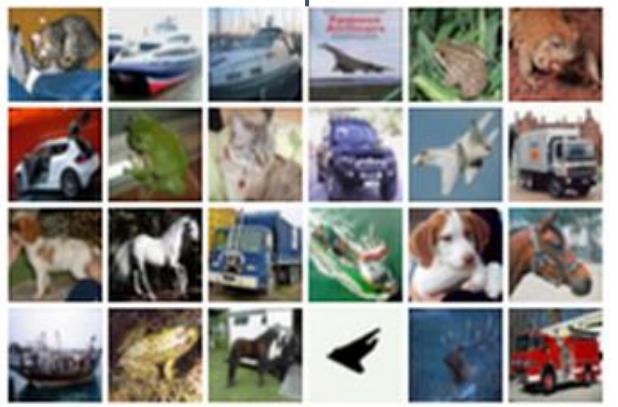
What's this?

Decoder:

4 tconv layers

Encoder:

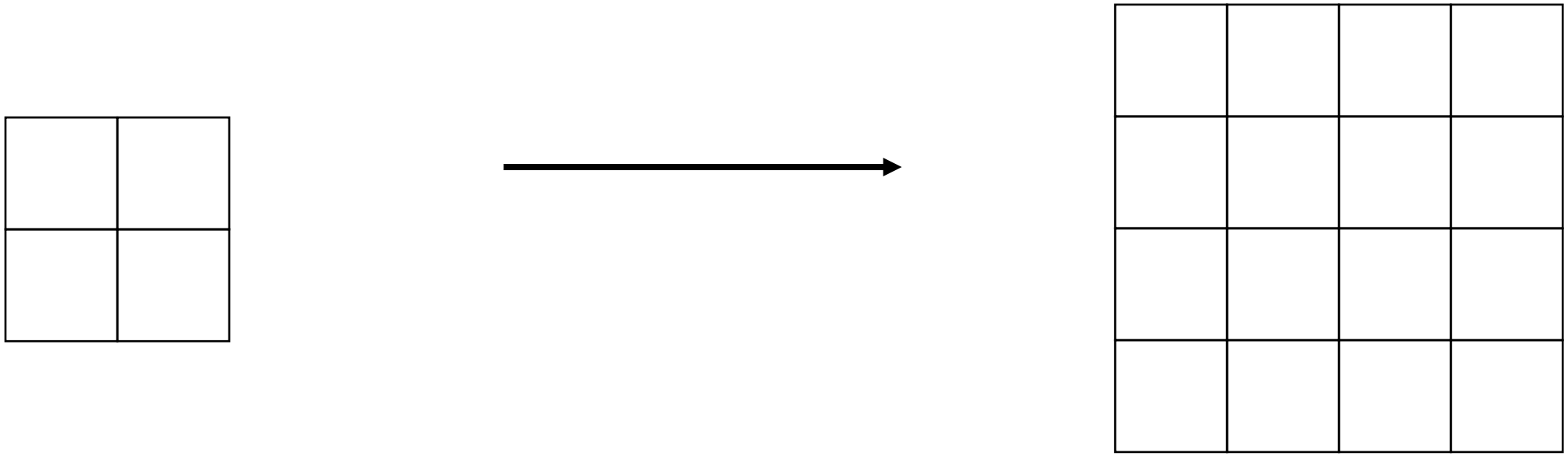
4 conv layers



Input Data

Transposed Convolution

3 x 3 **convolution transpose**, stride 2

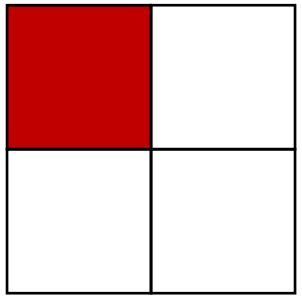


Input: 2 x 2

Output: 4 x 4

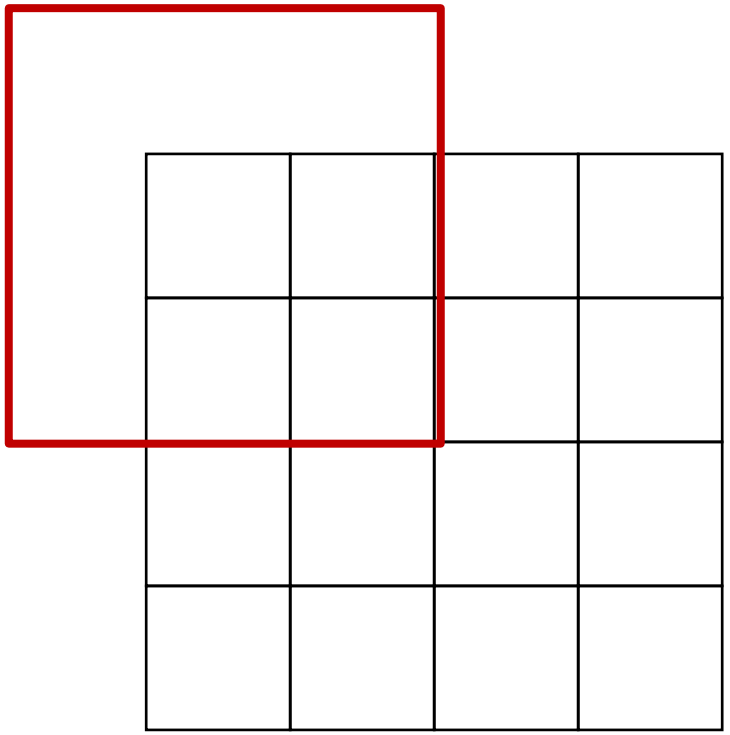
Transposed Convolution

3 x 3 **convolution transpose**, stride 2



Input: 2 x 2

Weight filter by input value and copy to output



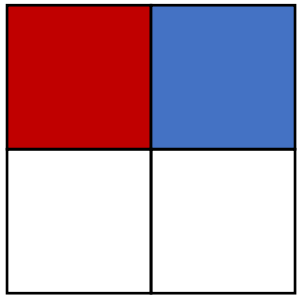
Output: 4 x 4

Transposed Convolution

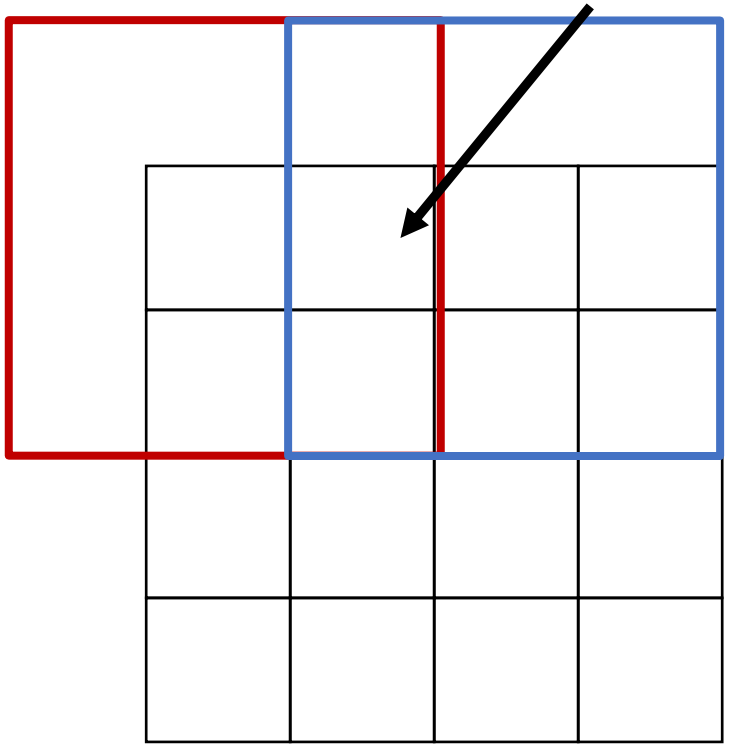
3 x 3 **convolution transpose**, stride 2

Sum where output overlaps

Filter moves 2 pixels in output for every 1 pixel in input



Weight filter by input value and copy to output



Input: 2 x 2

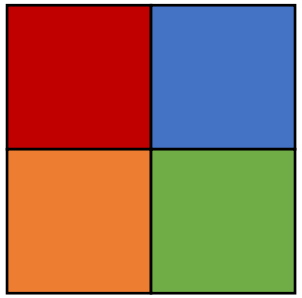
Output: 4 x 4

Transposed Convolution

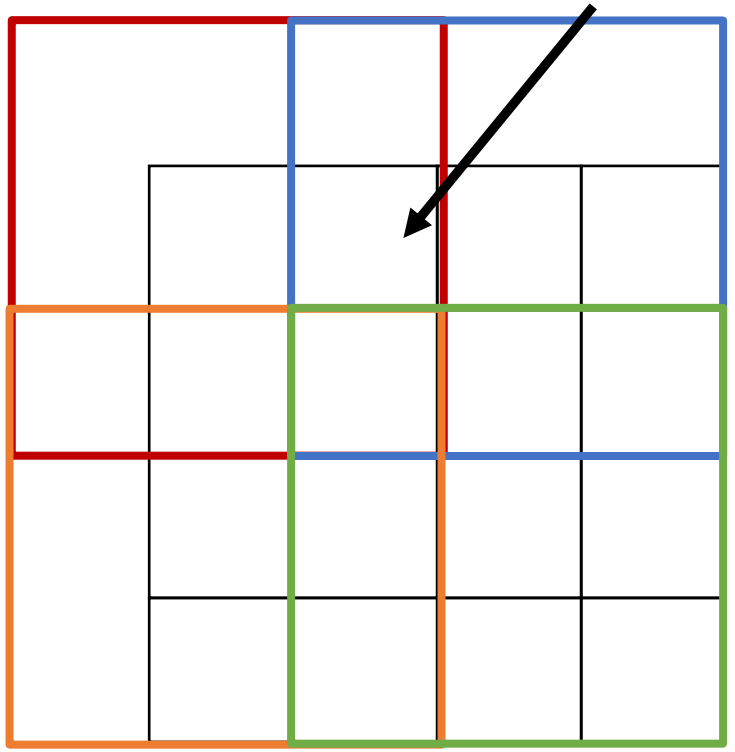
3 x 3 **convolution transpose**, stride 2

Sum where output overlaps

This gives 5x5 output – need to trim one pixel from top and left to give 4x4 output



Weight filter by input value and copy to output



Input: 2 x 2

Output: 4 x 4

Recall: PyTorch Convolution Layer

CLASS torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

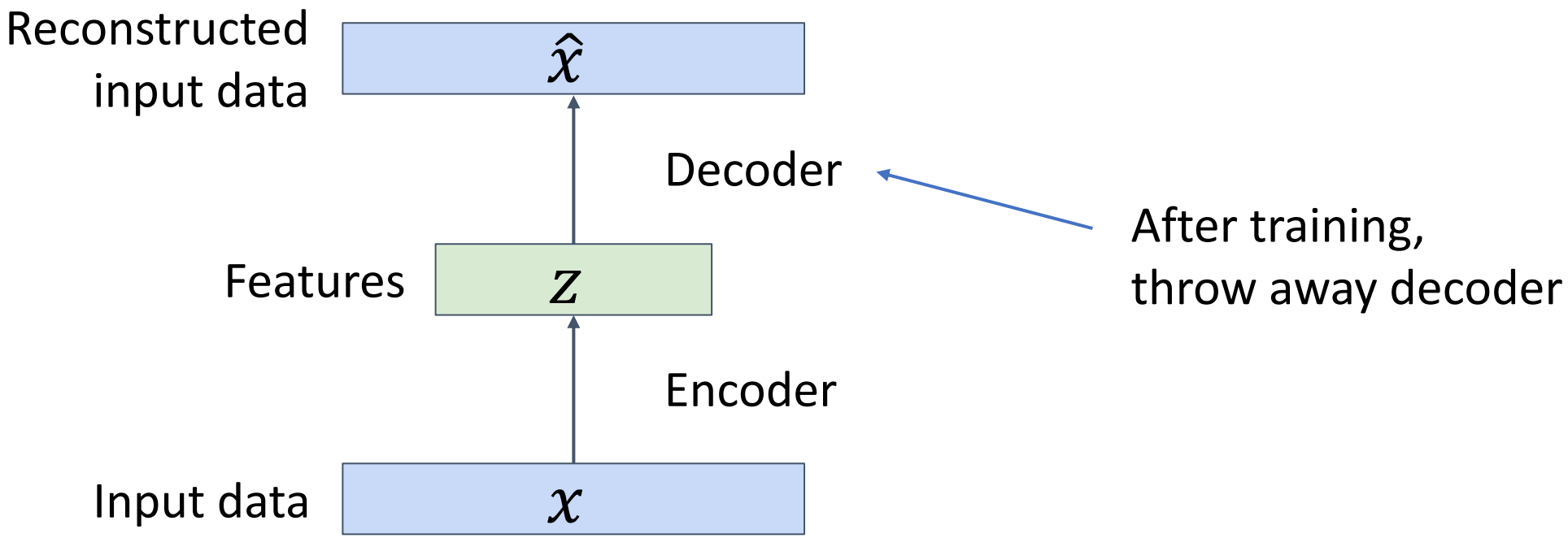
PyTorch Transposed Convolution Layer

CLASS torch.nn.ConvTranspose2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *output_padding=0*, *groups=1*, *bias=True*, *dilation=1*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

- Transposed convolution is used to revert convolution, in terms of the size of feature map.

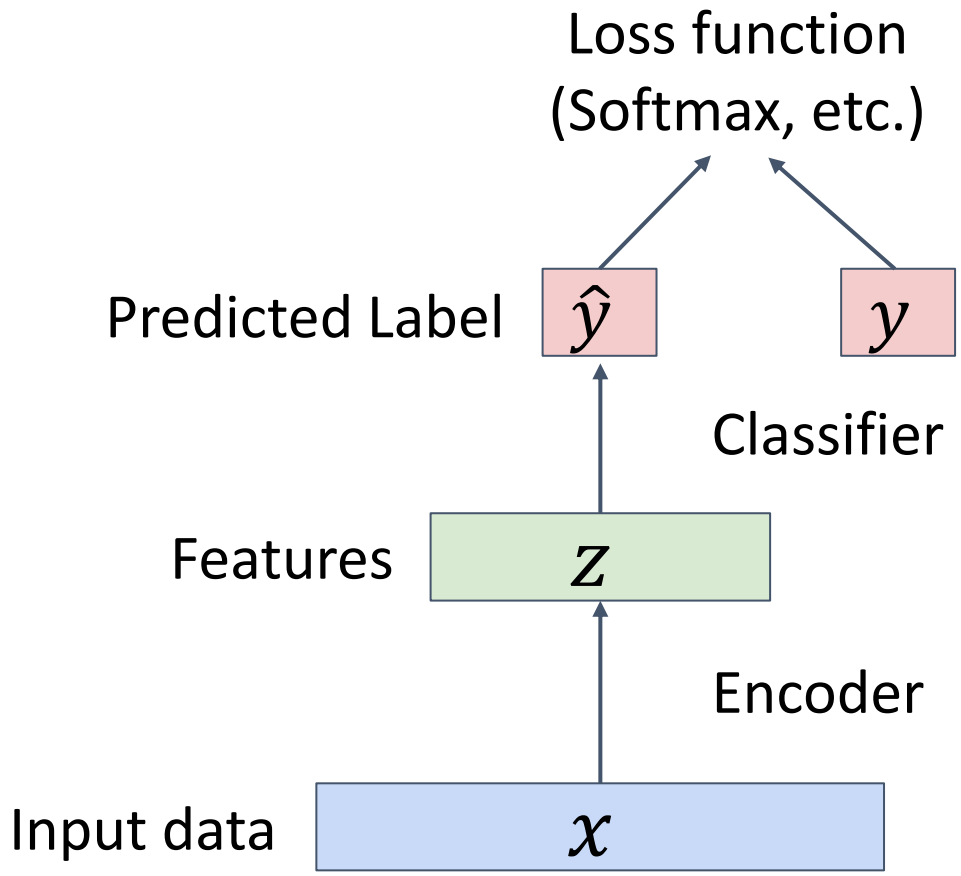
(Regular, non-variational) Autoencoders

After training, **throw away decoder** and use encoder for a downstream task



(Regular, non-variational) Autoencoders

After training, **throw away decoder** and use encoder for a downstream task



Fine-tune encoder jointly with classifier

Encoder can be used to initialize a **supervised** model

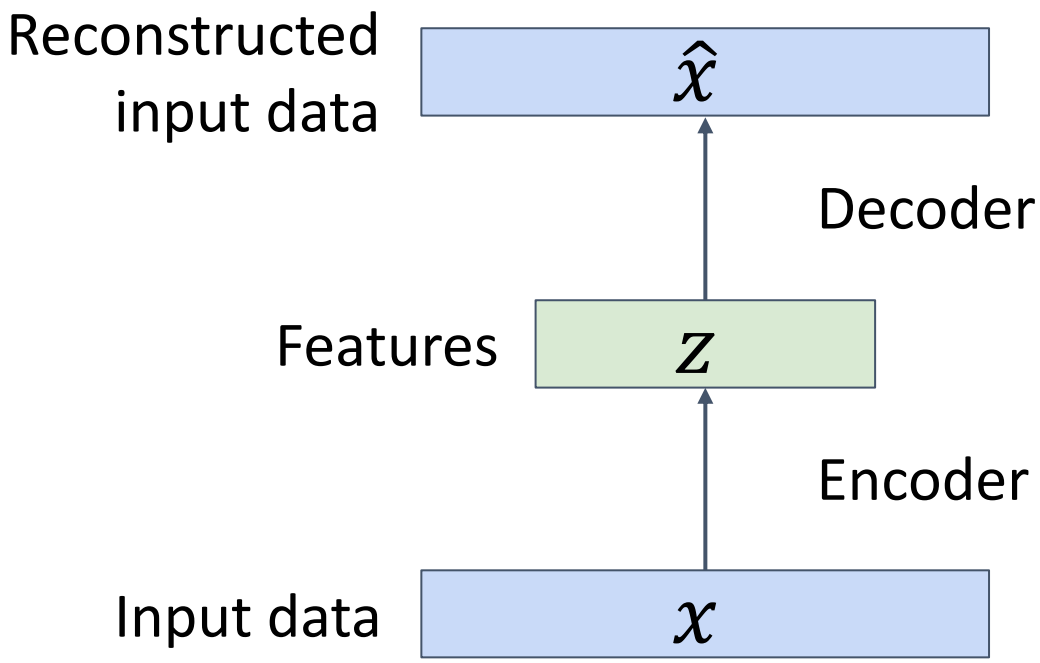
Train for final task (sometimes with small data)

(Regular, non-variational) Autoencoders

Autoencoders learn **latent features** for data without any labels!

Can use features to initialize a **supervised** model

Not probabilistic: No way to sample new data from learned model



Variational Autoencoders

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders

Probabilistic spin on autoencoders:

1. Learn latent features z from raw data
2. Sample from the model to generate new data

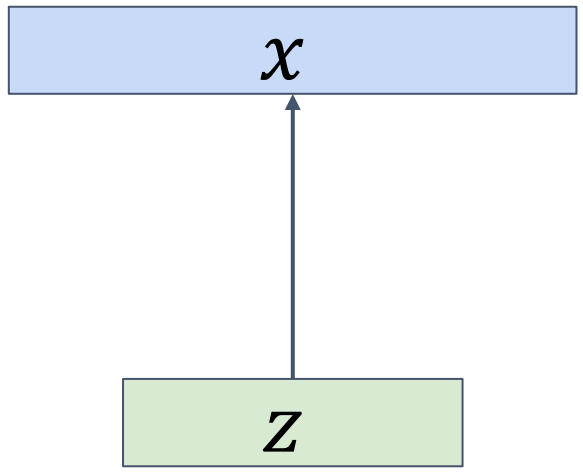
After training, sample new data like this:

Sample from conditional

$$p_{\theta^*}(x \mid z^{(i)})$$

Sample z from prior

$$p_{\theta^*}(z)$$



Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from unobserved (latent) representation z

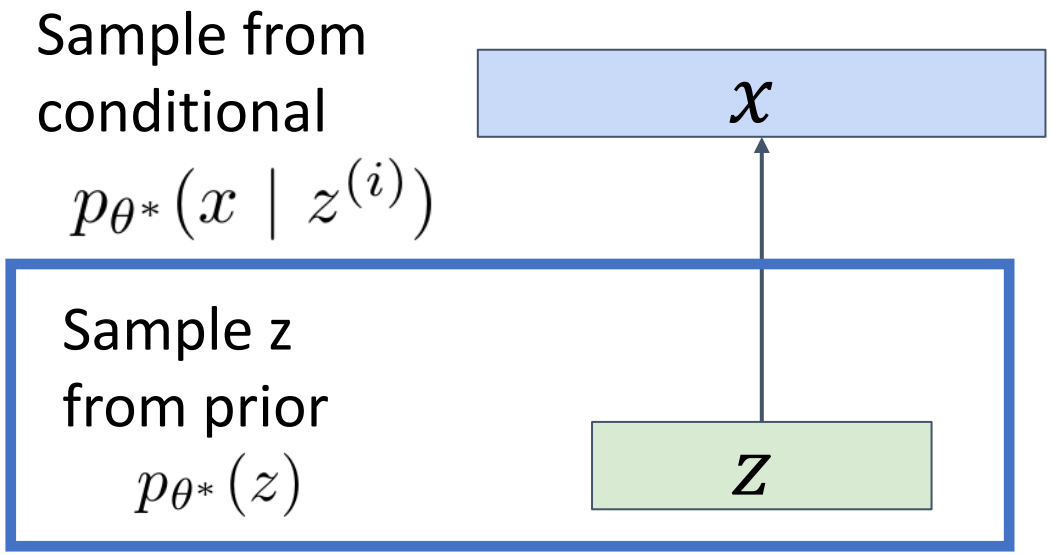
Intuition: x is an image, z is latent factors used to generate x: attributes, orientation, etc.

Variational Autoencoders

Probabilistic spin on autoencoders:

1. Learn latent features z from raw data
2. Sample from the model to generate new data

After training, sample new data like this:



Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from unobserved (latent) representation z

Intuition: x is an image, z is latent factors used to generate x: attributes, orientation, etc.

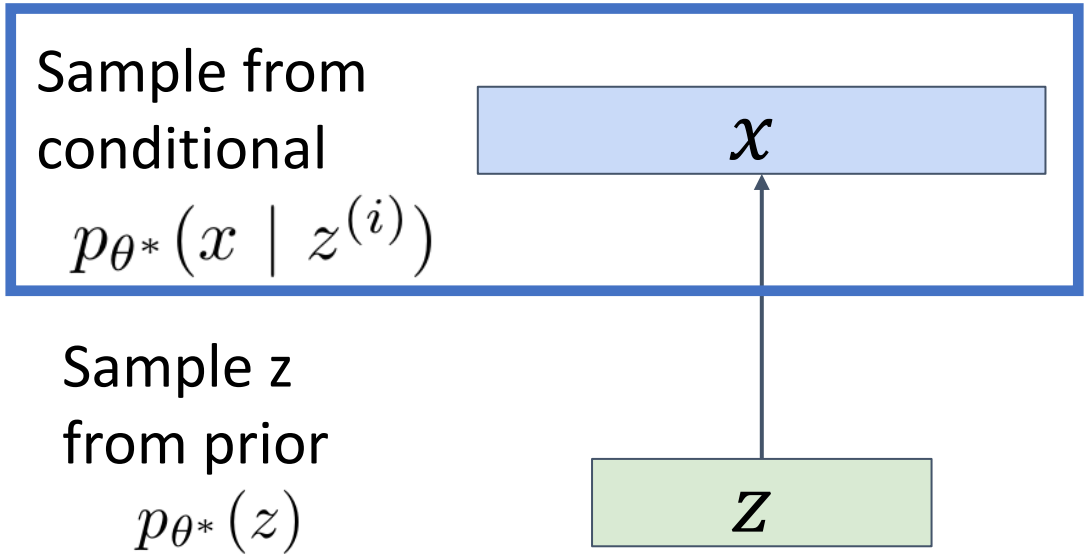
Assume simple prior $p(z)$, e.g., Gaussian

Variational Autoencoders

Probabilistic spin on autoencoders:

1. Learn latent features z from raw data
2. Sample from the model to generate new data

After training, sample new data like this:



Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from unobserved (latent) representation z

Intuition: x is an image, z is latent factors used to generate x: attributes, orientation, etc.

Assume simple prior $p(z)$, e.g., Gaussian

Represent $p(x|z)$ with a neural network (Similar to **decoder** from autoencoder)

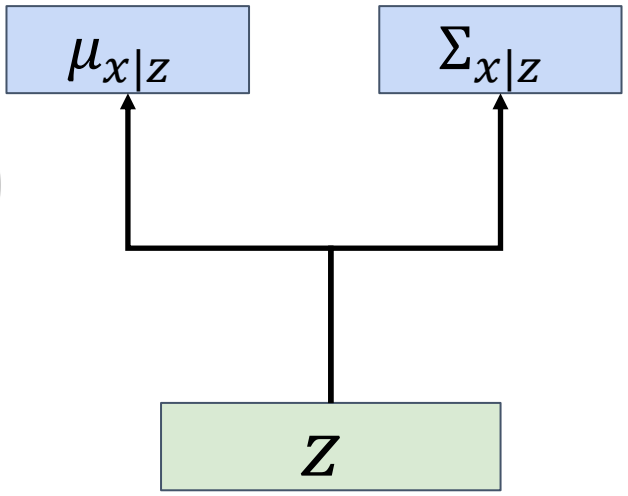
Variational Autoencoders

Decoder must be **probabilistic**:
Decoder inputs z, outputs mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample from conditional $p_{\theta^*}(x \mid z^{(i)})$

Sample z from prior $p_{\theta^*}(z)$



Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from unobserved (latent) representation z

Intuition: x is an image, z is latent factors used to generate x: attributes, orientation, etc.

Assume simple prior $p(z)$, e.g., Gaussian

Represent $p(x|z)$ with a neural network (Similar to **decoder** from autoencoder)

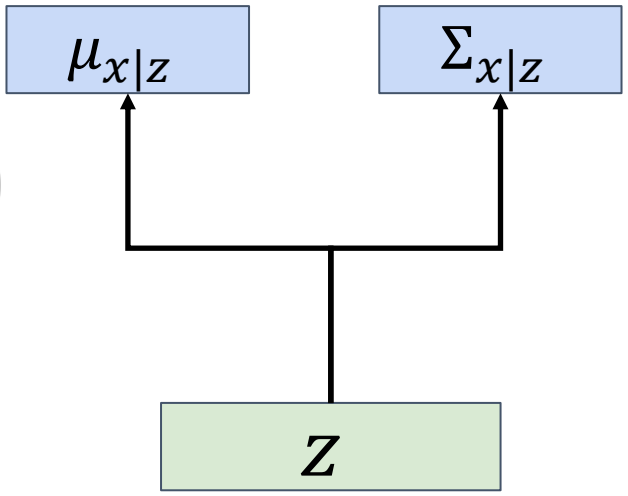
Variational Autoencoders

Decoder must be **probabilistic**:
Decoder inputs z, outputs mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample from conditional $p_{\theta^*}(x \mid z^{(i)})$

Sample z from prior $p_{\theta^*}(z)$



Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from unobserved (latent) representation z

How to train this model?

Basic idea: **maximize likelihood of data**

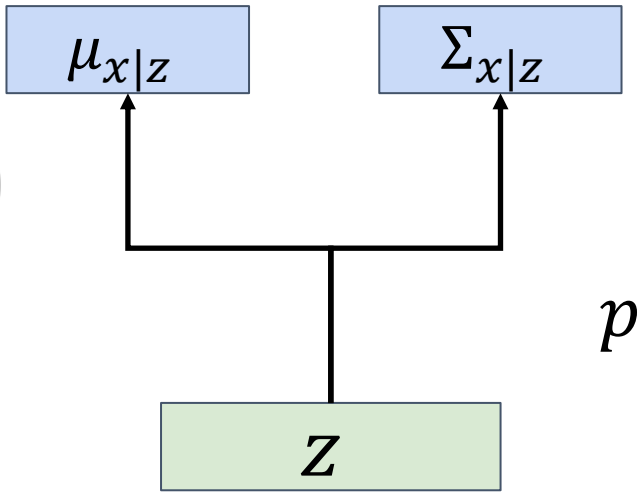
If we could observe the z for each x, then could train a *conditional generative model* $p(x|z)$

Variational Autoencoders

Decoder must be **probabilistic**:
Decoder inputs z, outputs mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample from conditional $p_{\theta^*}(x \mid z^{(i)})$



Sample z from prior $p_{\theta^*}(z)$

Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from unobserved (latent) representation z

How to train this model?

Basic idea: **maximize likelihood of data**

We don't observe z, so need to marginalize:

$$p_\theta(x) = \int p_\theta(x,z)dz = \int p_\theta(x|z)p_\theta(z)dz$$

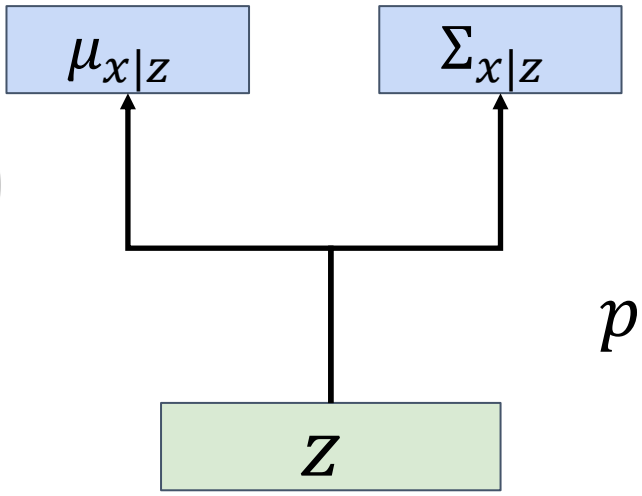
Variational Autoencoders

Decoder must be **probabilistic**:
Decoder inputs z, outputs mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample from conditional $p_{\theta^*}(x \mid z^{(i)})$

Sample z from prior $p_{\theta^*}(z)$



Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from unobserved (latent) representation z

How to train this model?

Basic idea: **maximize likelihood of data**

We don't observe z, so need to marginalize:

$$p_\theta(x) = \int p_\theta(x,z)dz = \int p_\theta(x|z)p_\theta(z)dz$$

Ok, can compute this with decoder network

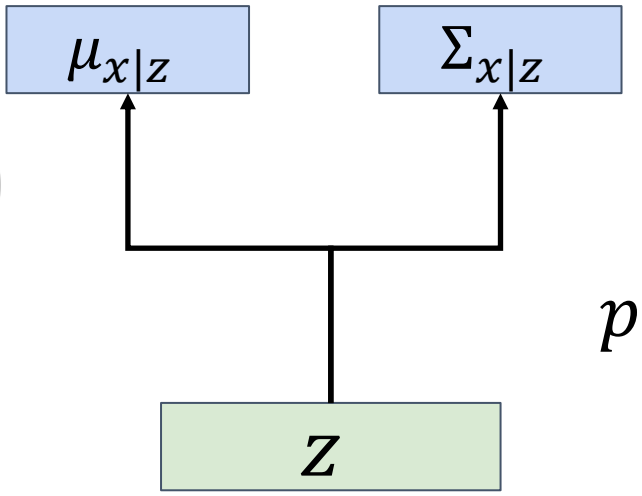
Variational Autoencoders

Decoder must be **probabilistic**:
Decoder inputs z, outputs mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample from conditional $p_{\theta^*}(x \mid z^{(i)})$

Sample z from prior $p_{\theta^*}(z)$



Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from unobserved (latent) representation z

How to train this model?

Basic idea: **maximize likelihood of data**

We don't observe z, so need to marginalize:

$$p_\theta(x) = \int p_\theta(x,z)dz = \int p_\theta(x|z)p_\theta(z)dz$$

Ok, we assumed Gaussian prior for z

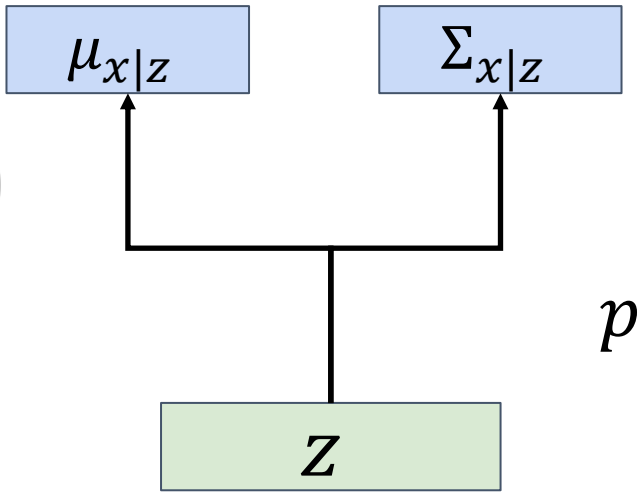
Variational Autoencoders

Decoder must be **probabilistic**:
Decoder inputs z, outputs mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample from conditional $p_{\theta^*}(x \mid z^{(i)})$

Sample z from prior $p_{\theta^*}(z)$



Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from unobserved (latent) representation z

How to train this model?

Basic idea: **maximize likelihood of data**

We don't observe z, so need to marginalize:

$$p_\theta(x) = \int p_\theta(x,z)dz = \int p_\theta(x|z)p_\theta(z)dz$$

Problem: Impossible to integrate over all z!

Recall $p(x, z) = p(x \mid z)p(z) = p(z \mid x)p(x)$

Variational Autoencoders

Decoder must be **probabilistic**:
Decoder inputs z, outputs mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

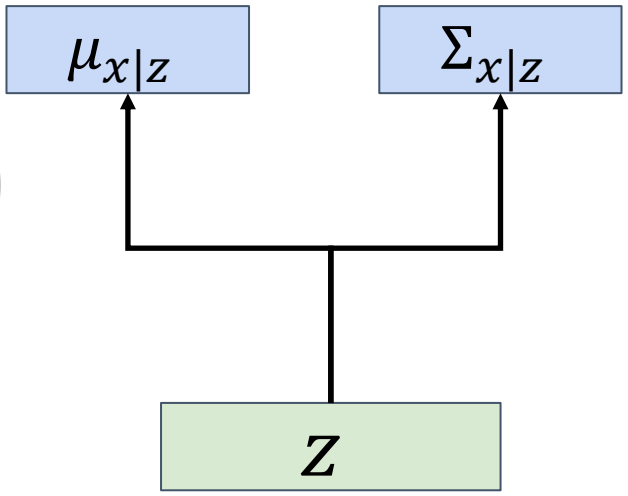
Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from unobserved (latent) representation z

How to train this model?

Sample from conditional $p_{\theta^*}(x \mid z^{(i)})$

Sample z from prior $p_{\theta^*}(z)$



Basic idea: **maximize likelihood of data**

Another idea: Try Bayes' Rule:

$$p_\theta(x) = \frac{p_\theta(x \mid z)p_\theta(z)}{p_\theta(z \mid x)}$$

Recall $p(x, z) = p(x \mid z)p(z) = p(z \mid x)p(x)$

Variational Autoencoders

Decoder must be **probabilistic**:
Decoder inputs z, outputs mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

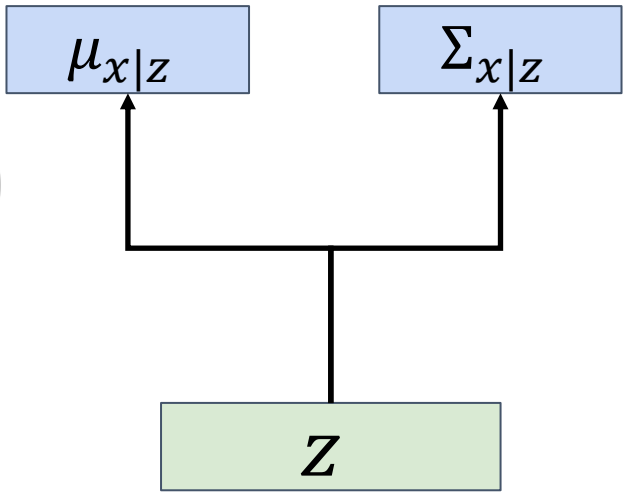
Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from unobserved (latent) representation z

How to train this model?

Sample from conditional $p_{\theta^*}(x \mid z^{(i)})$

Sample z from prior $p_{\theta^*}(z)$



Basic idea: **maximize likelihood of data**

Another idea: Try Bayes' Rule:

$$p_\theta(x) = \frac{p_\theta(x \mid z)p_\theta(z)}{p_\theta(z \mid x)}$$

Ok, compute with decoder network

Recall $p(x, z) = p(x \mid z)p(z) = p(z \mid x)p(x)$

Variational Autoencoders

Decoder must be **probabilistic**:
Decoder inputs z, outputs mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

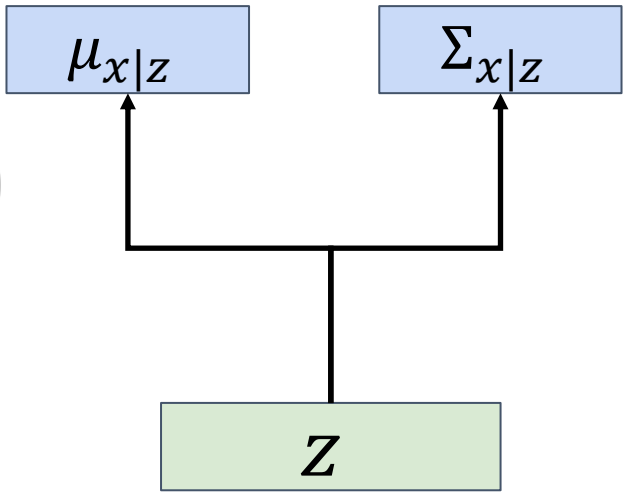
Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from unobserved (latent) representation z

How to train this model?

Sample from conditional $p_{\theta^*}(x \mid z^{(i)})$

Sample z from prior $p_{\theta^*}(z)$



Basic idea: **maximize likelihood of data**

Another idea: Try Bayes' Rule:

$$p_\theta(x) = \frac{p_\theta(x \mid z)p_\theta(z)}{p_\theta(z \mid x)}$$

Ok, we assumed Gaussian prior

Recall $p(x, z) = p(x \mid z)p(z) = p(z \mid x)p(x)$

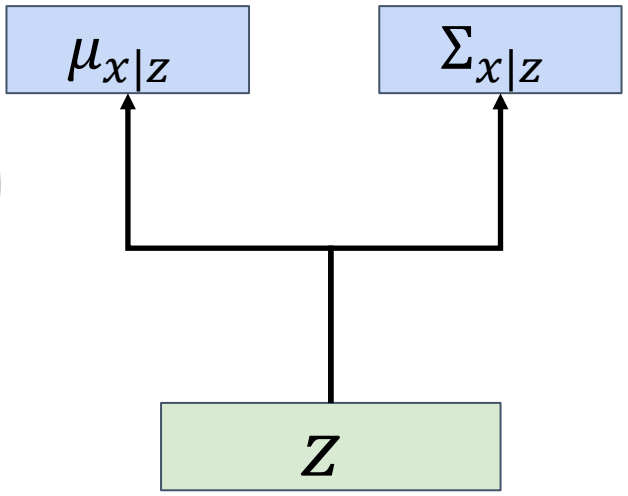
Variational Autoencoders

Decoder must be **probabilistic**:
Decoder inputs z, outputs mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample from conditional $p_{\theta^*}(x \mid z^{(i)})$

Sample z from prior $p_{\theta^*}(z)$



Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from unobserved (latent) representation z

How to train this model?

Basic idea: **maximize likelihood of data**

Another idea: Try Bayes' Rule:

$$p_\theta(x) = \frac{p_\theta(x \mid z)p_\theta(z)}{p_\theta(z \mid x)}$$

Problem: No way to compute this!

Recall $p(x, z) = p(x \mid z)p(z) = p(z \mid x)p(x)$

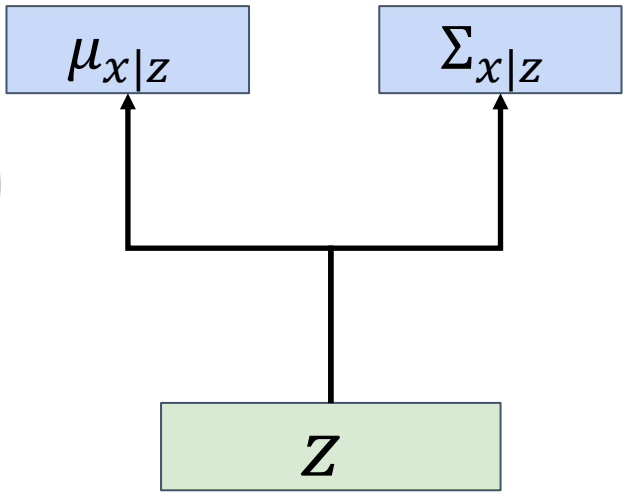
Variational Autoencoders

Decoder must be **probabilistic**:
Decoder inputs z, outputs mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample from conditional $p_{\theta^*}(x \mid z^{(i)})$

Sample z from prior $p_{\theta^*}(z)$



Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from unobserved (latent) representation z

How to train this model?

Basic idea: **maximize likelihood of data**

Another idea: Try Bayes' Rule:

$$p_\theta(x) = \frac{p_\theta(x \mid z)p_\theta(z)}{p_\theta(z \mid x)}$$

Solution: Train another network **(encoder)** that learns $q_\phi(z \mid x) \approx p_\theta(z \mid x)$

Recall $p(x, z) = p(x \mid z)p(z) = p(z \mid x)p(x)$

Variational Autoencoders

Decoder must be **probabilistic**:
Decoder inputs z, outputs mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

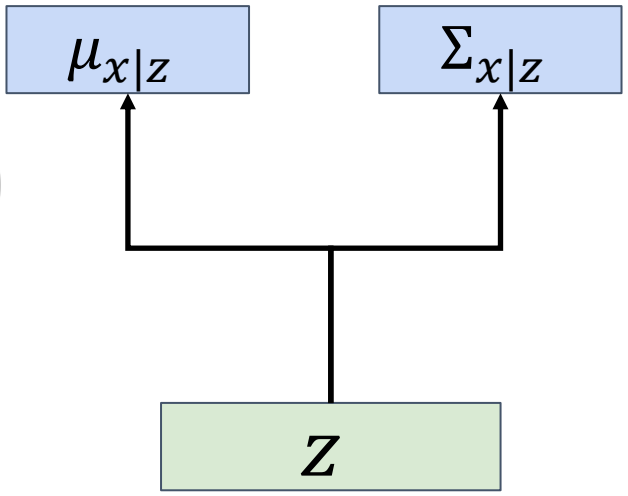
Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Assume training data $\{x^{(i)}\}_{i=1}^{N}$ is generated from unobserved (latent) representation z

How to train this model?

Sample from conditional $p_{\theta^*}(x \mid z^{(i)})$

Sample z from prior $p_{\theta^*}(z)$



Basic idea: **maximize likelihood of data**

Another idea: Try Bayes' Rule:

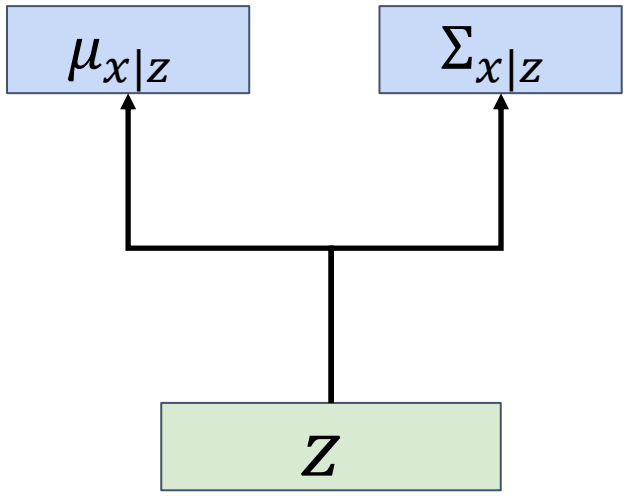
$$p_\theta(x) = \frac{p_\theta(x \mid z)p_\theta(z)}{p_\theta(z \mid x)} \approx \frac{p_\theta(x \mid z)p_\theta(z)}{q_\phi(z \mid x)}$$

Use **encoder** to compute $q_\phi(z \mid x) \approx p_\theta(z \mid x)$

Variational Autoencoders

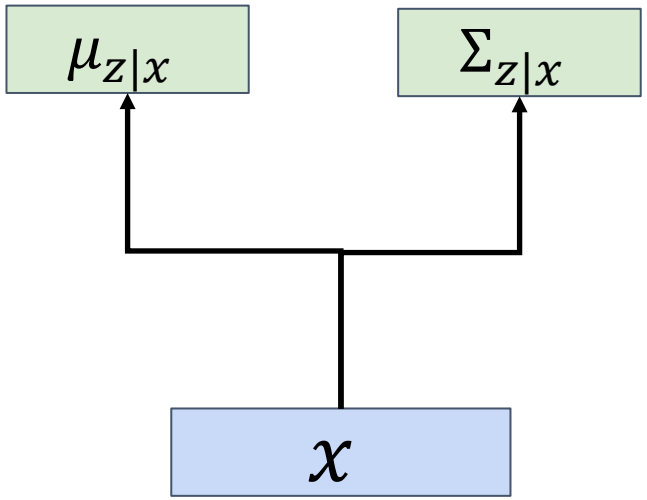
Decoder network inputs latent code z, gives distribution over data x

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



Encoder network inputs data x, gives distribution over latent codes z

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



If we can ensure that $q_\phi(z \mid x) \approx p_\theta(z \mid x)$,

then we can approximate

$$p_\theta(x) \approx \frac{p_\theta(x \mid z)p(z)}{q_\phi(z \mid x)}$$

Idea: Jointly train both encoder and decoder

Variational Autoencoders

$$\log p_\theta(x) = \log \frac{p_\theta(x \mid z)p(z)}{p_\theta(z \mid x)}$$

Bayes' Rule

Variational Autoencoders

$$\log p_\theta(x) = \log \frac{p_\theta(x \mid z)p(z)}{p_\theta(z \mid x)} = \log \frac{p_\theta(x|z)p(z)q_\phi(z|x)}{p_\theta(z|x)q_\phi(z|x)}$$

Multiply top and bottom by $q_\phi(z|x)$

Variational Autoencoders

$$\log p_\theta(x) = \log \frac{p_\theta(x \mid z)p(z)}{p_\theta(z \mid x)} = \log \frac{p_\theta(x|z)\,p(z)\,q_\phi(z|x)}{p_\theta(z|x)\,q_\phi(z|x)}$$

$$= \log p_\theta(x|z) - \log \frac{q_\phi(z|x)}{p(z)} + \log \frac{q_\phi(z|x)}{p_\theta(z|x)}$$

Split up using rules for logarithms

Variational Autoencoders

$$\log p_\theta(x) = \log \frac{p_\theta(x \mid z)p(z)}{p_\theta(z \mid x)} = \log \frac{p_\theta(x|z)p(z)q_\phi(z|x)}{p_\theta(z|x)q_\phi(z|x)}$$

$$= \log p_\theta(x|z) - \log \frac{q_\phi(z|x)}{p(z)} + \log \frac{q_\phi(z|x)}{p_\theta(z|x)}$$

$$\log p_\theta(x) = E_{z\sim q_\phi(z|x)}[\log p_\theta(x)]$$

We can wrap in an expectation since it doesn't depend on z

Variational Autoencoders

$$\log p_\theta(x) = \log \frac{p_\theta(x \mid z)p(z)}{p_\theta(z \mid x)} = \log \frac{p_\theta(x|z)p(z)q_\phi(z|x)}{p_\theta(z|x)q_\phi(z|x)}$$

$$= E_z[\log p_\theta(x|z)] - E_z\left[\log \frac{q_\phi(z|x)}{p(z)}\right] + E_z\left[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}\right]$$

$$\log p_\theta(x) = E_{z\sim q_\phi(z|x)}[\log p_\theta(x)]$$

We can wrap in an expectation since it doesn't depend on z

Variational Autoencoders

$$\log p_\theta(x) = \log \frac{p_\theta(x \mid z)p(z)}{p_\theta(z \mid x)} = \log \frac{p_\theta(x|z)p(z)q_\phi(z|x)}{p_\theta(z|x)q_\phi(z|x)}$$

$$= E_z[\log p_\theta(x|z)] - E_z\left[\log \frac{q_\phi(z|x)}{p(z)}\right] + E_z\left[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}\right]$$

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

Data reconstruction

Variational Autoencoders

$$\log p_\theta(x) = \log \frac{p_\theta(x \mid z)p(z)}{p_\theta(z \mid x)} = \log \frac{p_\theta(x|z)p(z)q_\phi(z|x)}{p_\theta(z|x)q_\phi(z|x)}$$

$$= E_z[\log p_\theta(x|z)] - E_z\left[\log \frac{q_\phi(z|x)}{p(z)}\right] + E_z\left[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}\right]$$

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

KL divergence between prior, and
samples from the encoder network

Variational Autoencoders

$$\log p_\theta(x) = \log \frac{p_\theta(x \mid z)p(z)}{p_\theta(z \mid x)} = \log \frac{p_\theta(x|z)p(z)q_\phi(z|x)}{p_\theta(z|x)q_\phi(z|x)}$$

$$= E_z[\log p_\theta(x|z)] - E_z\left[\log \frac{q_\phi(z|x)}{p(z)}\right] + E_z\left[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}\right]$$

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

KL divergence between encoder and posterior of decoder

Variational Autoencoders

$$\log p_\theta(x) = \log \frac{p_\theta(x \mid z)p(z)}{p_\theta(z \mid x)} = \log \frac{p_\theta(x|z)p(z)q_\phi(z|x)}{p_\theta(z|x)q_\phi(z|x)}$$

$$= E_z[\log p_\theta(x|z)] - E_z\left[\log \frac{q_\phi(z|x)}{p(z)}\right] + E_z\left[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}\right]$$

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

KL is >= 0, so dropping this term gives a **lower bound** on the data likelihood:

Variational Autoencoders

$$\log p_\theta(x) = \log \frac{p_\theta(x \mid z)p(z)}{p_\theta(z \mid x)} = \log \frac{p_\theta(x|z)p(z)q_\phi(z|x)}{p_\theta(z|x)q_\phi(z|x)}$$

$$= E_z[\log p_\theta(x|z)] - E_z\left[\log \frac{q_\phi(z|x)}{p(z)}\right] + E_z\left[\log \frac{q_\phi(z|x)}{p_\theta(z|x)}\right]$$

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

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

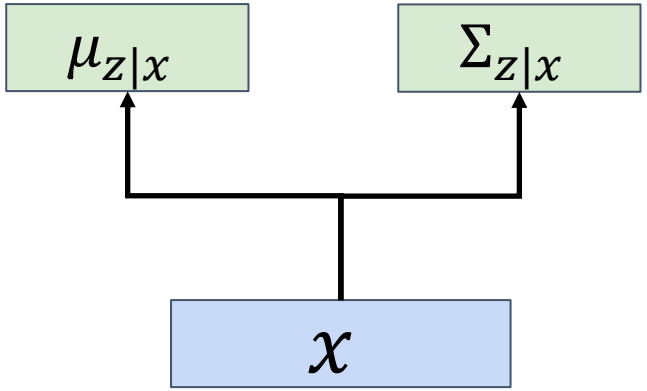
Variational Autoencoders

Jointly train **encoder** q and **decoder** p to maximize the **variational lower bound** on the data likelihood
Also called **Evidence Lower Bound** (**ELBo**)

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

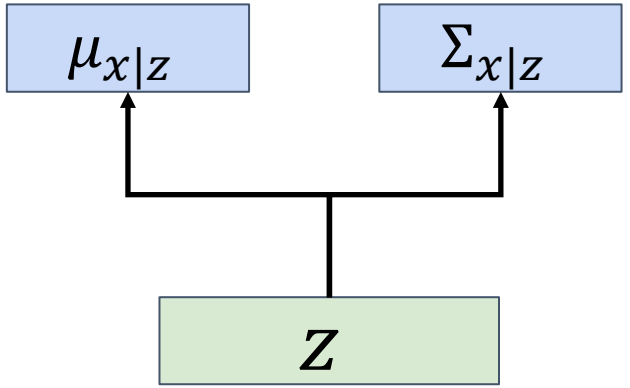
Encoder Network

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



Decoder Network

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



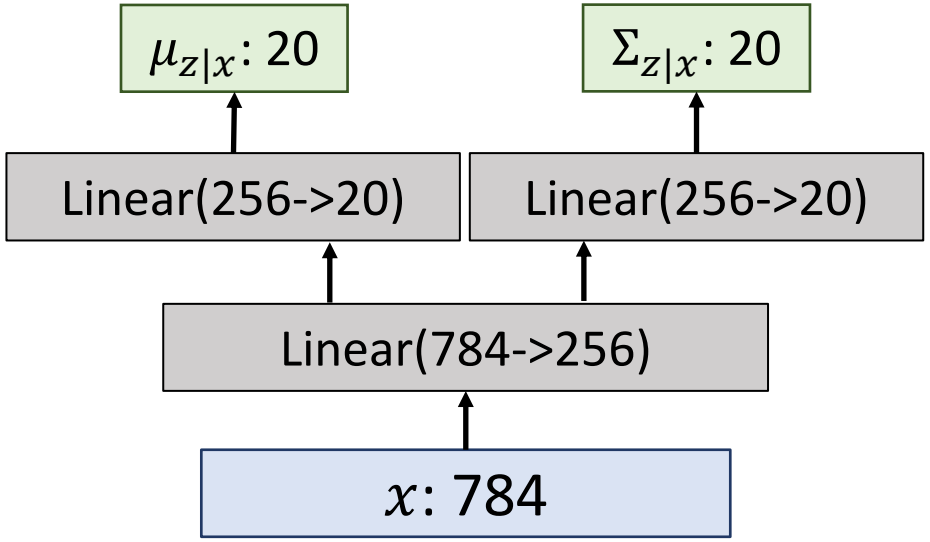
Example: Fully-Connected VAE

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

z: 20-dim vector

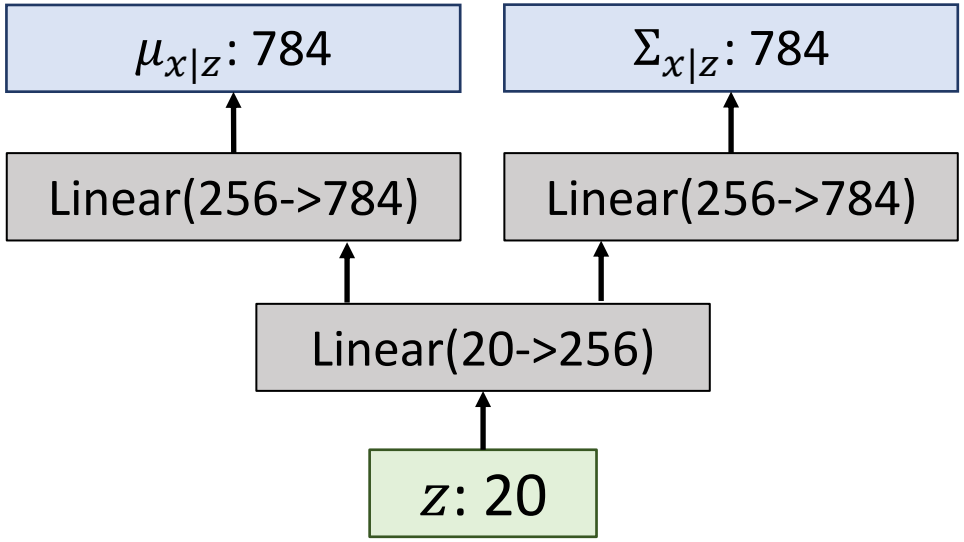
Encoder Network

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



Decoder Network

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

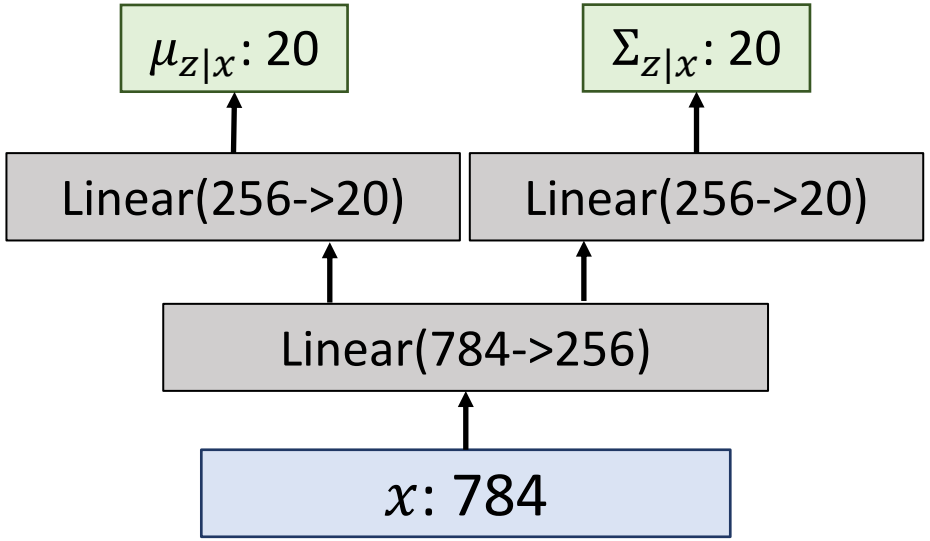


Example: Fully-Connected VAE (in Practice)

Deterministic decoder: Sampling from $z \sim p(z)$ is already stochastic

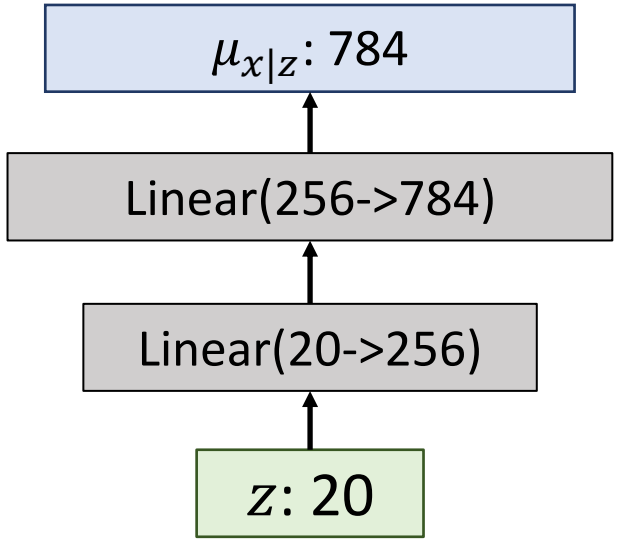
Encoder Network

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



Decoder Network

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



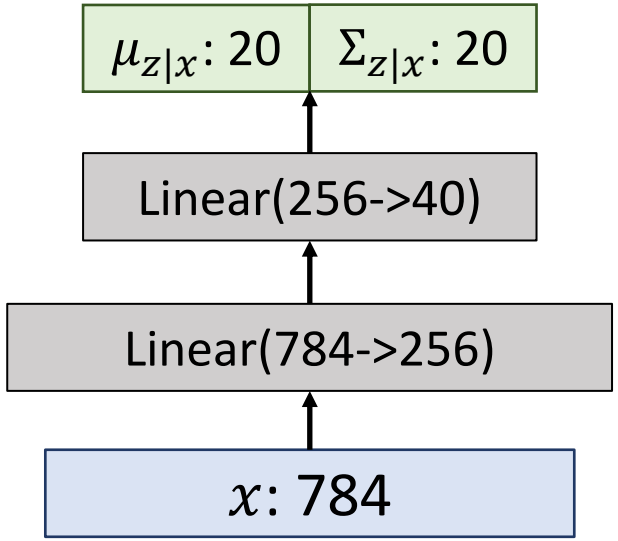
Example: Fully-Connected VAE (in Practice)

Split latent vector to $\mu_{z|x}$ and $\Sigma_{z|x}$ to make the encoder sequential

- Not so common; just have two parallel layers for $\mu_{z|x}$ and $\Sigma_{z|x}$

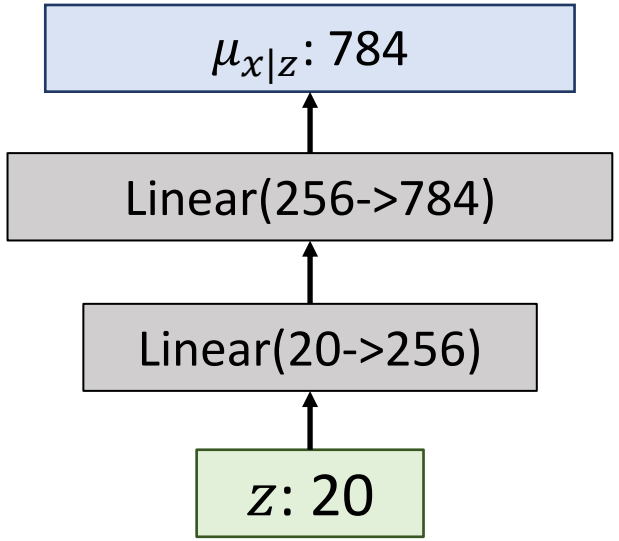
Encoder Network

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



Decoder Network

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

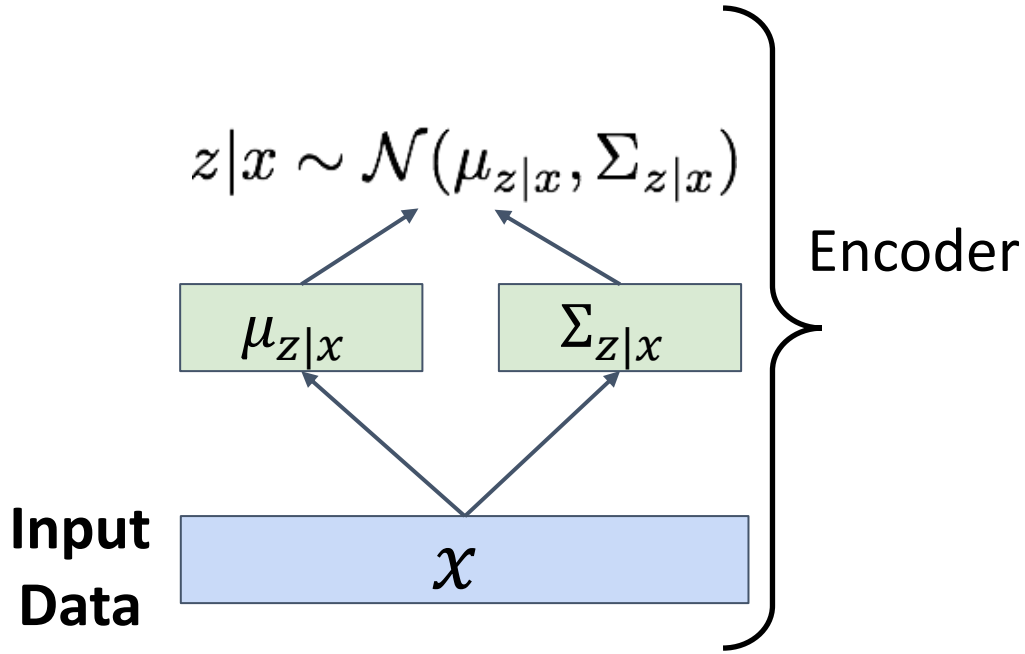


Variational Autoencoders

Train by maximizing the
variational lower bound

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

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



Variational Autoencoders

Train by maximizing the **variational lower bound**

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

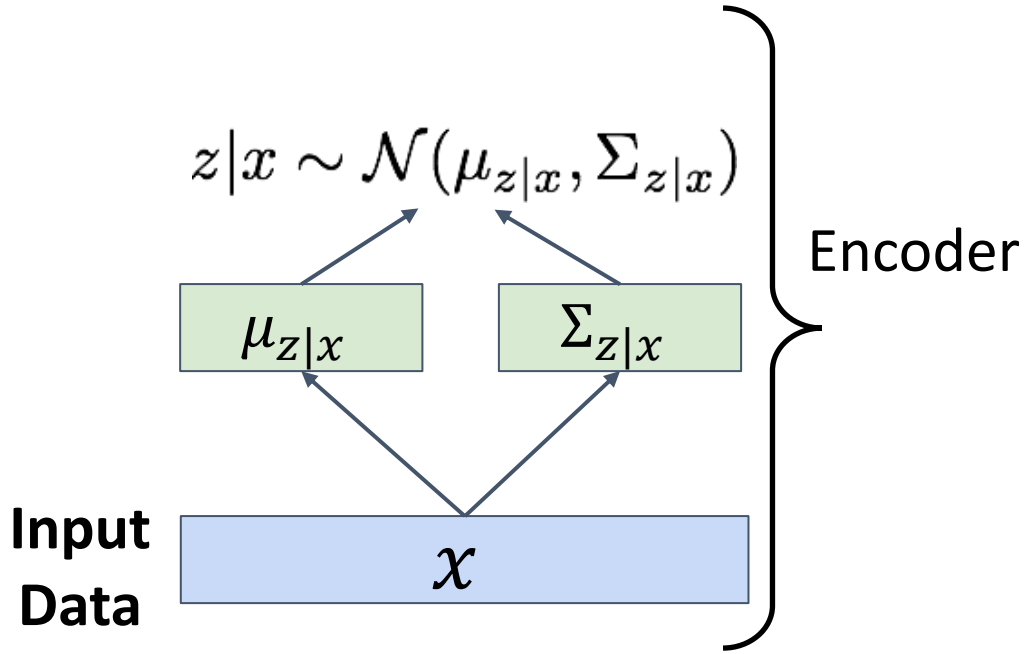
1. Run input data through **encoder** to get a distribution over latent codes
2. Encoder output should match the prior $p(z)$!

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

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

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

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

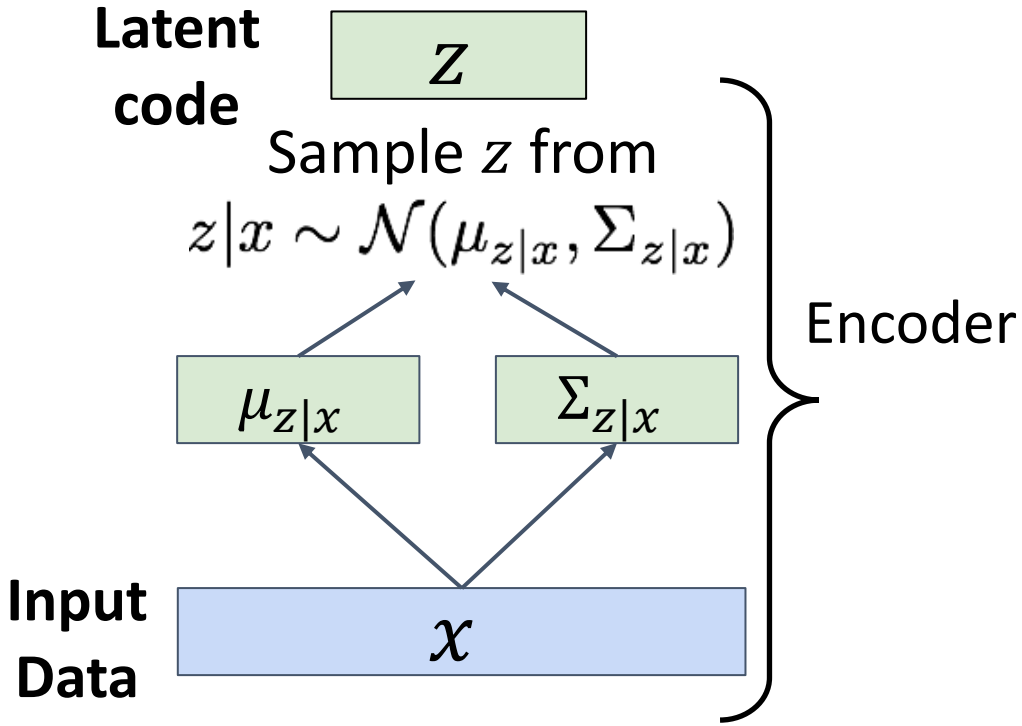


Variational Autoencoders

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
   - **Reparameterization trick:**
     sample $\varepsilon \sim \mathcal{N}(0, I)$ and compute
     $z = \mu_{z|x} + \Sigma_{z|x}^{1/2} \cdot \varepsilon \sim \mathcal{N}\left(\mu_{z|x}, \Sigma_{z|x}\right)$
     such that the encoder gets gradient!
     (Sampled variables don't propagate grad)

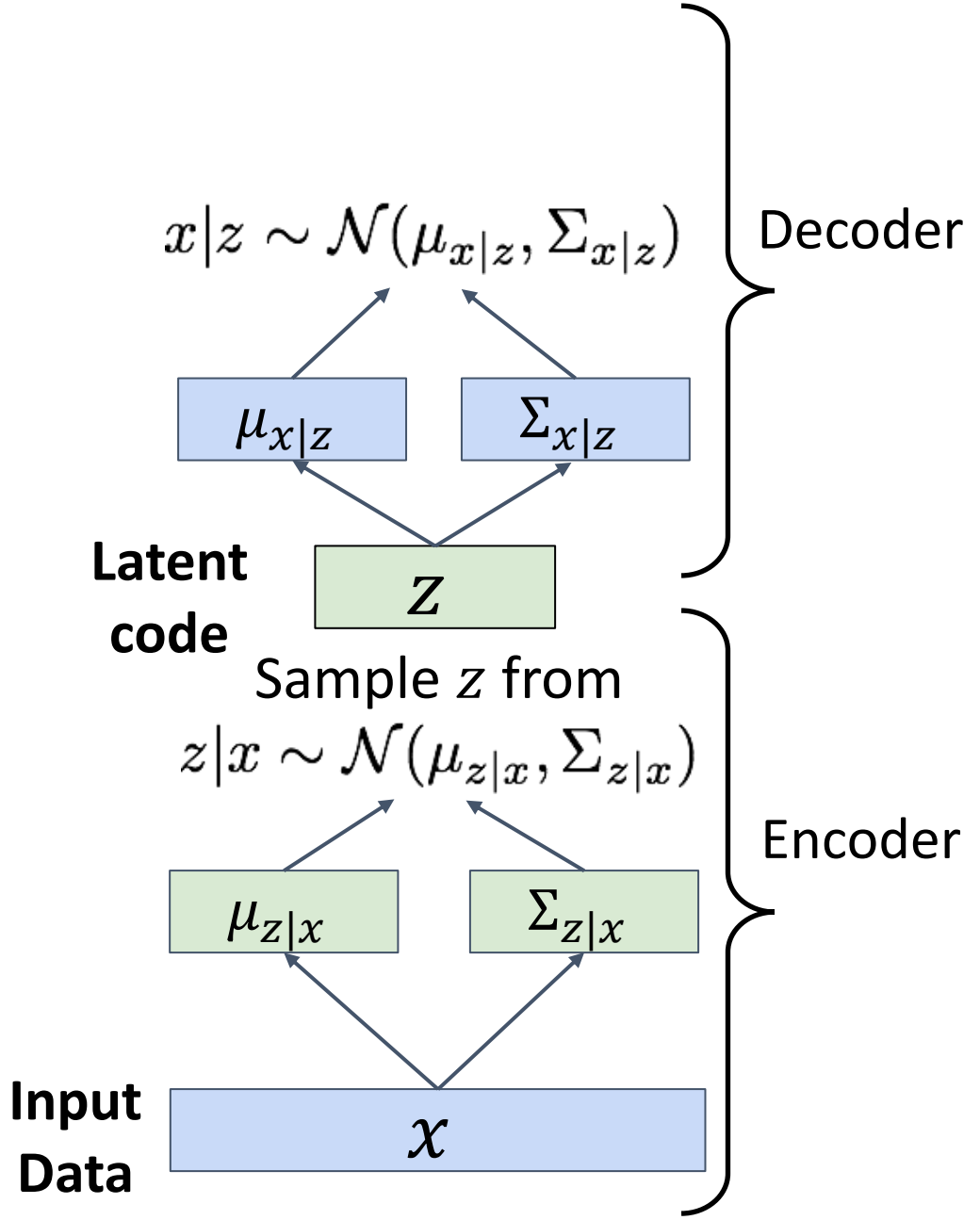


Variational Autoencoders

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

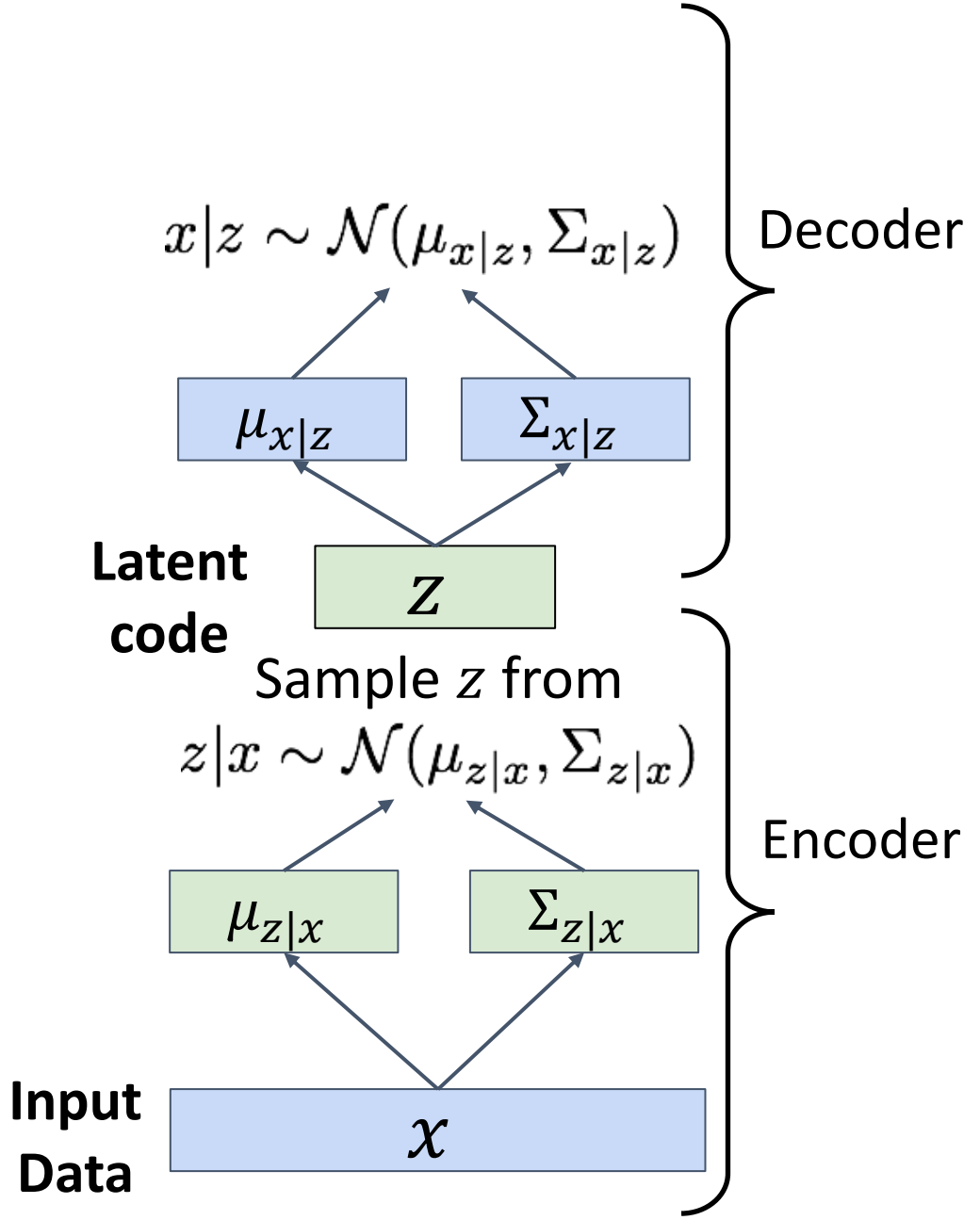


Variational Autoencoders

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)!
   - **Binary cross-entropy loss** or often **L2 loss**

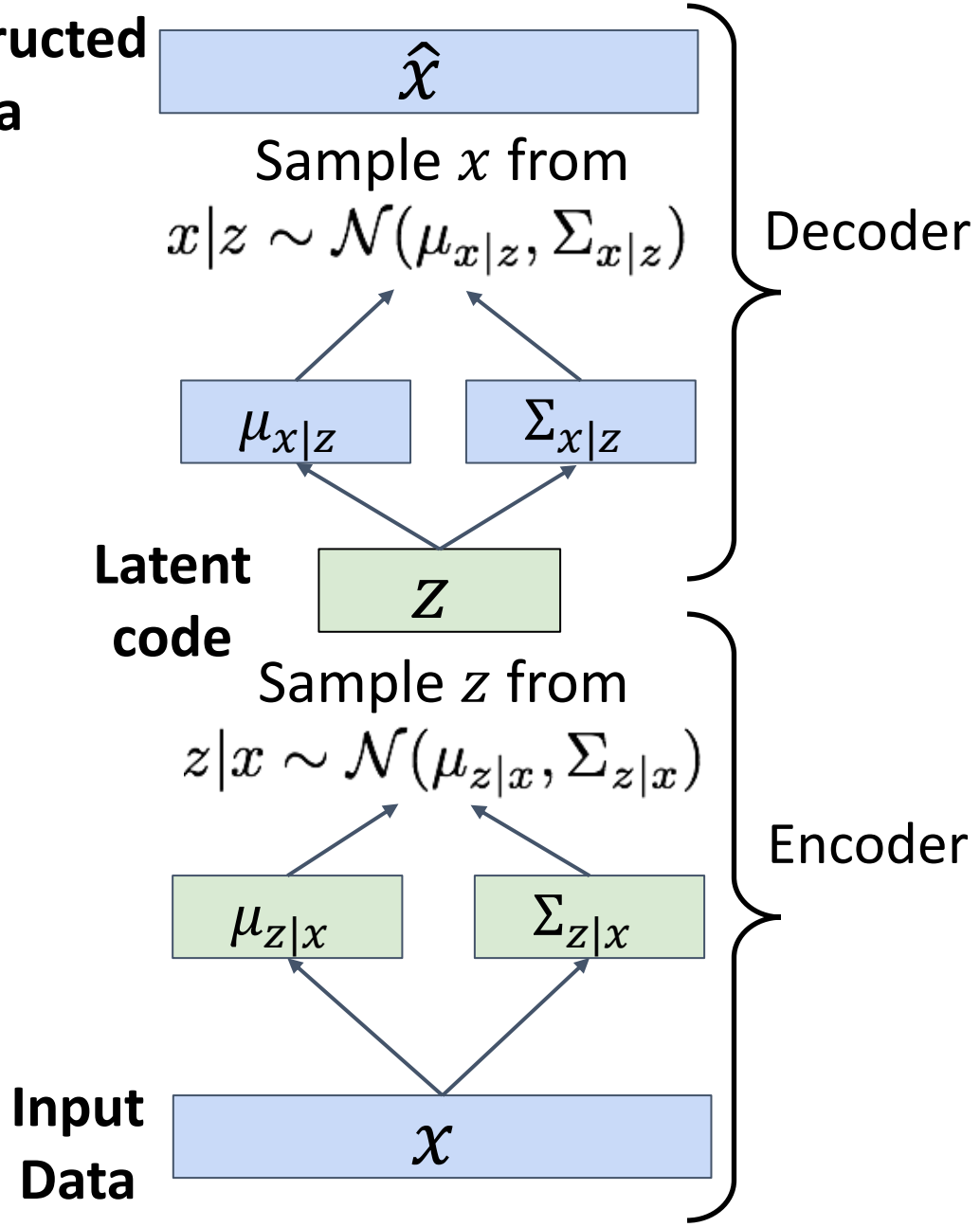


Variational Autoencoders

Train by maximizing the **variational lower bound**

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

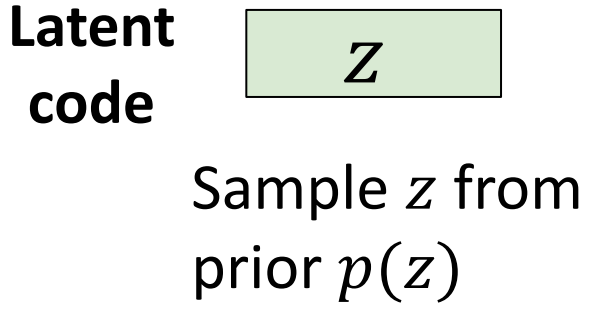
1. Run input data through **encoder** to get a distribution over latent codes
2. Encoder output should match the prior $p(z)$!
3. Sample code z from encoder output
4. Run sampled code through **decoder** to get a distribution over data samples
5. Original input data should be likely under the distribution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



Variational Autoencoders: Generating Data

After training we can generate new data!

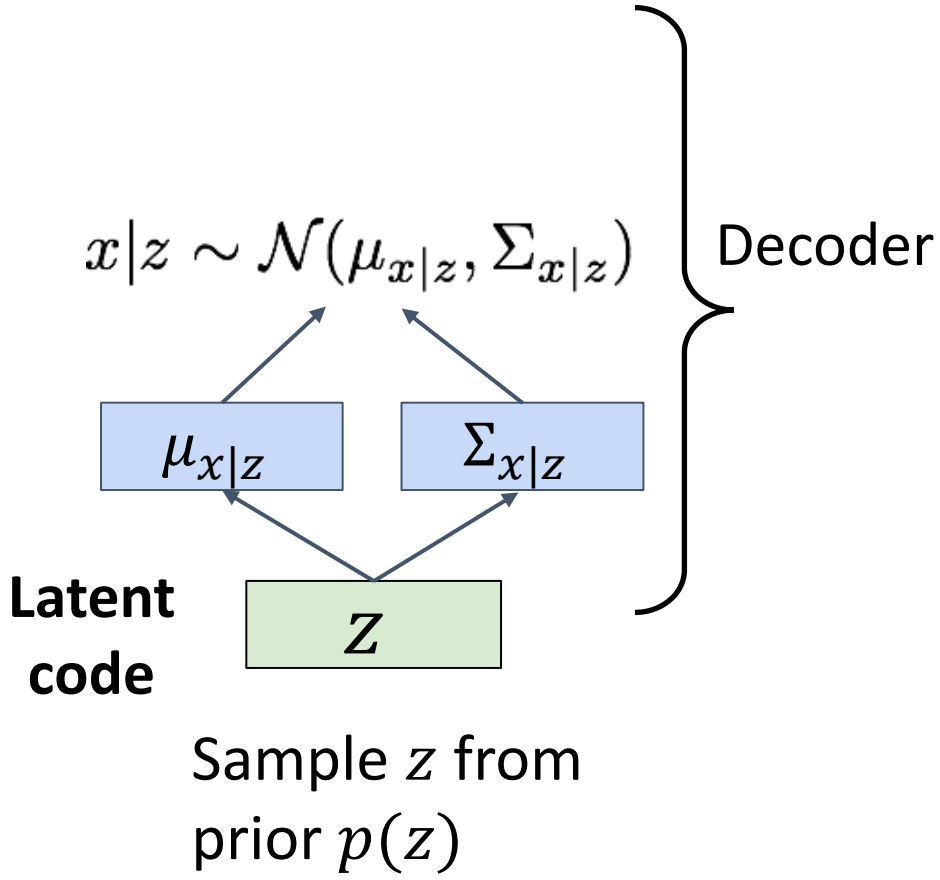
1. Sample z from prior $p(z)$



Variational Autoencoders: Generating Data

After training we can generate new data!

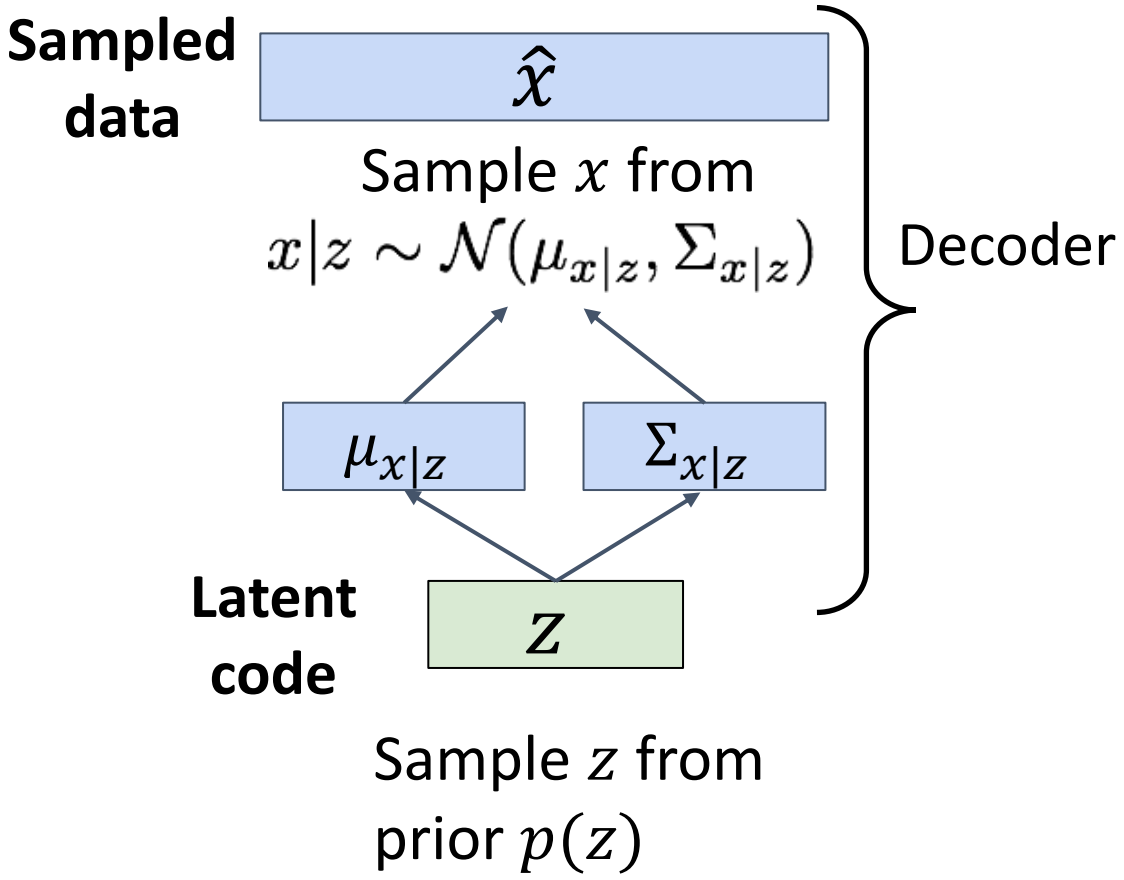
1. Sample z from prior $p(z)$
2. Run sampled z through decoder to get distribution over data x



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

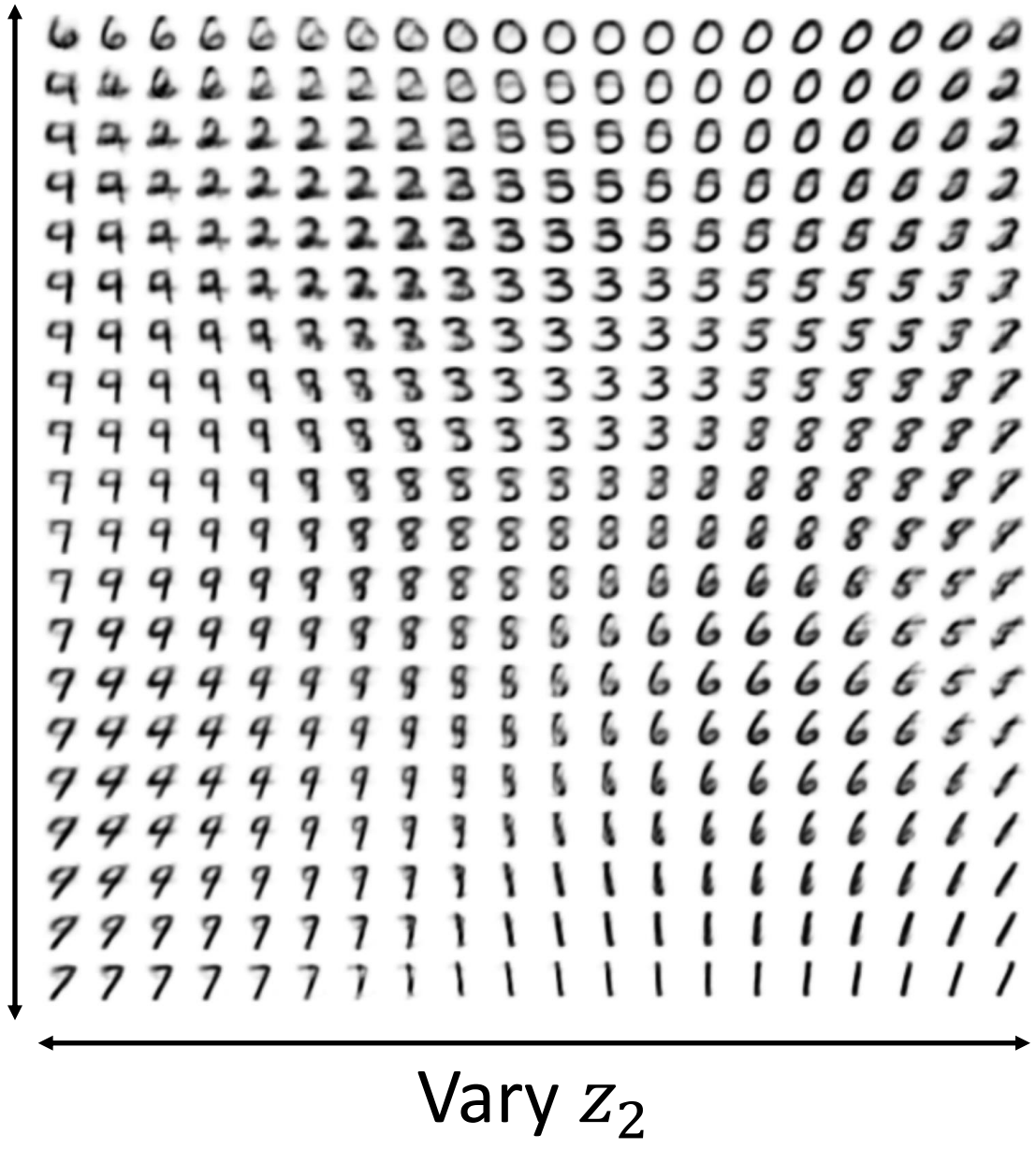
Labeled Faces in the Wild

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"

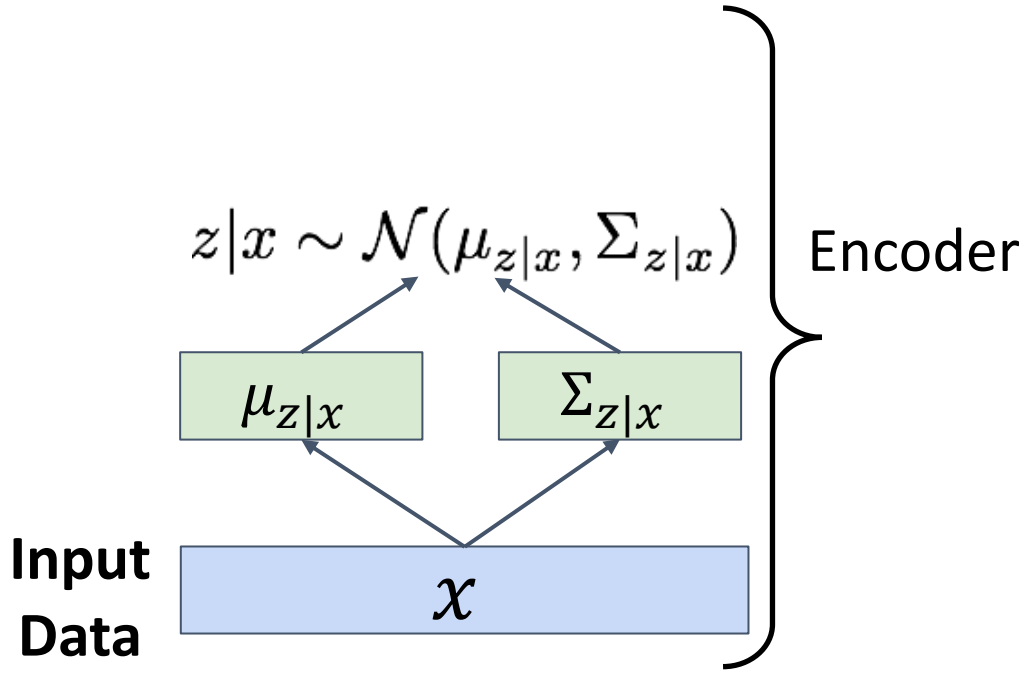


Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders

After training we can **edit images**

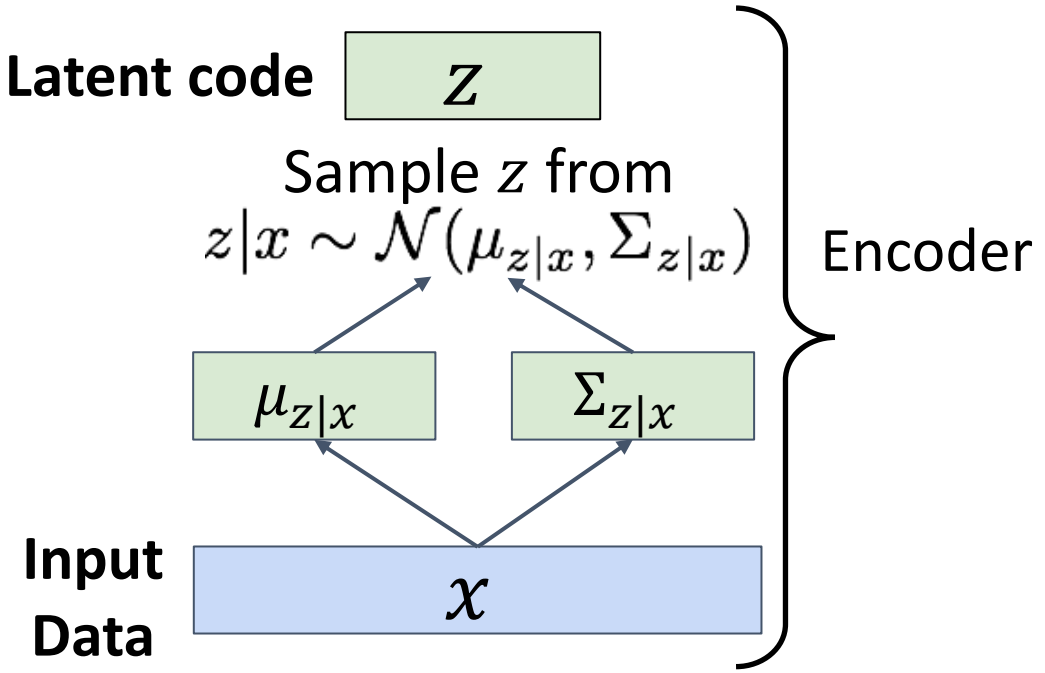
1. Run input data through **encoder** to get a distribution over latent codes



Variational Autoencoders

After training we can **edit images**

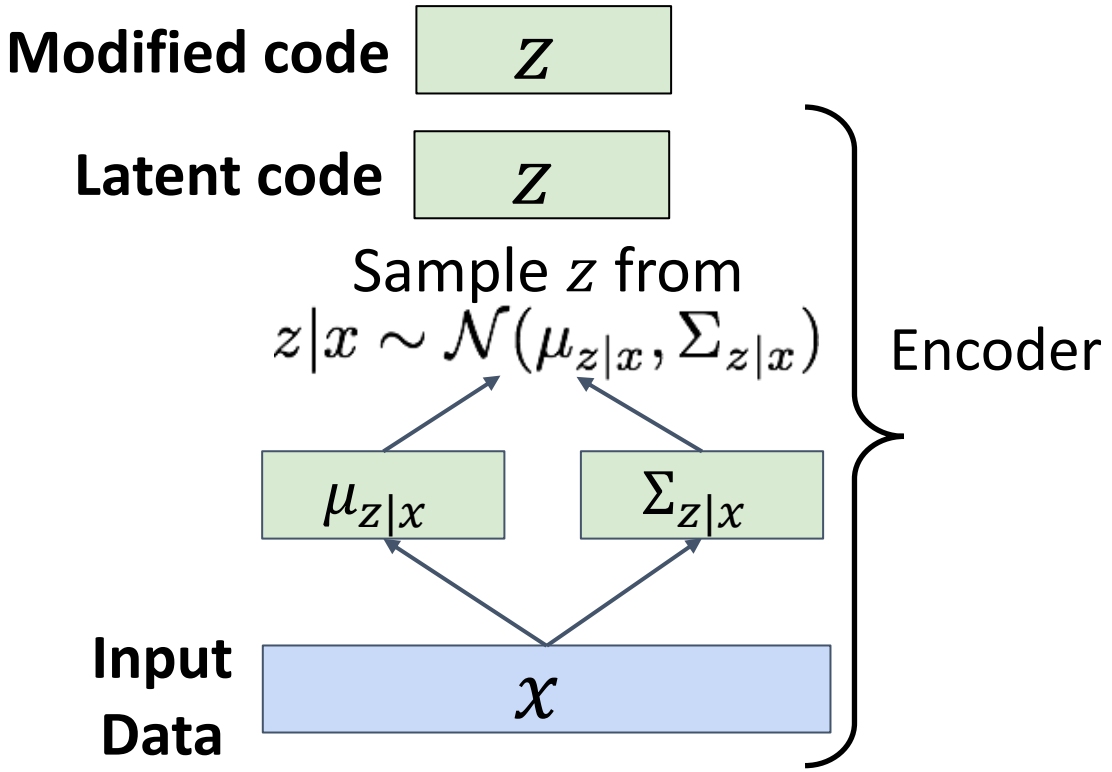
1. Run input data through **encoder** to get a distribution over latent codes
2. Sample code z from encoder output



Variational Autoencoders

After training we can **edit images**

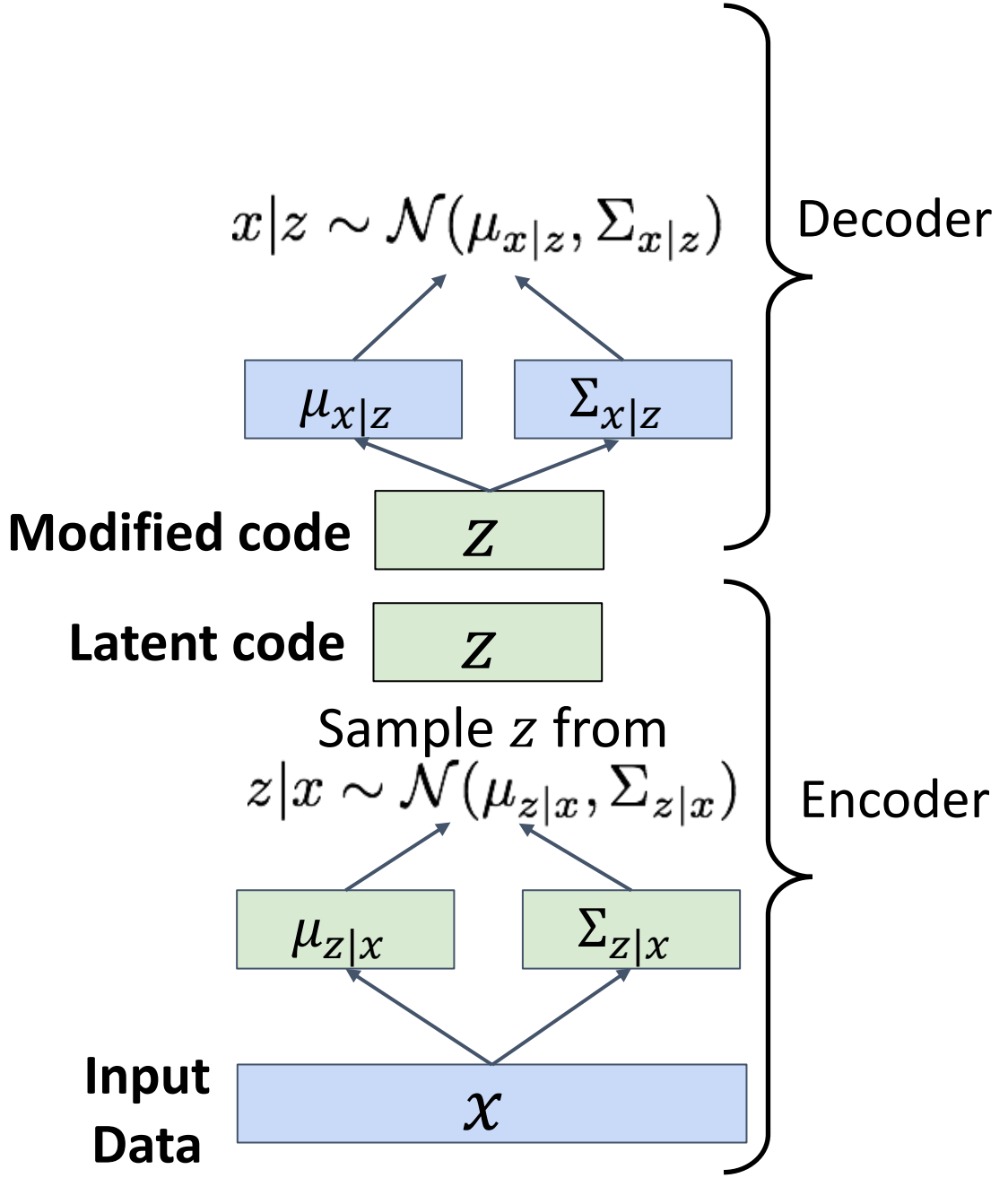
1. Run input data through **encoder** to get a distribution over latent codes
2. Sample code z from encoder output
3. Modify some dimensions of sampled code



Variational Autoencoders

After training we can **edit images**

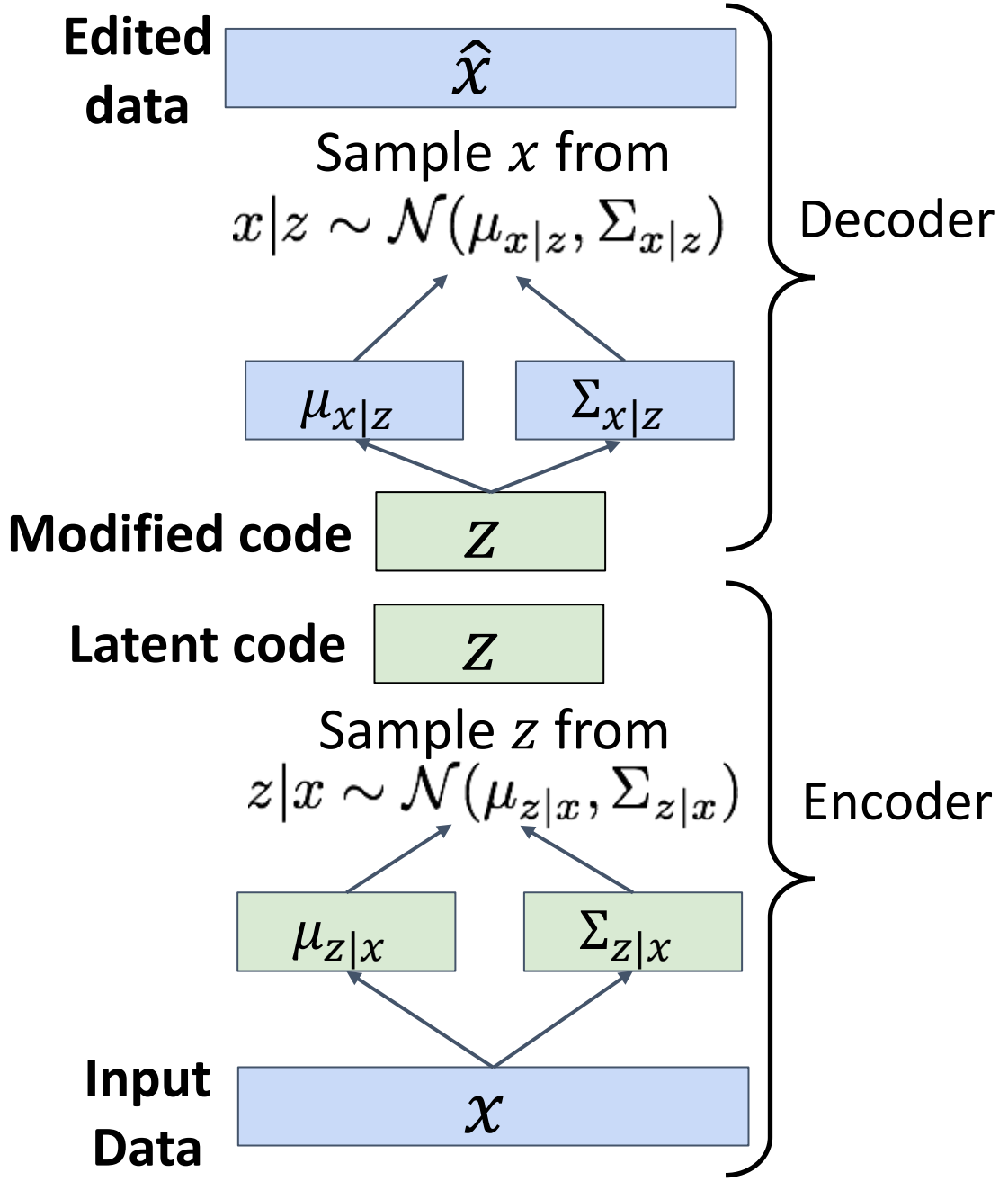
1. Run input data through **encoder** to get a distribution over latent codes
2. Sample code z from encoder output
3. Modify some dimensions of sampled code
4. Run modified z through **decoder** to get a distribution over data samples



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)



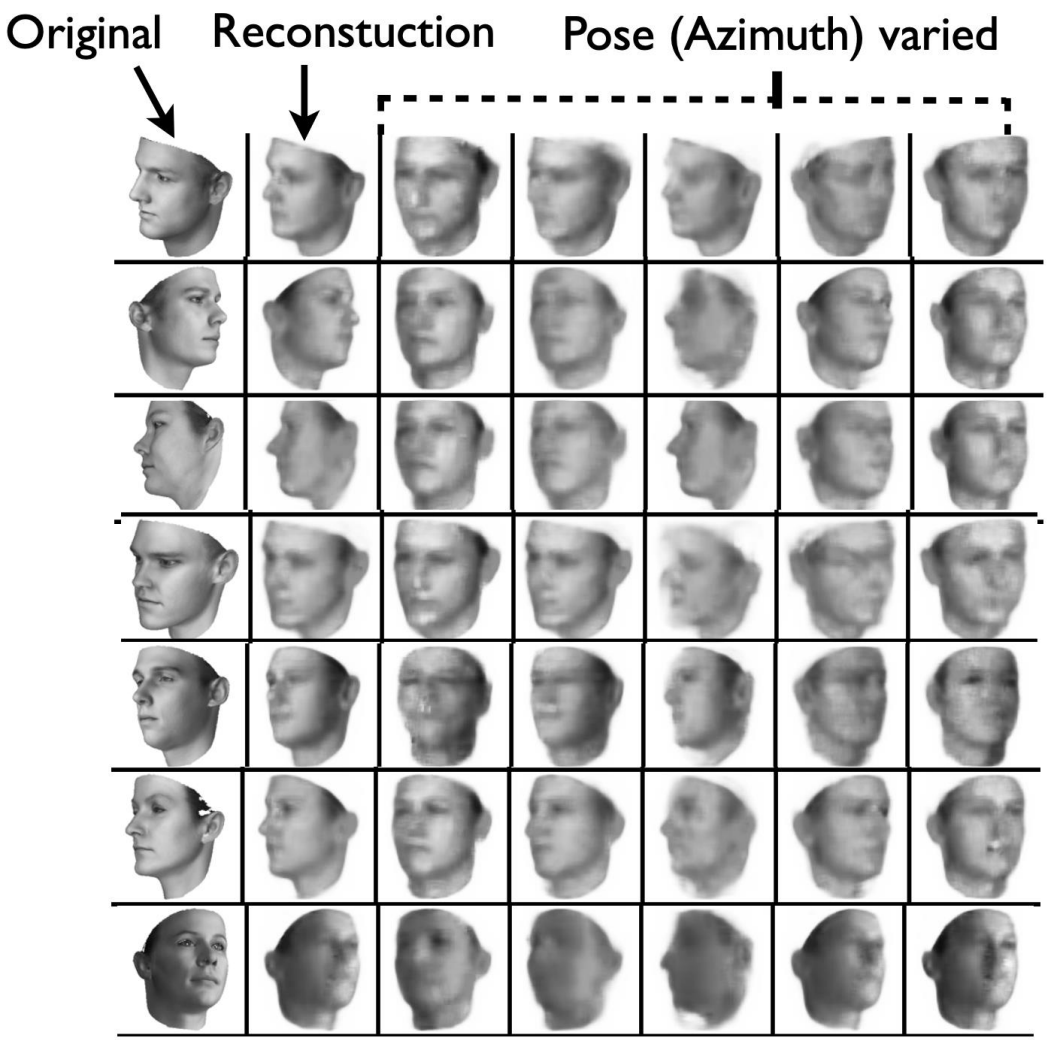
Variational Autoencoders

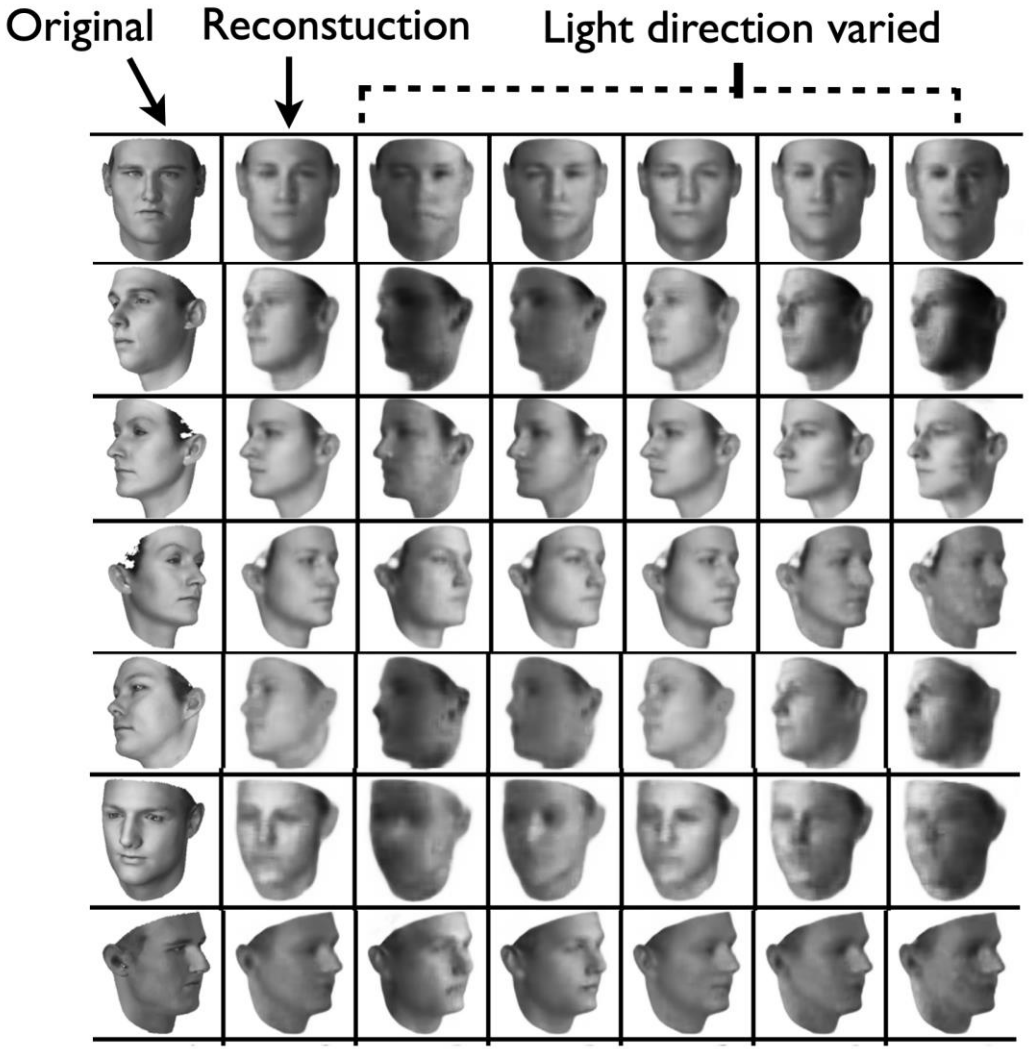
The diagonal prior on $p(z)$ causes dimensions of z to be independent

"Disentangling factors of variation"

Kingma and Welling, "Auto-Encoding Variational Bayes", 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
- 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

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

Text to Image: DALL-E 2

A rabbit detective sitting on a park bench and reading a newspaper in a victorian setting

A shark and a dolphin cruise hand-in-hand with an undersea city in the background

Robot dinosaurs versus monster trucks in the colosseum

Ramesh et al., "DALL·E 2." 2022. https://openai.com/dall-e-2/
Source: https://twitter.com/sama/status/1511724264629678084

Diffusion model (instead of VQ-VAE) + many other improvements!

Diffusion Models

Credit: CVPR 2022 Tutorial on
Denoising Diffusion-based Generative Modeling: Foundations and Applications
https://cvpr2022-tutorial-diffusion-models.github.io/

Denoising Diffusion Models

- Learning to generate by denoising
- Denoising diffusion models consist of two processes:
  - Forward diffusion process that gradually adds noise to input
  - Reverse denoising process that learns to generate data by denoising

Forward diffusion process (fixed)

Data

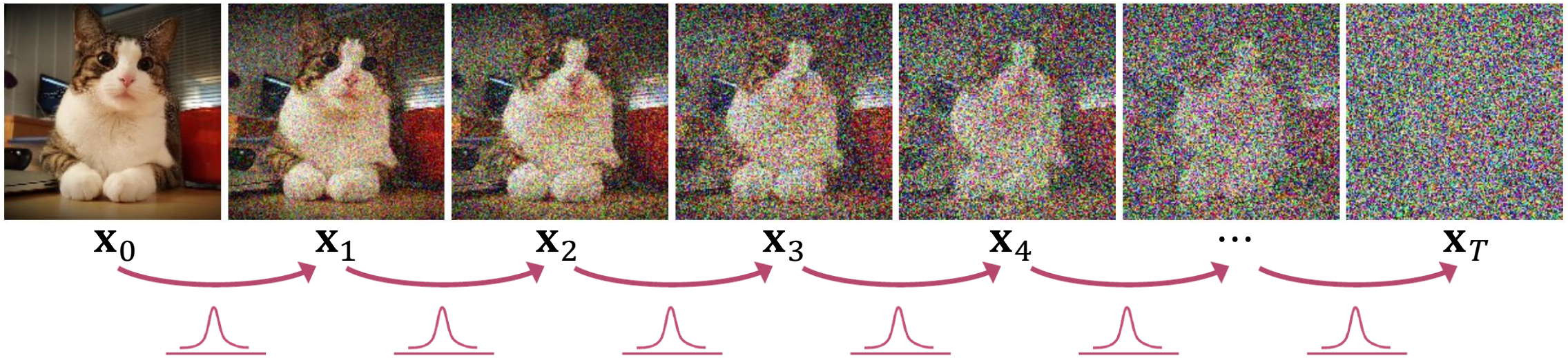
Noise

Reverse denoising process (generative)

Ho et al. “Denoising Diffusion Probabilistic Models”, NeurIPS 2020

Forward Diffusion Process

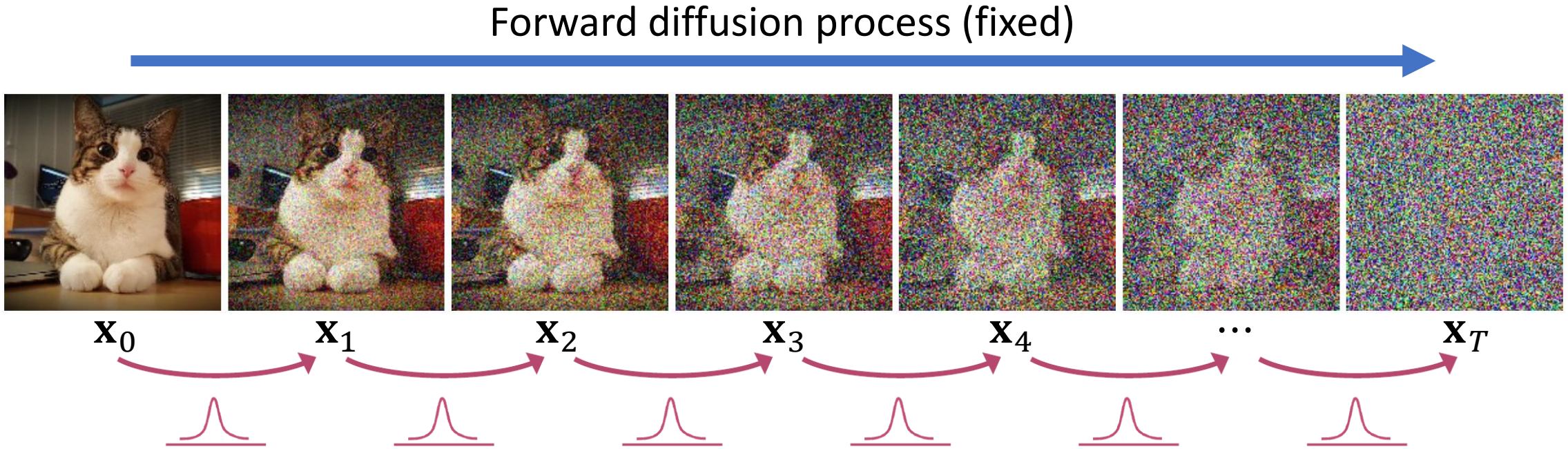
Forward diffusion process (fixed)



- Each forward step: $q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}\left(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t I\right)$
- Joint: $q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$
- With $\bar{\alpha}_t = \prod_{s=1}^{t} 1-\beta_s$, $q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)I\right)$

Ho et al. "Denoising Diffusion Probabilistic Models", NeurIPS 2020

Forward Diffusion Process

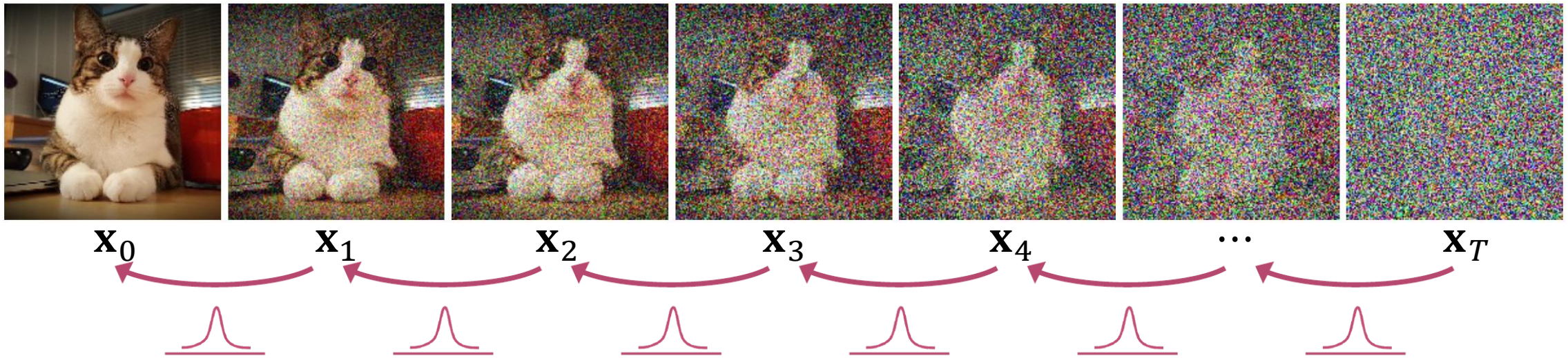


- With $\bar{\alpha}_t = \prod_{s=1}^{t} 1 - \beta_s$, $q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}\left(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)I\right)$
- Sampling: $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t} \cdot \varepsilon$ where $\varepsilon \sim \mathcal{N}(\mathbf{0}, I)$
- β_t is scheduled such that $\bar{\alpha}_T \to 0$ and $q(\mathbf{x}_T|\mathbf{x}_0) \simeq \mathcal{N}(\mathbf{x}_T; \mathbf{0}, I)$

Ho et al. “Denoising Diffusion Probabilistic Models”, NeurIPS 2020

Reverse Denoising Process

Reverse denoising process (generative)



- $p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, I)$
- $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \sigma_t^2 I)$

$$\Big\} \; p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$$

- $\mu_\theta(\mathbf{x}_t, t)$ is a learnable model (e.g., U-Net, denoising autoencoder)

Ho et al. "Denoising Diffusion Probabilistic Models", NeurIPS 2020

Learning Denoising Diffusion Models

- Variational lower bound:

$$\mathbb{E}_{q(\mathbf{x}_0)}[-\log p_\theta(\mathbf{x}_0)] \leq \mathbb{E}_{q(\mathbf{x}_0)q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] = L$$

- It can be decomposed as

$$L = \mathbb{E}_q\left[\underbrace{D_{\mathrm{KL}}\big(q(\mathbf{x}_T|\mathbf{x}_0) \,||\, p(\mathbf{x}_T)\big)}_{L_T} + \sum_{t>1} \underbrace{D_{\mathrm{KL}}\big(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0) \,||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)\big)}_{L_{t-1}} \underbrace{-\log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0}\right]$$

- where $q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)$ is the tractable posterior distribution:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0) = \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{x}_0)q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)} = \mathcal{N}\big(\mathbf{x}_{t-1};\, \tilde{\mu}_t(\mathbf{x}_t,\mathbf{x}_0), \tilde{\beta}_t I\big)$$

  - where $\tilde{\mu}_t(\mathbf{x}_t,\mathbf{x}_0) := \frac{\sqrt{\bar{\alpha}_t}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{1-\beta_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t$ and $\tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$

- With $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\cdot\varepsilon$, $\tilde{\mu}_t(\mathbf{x}_t,\mathbf{x}_0) = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\varepsilon\right)$

Ho et al. “Denoising Diffusion Probabilistic Models”, NeurIPS 2020

Learning Denoising Diffusion Models

$$L = \mathbb{E}_q \left[\underbrace{D_{\mathrm{KL}}\big(q(\mathbf{x}_T|\mathbf{x}_0) \,||\, p(\mathbf{x}_T)\big)}_{L_T} + \sum_{t>1} \underbrace{D_{\mathrm{KL}}\big(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \,||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)\big)}_{L_{t-1}} \underbrace{- \log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0} \right]$$

- Both $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ and $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ are Normal distributions:

$$L_{t-1} = D_{\mathrm{KL}}\big(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \,||\, p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)\big) = \mathbb{E}_q \left[\frac{1}{2\sigma_t^2} \|\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t, t)\|^2 \right] + C$$

- Recall that $\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{1}{\sqrt{1-\beta_t}} \left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \varepsilon \right)$
- With a noise-prediction network $\varepsilon_\theta(\mathbf{x}_t, t)$:

$$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{1-\beta_t}} \left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \varepsilon_\theta(\mathbf{x}_t, t) \right)$$

- With this parameterization:

$$L_{t-1} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \varepsilon \sim \mathcal{N}(\mathbf{0}, I)} \left[\frac{\beta_t^2}{2\sigma_t^2 (1-\beta_t)(1-\bar{\alpha}_t)} \left\| \varepsilon - \varepsilon_\theta\big(\underbrace{\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}}_{\mathbf{x}_t}, t \big) \right\|^2 \right] + C$$

Ho et al. “Denoising Diffusion Probabilistic Models”, NeurIPS 2020

Learning Denoising Diffusion Models

$$L_{t-1} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \varepsilon \sim \mathcal{N}(\mathbf{0}, I)} \left[\underbrace{\frac{\beta_t^2}{2\sigma_t^2(1-\beta_t)(1-\bar{\alpha}_t)}}_{\lambda_t} \left\| \varepsilon - \varepsilon_\theta \left(\underbrace{\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}}_{\mathbf{x}_t}, t \right) \right\|^2 \right] + C$$

- The time dependent $\lambda_t := \frac{\beta_t^2}{2\sigma_t^2(1-\beta_t)(1-\bar{\alpha}_t)}$ ensures that the training objective is weighted properly for the maximum data likelihood.
- But this is often large for small t's, and $\lambda_t = 1$ improves the sample quality (Ho et al., 2020).

$$L_{\text{simple}} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \varepsilon \sim \mathcal{N}(\mathbf{0}, I), t \sim \mathcal{U}(1, T)} \left[\left\| \varepsilon - \varepsilon_\theta \left(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}, t \right) \right\|^2 \right]$$

Ho et al. "Denoising Diffusion Probabilistic Models", NeurIPS 2020

Learning Denoising Diffusion Models

- Algorithms for Denoising Diffusion Probabilistic Models (DDPM)

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\underbrace{\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}}_{\mathbf{x}_t}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \underbrace{\frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right)}_{\mu_\theta(\mathbf{x}_t, t)} + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Ho et al. “Denoising Diffusion Probabilistic Models”, NeurIPS 2020

VAE vs. Diffusion Model

- Diffusion models can be considered as a kind of hierarchical VAEs.

- However, diffusion models are different from VAEs in that:
  - The encoder is fixed. (encoding == noise injection)
  - The latent variables have the same dimension as the data.
  - The denoising model is shared across different timestep.
  - The model is trained with some reweighting of the variational bound.

Advanced Diffusion Models

- Denoising Diffusion Probabilistic Models (DDPM, NeurIPS 2020)
- Denoising Diffusion Implicit Models (DDIM, ICLR 2021)
- Diffusion via Stochastic Differential Equations (Score SDE; ICLR 2021)
- Cascaded Diffusion Models (CDM; JMLR 2023)
- Latent diffusion model (LDM; CVPR 2022)
- Progressive Distillation (ICLR 2022)
- Diffusion Transformer (DiT; ICCV 2023)
- Consistency Models (NeurIPS 2023)
- …
- Diffusion models are state-of-the-art generative models!

DALL-E 2: Text-to-Image Generation

A rabbit detective sitting on a park bench and reading a newspaper in a victorian setting

A shark and a dolphin cruise hand-in-hand with an undersea city in the background

Robot dinosaurs versus monster trucks in the colosseum

Ramesh et al., "DALL·E 2." 2022. https://openai.com/dall-e-2/
Source: https://twitter.com/sama/status/1511724264629678084

Diffusion model (instead of VQ-VAE) + many other improvements!

DALL-E 3: Image Generation with Recaptioning

A rabbit detective sitting on a park bench and reading a newspaper in a victorian setting

A shark and a dolphin cruise hand-in-hand with an undersea city in the background

Robot dinosaurs versus monster trucks in the colosseum

Betker et al., “Improving Image Generation with Better Captions.” 2023.
Source: Kibok Lee subscribing ChatGPT Plus

Generative Adversarial Networks

Generative Adversarial Networks

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

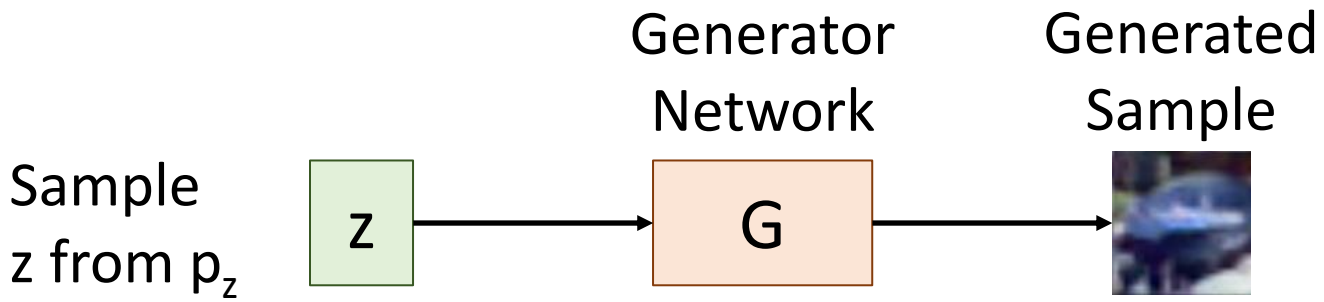
Idea: Introduce a latent variable z with simple prior p(z).
Sample $z \sim p(z)$ and pass to a **Generator Network** x = G(z)
Then x is a sample from the **Generator distribution** p_G. Want $p_G = p_{data}$!

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

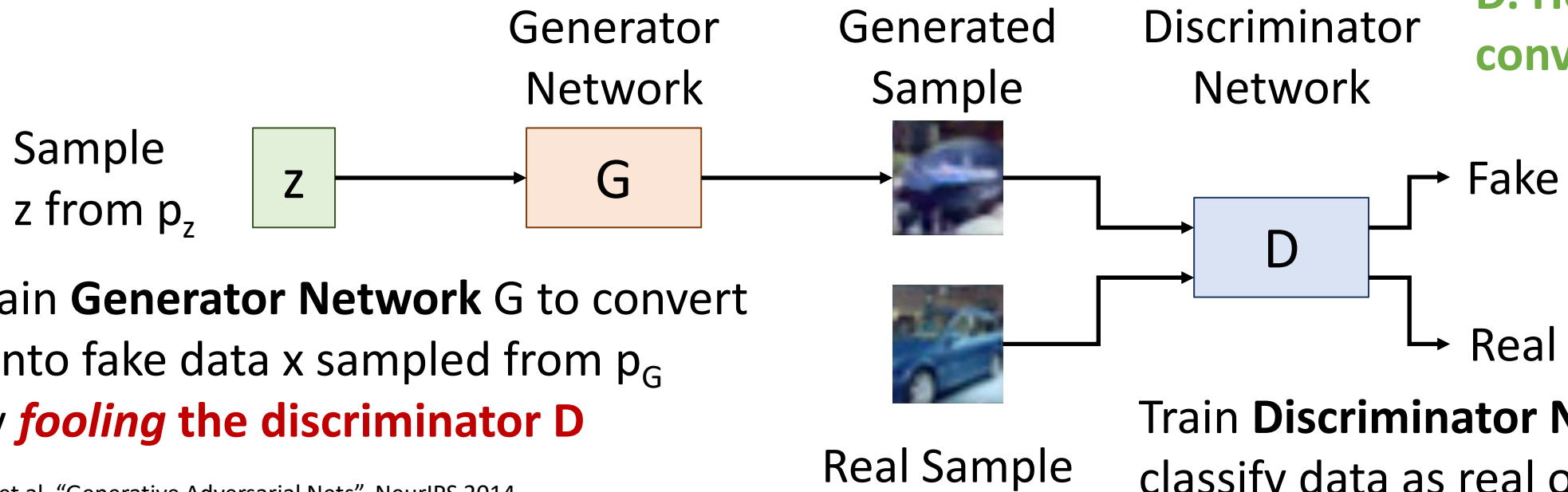
Goodfellow et al, "Generative Adversarial Nets", NeurIPS 2014

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 $p_G = p_{data}$!

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

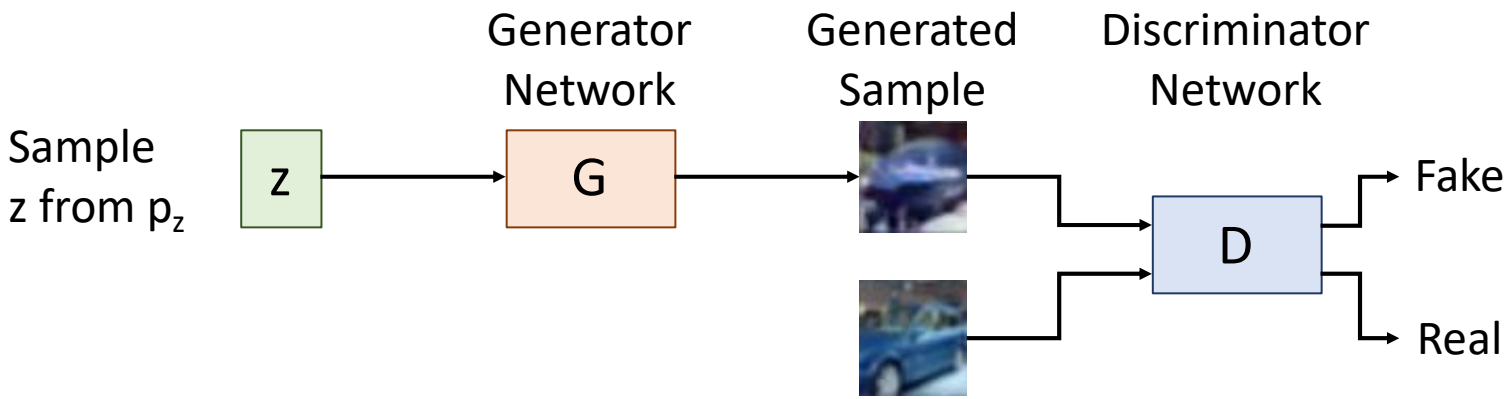
Jointly train generator G and discriminator D with a **minimax game**

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

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

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

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

For t in 1, … T:

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

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

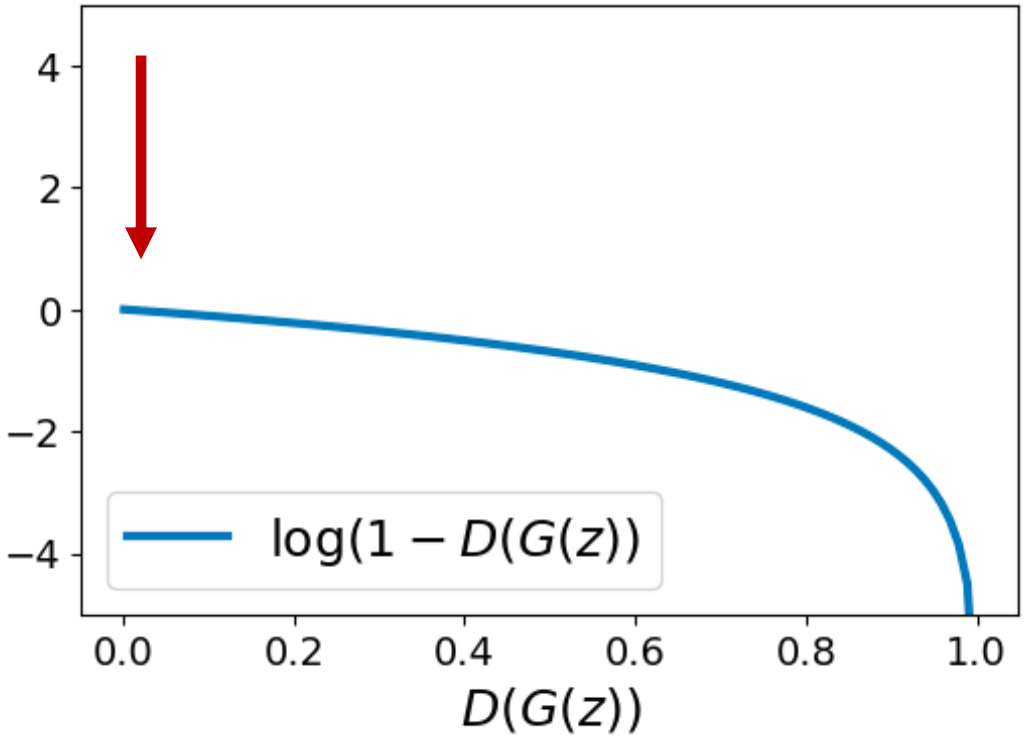
Generative Adversarial Networks: Training Objective

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



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

Generative Adversarial Networks: Training Objective

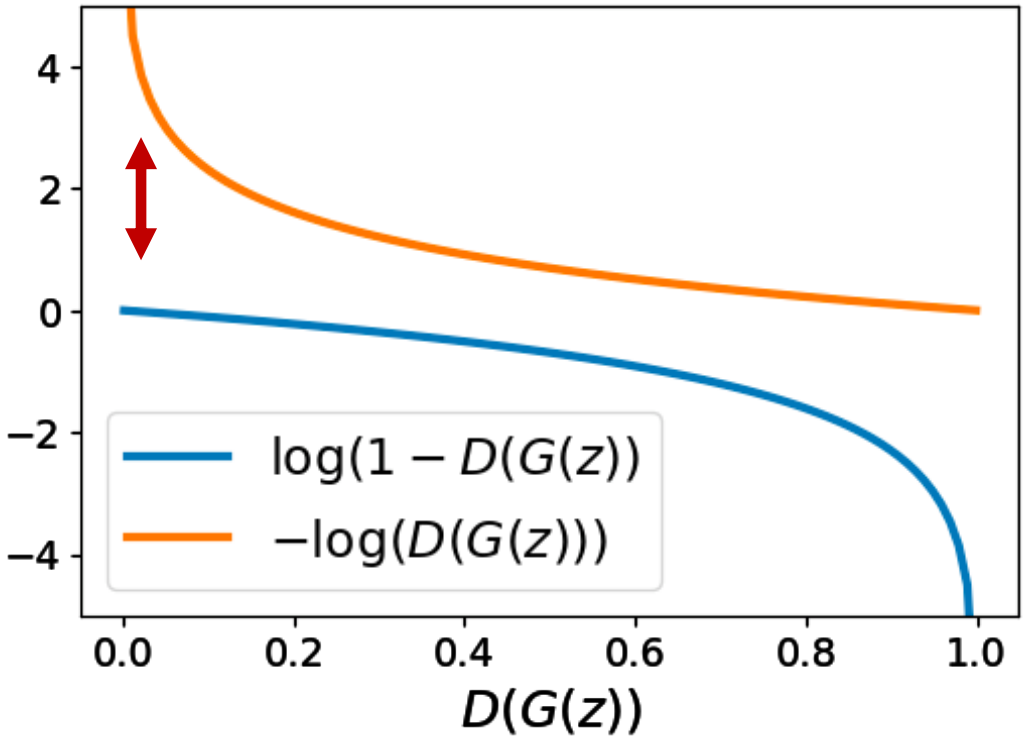
Jointly train generator G and discriminator D with a **minimax game**

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

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

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

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

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

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

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

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

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

(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

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

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

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

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

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

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

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

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

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

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

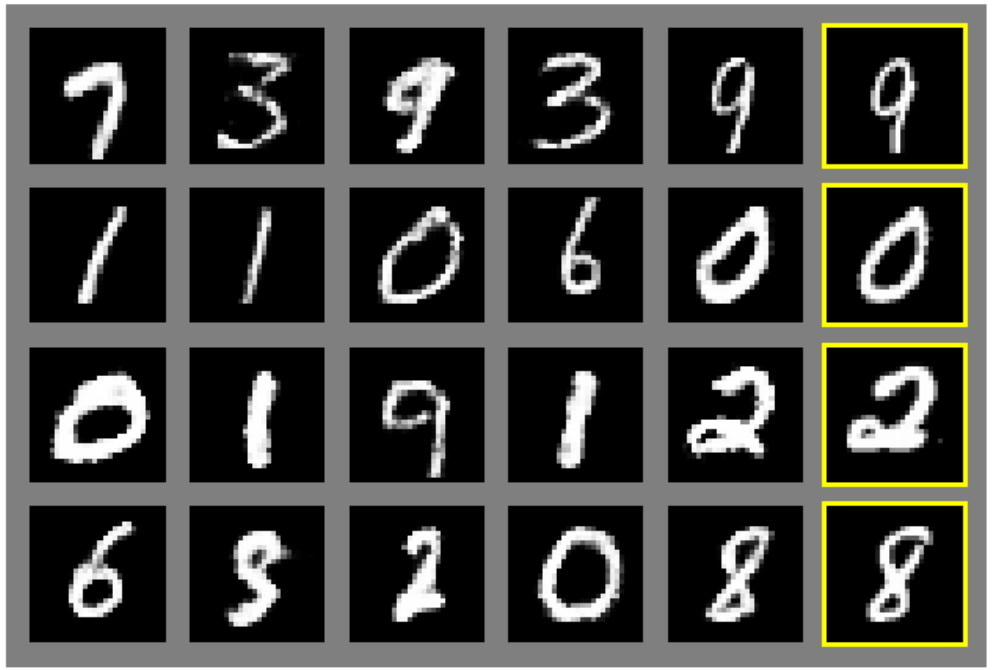
1. $D_G^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$ (Optimal discriminator for any G)
2. $p_G(x) = p_{data}(x)$ (Optimal generator for optimal D)

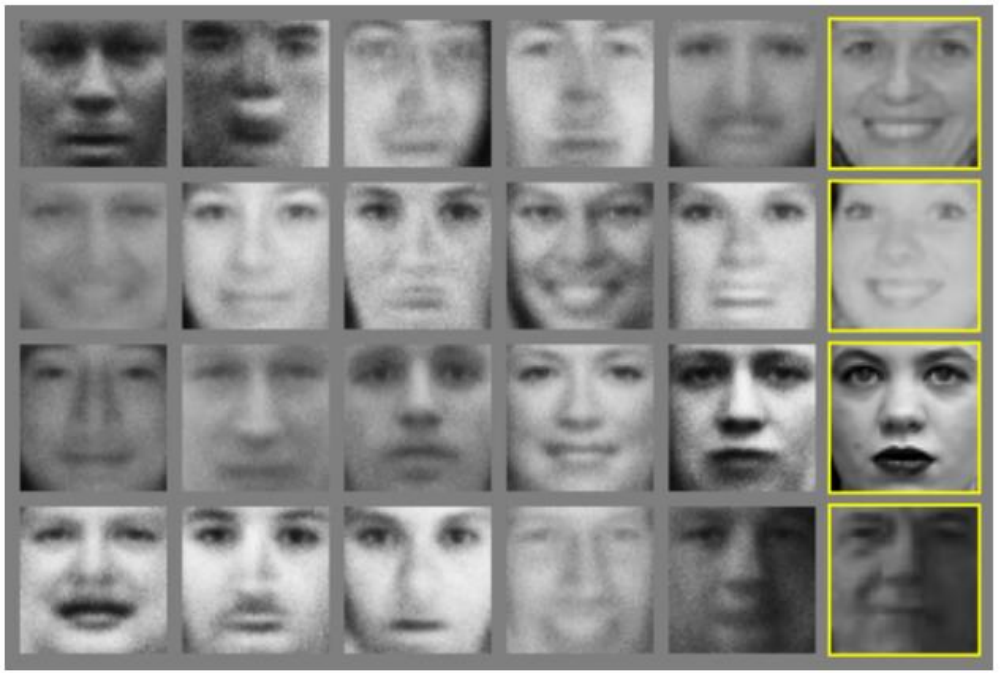
Caveats:

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

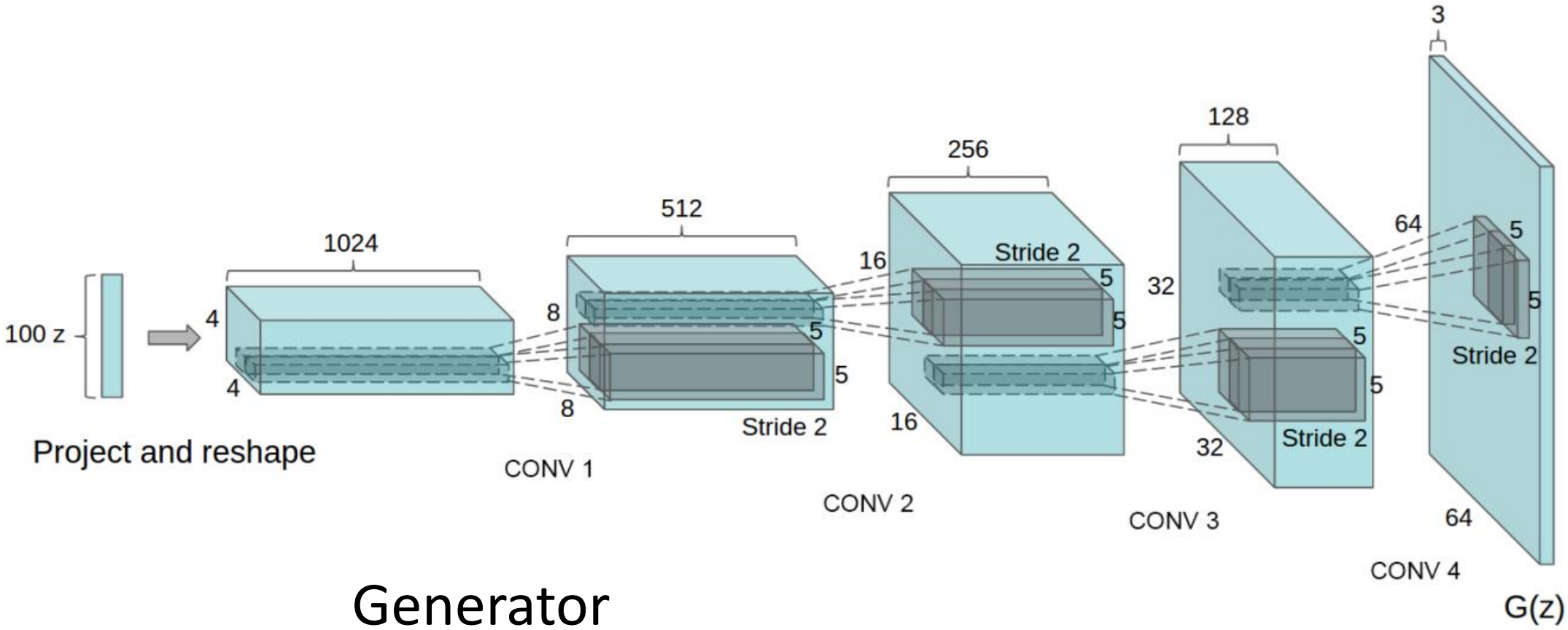




Nearest neighbor from training set

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

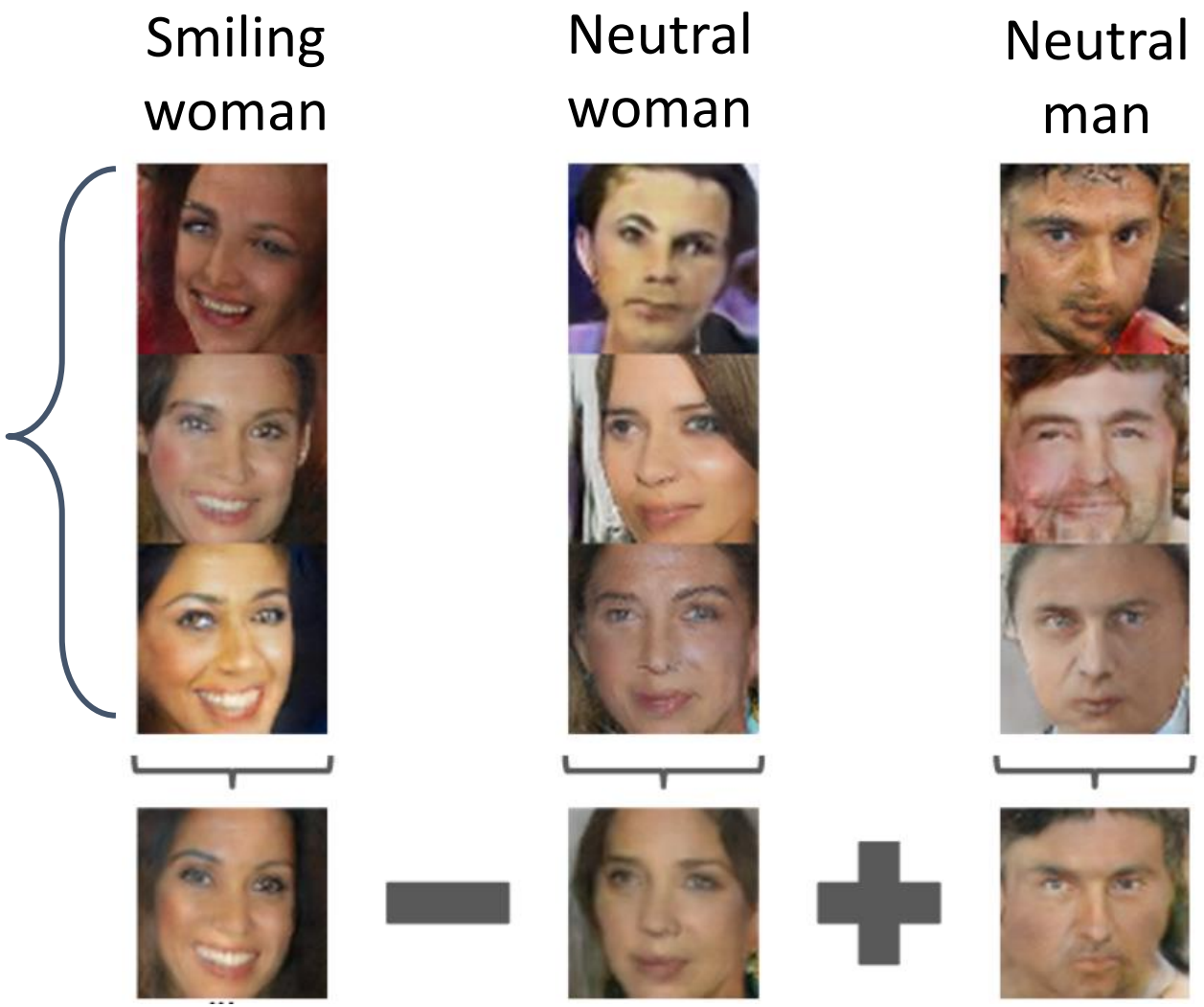
Neutral woman

Neutral man

Samples from the model

Radford et al, ICLR 2016

Generative Adversarial Networks: Vector Math



Radford et al, ICLR 2016

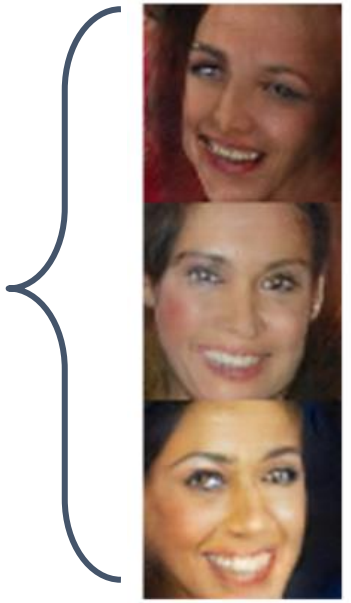
Generative Adversarial Networks: Vector Math

Smiling woman

Neutral woman

Neutral man

Samples from the model

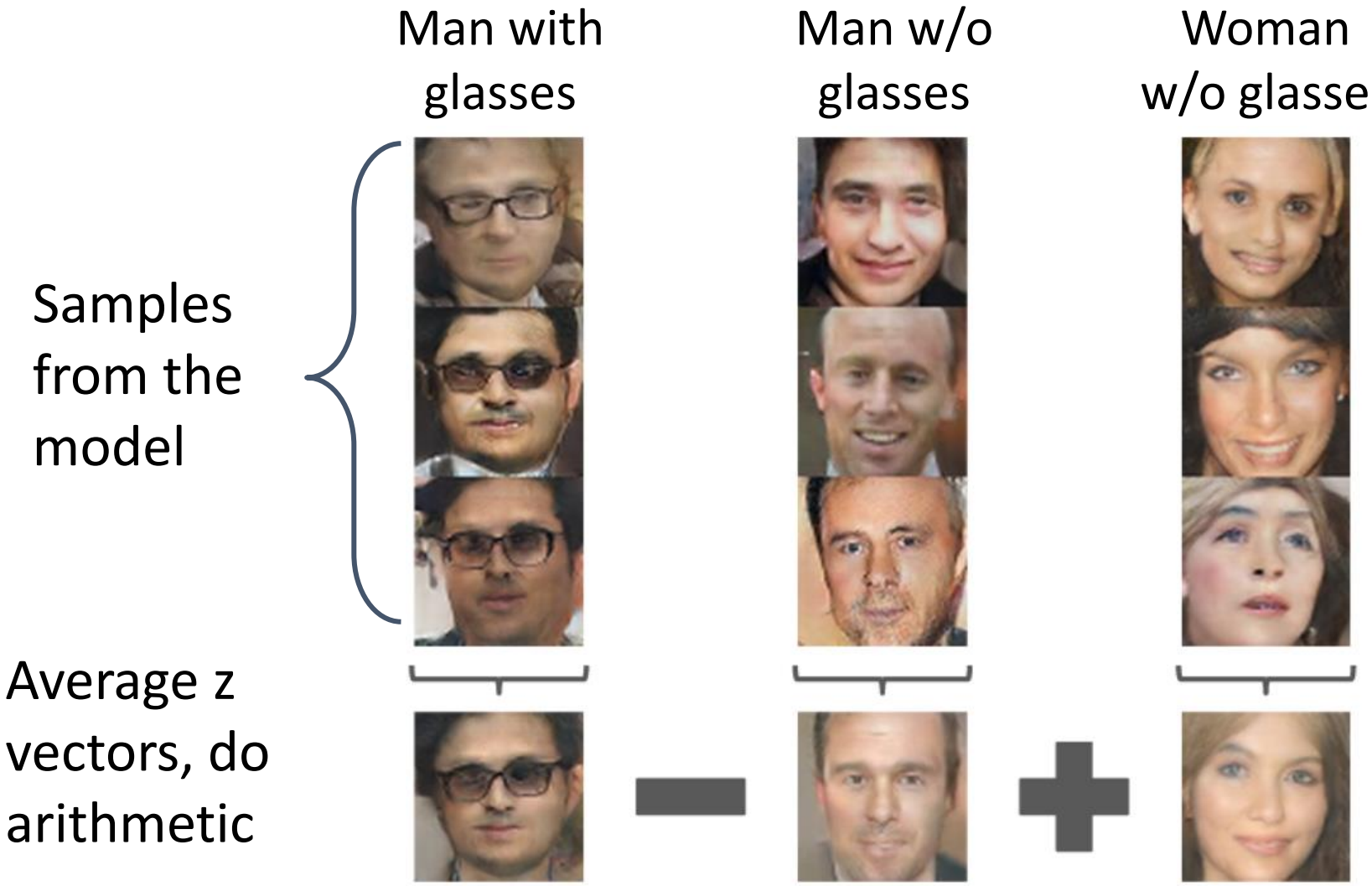


Average z vectors, do arithmetic

Smiling Man

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

−

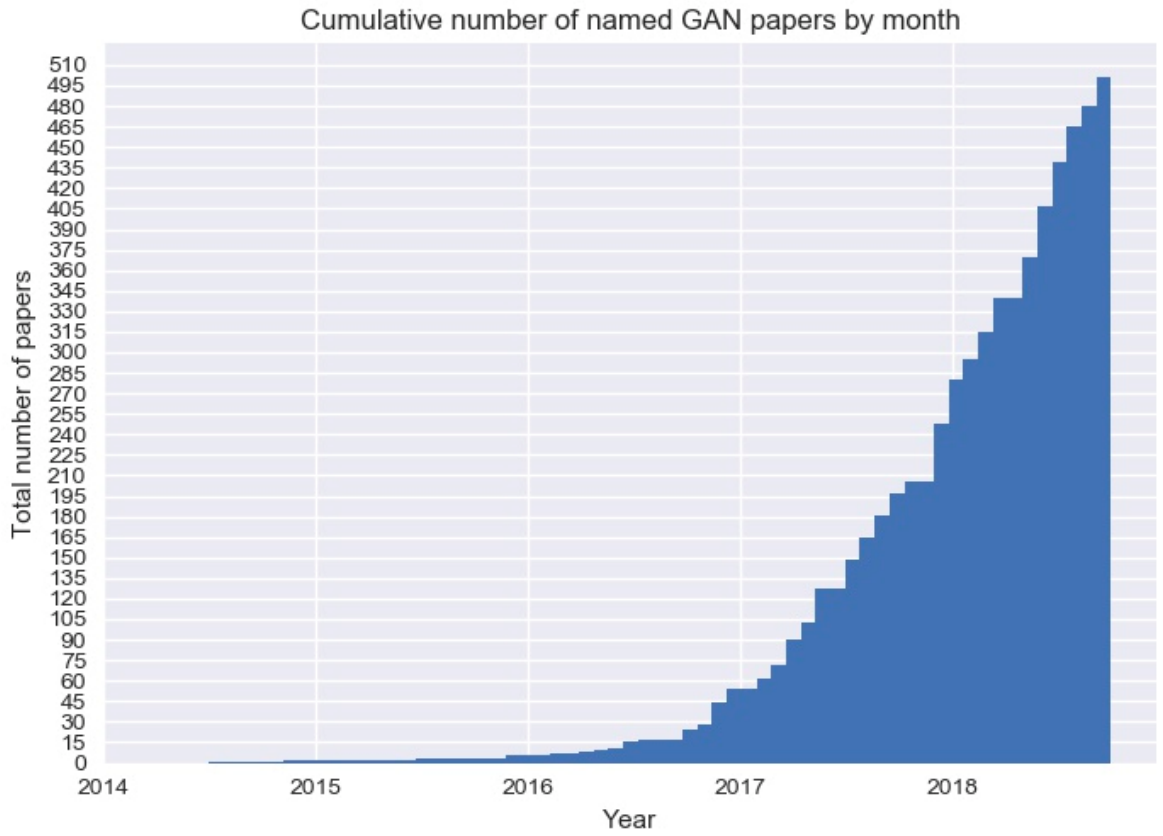
+

=

Woman with glasses

Radford et al, ICLR 2016

2017 to present: Explosion of GANs



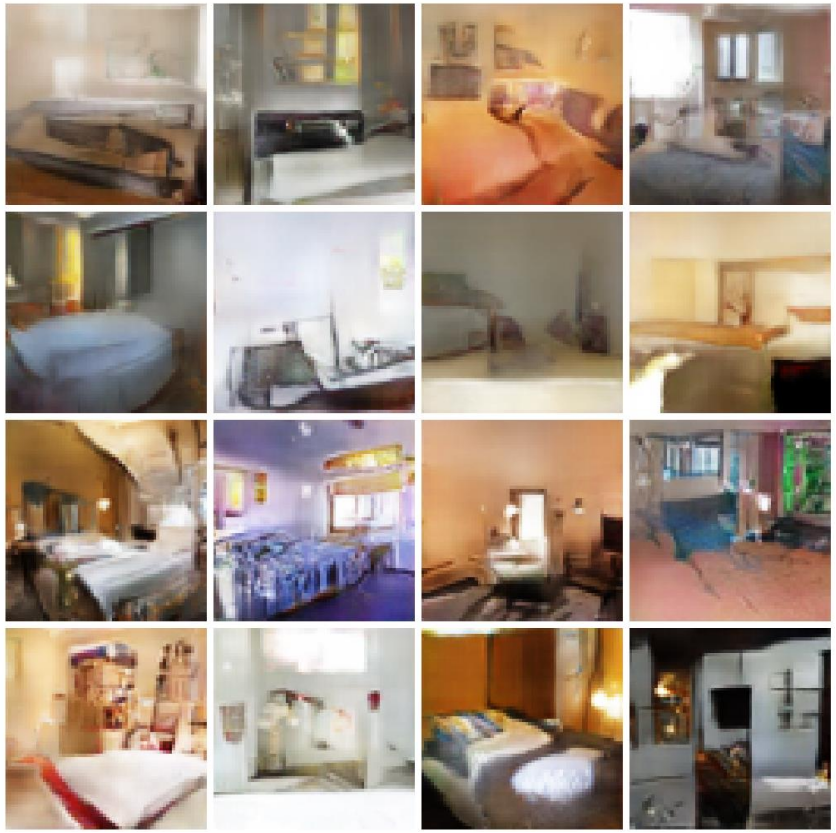
https://github.com/hindupuravinash/the-gan-zoo

- 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: 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

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

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!

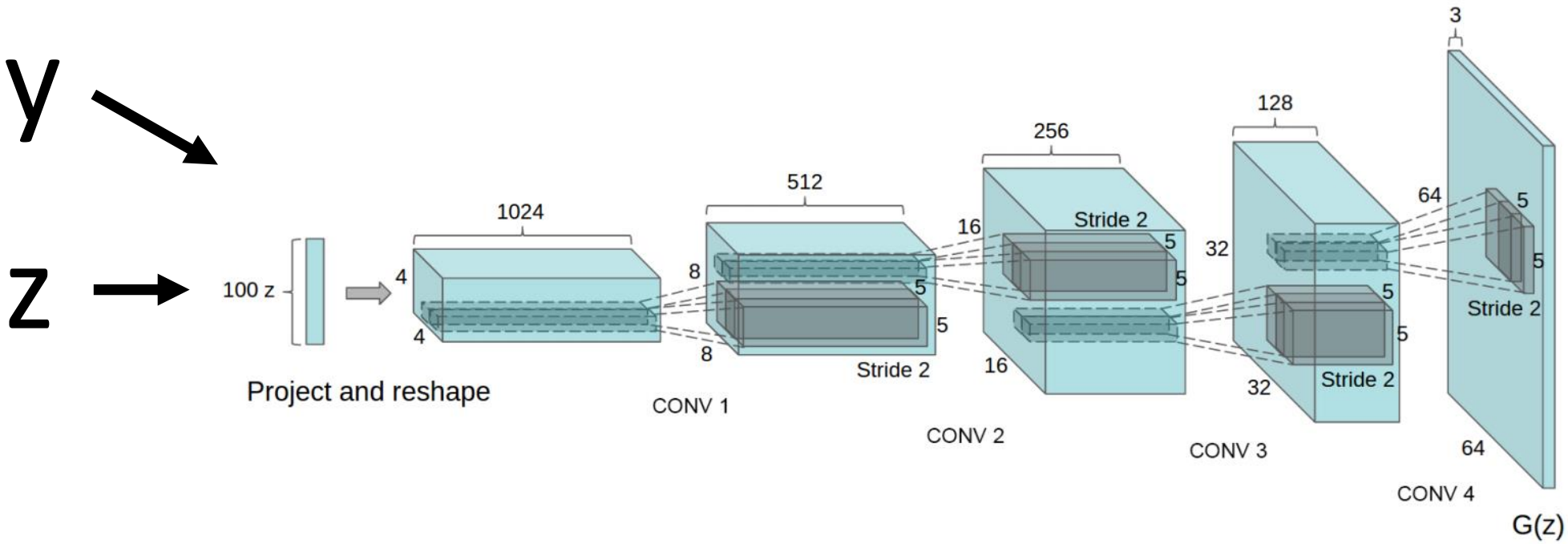


Figure credit: Radford et al, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", ICLR 2016

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

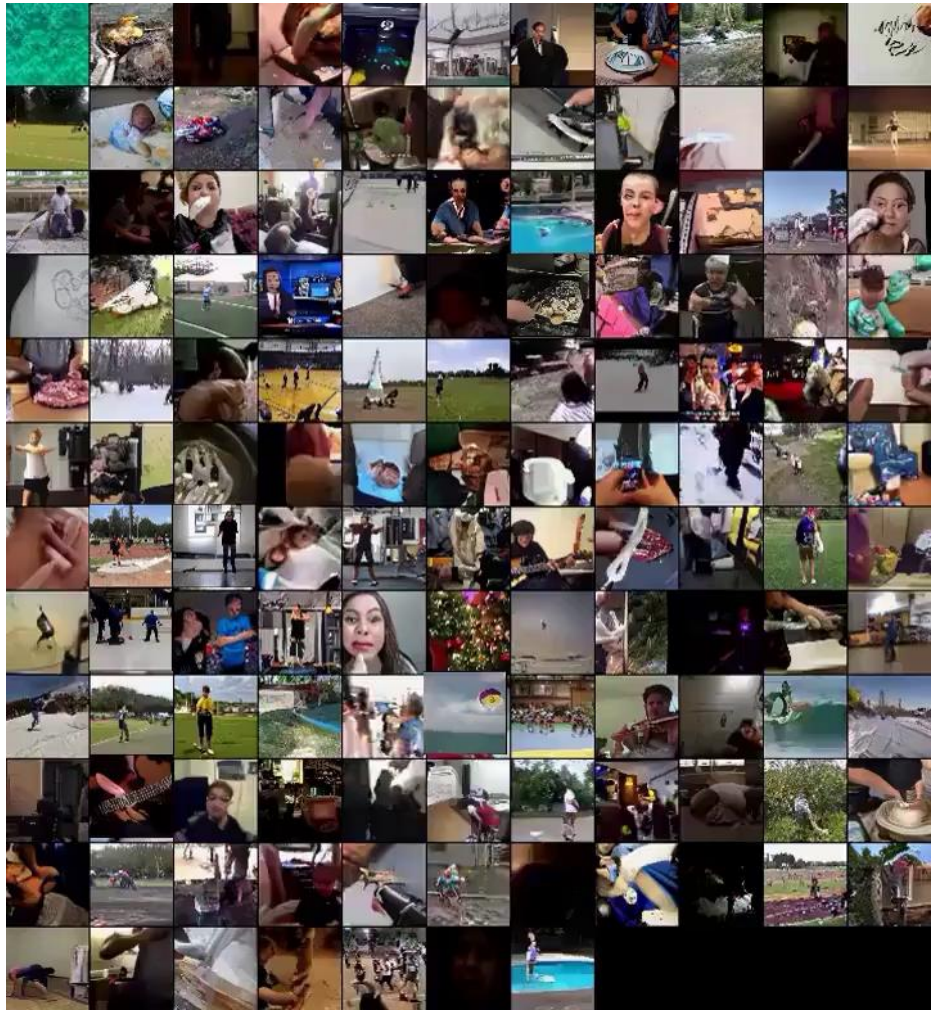
Conditional GANs: Spectral Normalization

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

128x128 images on ImageNet

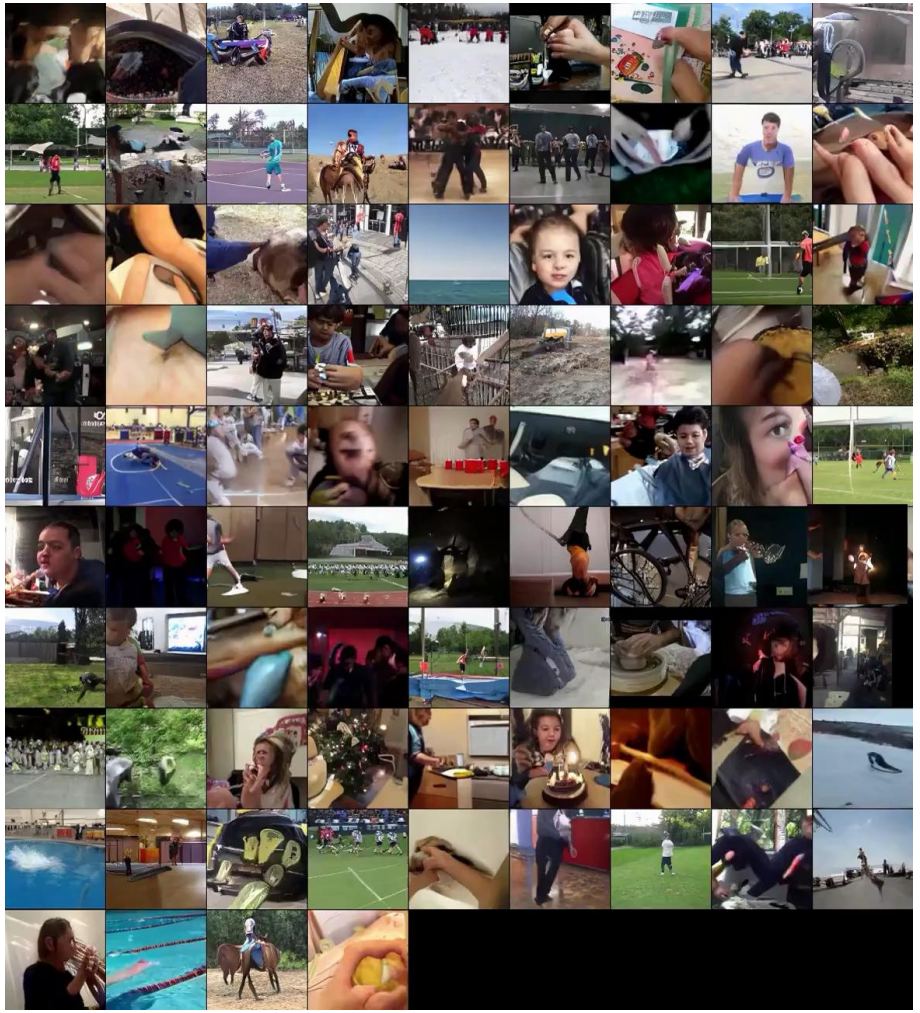
Generating Videos with GANs

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



64x64 images, 48 frames

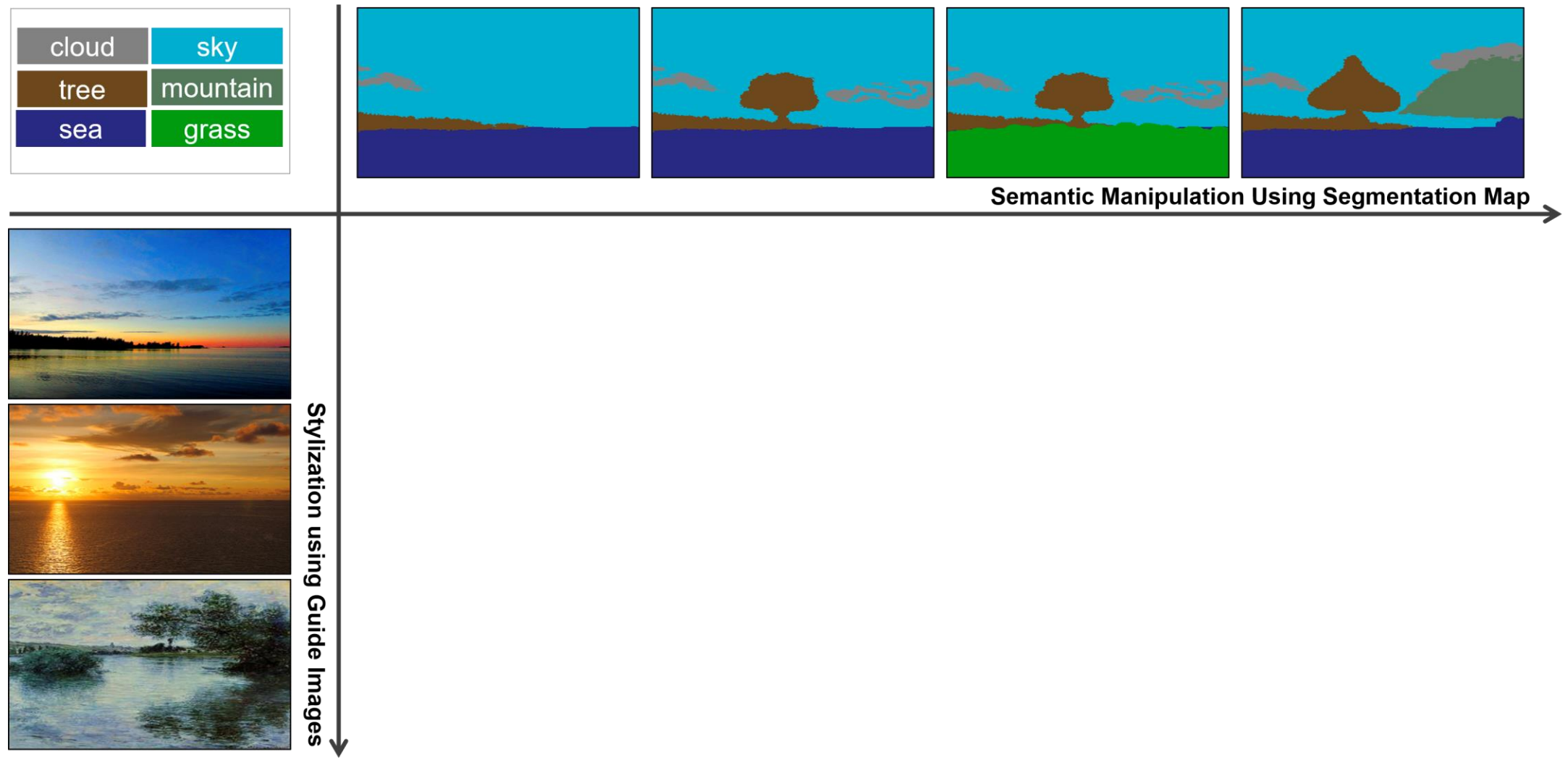
https://drive.google.com/file/d/1FjOQYdUuxPXvS8yeOhXdPQMapUQaklLi/view



128x128 images, 12 frames

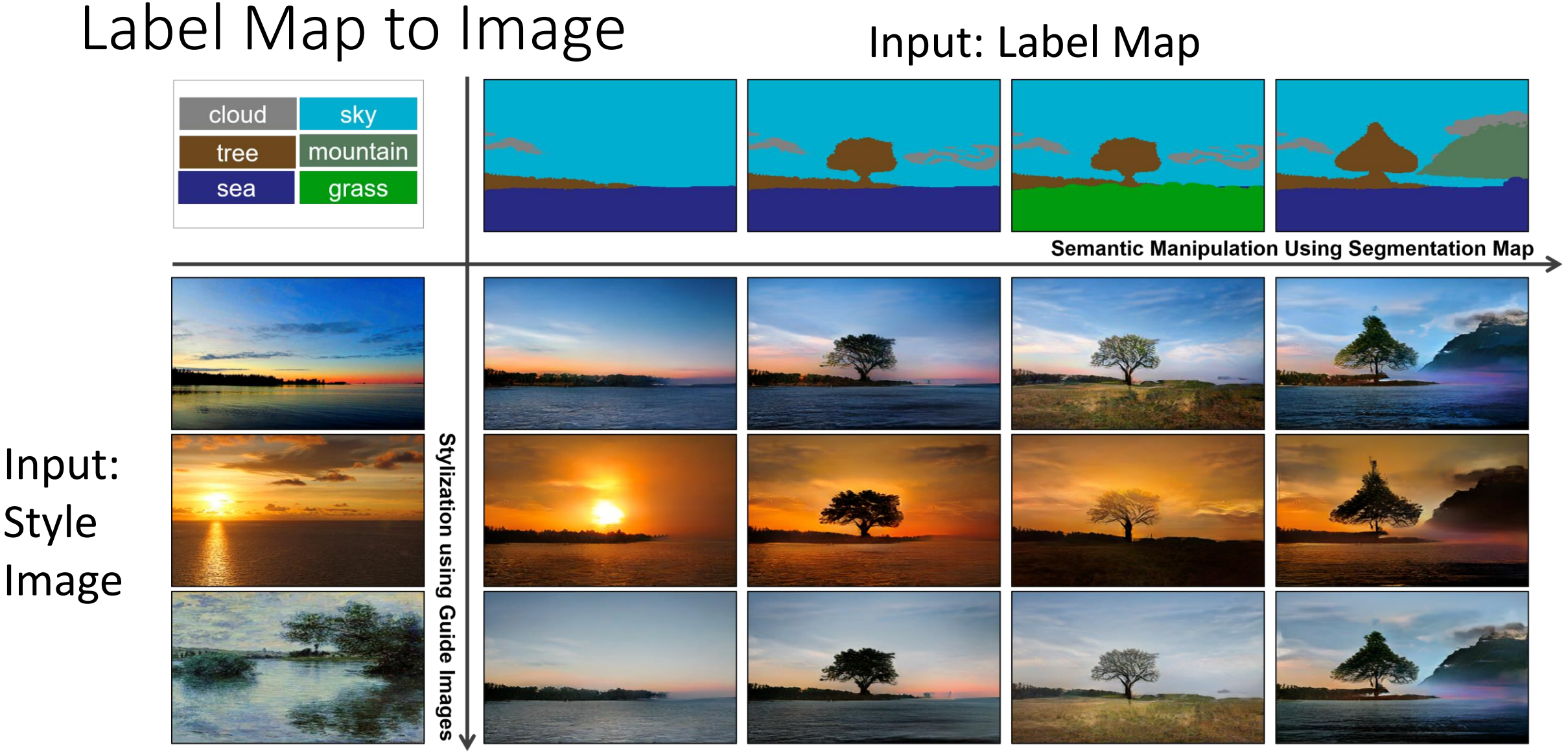
https://drive.google.com/file/d/165Yxuvvu3viOy-39LhhSDGtczbWphj_i/view

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

Text-to-Image Synthesis Using GAN

This bird is red and brown in color, with a stubby beak

The bird is short and stubby with yellow on its body

A bird with a medium orange bill white body gray wings and webbed feet

This small black bird has a short, slightly curved bill and long legs

A picture of a very clean living room

A group of people on skis stand in the snow

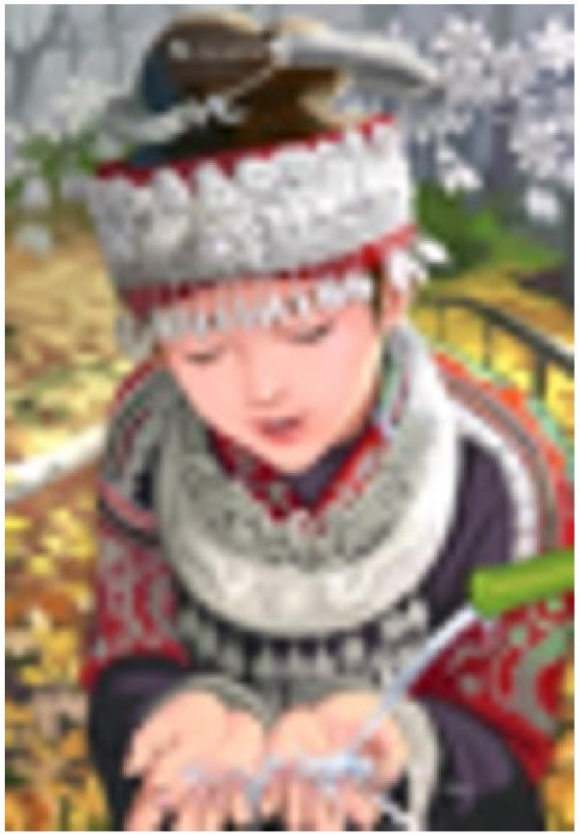
Eggs fruit candy nuts and meat served on white dish

A street sign on a stoplight pole in the middle of a day

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

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

bicubic
(21.59dB/0.6423)



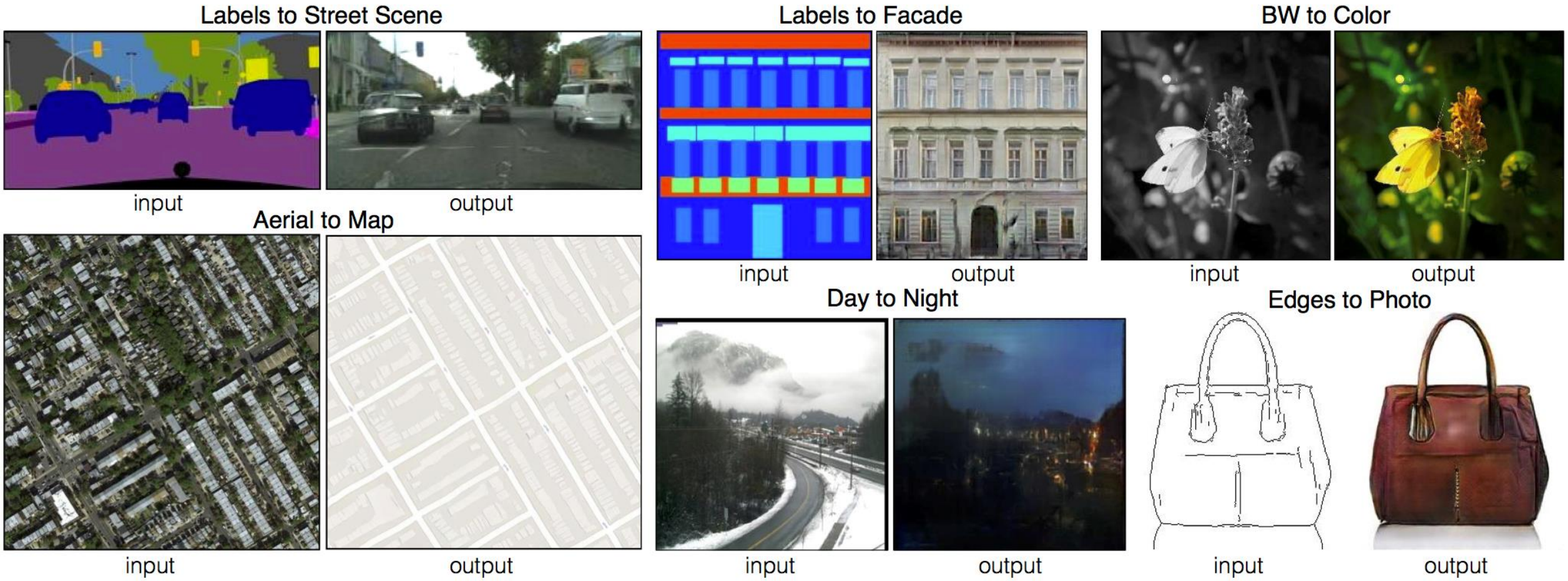
SRResNet
(23.53dB/0.7832)

SRGAN
(21.15dB/0.6868)

original

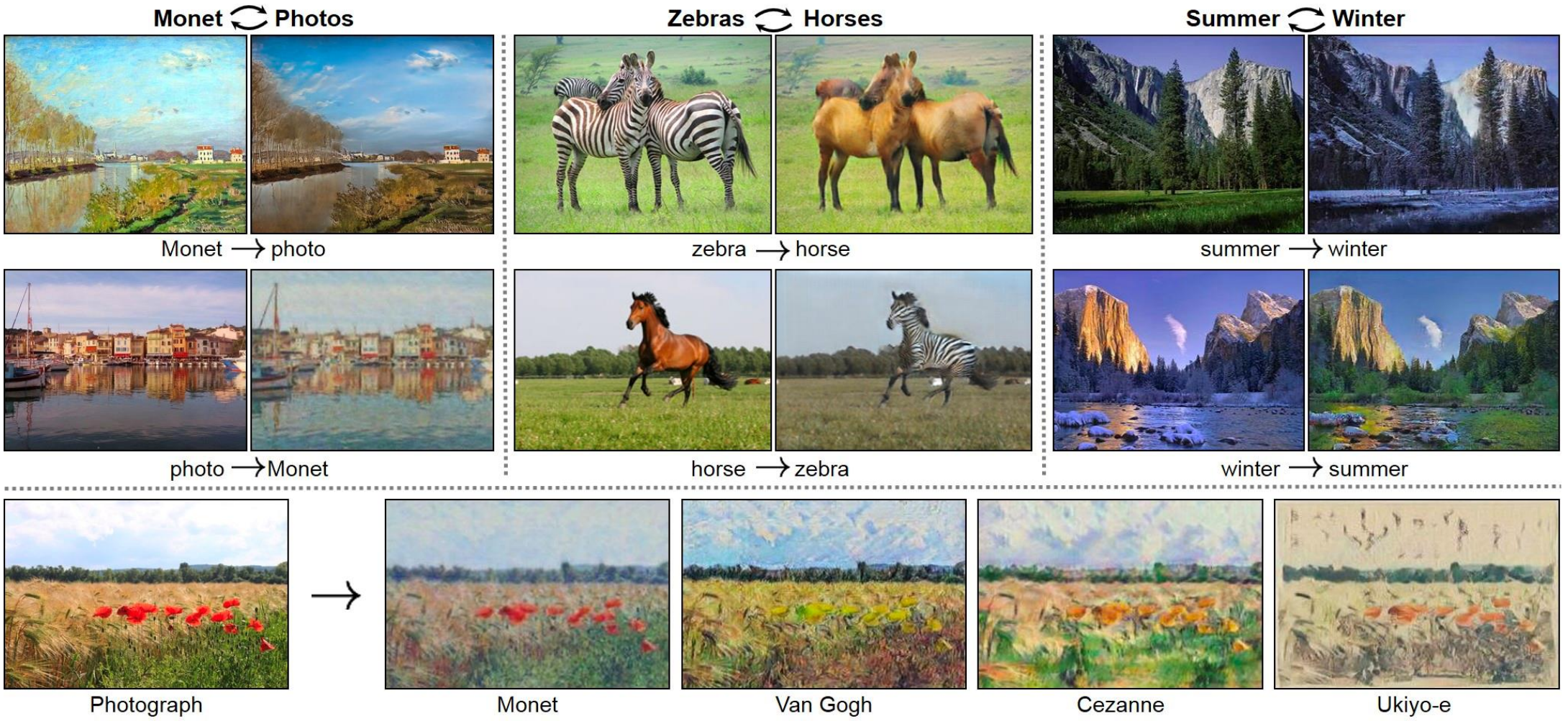
Ledig et al, “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network”, CVPR 2017

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

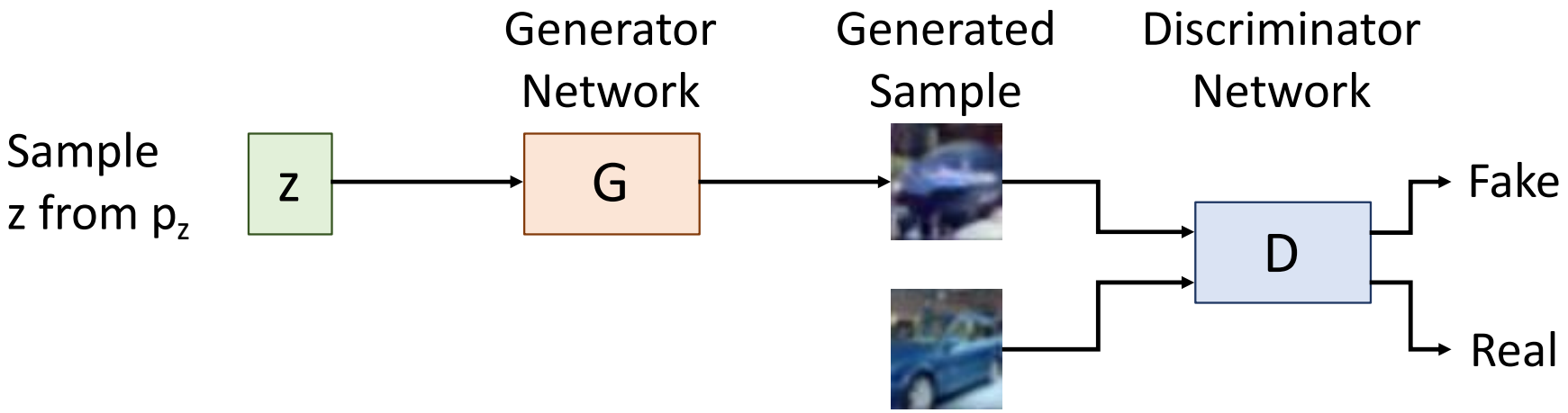
Output Video: Zebra

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

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

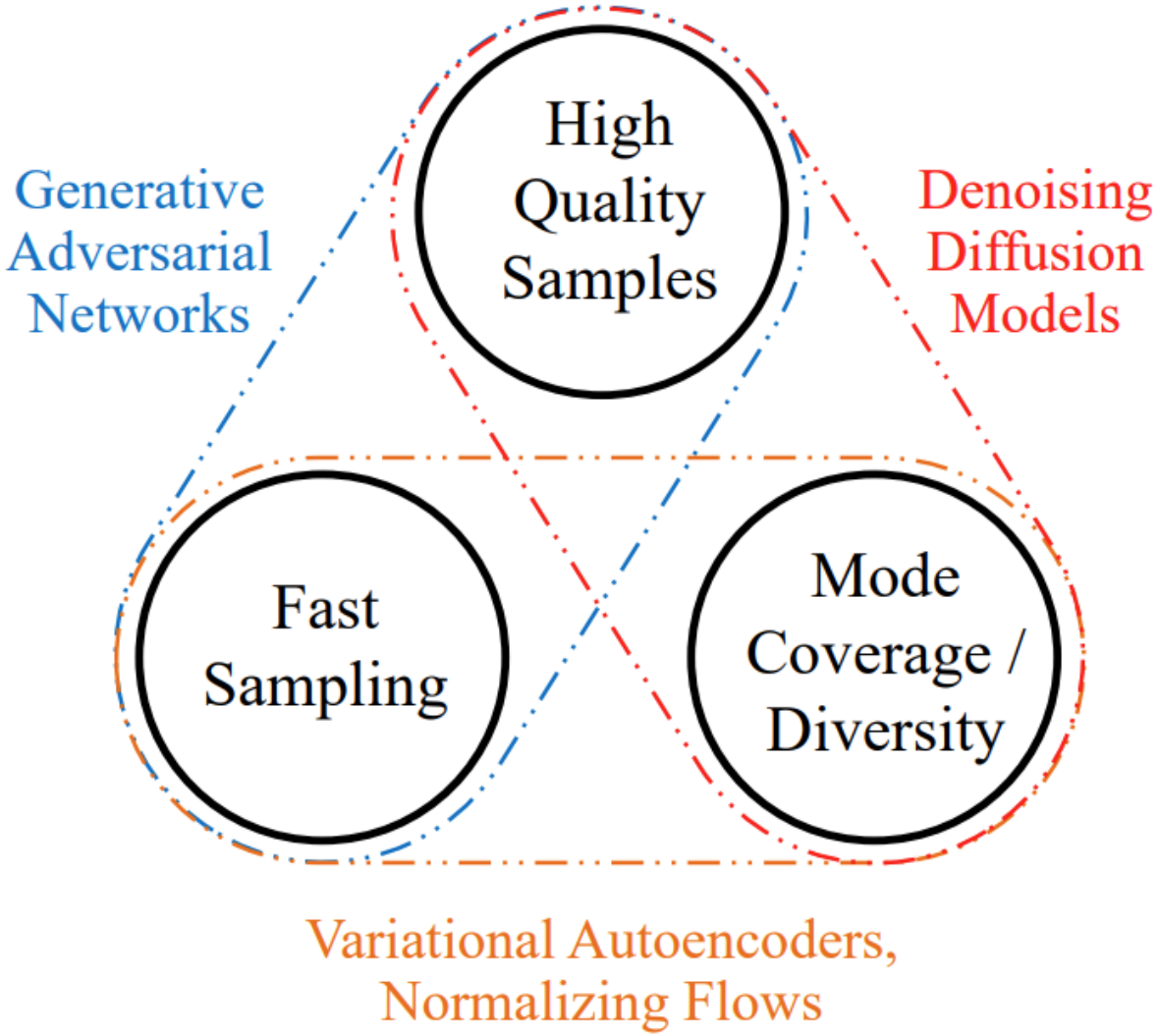
GAN Summary

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!

Generative Models Summary



Next: Recurrent Neural Networks