

# CS 2770: Generative Models

---

PhD. Nils Murrugarra-Llerena  
[nem177@pitt.edu](mailto:nem177@pitt.edu)



# Supervised vs Unsupervised Learning

## Supervised Learning

**Data:**  $(x, y)$

$x$  is data,  $y$  is label

**Goal:** Learn a function to map  $x \rightarrow y$

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



→ Cat

**Classification**

[This image](#) is [CC0 public domain](#)

Slide credit: Fei-Fei Li

# Supervised vs Unsupervised Learning

## Supervised Learning

**Data:**  $(x, y)$   
x is data, y is label

**Goal:** Learn a function to map  $x \rightarrow y$

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



A cat sitting on a suitcase on the floor

## Image captioning

Caption generated using [neuraltalk2](#)  
Image is [CC0 Public domain](#)

Slide credit: Fei-Fei Li

# Supervised vs Unsupervised Learning

## Supervised Learning

**Data:**  $(x, y)$

$x$  is data,  $y$  is label

**Goal:** Learn a function to map  $x \rightarrow y$

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



DOG, DOG, CAT

Object Detection

[This image](#) is [CC0 public domain](#)

# Supervised vs Unsupervised Learning

## Supervised Learning

**Data:**  $(x, y)$   
x is data, y is label

**Goal:** Learn a function to map  $x \rightarrow y$

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



GRASS, CAT, TREE,  
SKY

Semantic Segmentation

# Supervised vs Unsupervised Learning

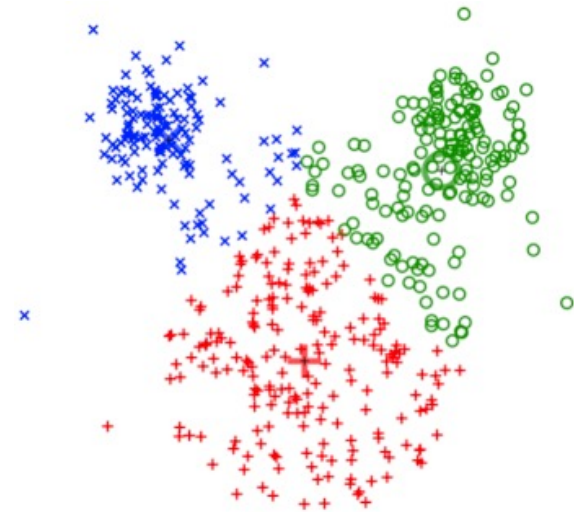
## Unsupervised Learning

**Data:**  $x$

Just data, no labels!

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

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



K-means clustering

[This image](#) is [CC0 public domain](#)

Slide credit: Fei-Fei Li

# Supervised vs Unsupervised Learning

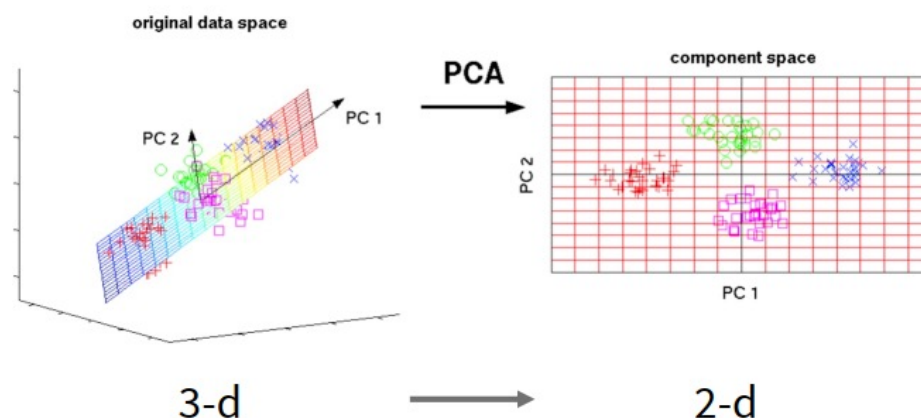
## Unsupervised Learning

**Data:**  $X$

Just data, no labels!

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

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



Principal Component Analysis  
(Dimensionality reduction)

[This image](#) from Matthias Scholz is  
[CC0 public domain](#)

Slide credit: Fei-Fei Li

# Supervised vs Unsupervised Learning

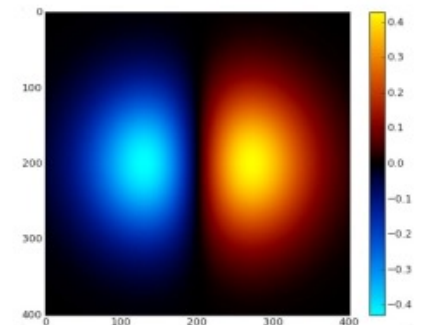
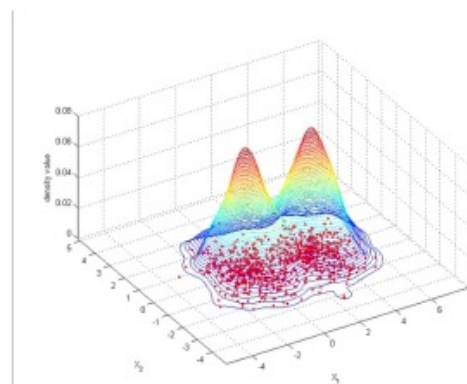
## Unsupervised Learning

**Data:**  $x$

Just data, no labels!

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

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



2-d density estimation

Modeling  $P(x)$

2-d density images [left](#) and [right](#)  
are [CC0 public domain](#)

Slide credit: Fei-Fei Li

# Supervised vs Unsupervised Learning

## Supervised Learning

**Data:**  $(x, y)$

$x$  is data,  $y$  is label

**Goal:** Learn a function to map  $x \rightarrow y$

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

## Unsupervised Learning

**Data:**  $x$

Just data, no labels!

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

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

# Supervised vs Unsupervised Learning

## Supervised Learning

**Data:**  $(x, y)$   
x is data, y is label

**Goal:** Learn a function to map  
 $x \rightarrow y$

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

## Unsupervised Learning

**Data:** x  
Just data, no labels!

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

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

## Self-Supervised Learning

**Data:**  $(x, \text{pseudo generated } y)$   
No manual labels!

**Goal:** Learn to generate good features (reduce the data to useful/generic features)

**Example:** Classification in downstream applications where we have limited data

# Generative vs Discriminative 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

# Generative vs Discriminative 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

$P(\text{cat} | \text{img})$



$P(\text{dog} | \text{img})$

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

Different values of  $x$   
**compete** for density

# Generative vs Discriminative Models

## Discriminative Model:

Learn a probability distribution  $p(y|x)$

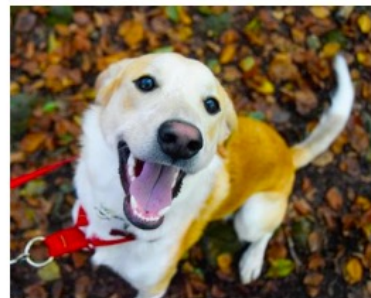
## Generative Model:

Learn a probability distribution  $p(x)$

**Conditional Generative Model:** Learn  $p(x|y)$



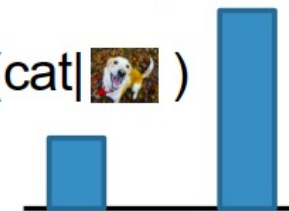
$P(\text{cat} | \text{image})$



$P(\text{dog} | \text{image})$



$P(\text{cat} | \text{image})$



Possible **labels** for each image compete for probability.  
No competition between **images**

[Monkey image](#) is CC0 Public Domain  
[Abstract image](#) is free to use under the  
[Pixabay license](#)

# Generative vs Discriminative Models

## Discriminative Model:

Learn a probability distribution  $p(y|x)$

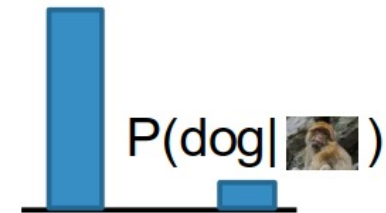
## Generative Model:

Learn a probability distribution  $p(x)$

**Conditional Generative Model:** Learn  $p(x|y)$



$P(\text{cat} | \text{img})$



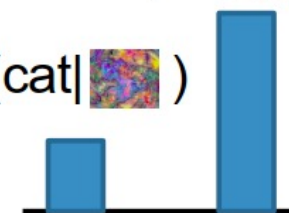
$P(\text{dog} | \text{img})$



$P(\text{dog} | \text{img})$



$P(\text{cat} | \text{img})$



No way to handle unreasonable inputs; must give a label distribution for all possible inputs

# Generative vs Discriminative Models

[Cat image](#) is [CC0 public domain](#)  
[Dog image](#) is CC0 Public Domain  
[Monkey image](#) is CC0 Public Domain  
[Abstract image](#) is free to use under the [Pixabay license](#)

## Discriminative Model:

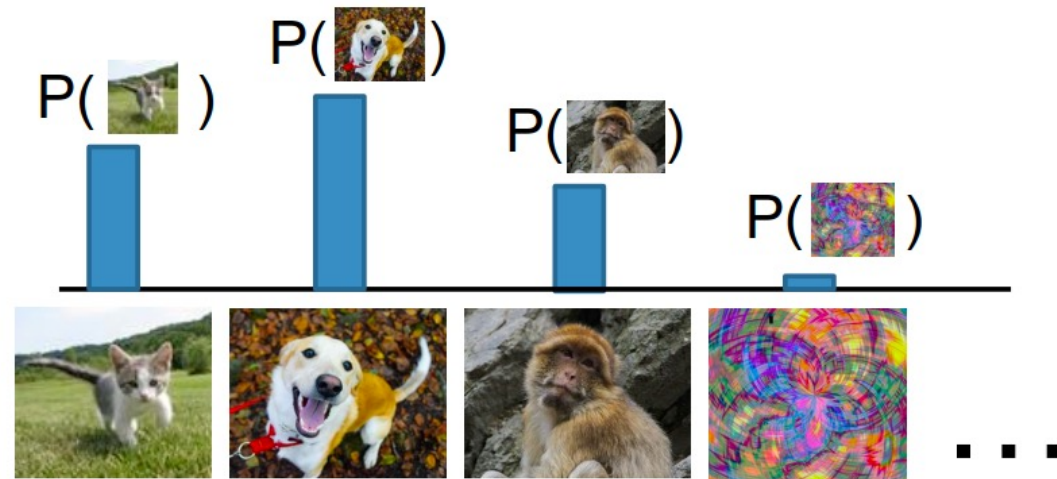
Learn a probability distribution  $p(y|x)$

## Generative Model:

Learn a probability distribution  $p(x)$

## Conditional Generative

Model: Learn  $p(x|y)$



All possible **images** compete for probability mass

# Generative vs Discriminative Models

[Cat image](#) is [CC0 public domain](#)  
[Dog image](#) is CC0 Public Domain  
[Monkey image](#) is CC0 Public Domain  
[Abstract image](#) is free to use under the [Pixabay license](#)

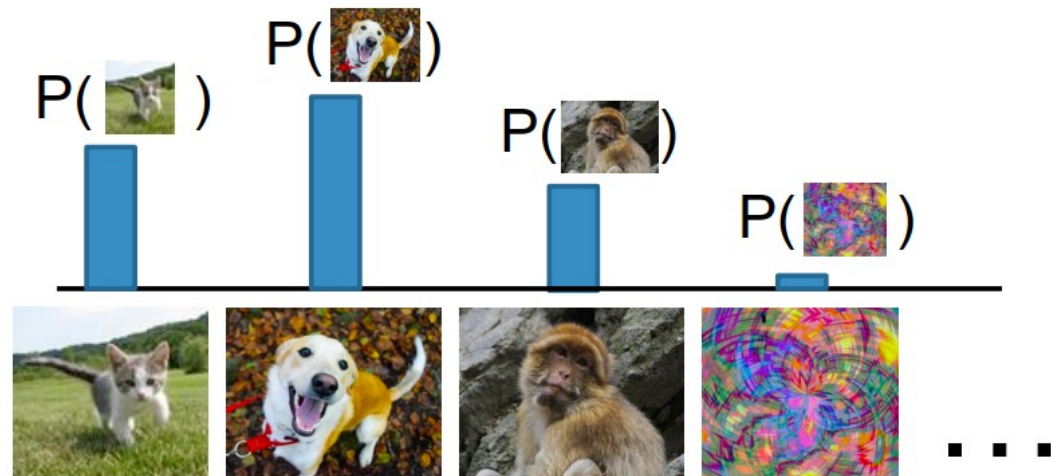
## Discriminative Model:

Learn a probability distribution  $p(y|x)$

## Generative Model:

Learn a probability distribution  $p(x)$

## Conditional Generative Model: Learn $p(x|y)$



All possible **images** compete for probability mass

Requires deep understanding: Is a dog more likely to sit or stand? Is a 3-legged dog more likely than a 3-armed monkey?

# Generative vs Discriminative Models

[Cat image](#) is [CC0 public domain](#)

[Dog image](#) is CC0 Public Domain

[Monkey image](#) is CC0 Public Domain

[Abstract image](#) is free to use under the [Pixabay license](#)

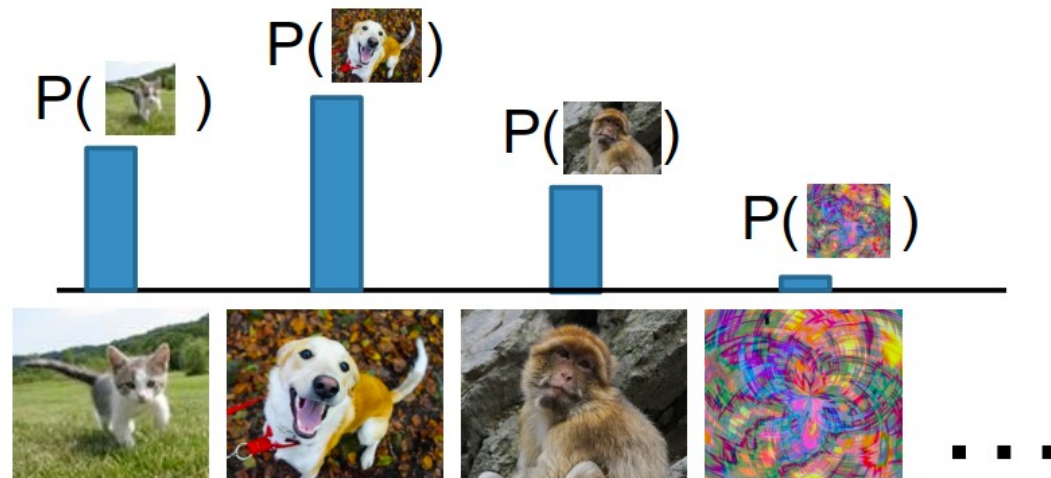
## Discriminative Model:

Learn a probability distribution  $p(y|x)$

## Generative Model:

Learn a probability distribution  $p(x)$

**Conditional Generative Model:** Learn  $p(x|y)$



All possible **images** compete for probability mass

Model can “reject” unreasonable inputs by giving them small probability mass

# Generative vs Discriminative Models

[Cat image](#) is [CC0 public domain](#)  
[Dog image](#) is CC0 Public Domain  
[Monkey image](#) is CC0 Public Domain  
[Abstract image](#) is free to use under the [Pixabay license](#)

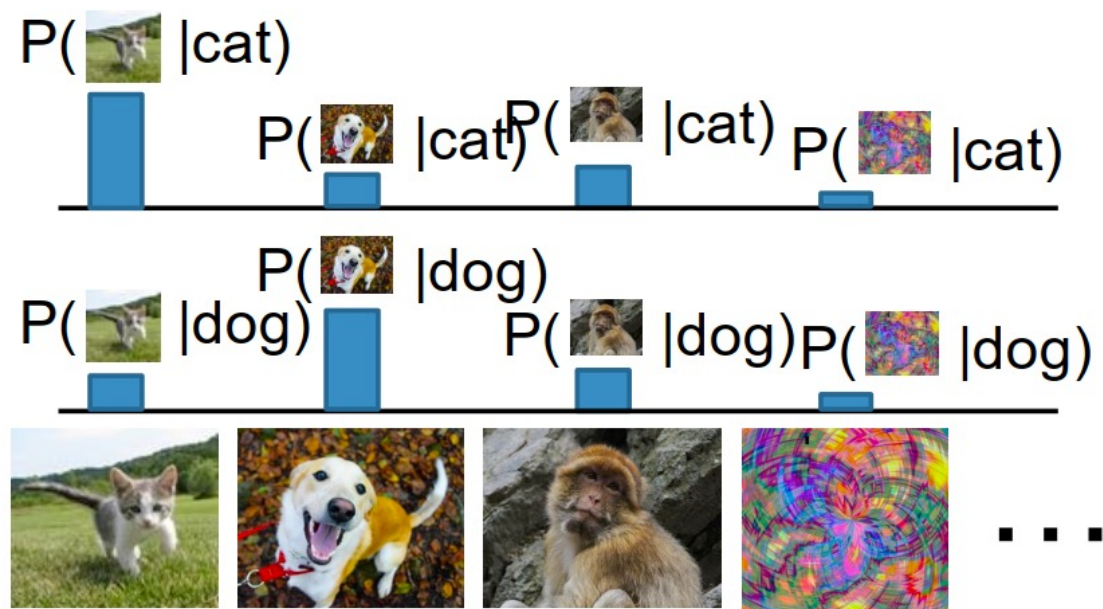
## Discriminative Model:

Learn a probability distribution  $p(y|x)$

## Generative Model:

Learn a probability distribution  $p(x)$

## Conditional Generative Model: Learn $p(x|y)$



Each possible **label** induces a competition across all possible **images**

# Generative vs Discriminative Models

**Discriminative Model:**

Learn a probability distribution  $p(y|x)$

**Generative Model:**

Learn a probability distribution  $p(x)$

**Conditional Generative Model:** Learn  $p(x|y)$

Recall **Bayes' Rule:**

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

# Generative vs Discriminative Models

## Discriminative Model:

Learn a probability distribution  $p(y|x)$

## Generative Model:

Learn a probability distribution  $p(x)$

**Conditional Generative Model:** Learn  $p(x|y)$

Recall **Bayes' Rule**:

$$\underbrace{P(x | y)}_{\substack{\text{Conditional} \\ \text{Generative Model}}} = \frac{\overbrace{P(y | x)}^{\text{Discriminative Model}}}{\underbrace{P(y)}_{\substack{\text{Prior over} \\ \text{labels}}}} \underbrace{P(x)}_{\substack{\text{(Unconditional)} \\ \text{Generative Model}}}$$

We can build a conditional generative model from other components ... but not common in practice

# Generative vs Discriminative Models

## **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

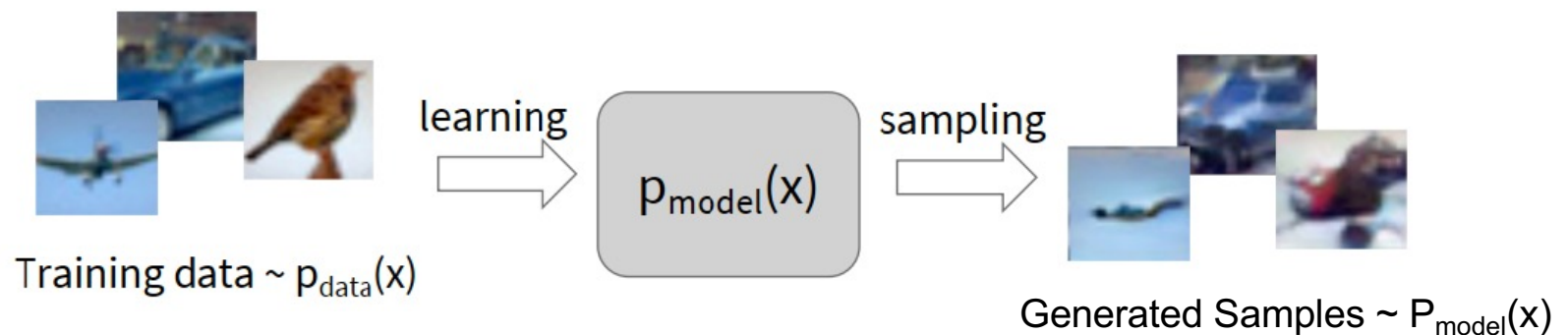
**Conditional Generative Model:** Learn  $p(x|y)$



Assign labels while rejecting outliers  
**Sample** to generate data from labels

# Generative Modeling

Given training data, generate new samples from same distribution



Objectives:

1. Learn  $p_{\text{model}}(x)$  that approximates  $p_{\text{data}}(x)$
2. Sampling new  $x$  from  $p_{\text{model}}(x)$

# Why Generative Models? Debiasing

Capable of uncovering **underlying features** in a dataset



Homogeneous skin color, pose

VS



Diverse skin color, pose, illumination

How can we use this information to create fair and representative datasets?

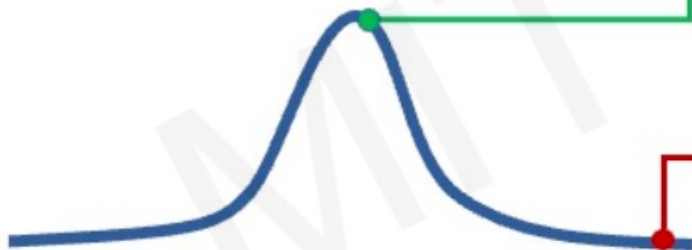
# Why Generative Models? Outlier Detection

- **Problem:** How can we detect when we encounter something new or rare?
- **Strategy:** Leverage generative models, detect outliers in the distribution
- Use outliers during training to improve even more!

**95% of Driving Data:**  
(1) sunny, (2) highway, (3) straight road



Detect outliers to avoid unpredictable behavior when training



Edge Cases



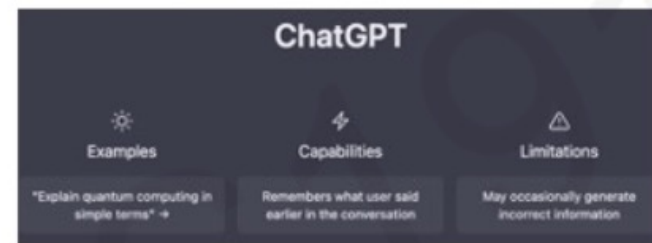
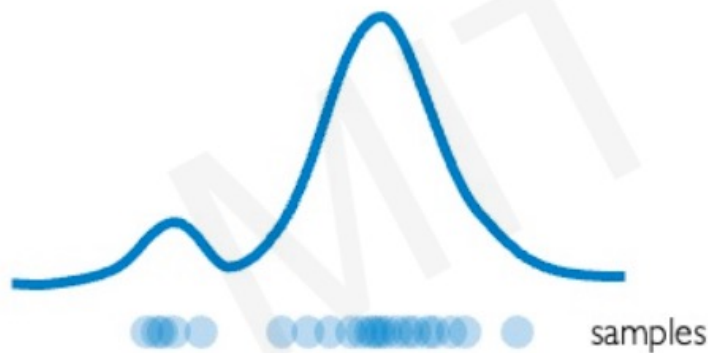
Harsh Weather



Pedestrians

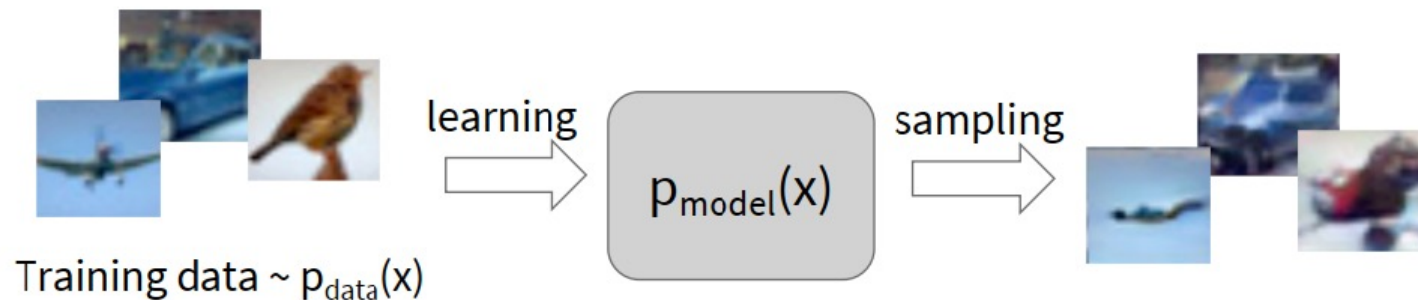
# Why Generative Models? Sample Generation

- **Generative models learn probability distributions**
- **Sampling** from that distribution → new data instances
- **Backbone of Generative AI:** generate language, images, and more



# Generative Modeling

Given training data, generate new samples from same distribution



Formulate as density estimation problems:

- Explicit density estimation: explicitly define and solve for  $p_{\text{model}}(x)$
- Implicit density estimation: learn model that can sample from  $p_{\text{model}}(x)$  without explicitly defining it.

# Generative Modeling: Applications



- Realistic samples for artwork, super-resolution, colorization, etc.
- Learn useful features for downstream tasks such as classification.
- Getting insights from high-dimensional data (physics, medical imaging, etc.)
- Modeling physical world for simulation and planning (robotics and reinforcement learning applications)
- Many more ...

# Taxonomy of Generative Models

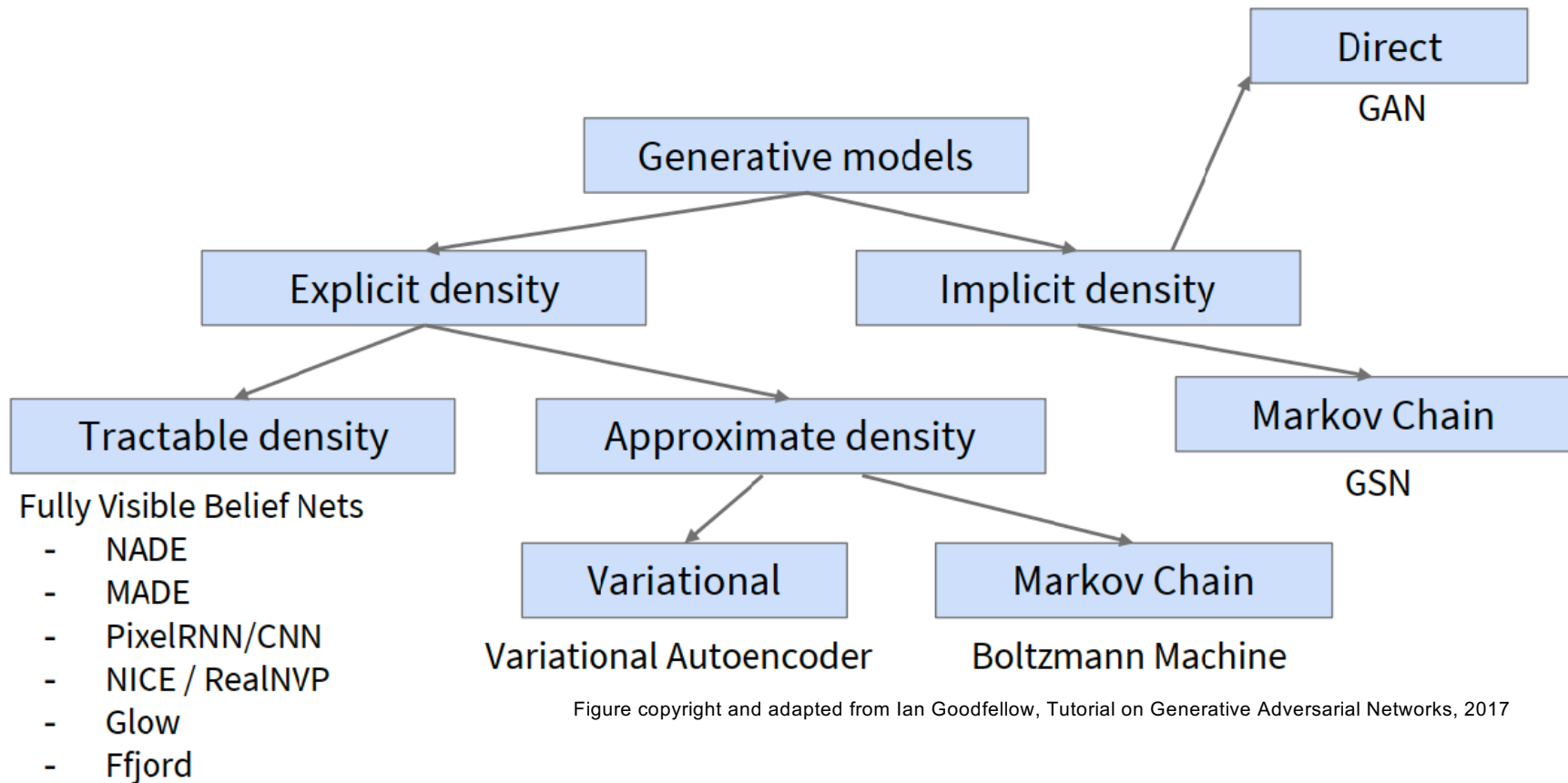


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017

Slide credit: Fei-Fei Li

# Taxonomy of Generative Models

Today: discuss 3 most popular types of generative models today

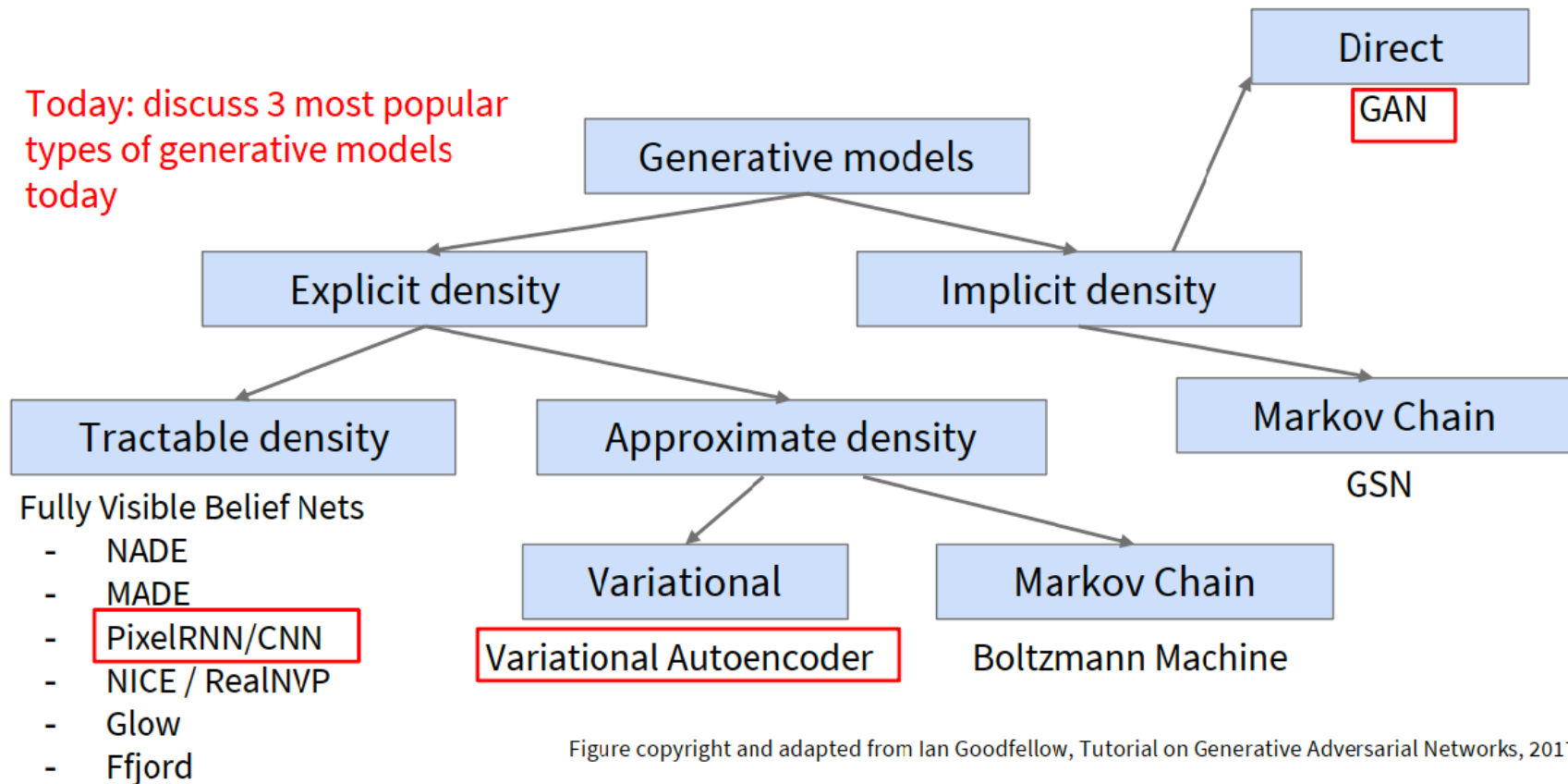


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Slide credit: Fei-Fei Li

# PixelRNN and PixelCNN



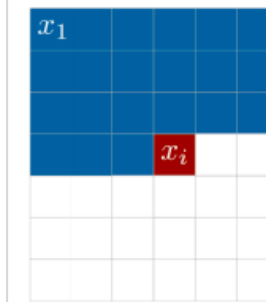
# Fully visible belief network (FVBN)

Explicit density model

Use chain rule to decompose likelihood of an image  $x$  into product of 1-d distributions:

$$p(x) = \prod_{i=1}^n p(x_i | x_1, \dots, x_{i-1})$$

↑
↑  
 Likelihood of image  $x$ 
Probability of  $i$ 'th pixel value given all previous pixels



Then maximize likelihood of training data

Slide credit: Fei-Fei Li

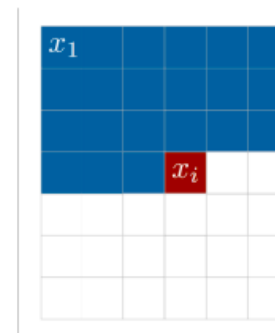
# Fully visible belief network (FVBN)

Explicit density model

Use chain rule to decompose likelihood of an image  $x$  into product of 1-d distributions:

$$p(x) = \prod_{i=1}^n p(x_i | x_1, \dots, x_{i-1})$$

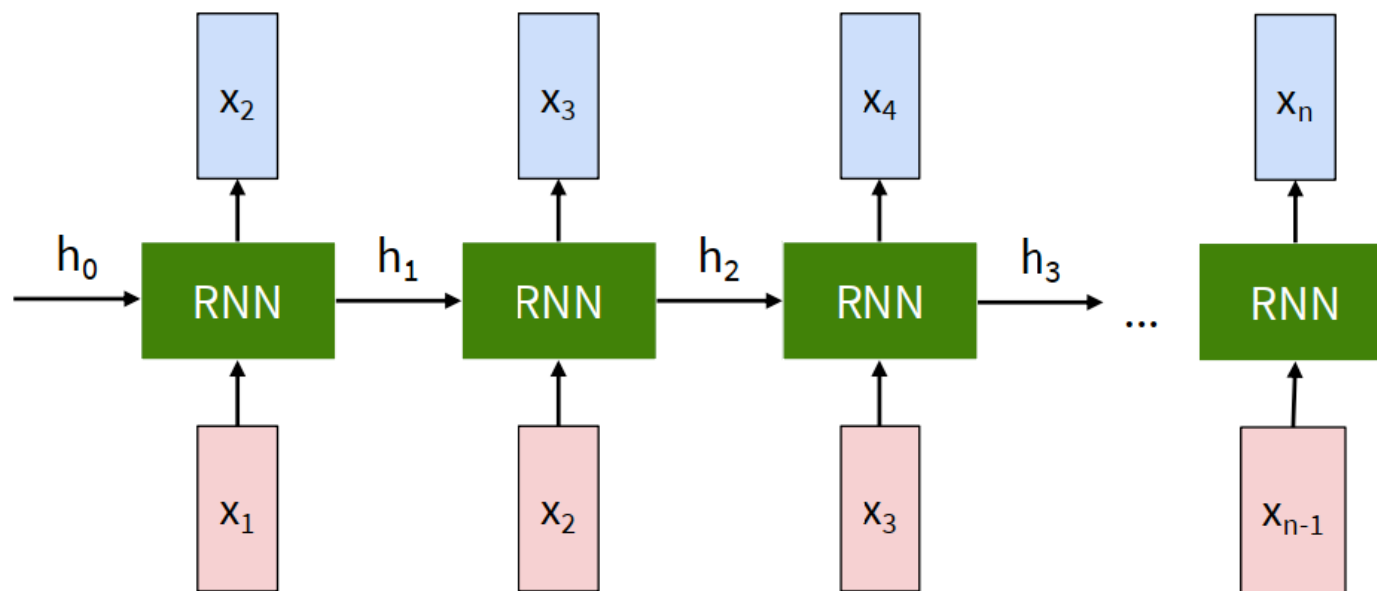
↑
↑  
 Likelihood of image  $x$ 
Probability of  $i$ 'th pixel value given all previous pixels



Complex distribution over pixel values  $\Rightarrow$  Express using a neural network!

Then maximize likelihood of training data

## Another solution: Recurrent Neural Network

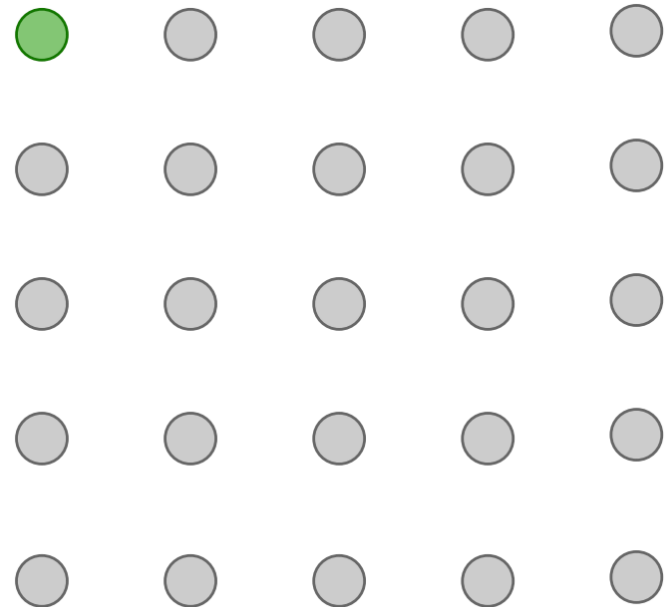


$$p(x_i | x_1, \dots, x_{i-1})$$

# PixelRNN [van der Oord et al. 2016]

Generate image pixels starting from corner

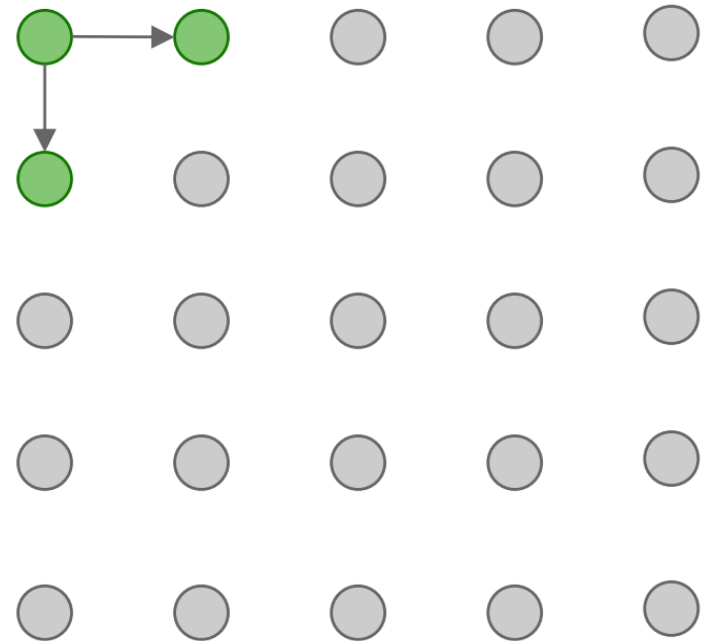
Dependency on previous pixels modeled using an RNN (LSTM)



# PixelRNN [van der Oord et al. 2016]

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

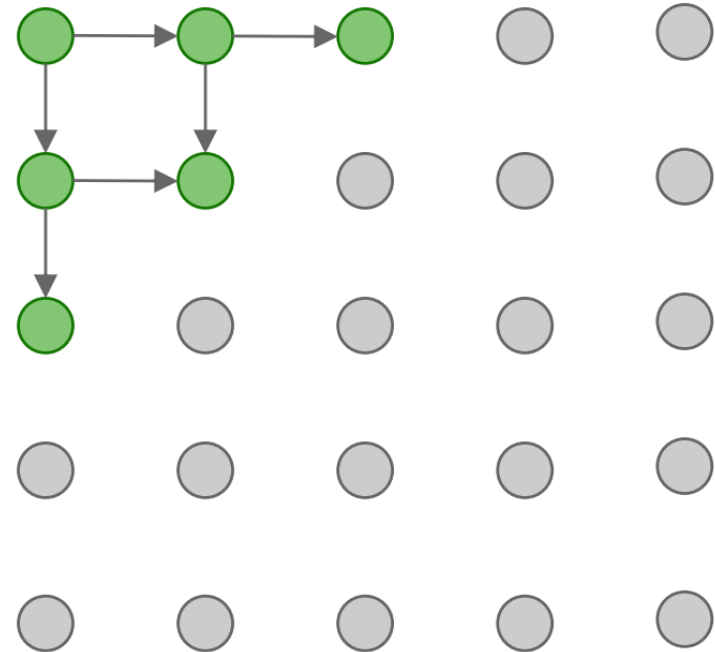


Slide credit: Fei-Fei Li

# PixelRNN [van der Oord et al. 2016]

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)



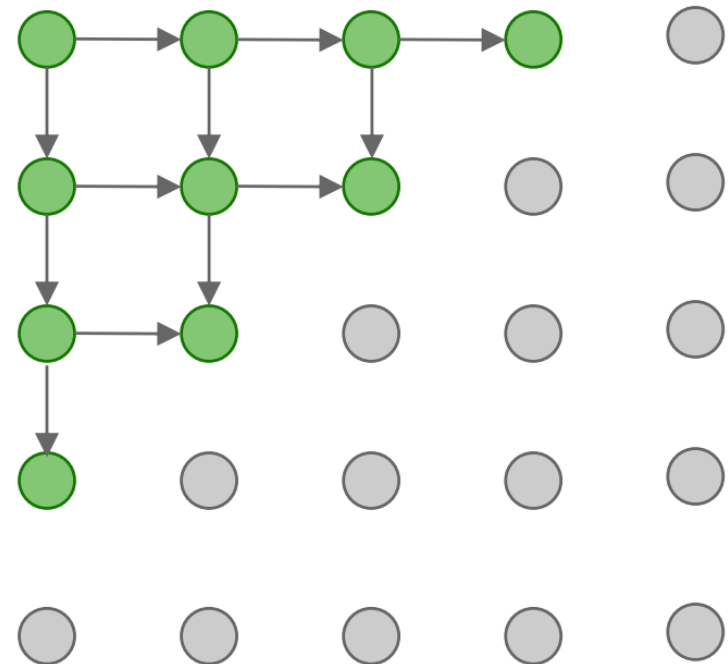
Slide credit: Fei-Fei Li

# PixelRNN [van der Oord et al. 2016]

Generate image pixels starting from corner

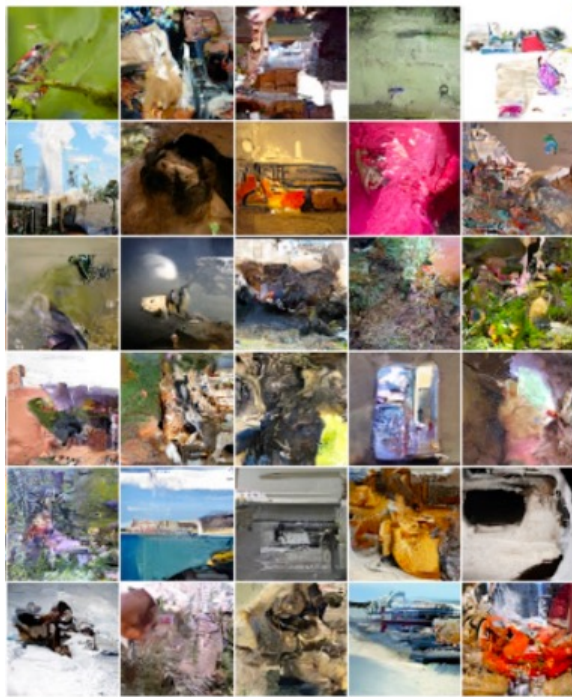
Dependency on previous pixels modeled using an RNN (LSTM)

Drawback: sequential generation is slow in both training and inference!



Slide credit: Fei-Fei Li

# PixelRNN [van der Oord et al. 2016]



Samples from PixelRNN

occluded

completions

original

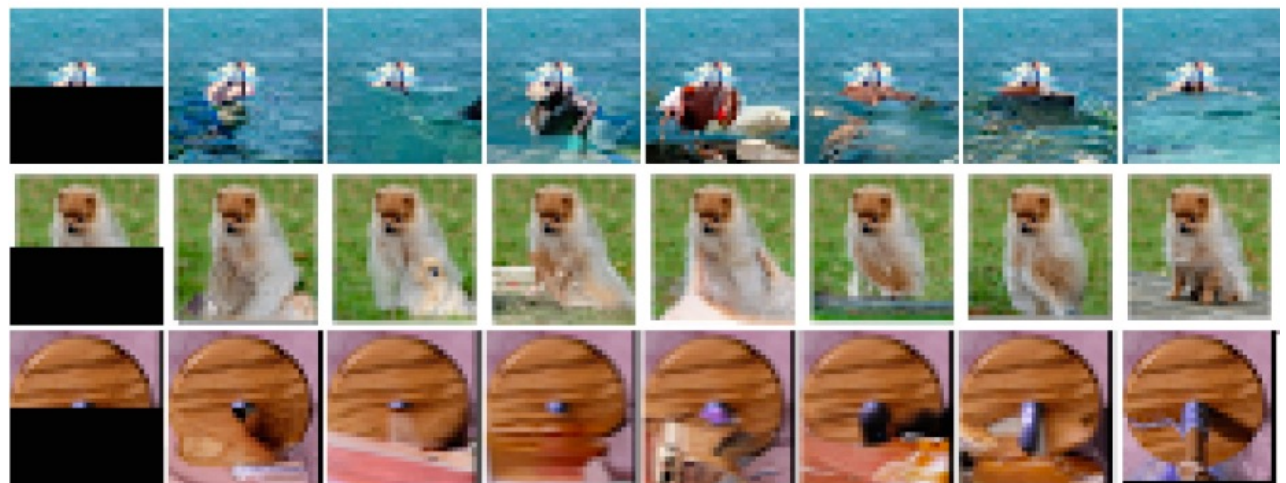
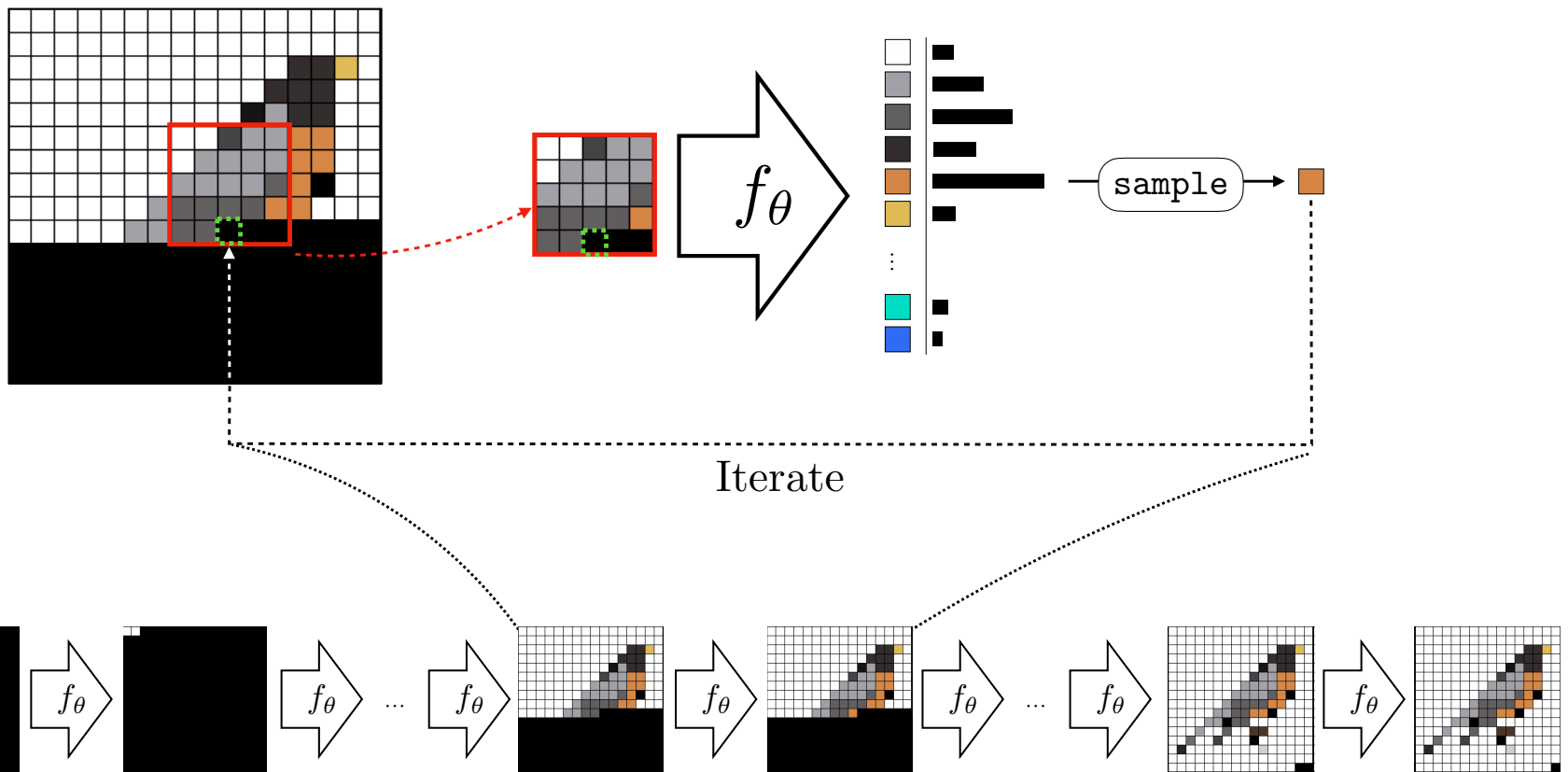


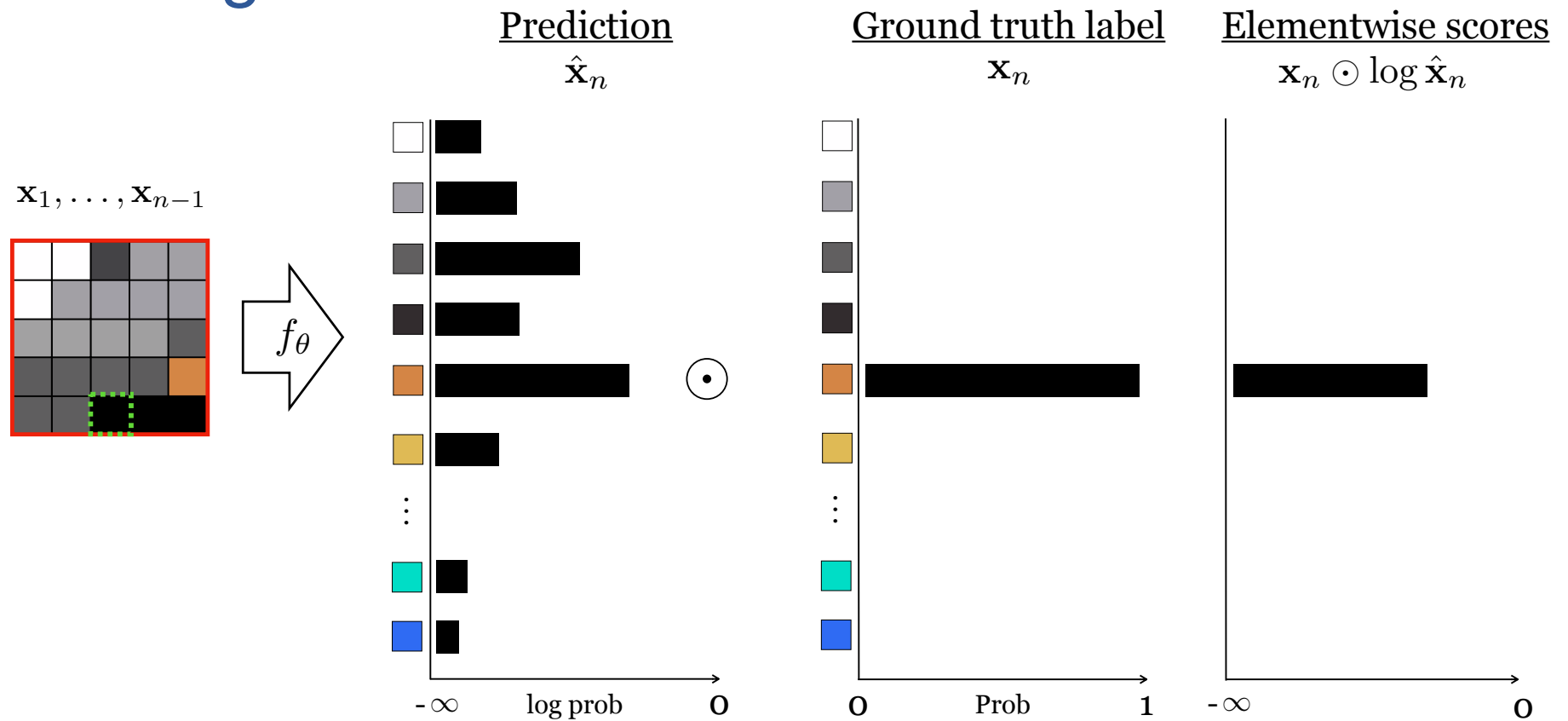
Image completions (conditional samples) from PixelRNN

[PixelRNN, van der Oord et al. 2016]

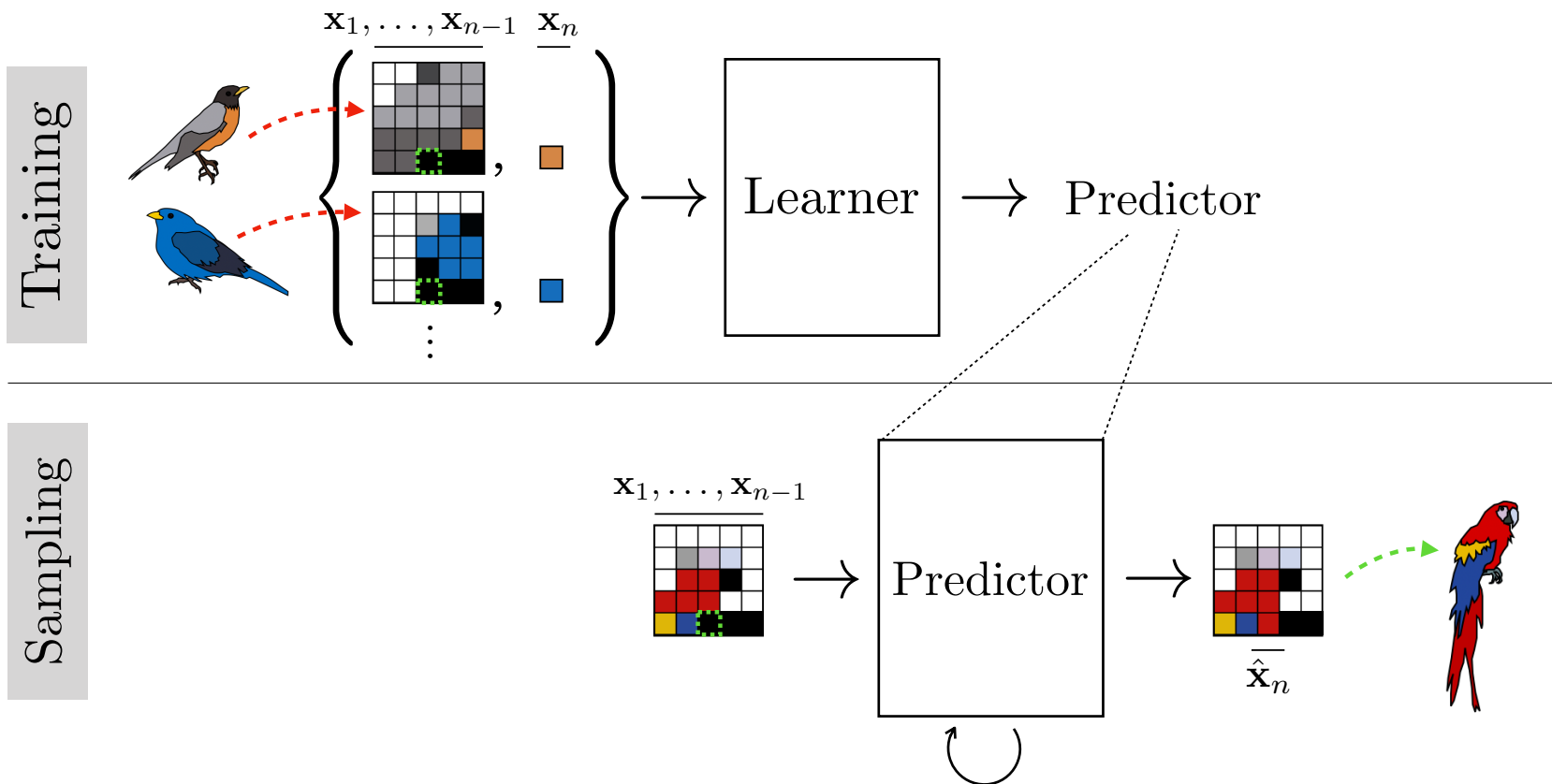
# Autoregressive Model of Pixels



# Autoregressive Model of Pixels



# Autoregressive Model of Pixels



# PixelCNN [van der Oord et al. 2016]

Still generate image pixels starting from corner

Dependency on previous pixels now model using a CNN over context region (masked convolution)

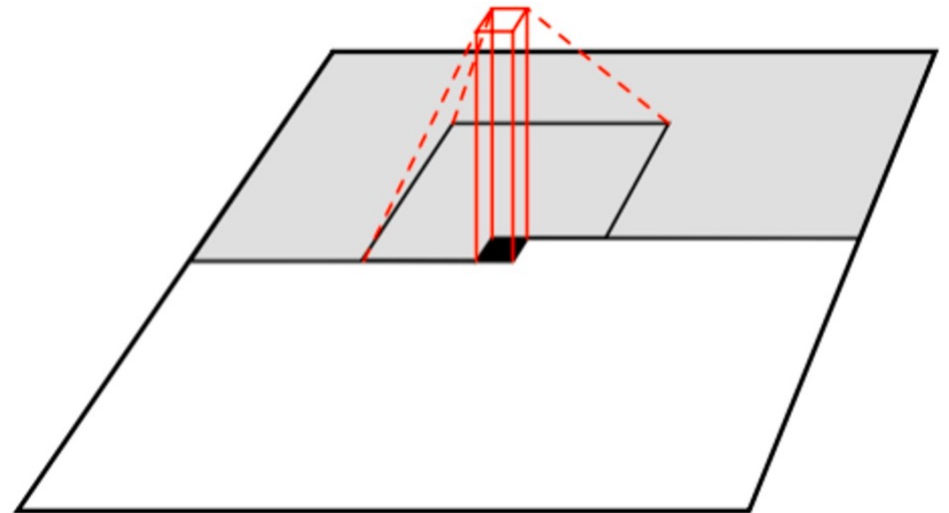


Figure copyright van der Oord et al., 2016

# PixelCNN [van der Oord et al. 2016]

Still generate image pixels starting from corner

Dependency on previous pixels now model using a CNN over context region (masked convolution)

Training is faster than PixelRNN (can parallelize convolutions since context Region values known from training images)

Generation is still slow:

For a 32x32 image, we need to do forward passes of the network 1024 times for a single image

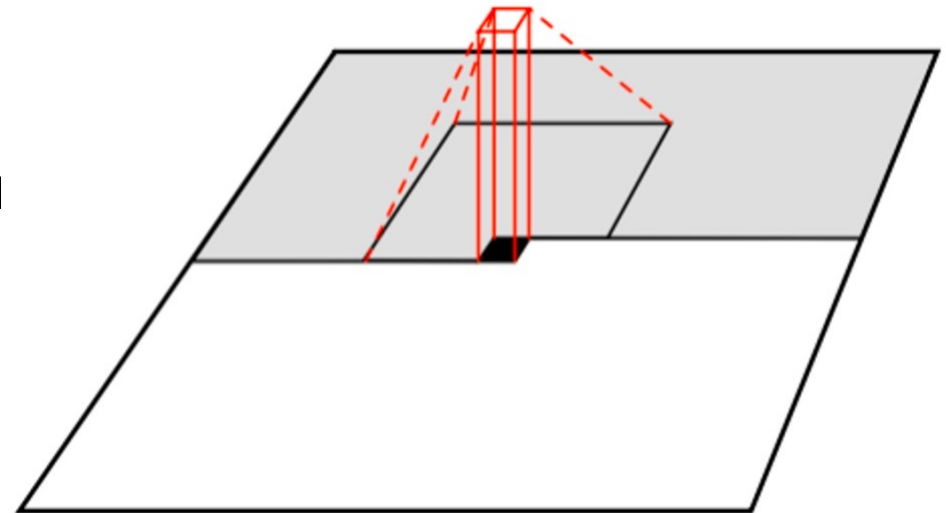
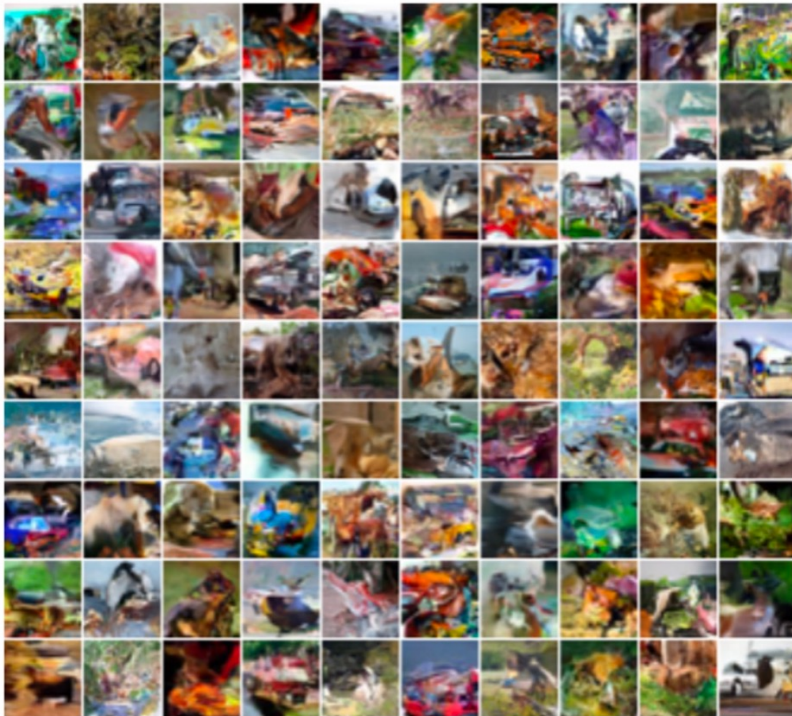
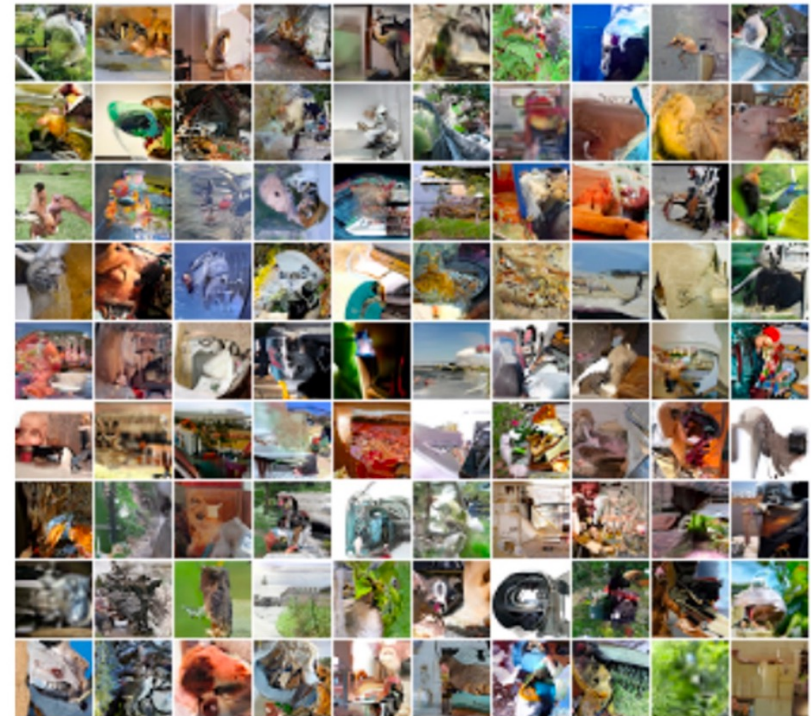


Figure copyright van der Oord et al., 2016

# Generation Samples



32x32 CIFAR-10



32x32 ImageNet

Figures copyright Aaron van der Oord et al., 2016

Slide credit: Fei-Fei Li

# PixelRNN and PixelCNN

## Pros:

- Can explicitly compute likelihood  $p(x)$
- Easy to optimize
- Good samples

## Cons:

- Sequential generation => slow

## Improving PixelCNN performance

- Gated convolutional layers
- Short-cut connections
- Discretized logistic loss
- Multi-scale
- Training tricks
- Etc...

## See

- Van der Oord et al. NIPS 2016
- Salimans et al. 2017 (PixelCNN++)

# How to improve PixelCNN?

What are the limitations of PixelCNN/RNN?

- Slow sampling time.
- May accumulate errors over multiple steps. (might not be a big issue for image completion)



How can we further improve results?

## VQ-VAE-2 :VAE+PixelCNN

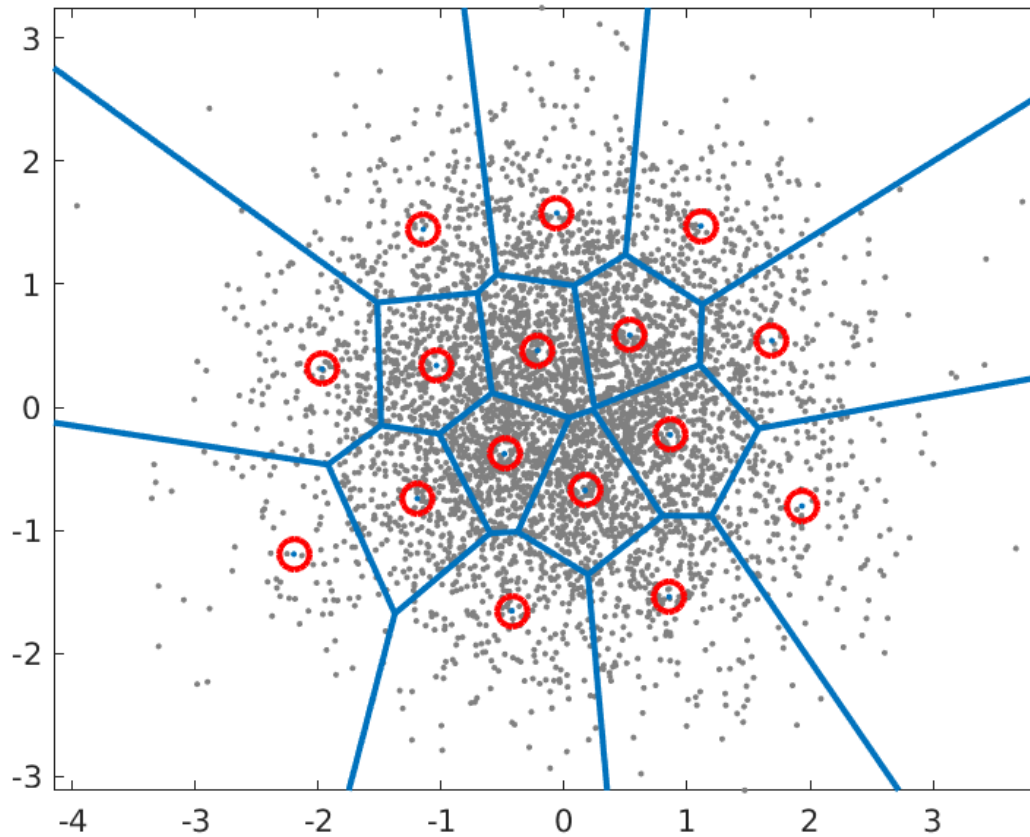


VQ (Vector quantization) maps continuous vectors into discrete codes

**Common methods:** clustering (e.g., k-means)

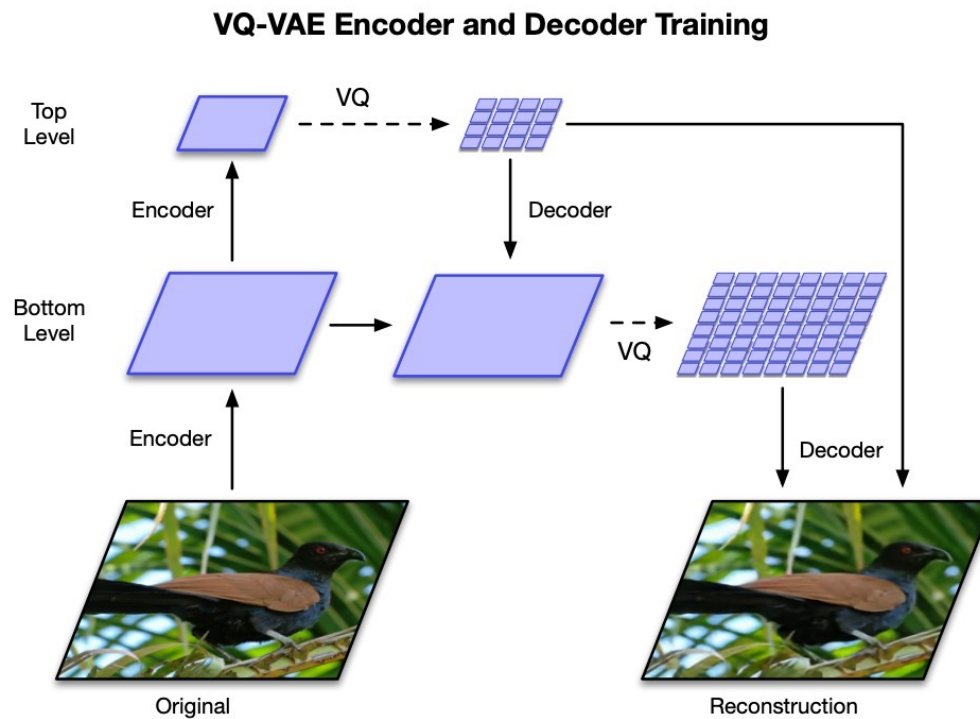
Generating Diverse High-Fidelity Images with VQ 45 -VAE-2 [Razavi et al., 2019]

## VQ-VAE-2 : Vector Quantization



K-means, EM (GMM),  
end-to-end learning

# VQ-VAE-2 : VAE+PixelCNN



**VAE+VQ:** learn a more compact codebook for PixelCNN (instead of pixels)

**PixelCNN:** use a more expressive bottleneck for VAE (instead of Gaussian)

Generating Diverse High-Fidelity Images with VQ 45 -VAE-2 [Razavi et al., 2019]

# Taxonomy of Generative Models

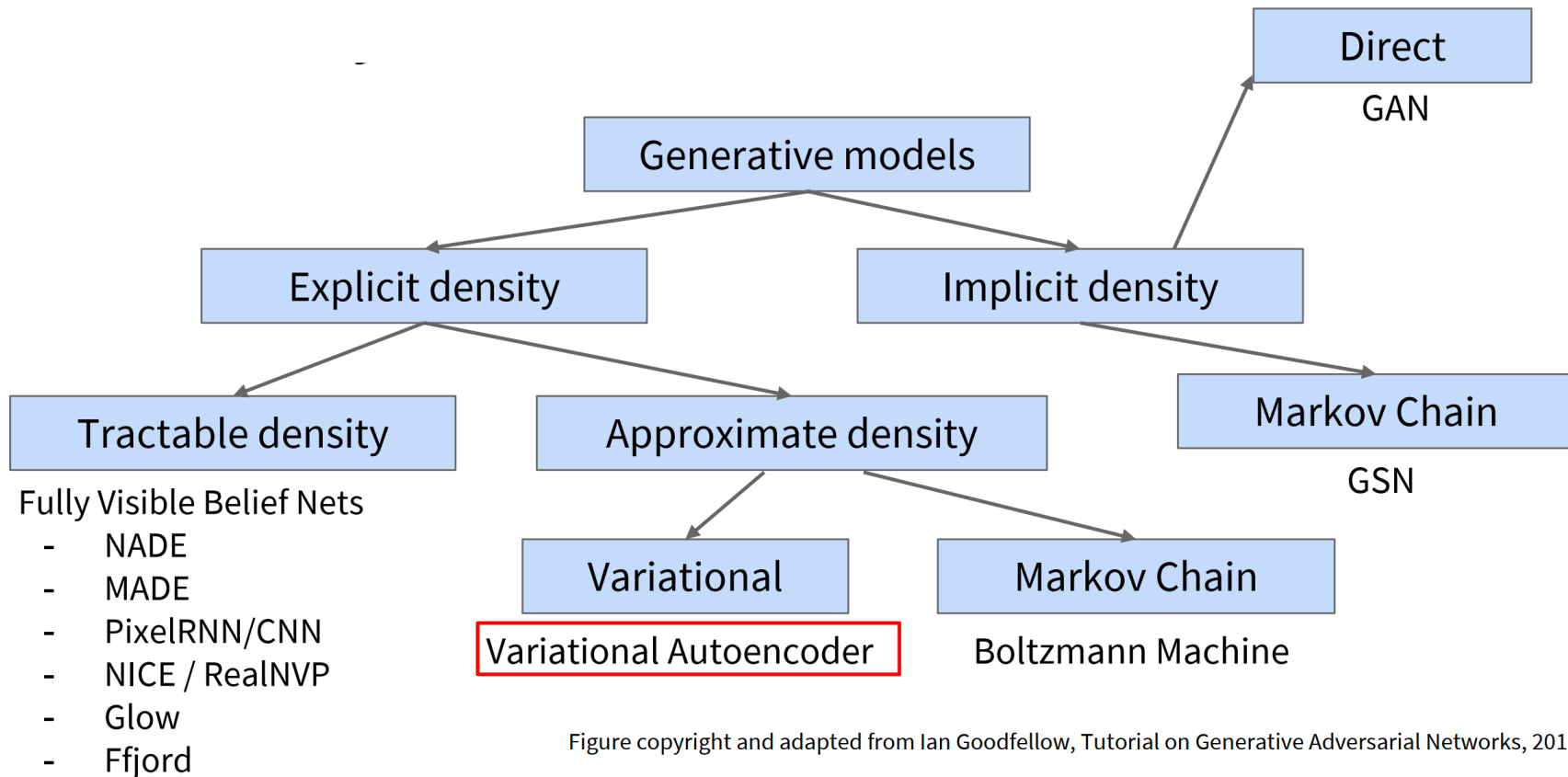


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Slide credit: Fei-Fei Li

# The Myth of the Cave



[https://www.youtube.com/watch?v=1RWOpQXTItA&ab\\_channel=TED-Ed](https://www.youtube.com/watch?v=1RWOpQXTItA&ab_channel=TED-Ed)

# What is a Latent Variable?



[https://en.wikipedia.org/wiki/Myth\\_of\\_the\\_Cave](https://en.wikipedia.org/wiki/Myth_of_the_Cave)

# The Myth of the Cave



[https://en.wikipedia.org/wiki/Myth\\_of\\_the\\_Cave](https://en.wikipedia.org/wiki/Myth_of_the_Cave)

Can we learn the **true explanatory factors** (e.g. latent variables) from only observed data?

© Alexander Amini and Ava Amini

MIT Introduction to Deep Learning

[IntroToDeepLearning.com](http://IntroToDeepLearning.com)

# Variational AutoEncoders (VAE)

PixelRNN/CNNs define tractable density function, optimize likelihood of training data:

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

Variational Autoencoders (VAEs) define intractable density function with latent  $z$ :

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

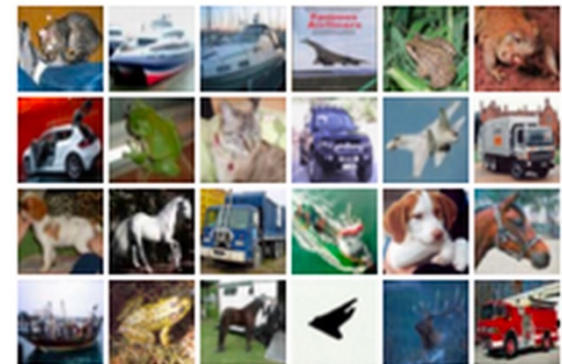
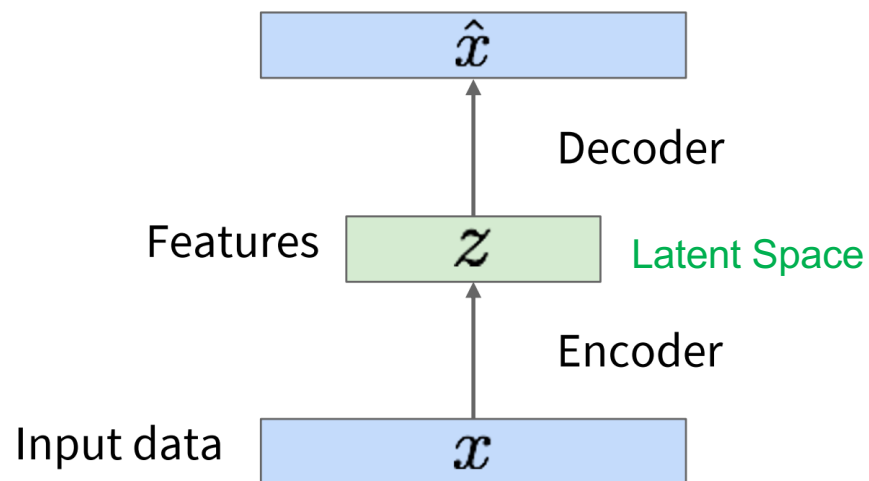
No dependencies among pixels, can generate all pixels at the same time!

Cannot optimize directly, derive and optimize lower bound on likelihood instead

## Why latent $z$ ?

# Background: AutoEncoders

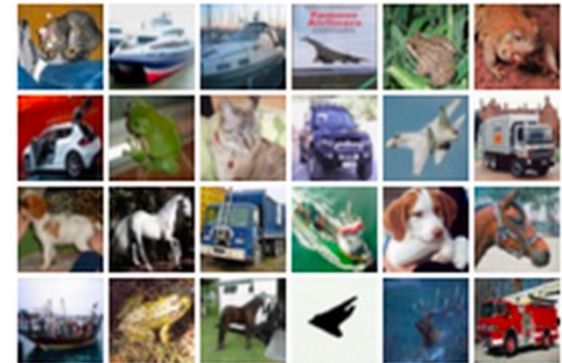
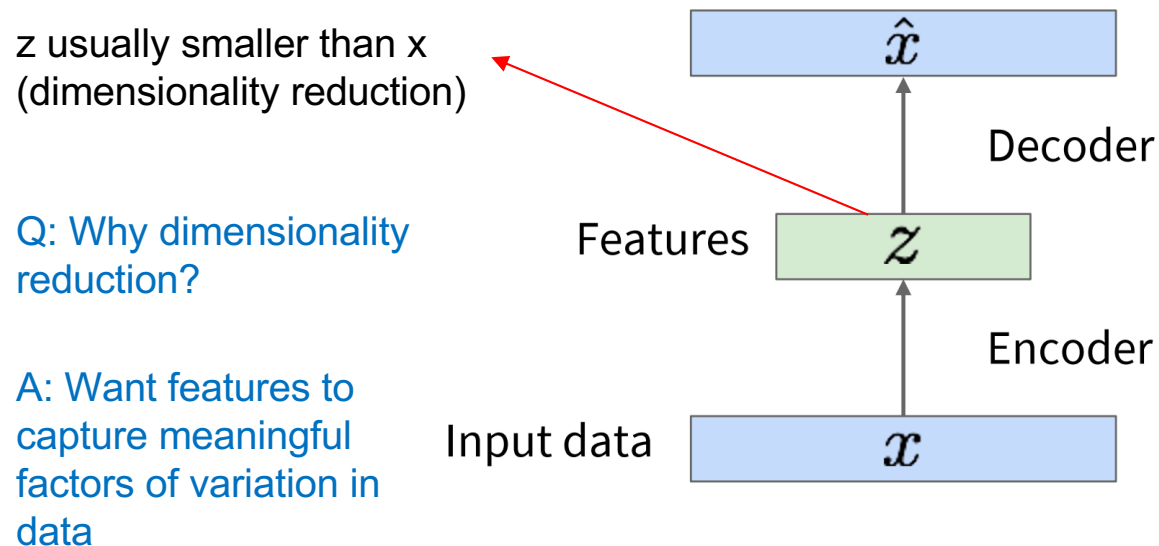
Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data



Slide credit: Fei-Fei Li

# Background: AutoEncoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

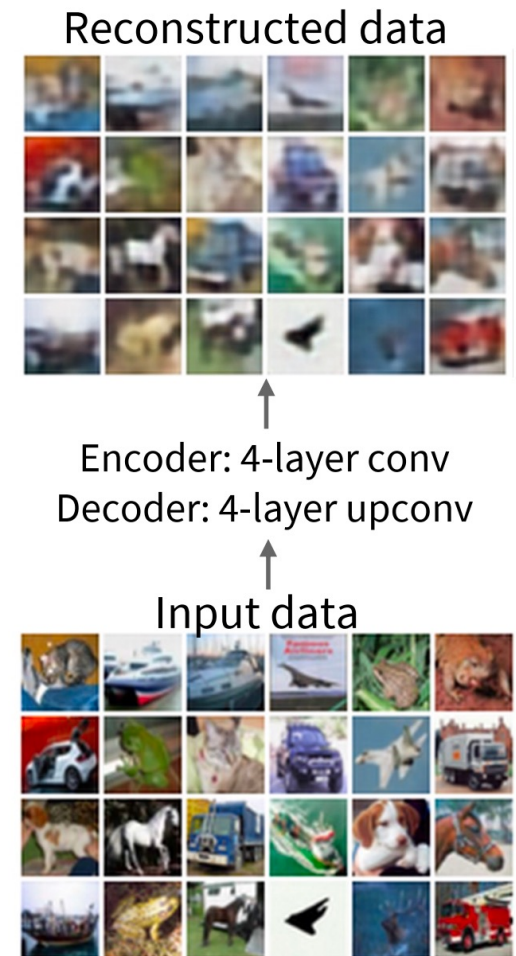
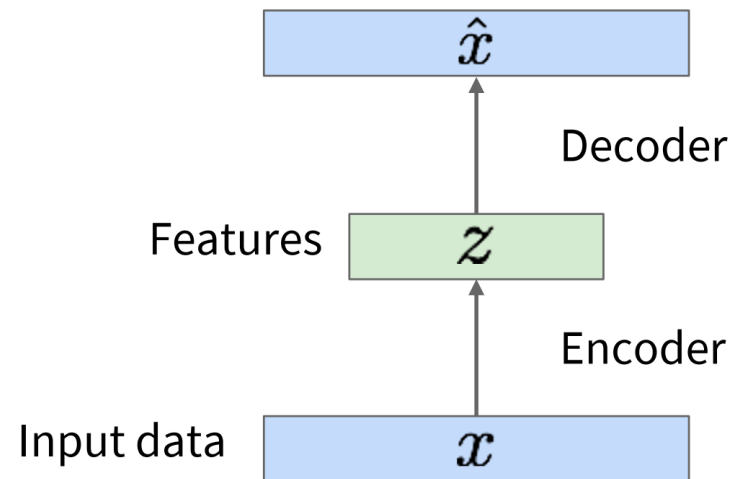


Slide credit: Fei-Fei Li

# Background: AutoEncoders

How to learn this feature representation?

Train such that features can be used to reconstruct original data  
 "Autoencoding" - encoding input itself

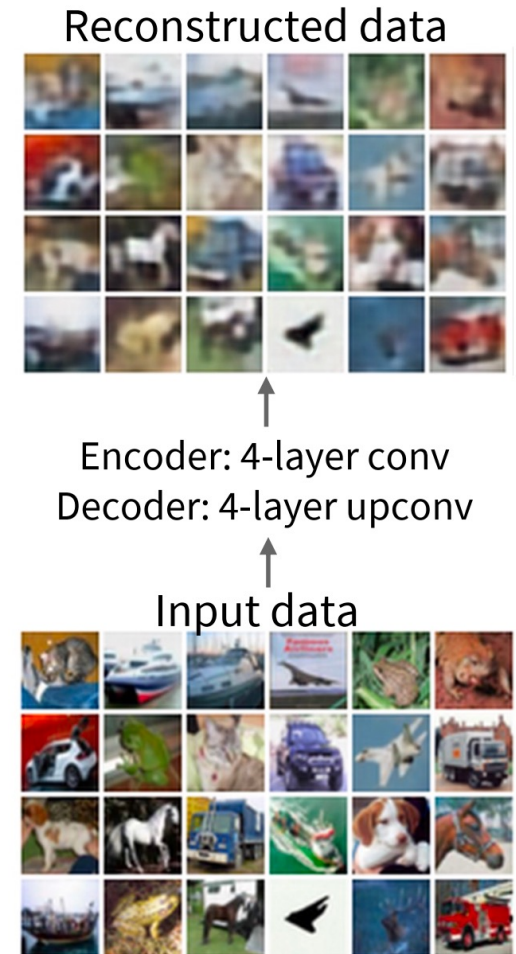
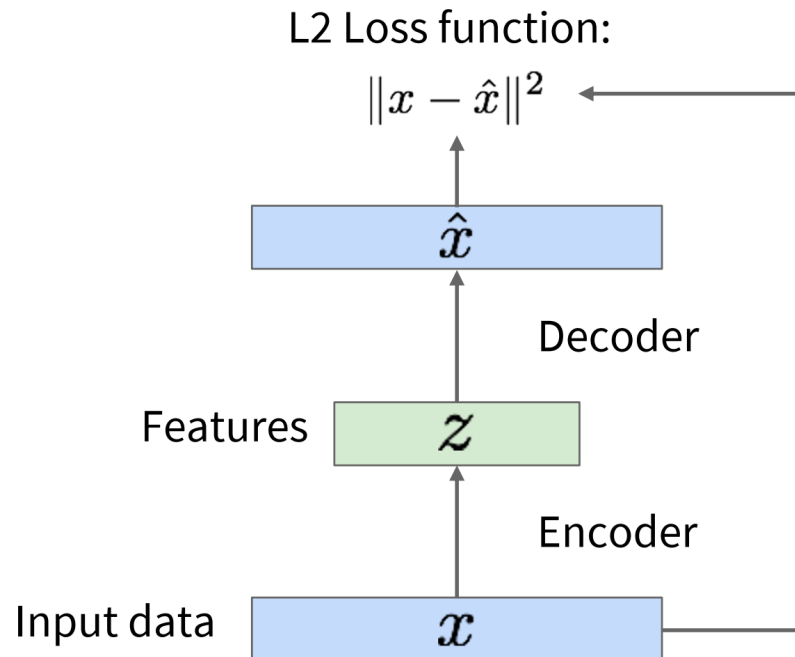


Slide credit: Fei-Fei Li

# Background: AutoEncoders

Train such that features can be used to reconstruct original data

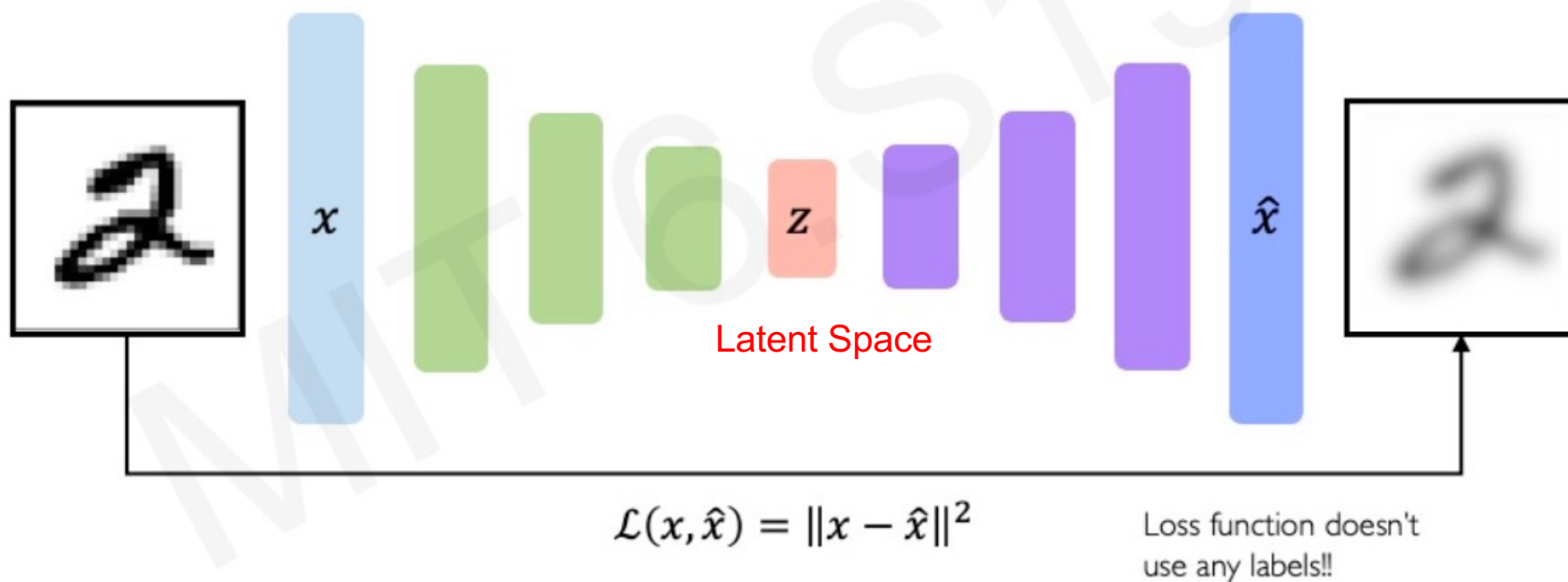
Doesn't use labels!



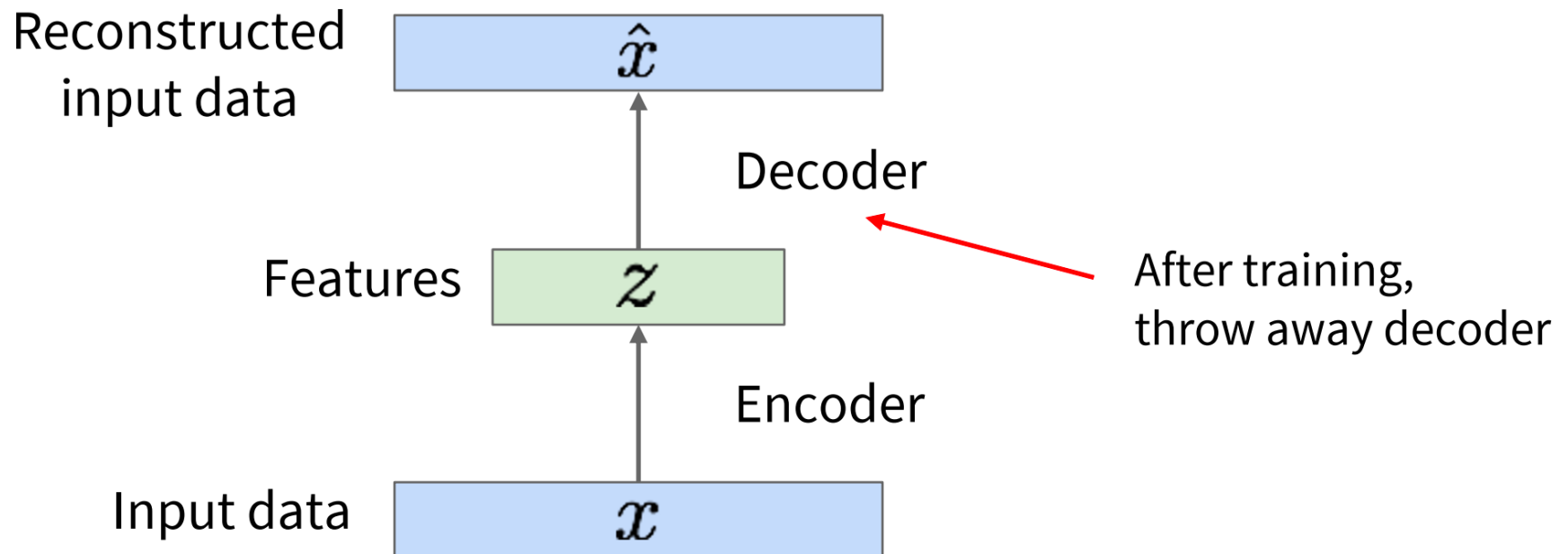
Slide credit: Fei-Fei Li

# Background: AutoEncoders

How can we learn this latent space?  
 Train the model to use these features to **reconstruct the original data**.



# Background: AutoEncoders



# Background: AutoEncoders – Reconstruction Quality

Autoencoding is a form of compression!  
Smaller latent space will force a larger training bottleneck

2D latent space



5D latent space

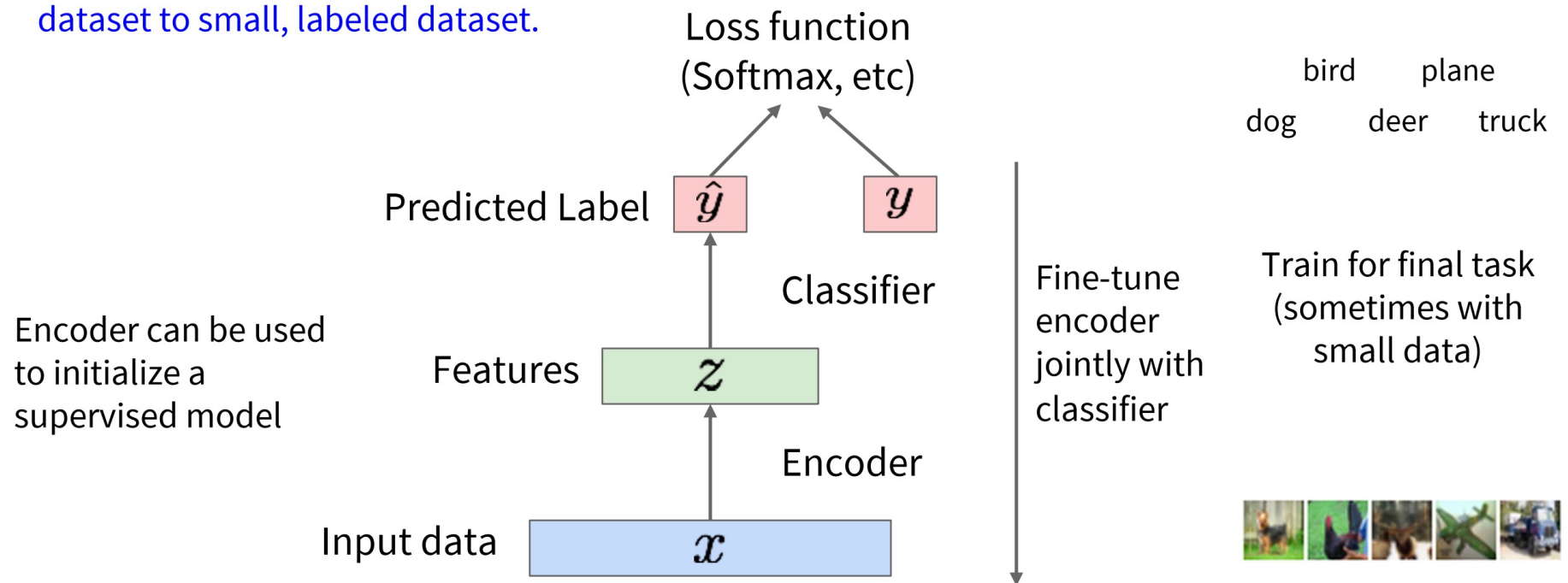


Ground Truth



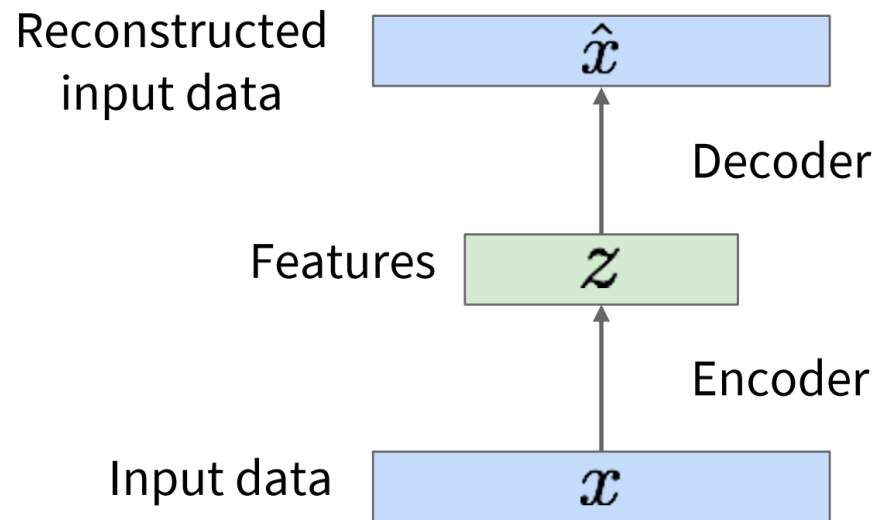
# Background: AutoEncoders

Transfer from large, unlabeled dataset to small, labeled dataset.



Slide credit: Fei-Fei Li

# Background: AutoEncoders



Autoencoders can reconstruct data, and can learn features to initialize a supervised model

Features capture factors of variation in training data.

But we can't generate new images from an autoencoder because we don't know the space of  $z$ .

How do we make autoencoder a generative model?

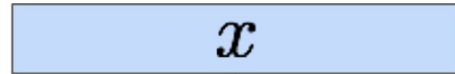
# Variational AutoEncoders

Probabilistic spin on autoencoders - will let us sample from the model to generate data!

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

Sample from  
true conditional

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



Sample from  
true prior

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

Intuition (remember from autoencoders!):  $x$  is an image,  $z$  is latent factors used to generate  $x$ : attributes, orientation, etc.

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

Slide credit: Fei-Fei Li

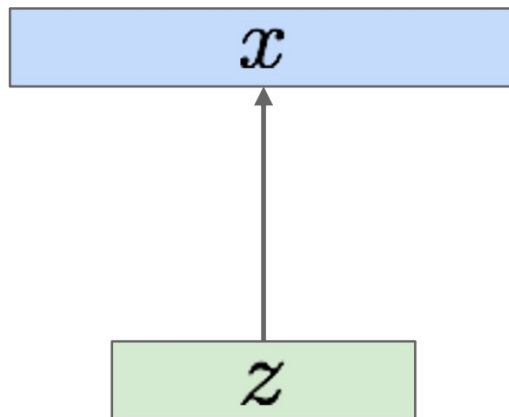
# Variational AutoEncoders

Sample from  
true conditional

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

Sample from  
true prior

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



We want to estimate the true parameters  $\theta^*$  of this generative model given training data  $x$ .

How should we represent this model?

Choose prior  $\mathbf{p}(z)$  to be simple, e.g. **Gaussian**. Reasonable for latent attributes, e.g. pose, how much smile.

Conditional  $\mathbf{p}(x|z)$  is complex (generates image) => represent with **neural network**

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

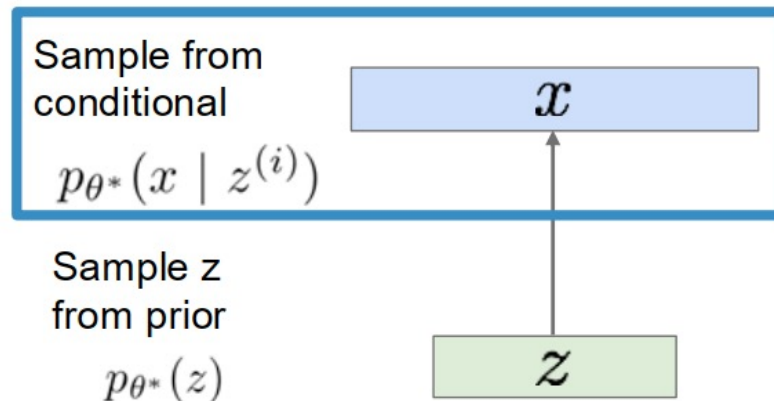
Slide credit: Fei-Fei Li

# 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  $z$

How can we train this?

Basic idea: **maximum likelihood**

We don't observe  $z$ , so **marginalize**:

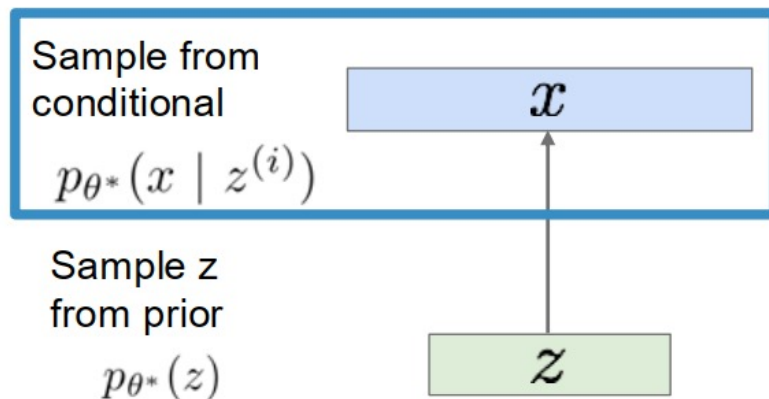
$$p_{\theta}(x) = \int p_{\theta}(x, z) dz = \int p_{\theta}(x|z)p_{\theta}(z) dz$$

# 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  $\mathbf{z}$

How can we train this?

Basic idea: **maximum likelihood**

We don't observe  $z$ , so **marginalize**:

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

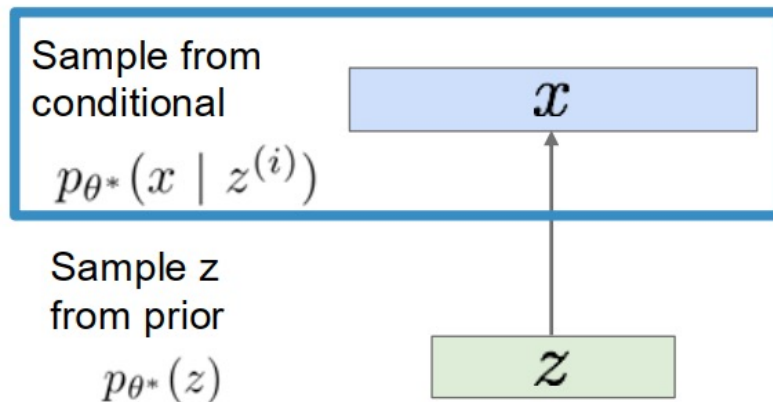
Ok, we can compute this with the decoder

# 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  $z$

How can we train this?

Basic idea: **maximum likelihood**

We don't observe  $z$ , so **marginalize**:

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

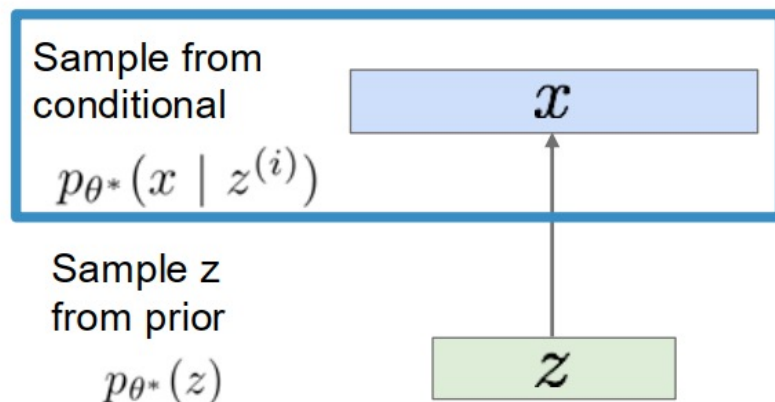
Ok, we assumed Gaussian prior for  $z$

# Variational AutoEncoders

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  $z$

How can we train this?

Basic idea: **maximum likelihood**

We don't observe  $z$ , so **marginalize**:

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

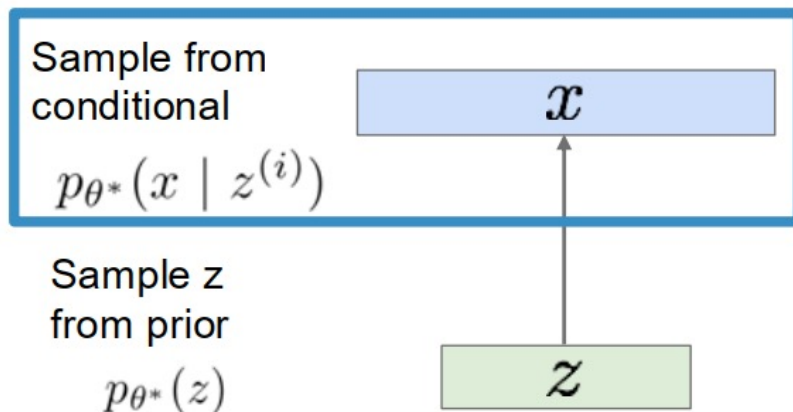
**Problem, we can't integrate over all  $z$**

# 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  $z$

How can we train this?

Basic idea: **maximum likelihood**

Another idea: Try Bayes' Rule:

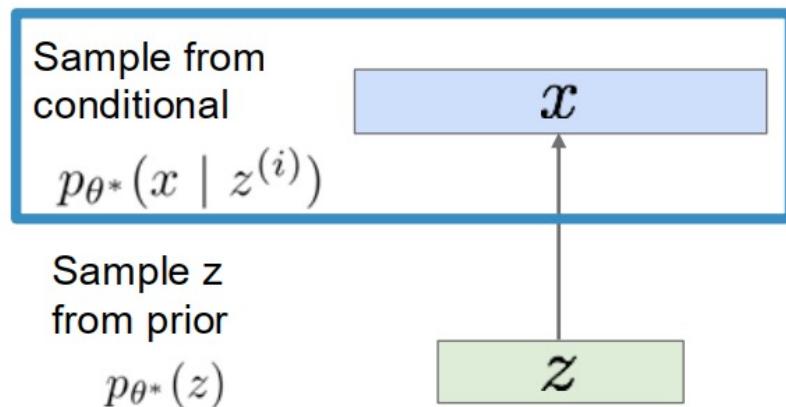
$$p_{\theta}(x) = \frac{p_{\theta}(x | z)p_{\theta}(z)}{p_{\theta}(z | x)}$$

# 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  $\mathbf{z}$

How can we train this?

Basic idea: **maximum likelihood**

Another idea: Try Bayes' Rule:

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

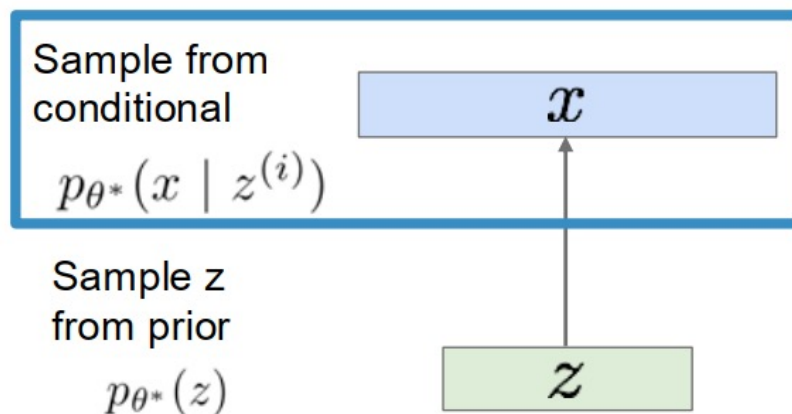
Ok, we can compute this with the decoder

# 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  $\mathbf{z}$

How can we train this?

Basic idea: **maximum likelihood**

Another idea: Try Bayes' Rule:

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

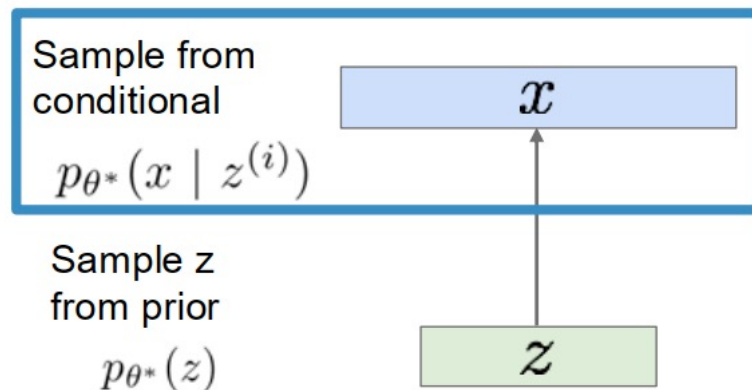
Ok, we assumed Gaussian prior for  $z$

# 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  $z$

How can we train this?

Basic idea: **maximum likelihood**

Another idea: Try Bayes' Rule:

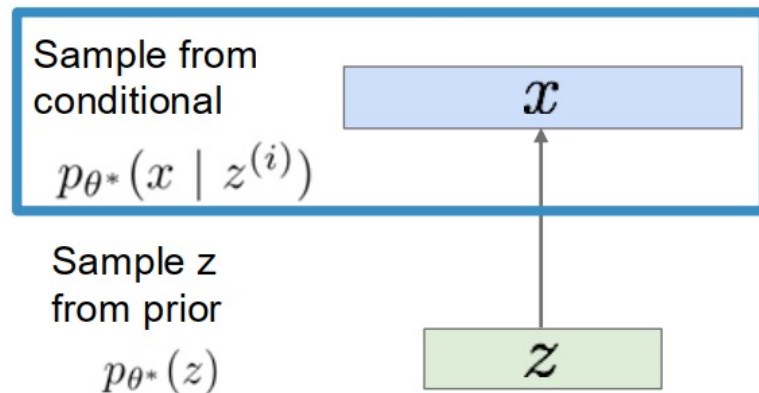
$$p_{\theta}(x) = \frac{p_{\theta}(x | z)p_{\theta}(z)}{p_{\theta}(z | x)} \quad \text{Problem: no way to compute this}$$

# 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  $z$

How can we train this?

Basic idea: **maximum likelihood**

Another idea: Try Bayes' Rule:

$$p_{\theta}(x) = \frac{p_{\theta}(x | z)p_{\theta}(z)}{p_{\theta}(z | x)} \quad \text{Problem: no way to compute this}$$

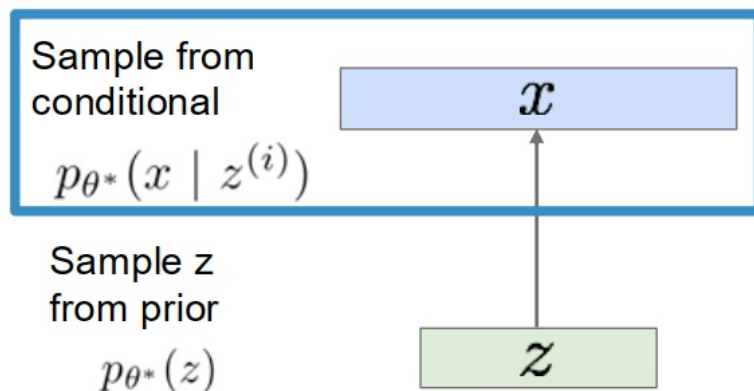
**Solution:** Train another network that learns  $q_{\phi}(z | x) \approx p_{\theta}(z | x)$

# 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  $\mathbf{z}$

How can we train this?

Basic idea: **maximum likelihood**

Another idea: Try Bayes' Rule:

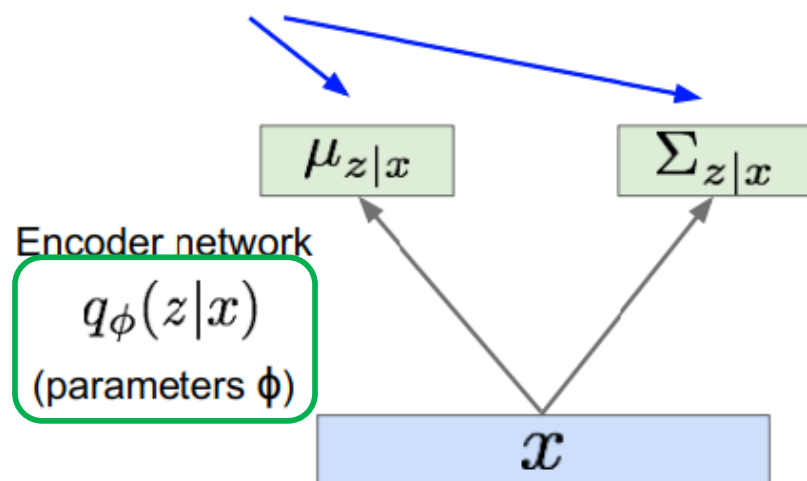
$$p_{\theta}(x) = \frac{p_{\theta}(x | z)p_{\theta}(z)}{p_{\theta}(z | x)} \approx \frac{p_{\theta}(x | z)p_{\theta}(z)}{q_{\phi}(z | x)}$$

**Solution:** Train another network that learns  $q_{\phi}(z | x) \approx p_{\theta}(z | x)$

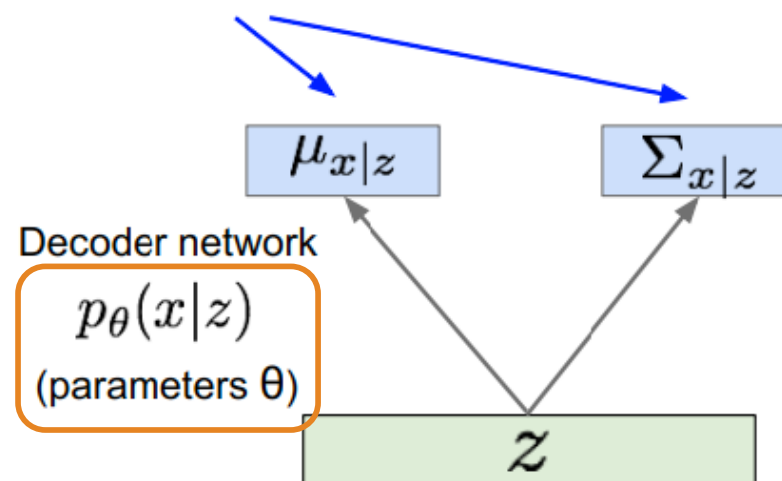
# Variational AutoEncoders

Since we're modeling probabilistic generation of data, encoder and decoder networks are probabilistic

Mean and (diagonal) covariance of  $\mathbf{z} | \mathbf{x}$



Mean and (diagonal) covariance of  $\mathbf{x} | \mathbf{z}$

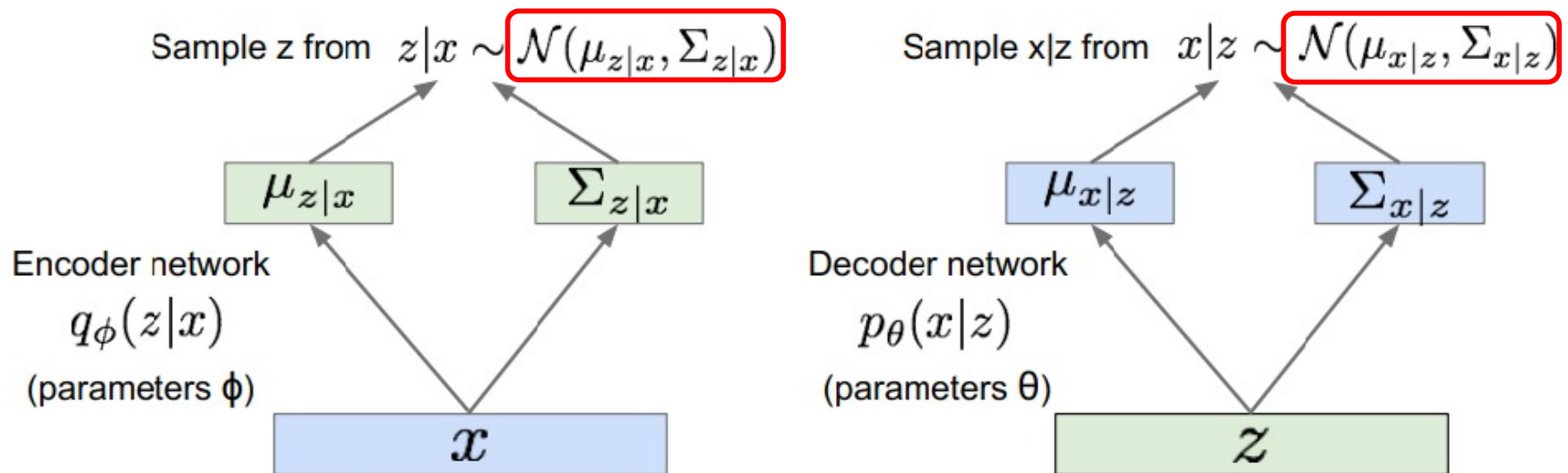


Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Slide credit: Fei-Fei Li

# Variational AutoEncoders

Since we're modeling probabilistic generation of data, encoder and decoder networks are probabilistic

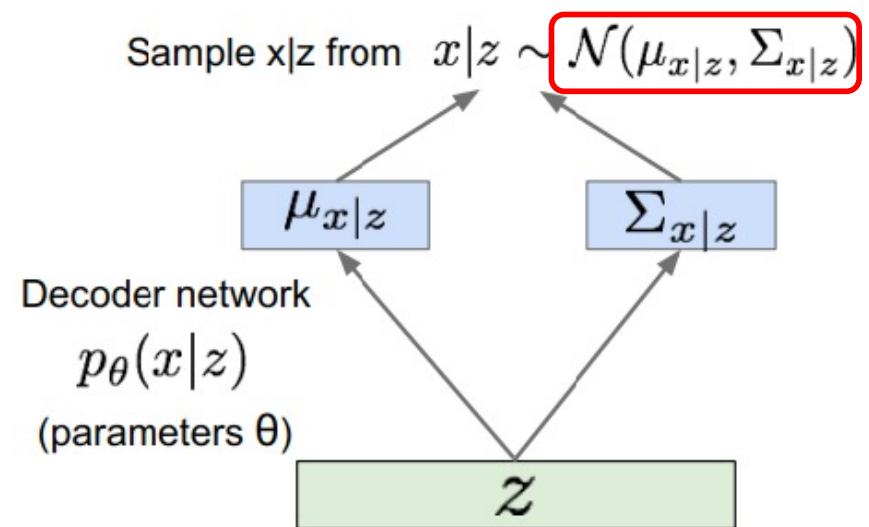
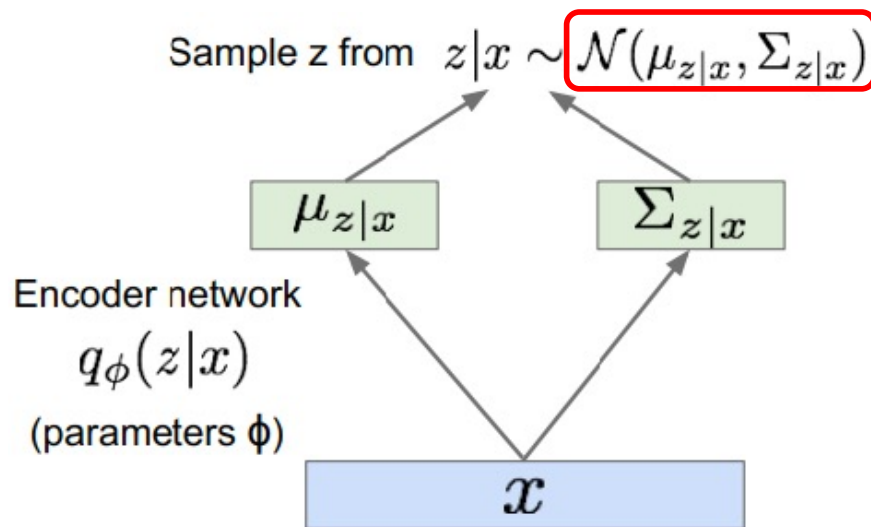


Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Slide credit: Fei-Fei Li

# Variational AutoEncoders

Since we're modeling probabilistic generation of data, encoder and decoder networks are probabilistic

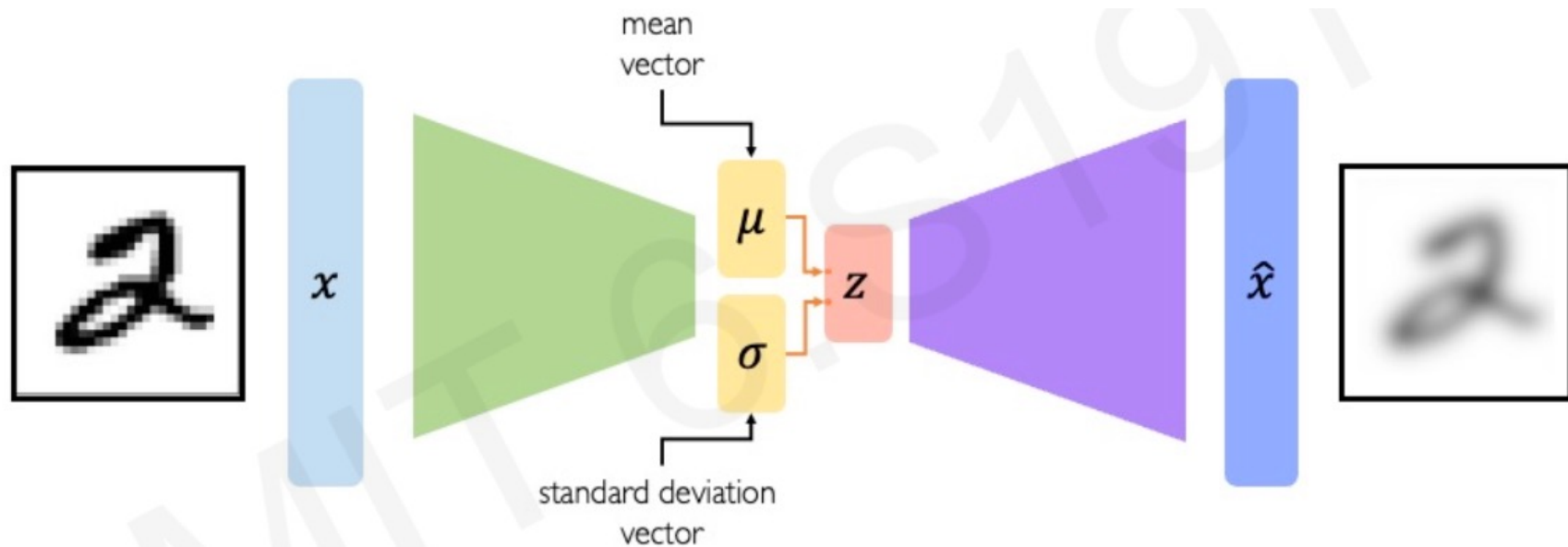


Encoder and decoder networks also called “recognition”/“inference” and “generation” networks

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

Slide credit: Fei-Fei Li

# Variational AutoEncoders: Key Difference with AutoEncoders



**Variational autoencoders are a probabilistic twist on autoencoders!**  
Sample from the mean and standard deviation to compute latent sample

# Variational AutoEncoders: Learning

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

Bayes' Rule

## Variational AutoEncoders: Learning

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

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

## Variational AutoEncoders: Learning

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

Logarithms + rearranging

## Variational AutoEncoders: Learning

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

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

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

## Variational AutoEncoders: Learning

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

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

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

## Variational AutoEncoders: Learning

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

## Variational AutoEncoders: Learning

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

**Data reconstruction:**  $x \Rightarrow$  encoder  $\Rightarrow$  decoder should reconstruct  $x$   
 Can compute in closed form for Gaussians.

## Variational AutoEncoders: Learning

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

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

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

**Prior:** Encoder output should match prior over  $z$ .

Can compute in closed form for Gaussians.

## Variational AutoEncoders: Learning

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

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

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

**Posterior Approximation:** Encoder output  $q_{\phi}(z|x)$  should match  $p_{\theta}(z|x)$

We cannot compute this for Gaussians...

## Variational AutoEncoders: Learning

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

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

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

**Posterior Approximation:** Decoder output  $q_{\phi}(z|x)$  should match  $p_{\theta}(z|x)$

KL is  $\geq 0$ , so we can drop it to get a lower bound on likelihood

## Variational AutoEncoders: Learning

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

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

This is our VAE training objective

# Variational AutoEncoders: KL Divergence

$$D_{KL}(\mathcal{N}(\mu_{z|x}, \Sigma_{z|x}) || \mathcal{N}(0, I))$$

Inferred Latent  
Distribution

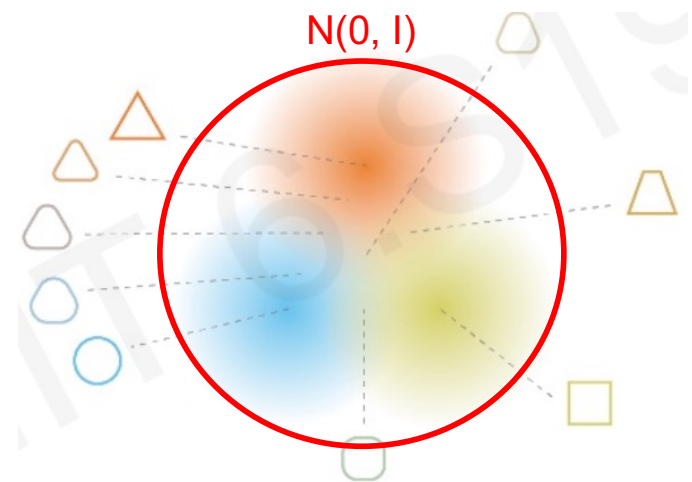
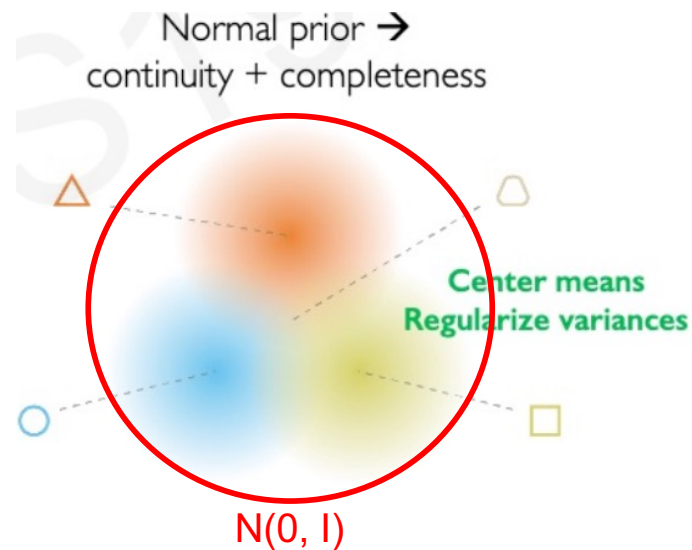
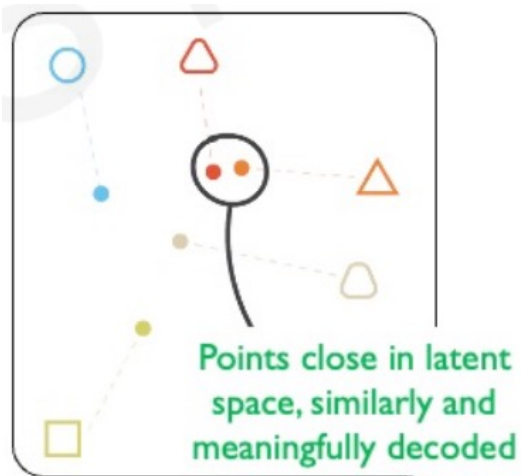
Fixed Prior on  
Latent Distribution

- Encourage encodings to distribute them evenly around the center of the latent space
- Penalize the network when it tries to “cheat” by clustering points in specific regions (i.e. memorizing data). However, it does not ensure smooth transition of data.

# Variational AutoEncoders: KL Divergence

What properties do we want to achieve from KL Divergence?

1. **Continuity:** Points that are close in latent space  $\rightarrow$  similar content after decoding
2. **Completeness:** sampling from latent space  $\rightarrow$  “meaningful” content after decoding



# Variational AutoEncoders

Putting it all together: maximizing the likelihood lower bound

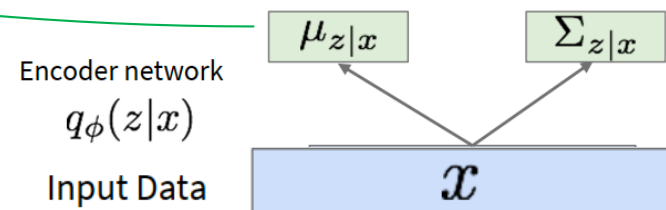
$$\underbrace{\mathbf{E}_z \left[ \log p_\theta(x^{(i)} | z) \right]}_{\mathcal{L}(x^{(i)}, \theta, \phi)} - \underbrace{D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\text{KL Divergence}}$$

Let's look at computing the KL divergence between the estimated posterior and the prior given some data

Make approximate posterior distribution close to prior

$$D_{KL}(\mathcal{N}(\mu_{z|x}, \Sigma_{z|x}) || \mathcal{N}(0, I))$$

Have analytical solution



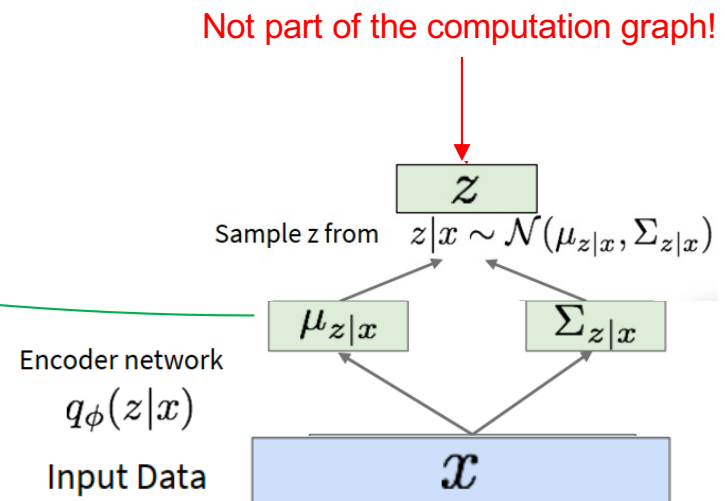
Slide credit: Fei-Fei Li

# Variational AutoEncoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z \left[ \log p_\theta(x^{(i)} | z) \right]}_{\mathcal{L}(x^{(i)}, \theta, \phi)} - \underbrace{D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\text{Make approximate posterior distribution close to prior}}$$

Make approximate posterior distribution close to prior



Slide credit: Fei-Fei Li

# Variational AutoEncoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

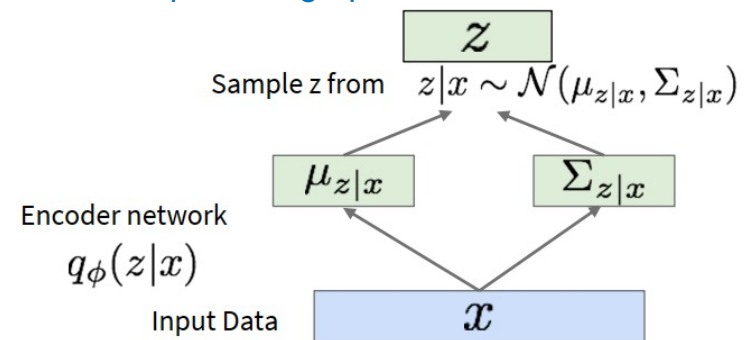
Reparameterization trick to make sampling differentiable:

Sample  $\epsilon \sim \mathcal{N}(0, I)$

$$z = \mu_{z|x} + \epsilon \sigma_{z|x}$$

Part of computation graph

Input to the graph

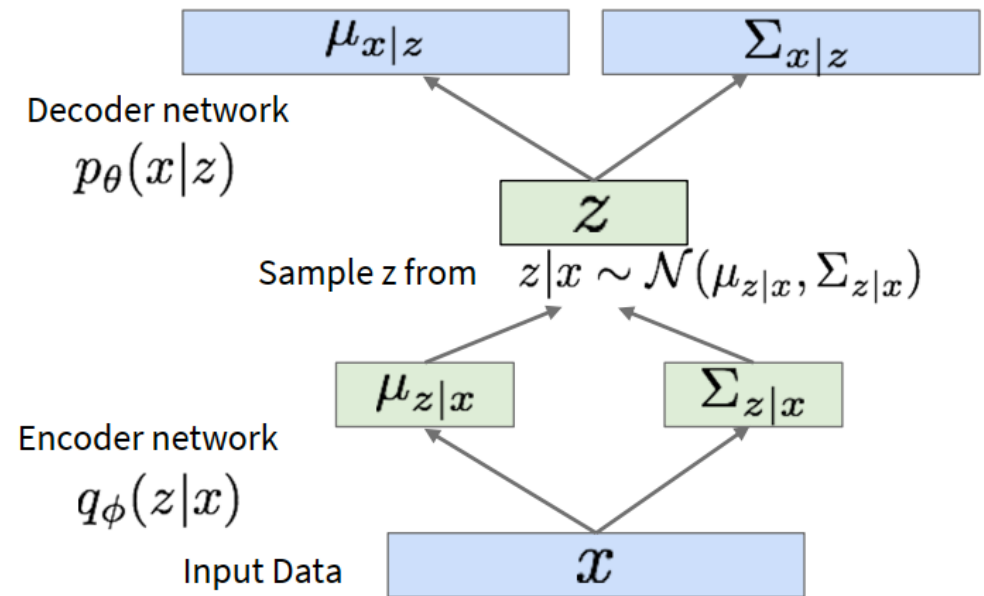


Slide credit: Fei-Fei Li

# Variational AutoEncoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z \left[ \log p_\theta(x^{(i)} | z) \right]}_{\mathcal{L}(x^{(i)}, \theta, \phi)} - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))$$

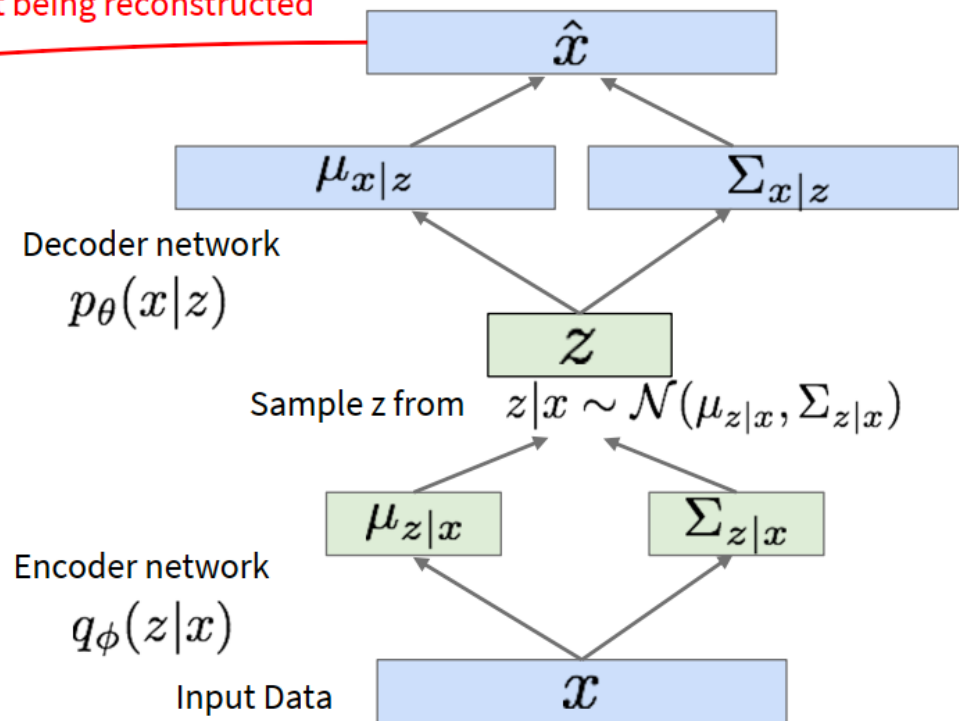


# Variational AutoEncoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - \mathcal{D}_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

Maximize likelihood of original input being reconstructed

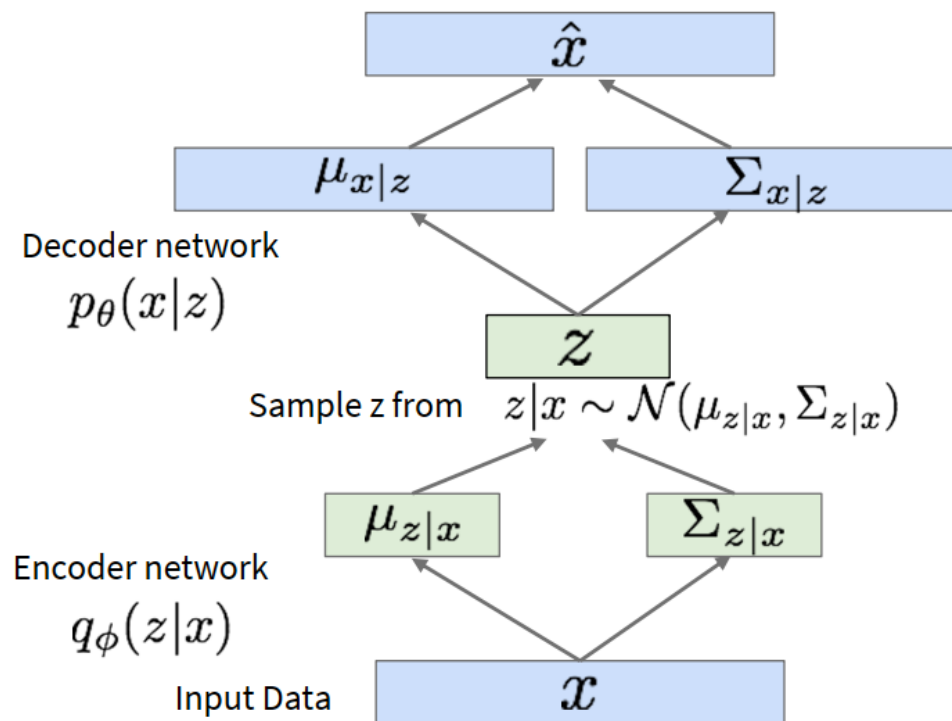


# Variational AutoEncoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z \left[ \log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

For every minibatch of input data: compute this forward pass, and then backprop!

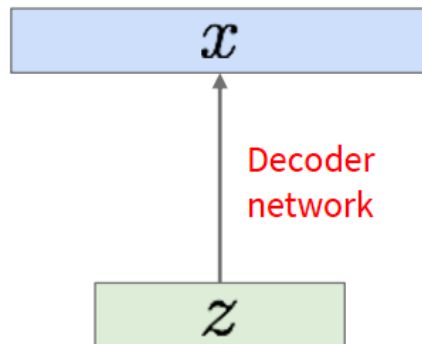


# Variational AutoEncoders: Generating Data!

Our assumption about data generation process

Sample from true conditional

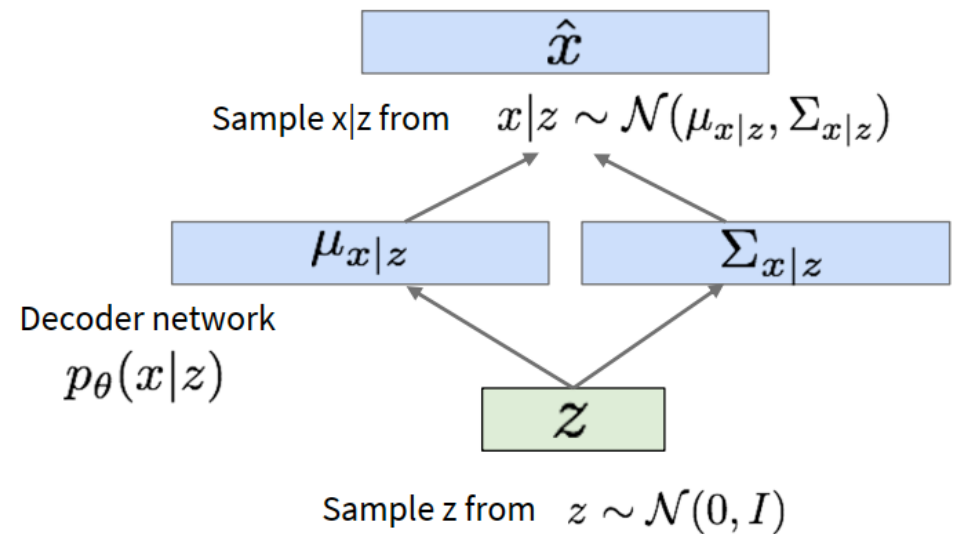
$$p_{\theta^*}(x | z^{(i)})$$



Sample from true prior

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

Now given a trained VAE:  
use decoder network & sample  $z$  from prior!



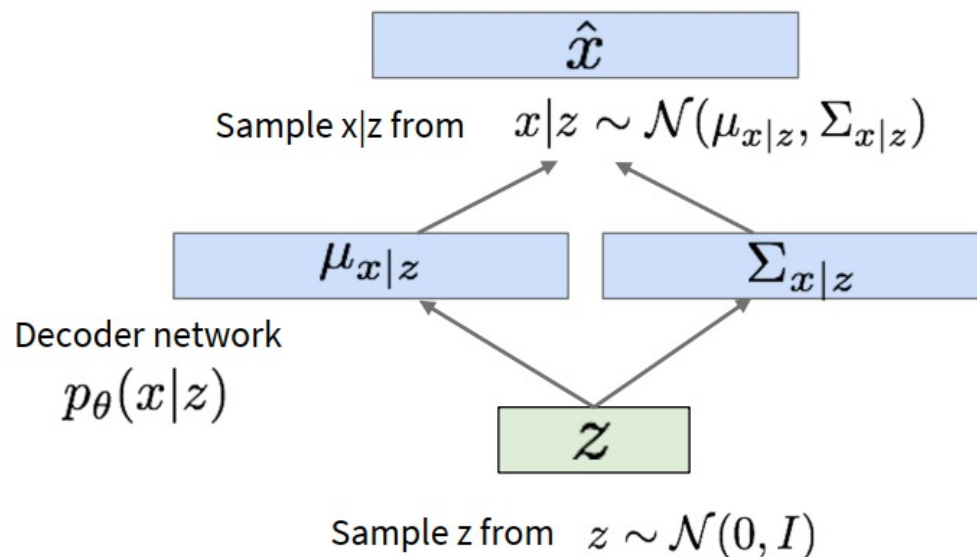
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Slide credit: Fei-Fei Li

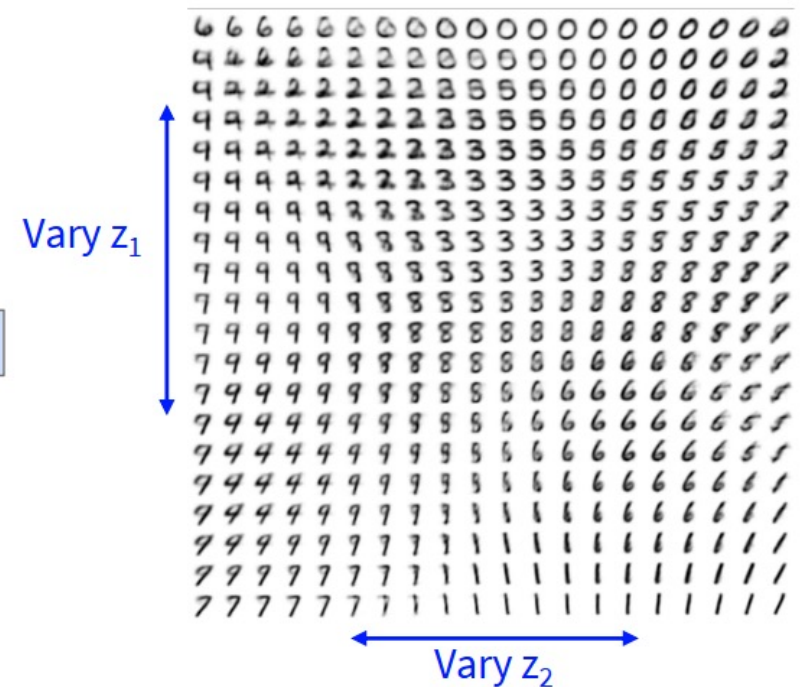


# Variational AutoEncoders: Generating Data!

Use decoder network. Now sample  $z$  from prior!



Data manifold for 2-d  $z$



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

Slide credit: Fei-Fei Li

# Variational AutoEncoders: Generating Data!

Diagonal prior on  $z$   
 => independent  
 latent variables

Different dimensions  
 of  $z$  encode  
 interpretable factors  
 of variation

Also good feature representation that  
 can be computed using  $q_\phi(z|x)$ !

Degree of smile

Vary  $z_1$



Vary  $z_2$

Head pose

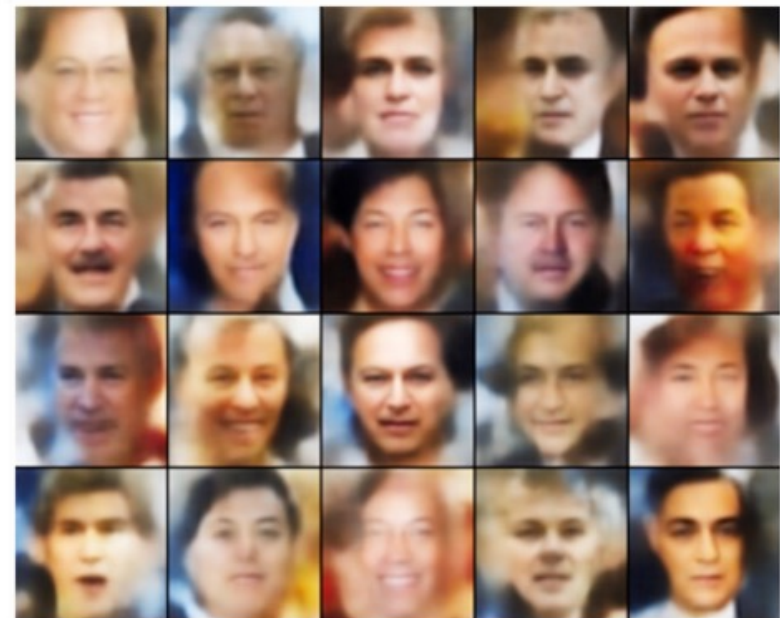
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Slide credit: Fei-Fei Li

# Variational AutoEncoders: Generating Data!



32x32 CIFAR-10



Labeled Faces in the Wild

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

Slide credit: Fei-Fei Li

# Variational AutoEncoders: Generating Data!

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
- Interpretable latent space.
- Allows inference of  $q(z|x)$ , can be useful feature representation for other tasks

## Cons:

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

## Active areas of research:

- More flexible approximations, e.g. richer approximate posterior instead of diagonal Gaussian, e.g., Gaussian Mixture Models (GMMs), Categorical Distributions.
- Learning disentangled representations.

# Taxonomy of Generative Models

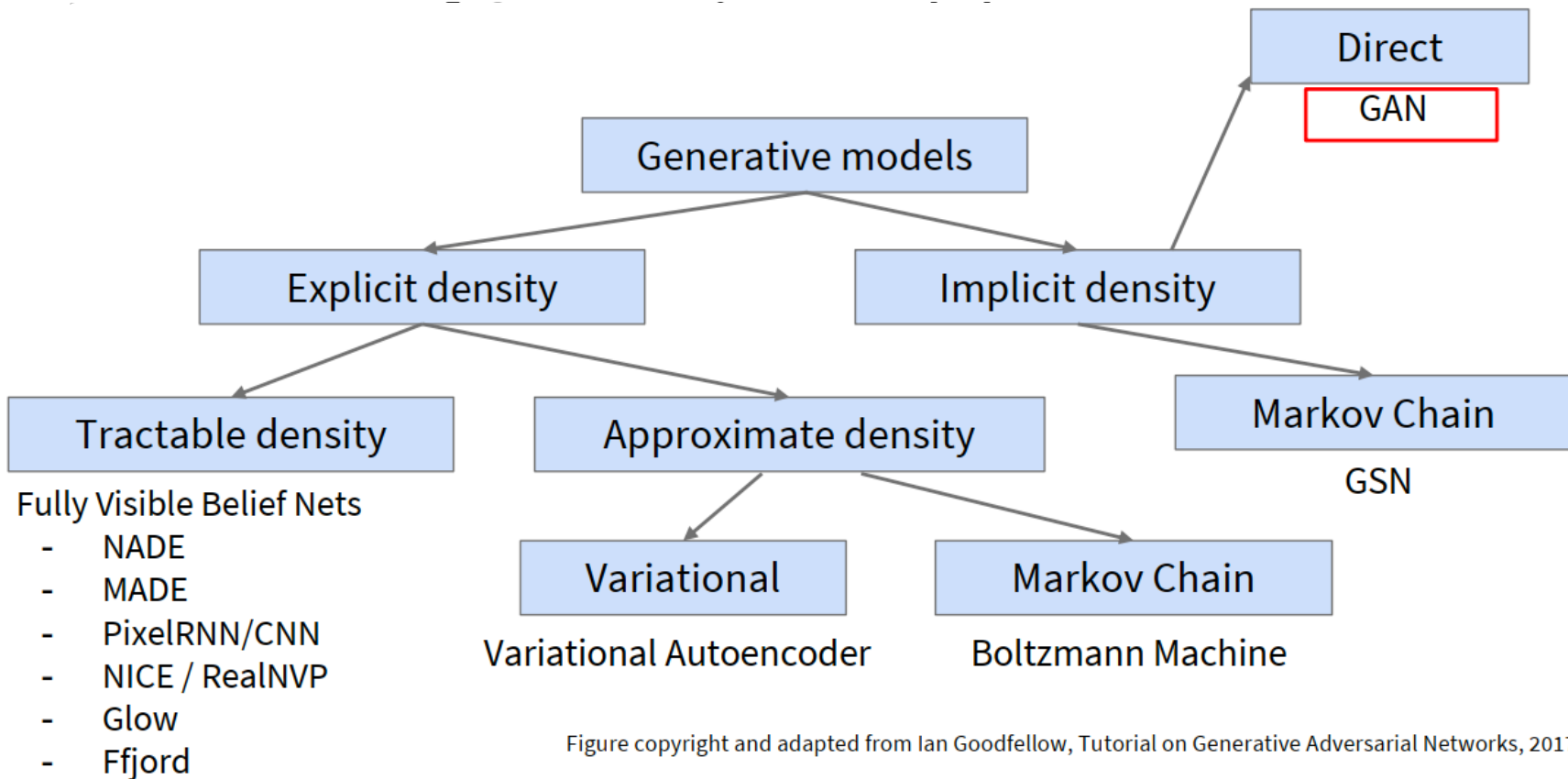


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Slide credit: Fei-Fei Li

# Generative Adversarial Networks (GANs)

PixelRNN/CNNs define tractable density function, optimize likelihood of training data:

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

VAEs define intractable density function with latent  $z$ :

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

Cannot optimize directly, derive and optimize lower bound on likelihood instead

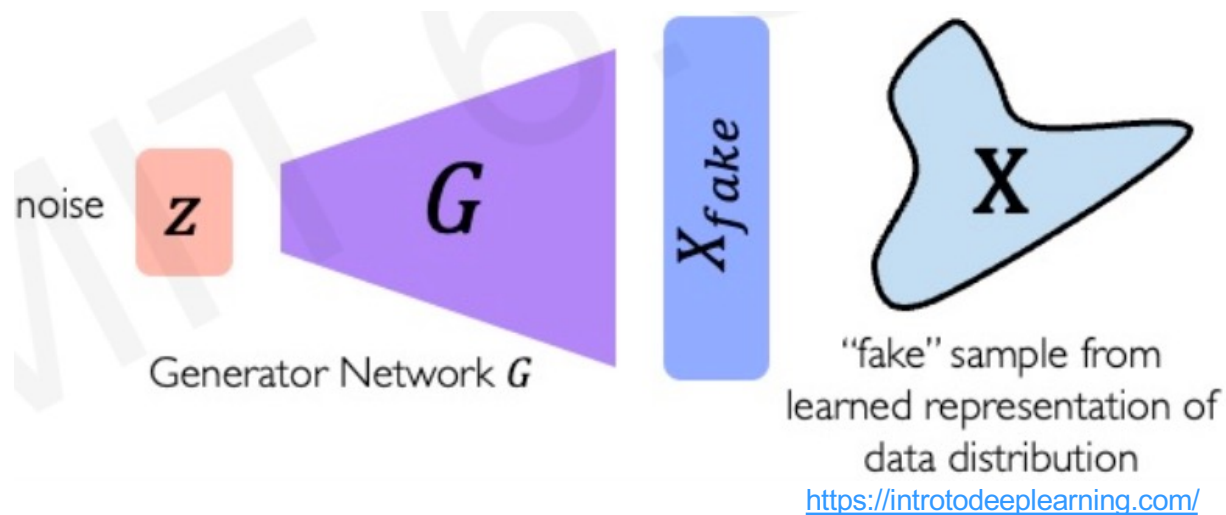
What if we give up on explicitly modeling density, and just want ability to sample?

GANs: not modeling any explicit density function!

# Generative Adversarial Networks (GANs)

**Problem:** Want to sample from complex, high-dimensional training distribution. No direct way to do this!

**Solution:** Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

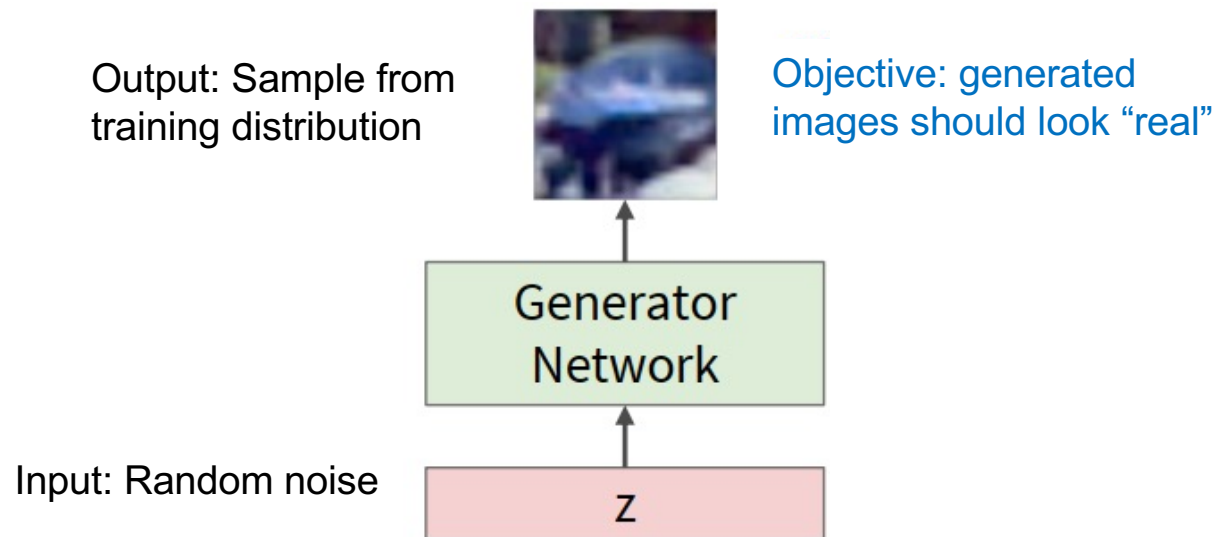


# Generative Adversarial Networks (GANs)

**Problem:** Want to sample from complex, high-dimensional training distribution. No direct way to do this!

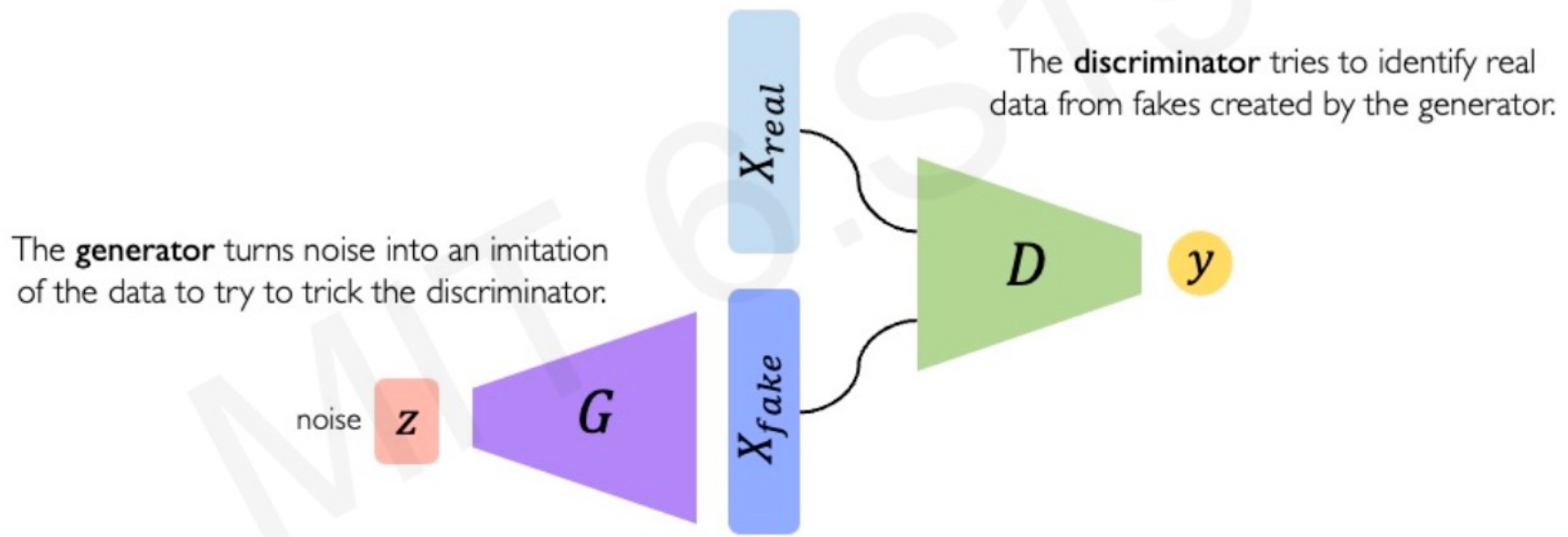
**Solution:** Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

But we don't know which sample  $z$  maps to which training image -> can't learn by reconstructing training images

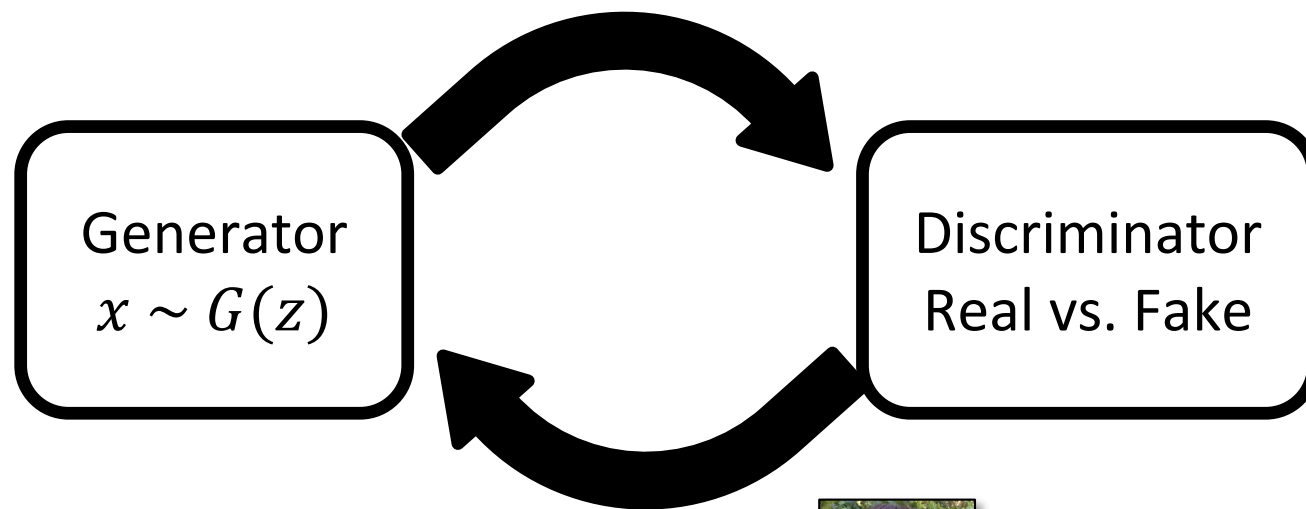


# Generative Adversarial Networks (GANs)

**Generative Adversarial Networks (GANs)** are a way to make a generative model by having two neural networks compete with each other.



# Adversarial Networks Framework

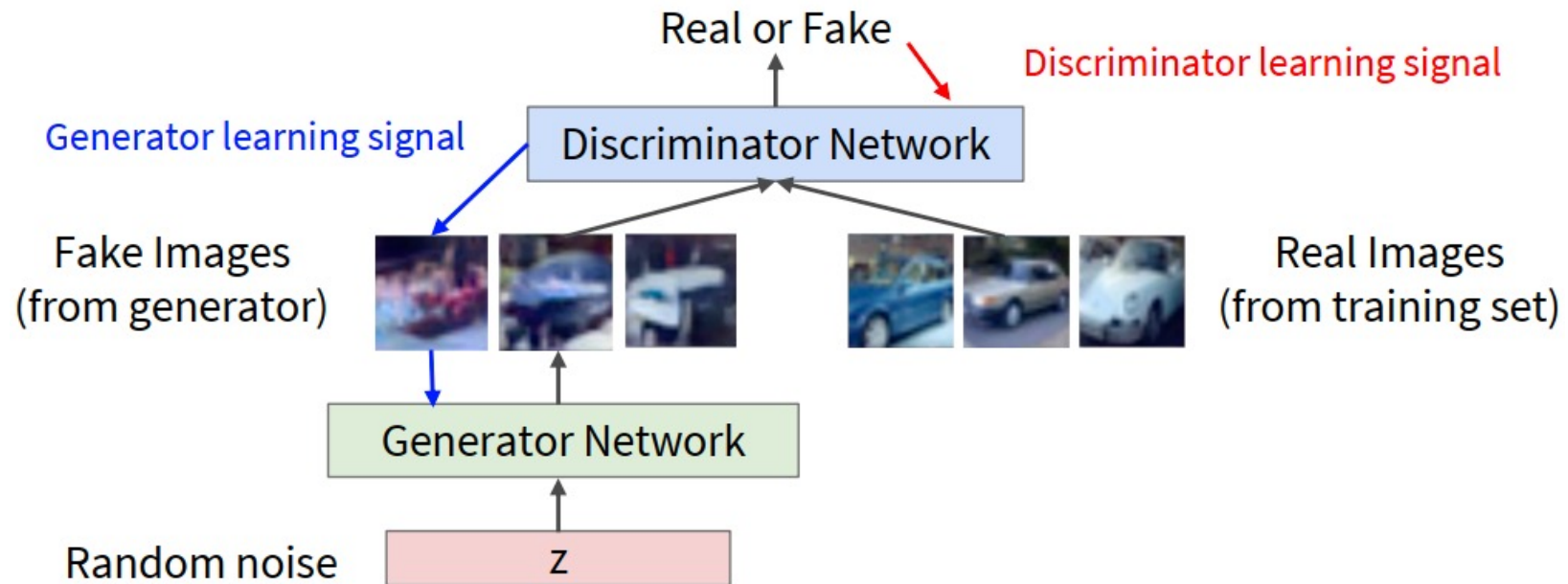


[Goodfellow et al. 2014]

# Training GANs: Two-player game

Discriminator network: try to distinguish between real and fake images

Generator network: try to fool the discriminator by generating real-looking images



# Training GANs: Two-player game

**Discriminator network:** try to distinguish between real and fake images

**Generator network:** try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

Minimax Objective Function:

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

Generator objective      Discriminator objective

# Training GANs: Two-player game

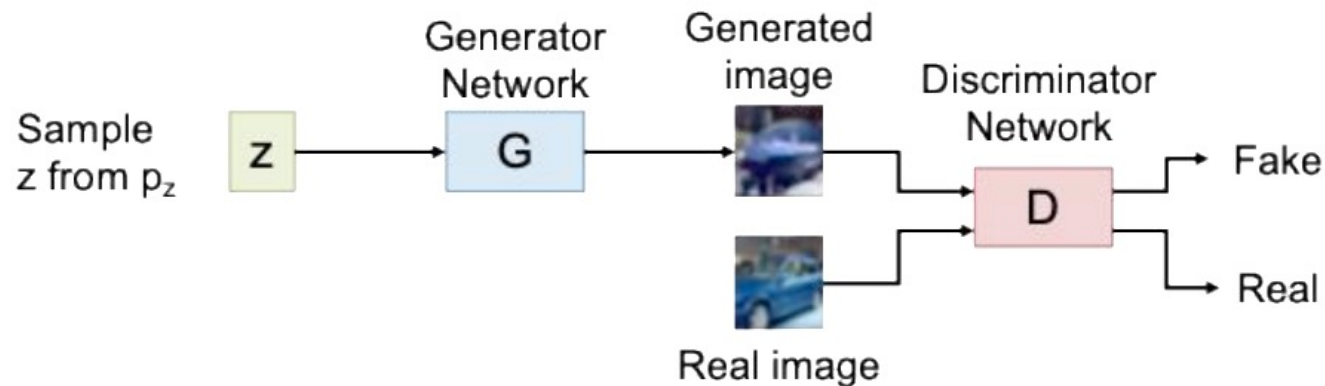
Jointly train generator  $G$  and discriminator  $D$  with a **minimax game**

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

# Training GANs: Two-player game

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

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



$$D(x) = P(x \text{ is real})$$

$$D(x) = 0 \Rightarrow \text{fake}$$

$$D(x) = 1 \Rightarrow \text{real}$$

# Training GANs: Two-player game

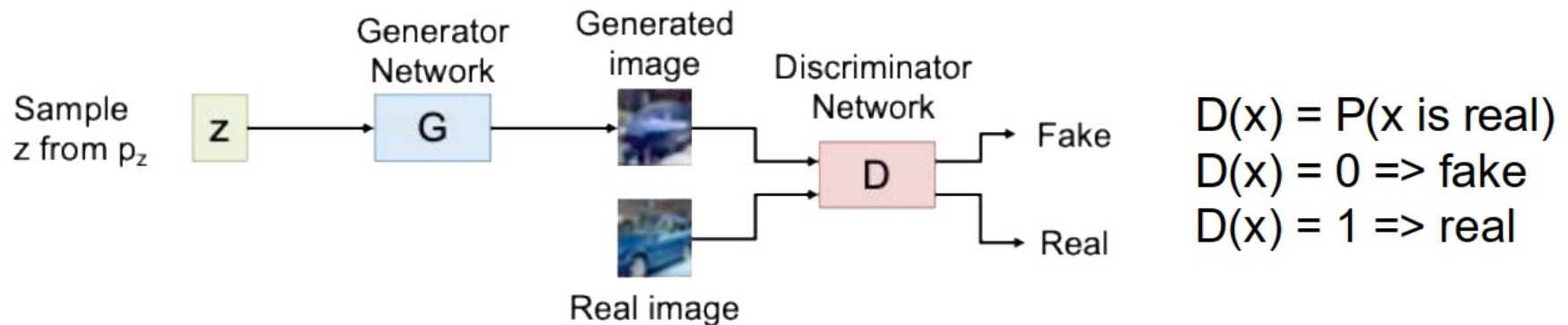
Jointly train generator  $G$  and discriminator  $D$  with a **minimax game**

Imagine fixing  $G$

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

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

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



# Training GANs: Two-player game

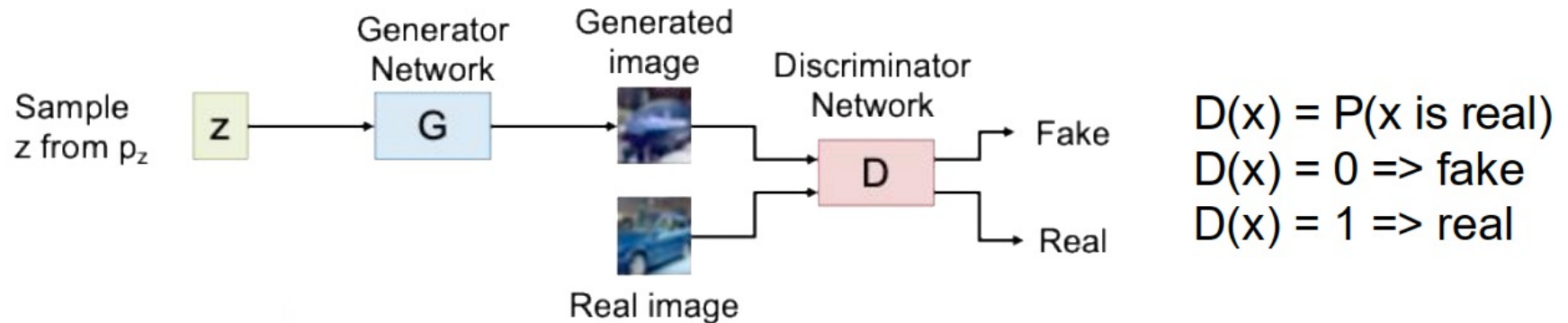
Jointly train generator  $G$  and discriminator  $D$  with a **minimax game**

Imagine fixing  $D$

This term does not depend on  $G$

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

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



# Training GANs: Two-player game

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

Train  $G$  and  $D$  using alternating gradient updates

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

$$= \min_G \max_D V(G, D)$$

While True:

$$D = D + \alpha_D \frac{dV}{dD}$$

$$G = G - \alpha_G \frac{dV}{dG}$$

# Training GANs: Two-player game

Minimax objective function:

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

Alternate between:

**1. Gradient ascent** on discriminator

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

**2. Gradient descent** on generator

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

# Training GANs: Two-player game

Minimax objective function:

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

Alternate between:

## 1. Gradient ascent on discriminator

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

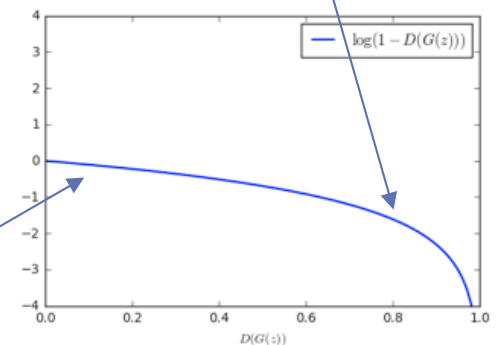
## 2. Gradient descent on generator

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

When sample is likely fake, want to learn from it to improve generator (move to the right on X axis).

But gradient in this region is relatively flat!

Gradient signal dominated by region where sample is already good



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

# Training GANs: Two-player game

Minimax objective function:

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

Alternate between:

1. Gradient ascent on discriminator

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

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

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

# Training GANs: Two-player game

Putting it together: GAN training algorithm

**for** number of training iterations **do**

**for**  $k$  steps **do**

- Sample minibatch of  $m$  noise samples  $\{z^{(1)}, \dots, z^{(m)}\}$  from noise prior  $p_g(z)$ .
- Sample minibatch of  $m$  examples  $\{x^{(1)}, \dots, x^{(m)}\}$  from data generating distribution  $p_{\text{data}}(x)$ .
- Update the discriminator by ascending its stochastic gradient:

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

**end for**

- Sample minibatch of  $m$  noise samples  $\{z^{(1)}, \dots, z^{(m)}\}$  from noise prior  $p_g(z)$ .
- Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

**end for**

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

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

# Training GANs: Other Divergence

## Least Squares GAN

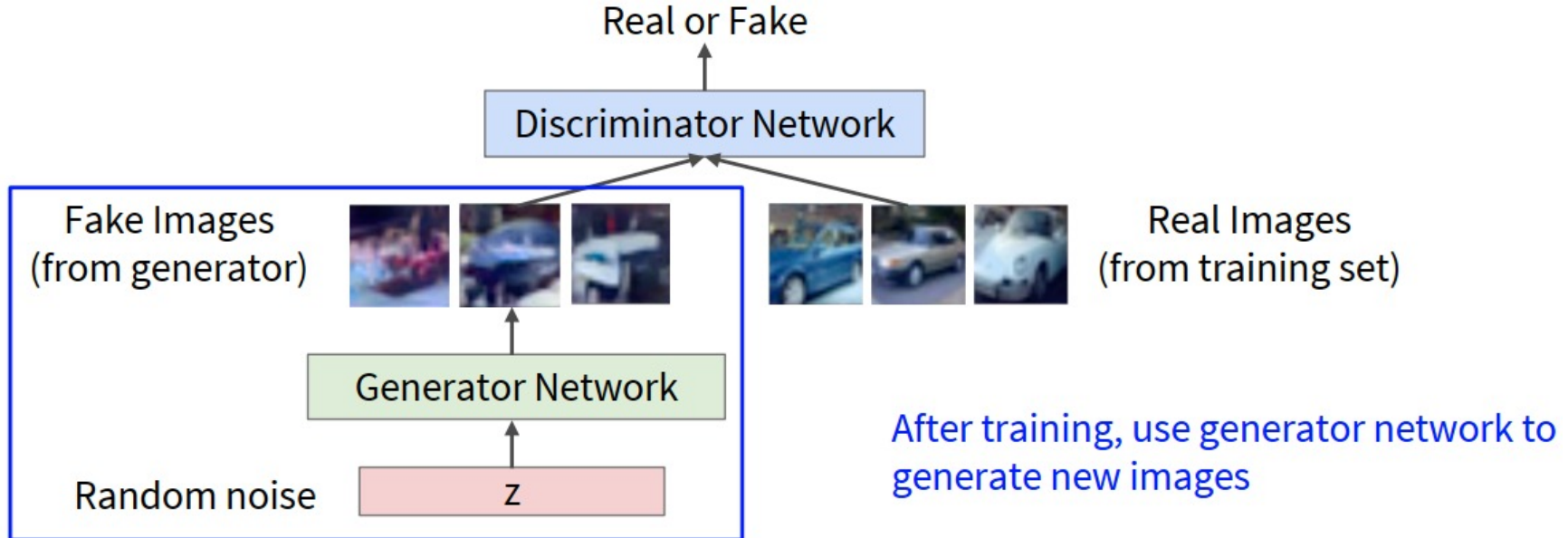
$$\min_D V_{\text{LSGAN}}(D) = \frac{1}{2} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [(D(\mathbf{x}) - 1)^2] + \frac{1}{2} \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [(D(G(\mathbf{z})))^2]$$
$$\min_G V_{\text{LSGAN}}(G) = \frac{1}{2} \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [(D(G(\mathbf{z}))) - 1]^2.$$

\* Employed for our Assignment.

# Training GANs: Two-player game

**Generator network:** try to fool the discriminator by generating real-looking images

**Discriminator network:** try to distinguish between real and fake images



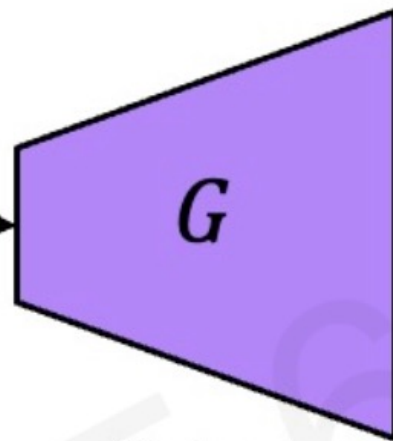
Fake and real images copyright Emily Denton et al. 2015

# GANs: Generating New Data

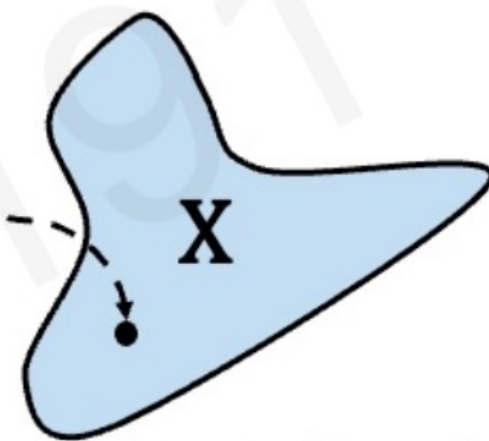
GANs are Distribution Transformers

Gaussian noise

$$z \sim \mathcal{N}(0,1)$$



Trained  
generator

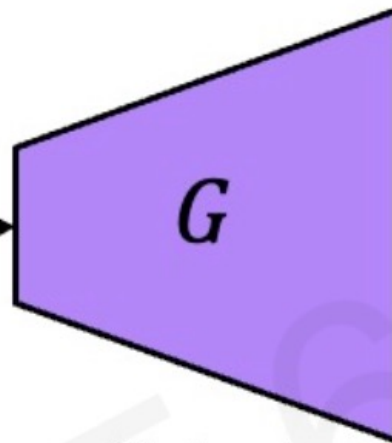
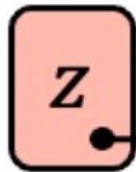


Learned target  
data distribution

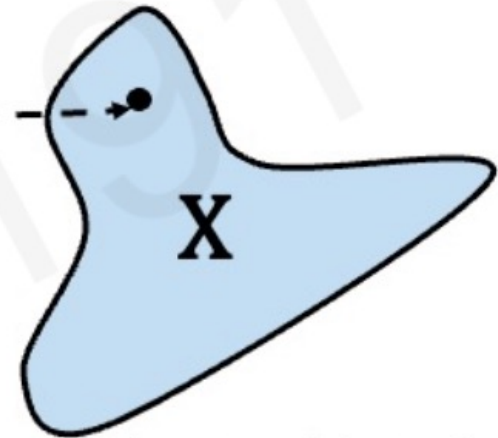
# GANs: Generating New Data

GANs are Distribution Transformers

Gaussian noise  
 $z \sim N(0,1)$



Trained  
generator

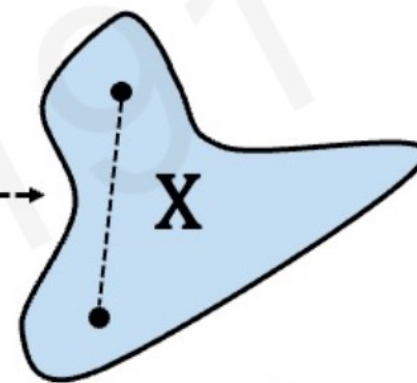
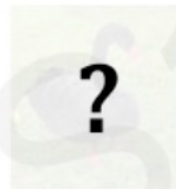
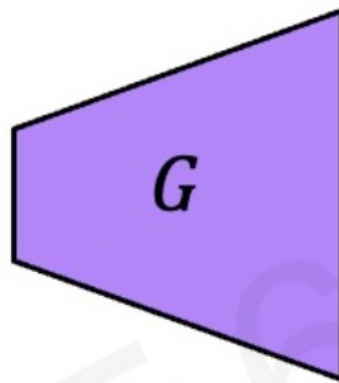


Learned target  
data distribution

# GANs: Generating New Data

GANs are Distribution Transformers

Gaussian noise  
 $z \sim N(0,1)$



Trained  
generator

Learned target  
data distribution



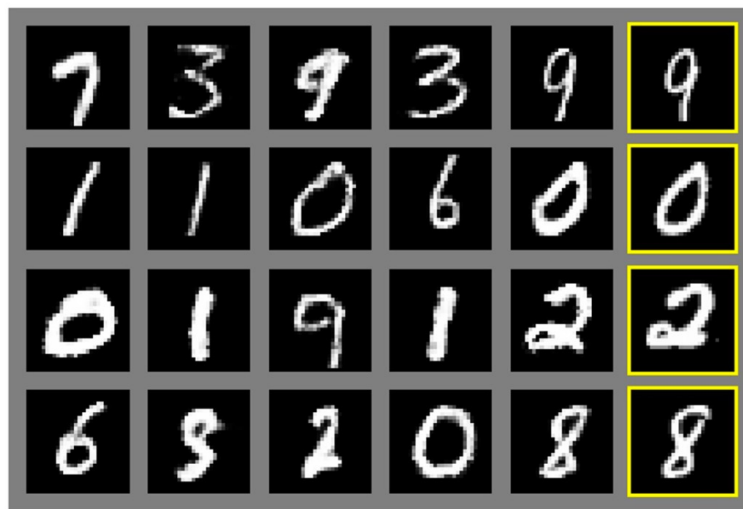
# Lab 9a: Generative Adversarial Networks (GANs) for Digits

Duration: 10 min



# Generative Adversarial Nets

Generated samples



Nearest neighbor from training set

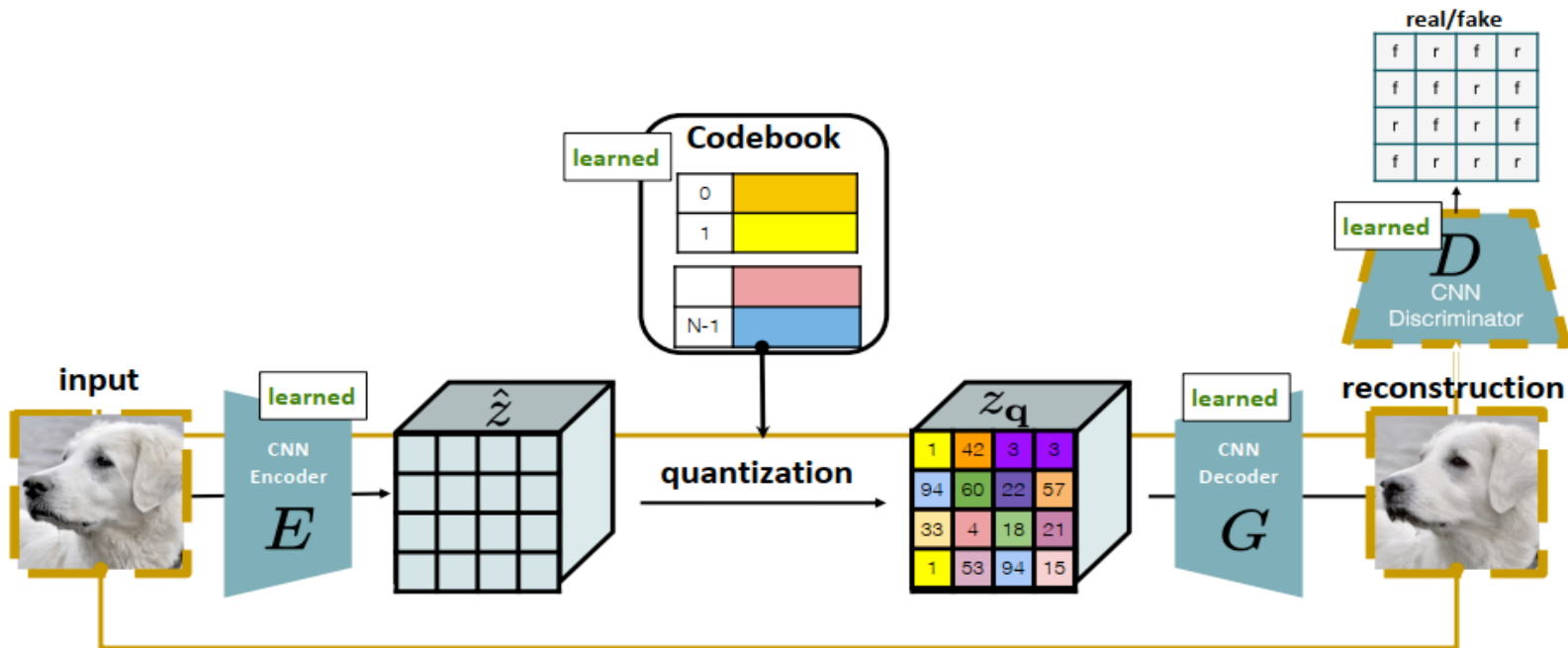
# Generative Adversarial Nets

Generated samples (CIFAR-10)



Nearest neighbor from training set

# From VQ-VAE to VQGAN



$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{rec}} + \mathcal{L}_{\text{VQ}} + \lambda \mathcal{L}_{\text{GAN}} \text{ where } \lambda = \frac{\nabla_{G_L}[\mathcal{L}_{\text{rec}}]}{\nabla_{G_L}[\mathcal{L}_{\text{GAN}}] + \delta}$$

# From VQ-VAE to VQGAN



Slide credit: Robin Rombach



# Generative Adversarial Nets: DC-GAN

**Generator** is an upsampling network with fractionally-strided convolutions

**Discriminator** is a convolutional network

## Architecture guidelines for stable Deep Convolutional GANs

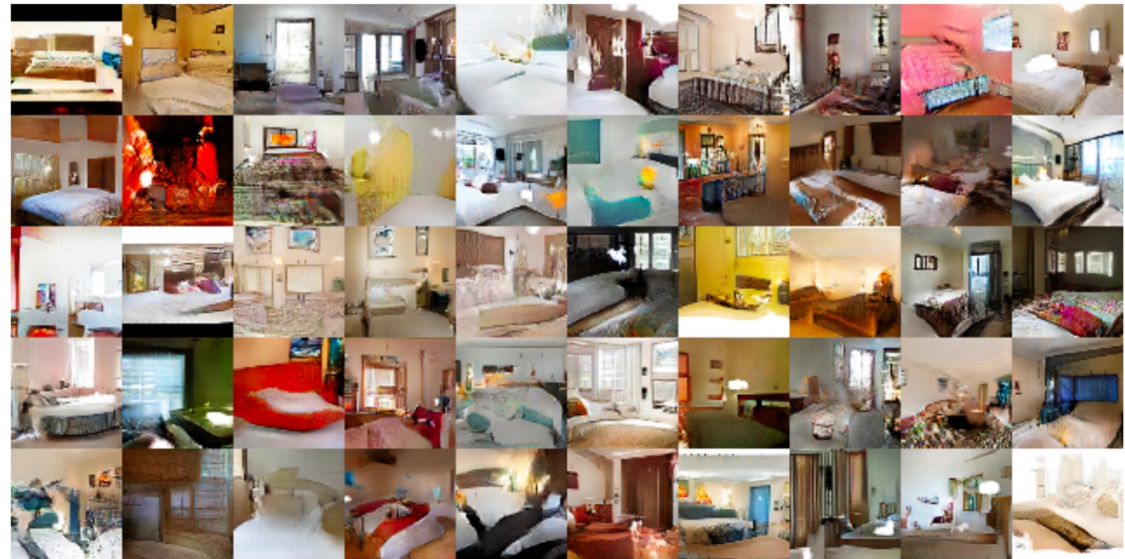
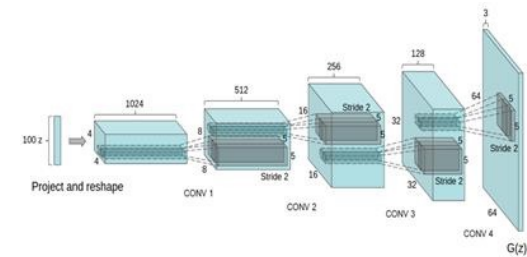
- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

# Generative Adversarial Nets: DC-GAN

Generator G and discriminator D are both neural networks

Usually CNNs ... GANs fell out of favor before ViT became popular

DC-GAN was the first GAN architecture that worked on non-toy data



# Generative Adversarial Nets: DC-GAN

---

## Improving Language Understanding by Generative Pre-Training

---

Alec Radford  
OpenAI  
alec@openai.com

Karthik Narasimhan  
OpenAI  
karthikn@openai.com

Tim Salimans  
OpenAI  
tim@openai.com

Ilya Sutskever  
OpenAI  
ilyasu@openai.com

---

## Language Models are Unsupervised Multitask Learners

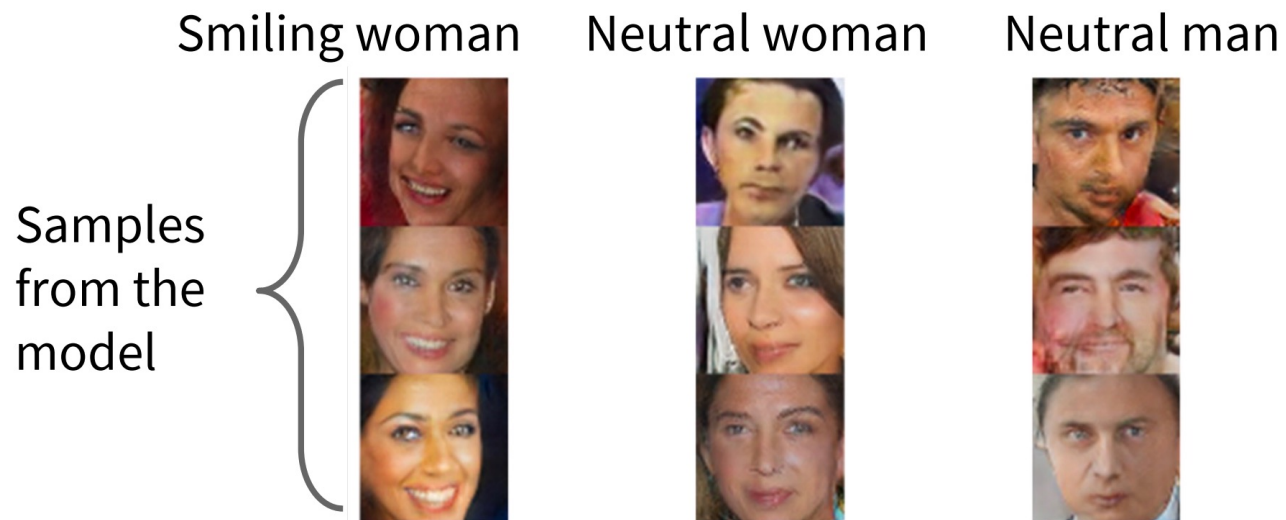
---

Alec Radford \*<sup>1</sup> Jeffrey Wu \*<sup>1</sup> Rewon Child<sup>1</sup> David Luan<sup>1</sup> Dario Amodei \*\*<sup>1</sup> Ilya Sutskever \*\*<sup>1</sup>



# GANs: Interpretable Vector Math

Radford et al, ICLR 2016



Adapted from Serena Young

# GANs: Interpretable Vector Math

Glasses man



No glasses man



No glasses woman



Radford et al,  
ICLR 2016

Woman with glasses



-

+

=

Adapted from Serena Young

# 2017: Explosion of GANs

## “The GAN Zoo”

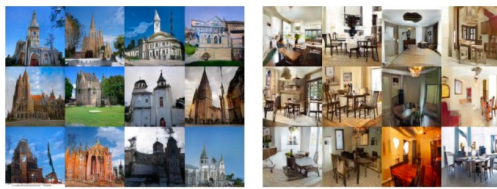
- GAN - Generative Adversarial Networks
- 3D-GAN - Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling
- acGAN - Face Aging With Conditional Generative Adversarial Networks
- AC-GAN - Conditional Image Synthesis With Auxiliary Classifier GANs
- AdaGAN - AdaGAN: Boosting Generative Models
- AEGAN - Learning Inverse Mapping by Autoencoder based Generative Adversarial Nets
- AffGAN - Amortised MAP Inference for Image Super-resolution
- AL-CGAN - Learning to Generate Images of Outdoor Scenes from Attributes and Semantic Layouts
- ALI - Adversarially Learned Inference
- AM-GAN - Generative Adversarial Nets with Labeled Data by Activation Maximization
- AnoGAN - Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery
- ArtGAN - ArtGAN: Artwork Synthesis with Conditional Categorical GANs
- b-GAN - b-GAN: Unified Framework of Generative Adversarial Networks
- Bayesian GAN - Deep and Hierarchical Implicit Models
- BEGAN - BEGAN: Boundary Equilibrium Generative Adversarial Networks
- BiGAN - Adversarial Feature Learning
- BS-GAN - Boundary-Seeking Generative Adversarial Networks
- CGAN - Conditional Generative Adversarial Nets
- CaloGAN - CaloGAN: Simulating 3D High Energy Particle Showers in Multi-Layer Electromagnetic Calorimeters with Generative Adversarial Networks
- CCGAN - Semi-Supervised Learning with Context-Conditional Generative Adversarial Networks
- CatGAN - Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks
- CoGAN - Coupled Generative Adversarial Networks
- Context-RNN-GAN - Contextual RNN-GANs for Abstract Reasoning Diagram Generation
- C-RNN-GAN - C-RNN-GAN: Continuous recurrent neural networks with adversarial training
- CS-GAN - Improving Neural Machine Translation with Conditional Sequence Generative Adversarial Nets
- CVAE-GAN - CVAE-GAN: Fine-Grained Image Generation through Asymmetric Training
- CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks
- DTN - Unsupervised Cross-Domain Image Generation
- DCGAN - Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
- DiscoGAN - Learning to Discover Cross-Domain Relations with Generative Adversarial Networks
- DR-GAN - Disentangled Representation Learning GAN for Pose-Invariant Face Recognition
- DualGAN - DualGAN: Unsupervised Dual Learning for Image-to-Image Translation
- EBGAN - Energy-based Generative Adversarial Network
- f-GAN - f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization
- FF-GAN - Towards Large-Pose Face Frontalization in the Wild
- GAWWN - Learning What and Where to Draw
- GeneGAN - GeneGAN: Learning Object Transfiguration and Attribute Subspace from Unpaired Data
- Geometric GAN - Geometric GAN
- GoGAN - Gang of GANs: Generative Adversarial Networks with Maximum Margin Ranking
- GP-GAN - GP-GAN: Towards Realistic High-Resolution Image Blending
- IAN - Neural Photo Editing with Introspective Adversarial Networks
- iGAN - Generative Visual Manipulation on the Natural Image Manifold
- IcGAN - Invertible Conditional GANs for image editing
- ID-CGAN - Image De-raining Using a Conditional Generative Adversarial Network
- Improved GAN - Improved Techniques for Training GANs
- InfoGAN - InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets
- LAGAN - Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis
- LAPGAN - Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks

Adapted from Serena Young

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

# 2017: Explosion of GANs

## Better training and generation



(a) Church outdoor.



(b) Dining room.

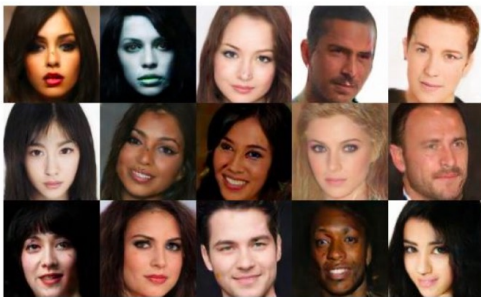


(c) Kitchen.



(d) Conference room.

LSGAN. Mao et al. 2017.



BEGAN. Bertholet et al. 2017.

Adapted from Serena Young

## Source->Target domain transfer



CycleGAN. Zhu et al. 2017.

## Text -> Image Synthesis

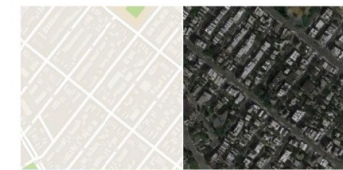
this small bird has a pink breast and crown, and black primaries and secondaries.

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



Reed et al. 2017.

## Many GAN applications



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

# Summary: GANs

Don't work with an explicit density function

Take game-theoretic approach: learn to generate from training distribution through 2-player game

## Pros:

- Beautiful samples!

## Cons:

- Trickier / more unstable to train
- Can't solve inference queries such as  $p(x)$ ,  $p(z|x)$

## Active areas of research:

- Better loss functions, more stable training (Wasserstein GAN, LSGAN, many others)
- Conditional GANs, GANs for all kinds of applications

# Lab 9b: Generative Adversarial Networks (GANs)

Duration: 10 min



To join, go to: [ahaslides.com/J3BW2](https://ahaslides.com/J3BW2) 

 AhaSlides

Please, from Lab 9b: GAN [Section 1.2], submit your interpolation plot result.

^ Get Feedback



 0  0/100 

# Summary: GANs - Scaling

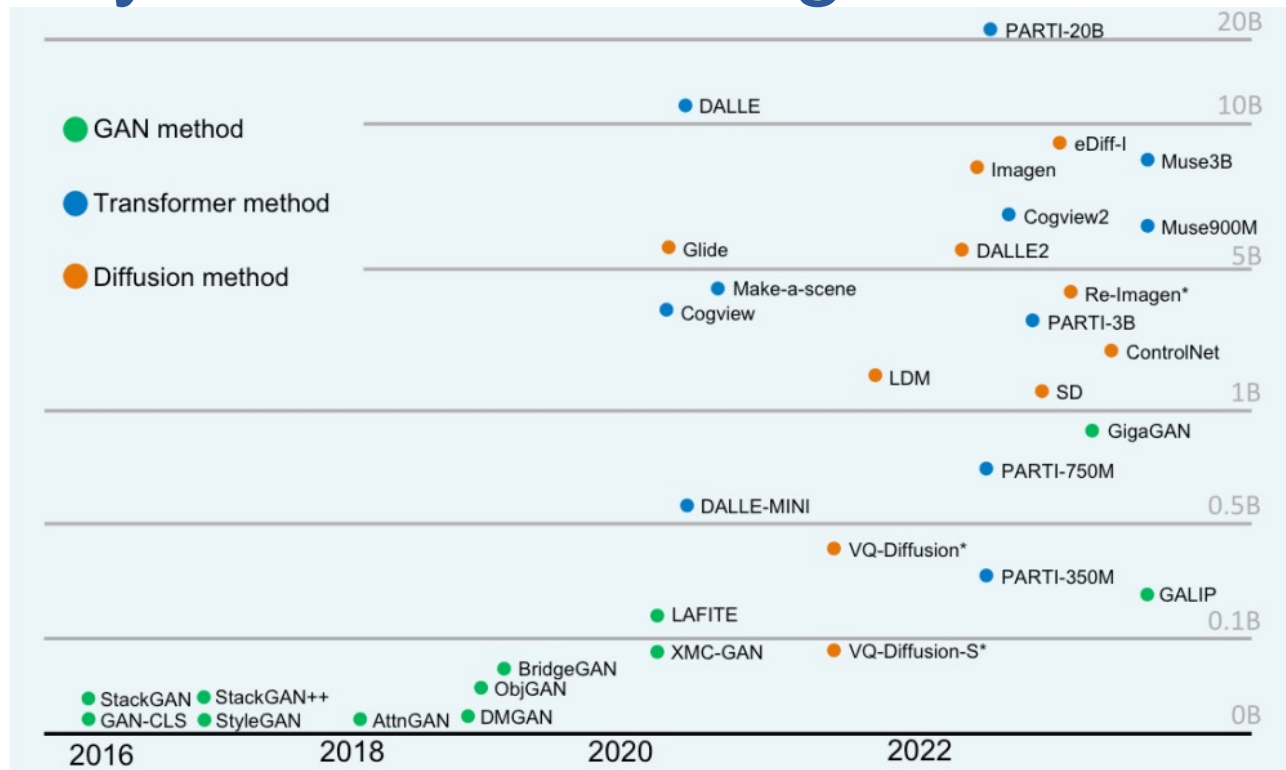


Fig. 5. Timeline of TTI model development, where green dots are GAN TTI models, blue dots are autoregressive Transformers and orange dots are Diffusion TTI models. Models are separated by their parameter, which are in general counted for all their components. Models with asterisk are calculated without the involvement of their text encoders.

# Taxonomy of Generative Models

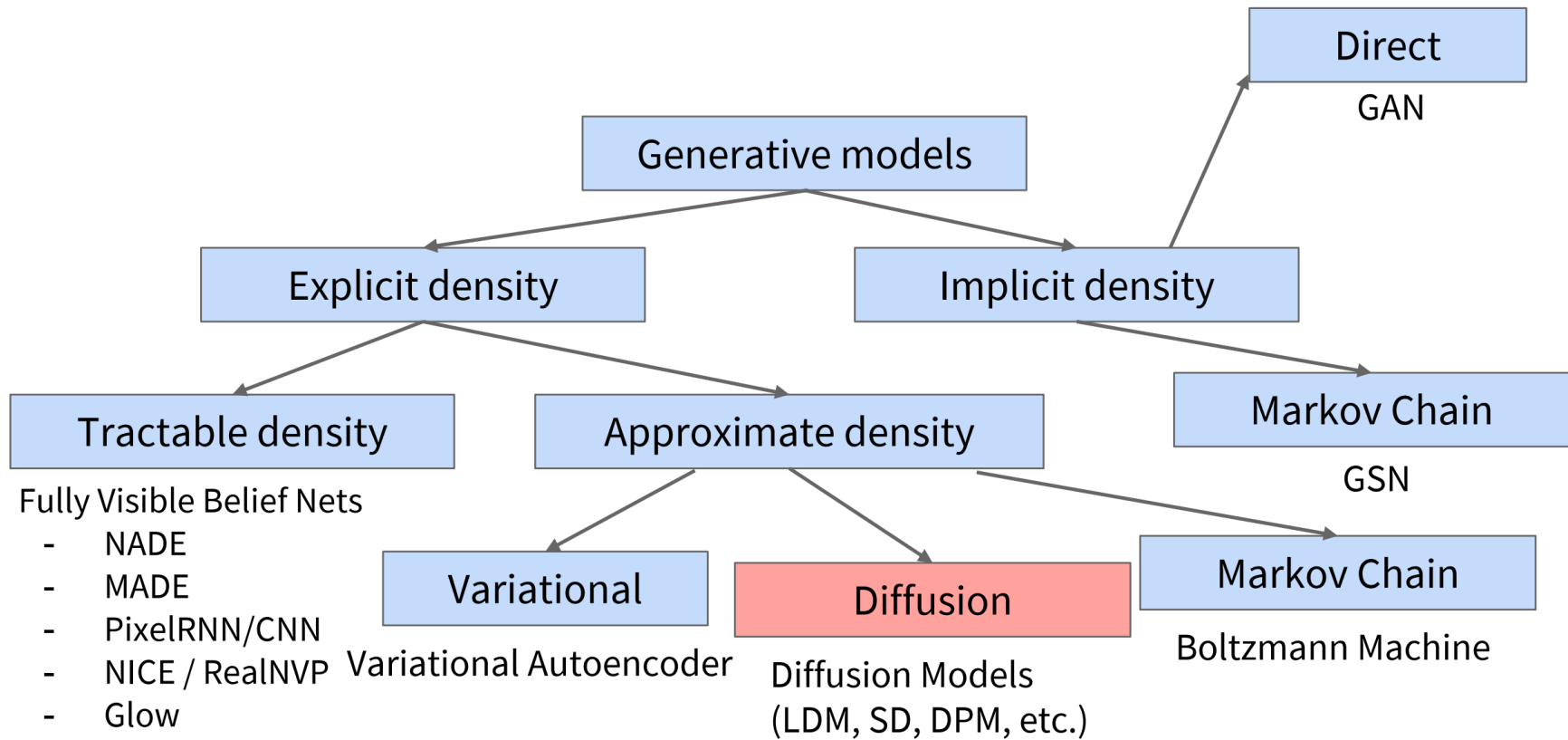


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Network  
Adapted from Fei-Fei Li

# Diffusion Models: Motivation

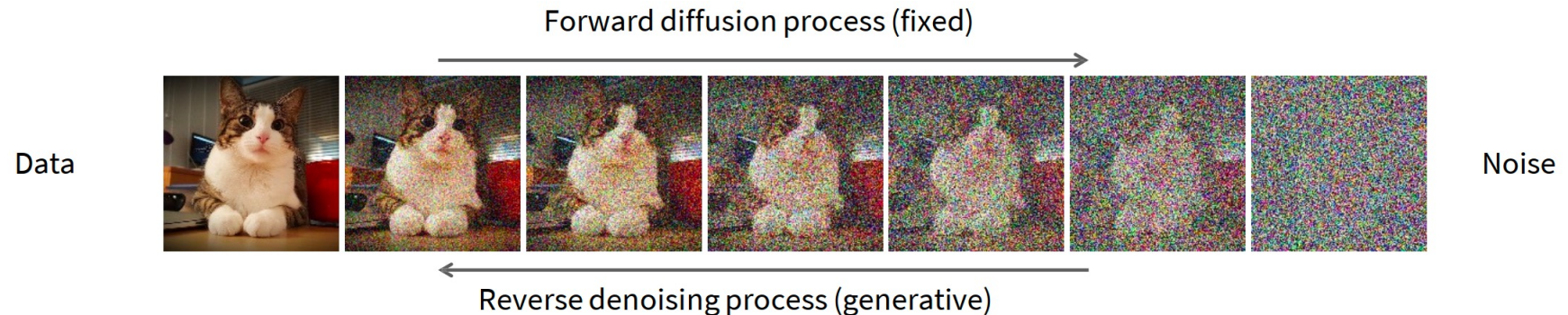
**How to create a sandcastle:**

# Denoising Diffusion Models

Learning to generate by denoising

Denoising diffusion models consist of two processes:

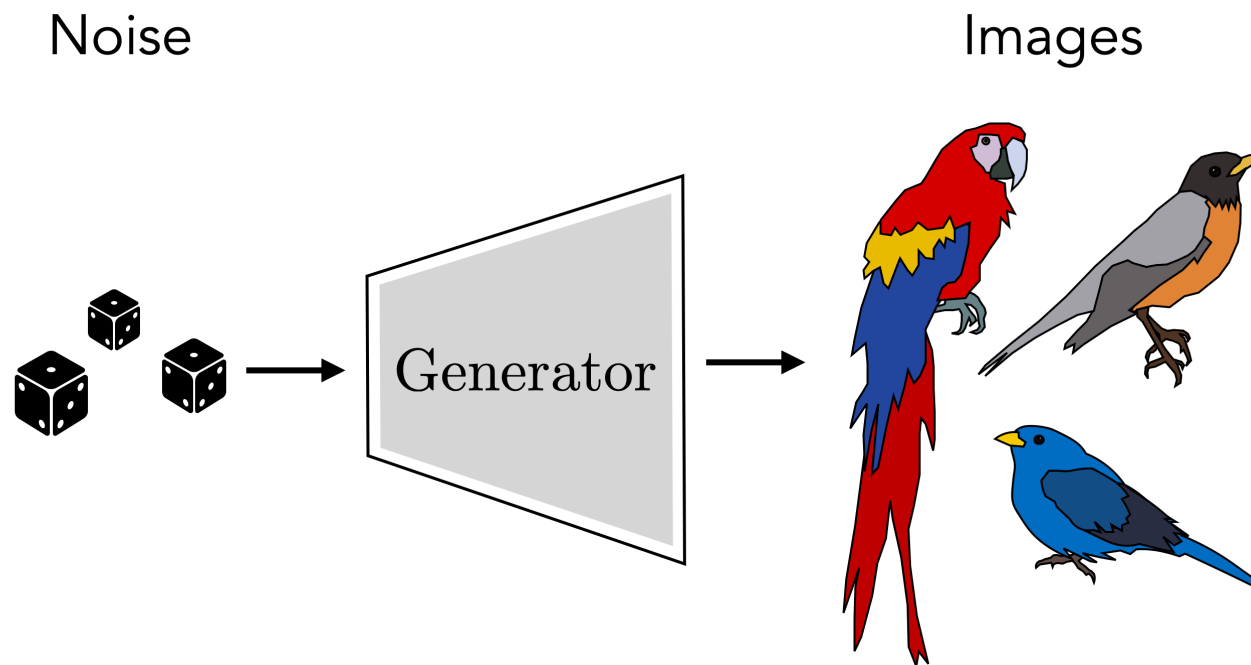
- Forward diffusion process that gradually adds noise to input
- Reverse denoising process that learns to generate data by denoising



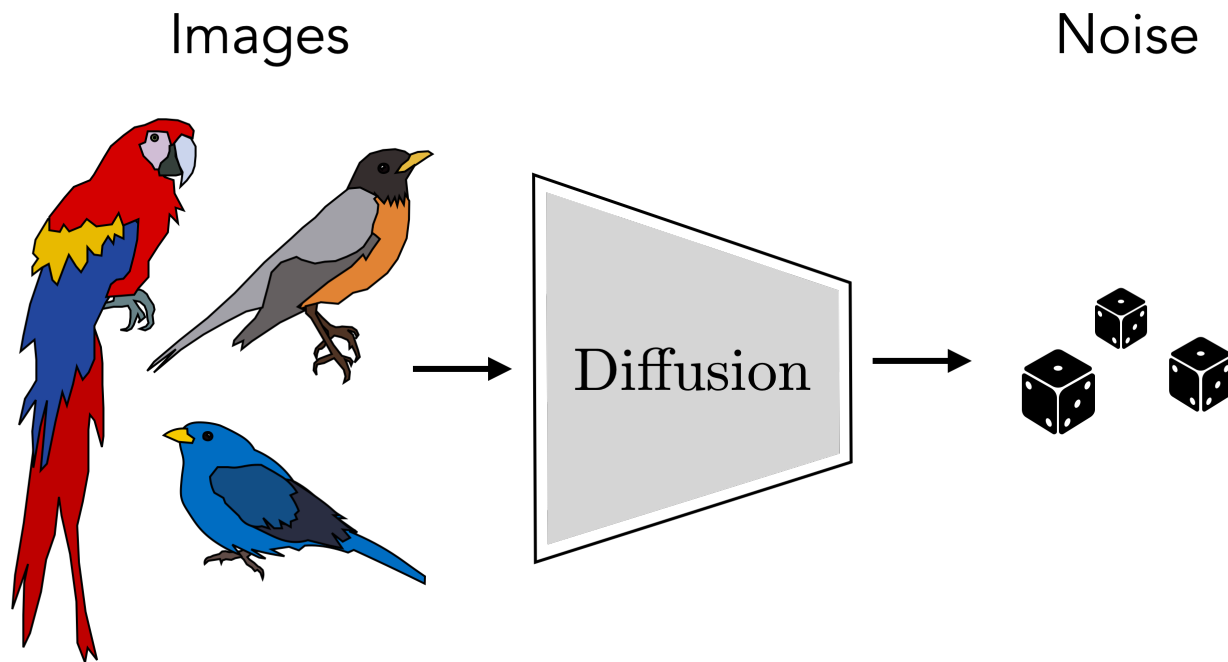
Sohl-Dickstein et al., Deep Unsupervised Learning using Nonequilibrium Thermodynamics, ICML 2015  
Ho et al., Denoising Diffusion Probabilistic Models, NeurIPS 2020  
Song et al., Score-Based Generative Modeling through Stochastic Differential Equations, ICLR 2021

slide from <https://cvpr2022-tutorial-diffusion-models.github.io/>  
Courtesy of Ruiqi Gao

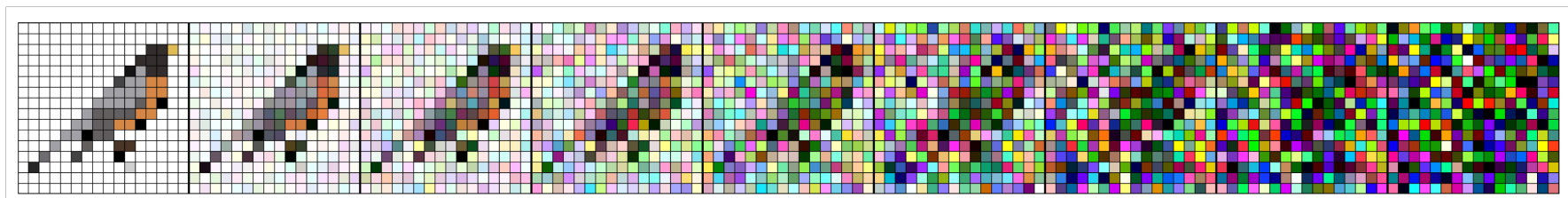
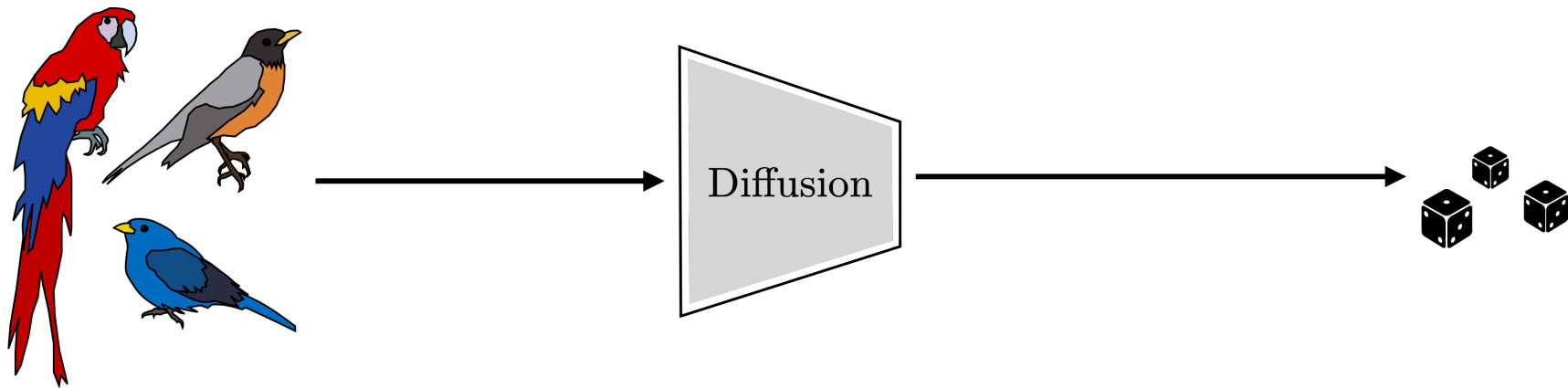
# Diffusion Models



# Diffusion Models

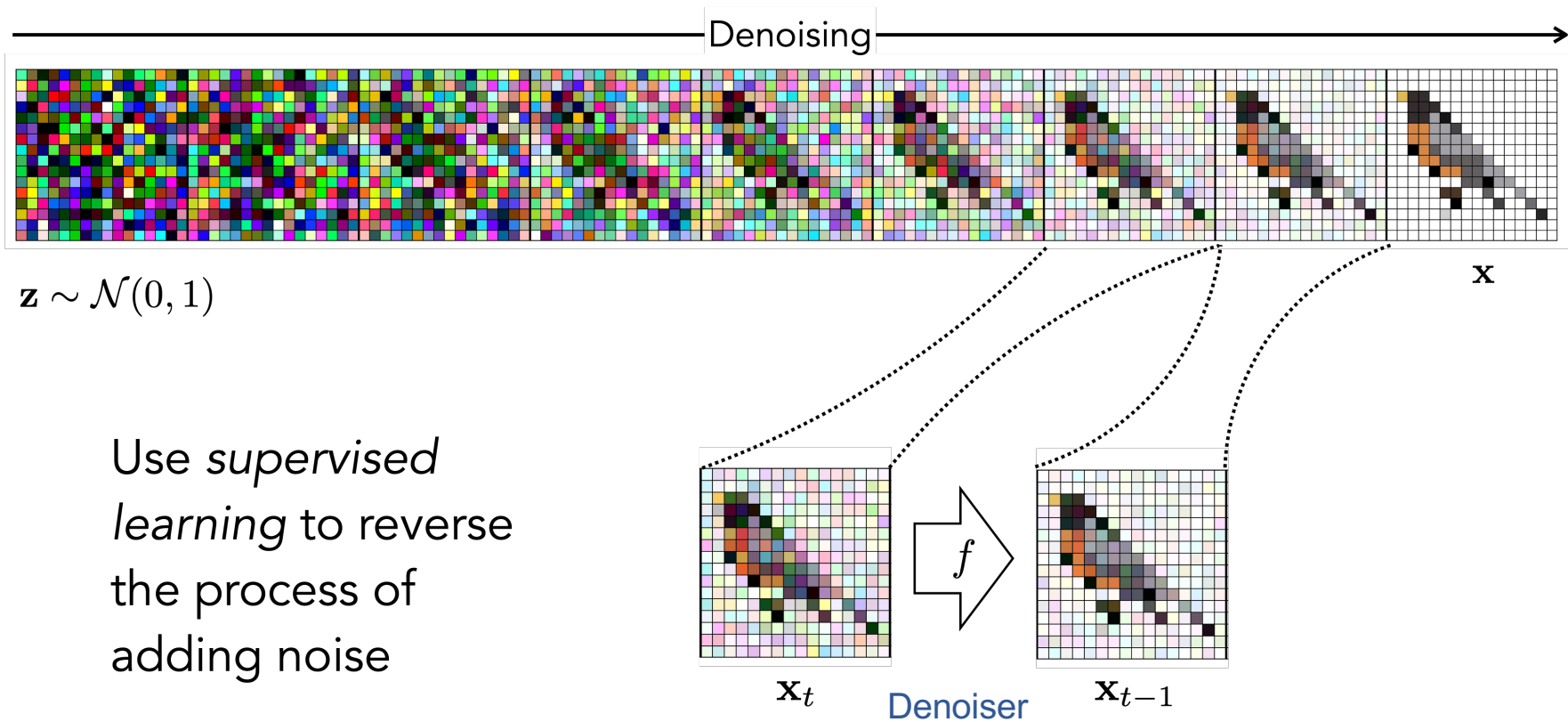


# Diffusion Models



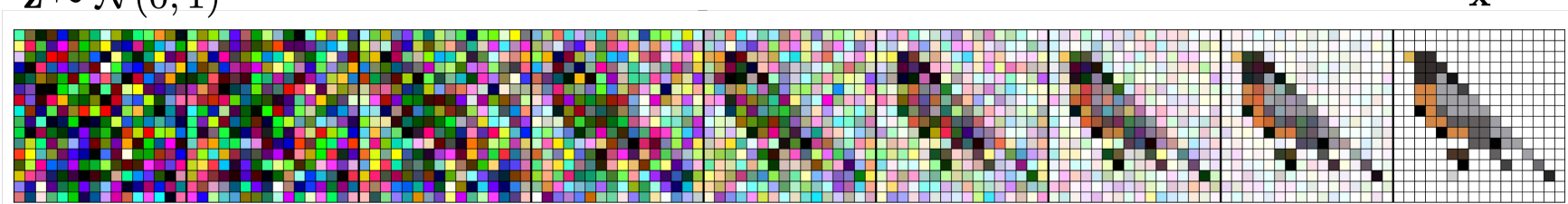
Diffusion: Just add noise

# Diffusion Models



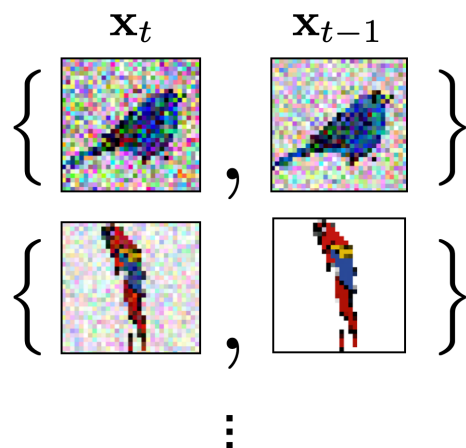
# Diffusion Models

$$\mathbf{z} \sim \mathcal{N}(0, 1)$$



Denoising

Training data

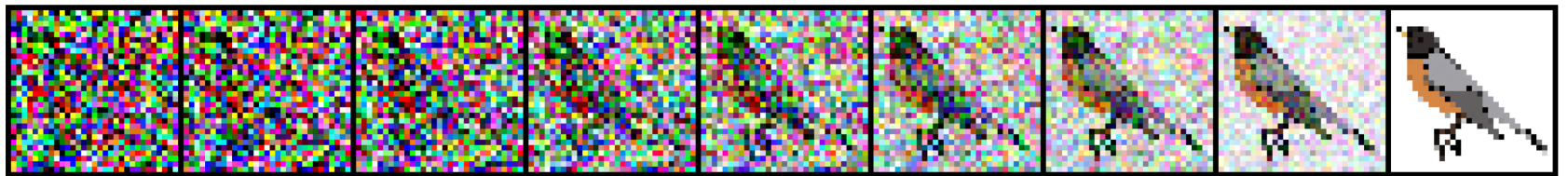


$$\arg \min_{f \in \mathcal{F}} \sum_{i=1}^N \mathcal{L}(f(\mathbf{x}_t), \mathbf{x}_{t-1})$$

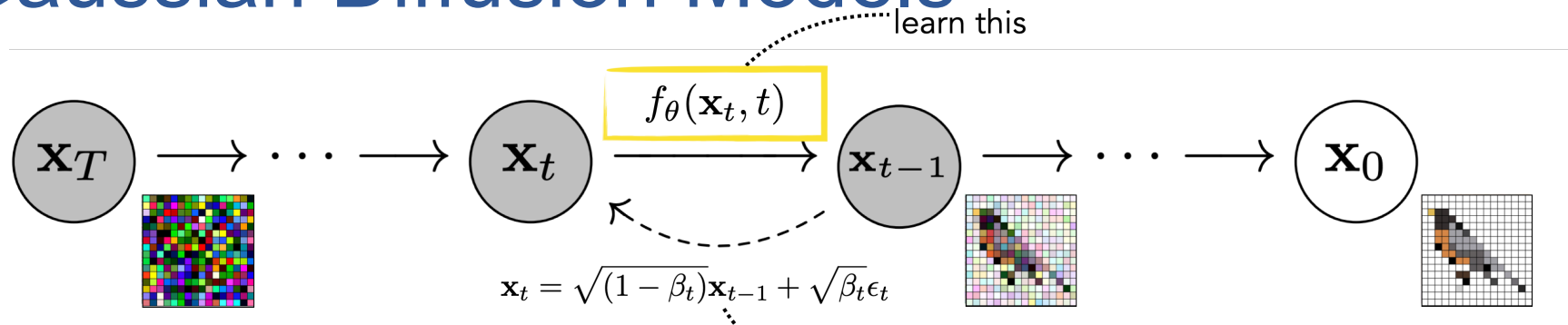
Converts generative modeling into a bunch of supervised prediction problems

# Diffusion Models

Different noise samples (dice rolls) result in different images



# Gaussian Diffusion Models



Forward process:

$$\epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \quad \mathbf{x}_t = \sqrt{(1 - \beta_t)}\mathbf{x}_{t-1} + \sqrt{\beta_t}\epsilon_t$$

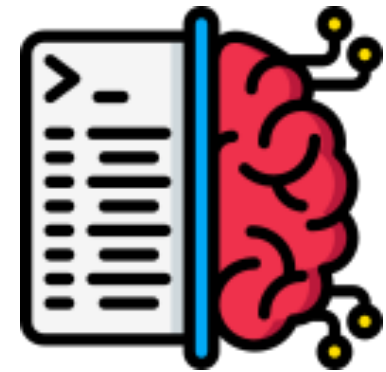
The variances, beta and sigma, are modeling choices. See Ho, Jain, and Abbeel for details.

Reverse process:

$$\mathbf{x}_{t-1} = f_\theta(\mathbf{x}_t, t)$$

# Diffusion Models: Training

1. Generate Training data by corrupting a bunch of images (**forward process**; **noising**)
2. Train a neural net to invert each step of corruption (**reverse process**; **denoising**)



# Diffusion Models: Training

**Algorithm 1.2:** Training a diffusion model.

- Forward Process. Noising
- 1 **Input:** training data  $\{\mathbf{x}^{(i)}\}_{i=1}^N$
  - 2 **Output:** trained model  $f_\theta$
  - 3 **Generate training sequences via diffusion:**
  - 4 **for**  $i = 1, \dots, N$  **do**
  - 5     **for**  $t = 1, \dots, T$  **do**
  - 6          $\epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  - 7          $\mathbf{x}_t^{(i)} \leftarrow \sqrt{(1 - \beta_t)}\mathbf{x}_{t-1}^{(i)} + \sqrt{\beta_t}\epsilon_t$
  - 8
  - 9 **Train denoiser**  $f_\theta$  **to reverse these sequences:**
  - 10  $\theta^* = \arg \min_{\theta} \sum_{i=1}^N \sum_{t=1}^T \mathcal{L}(f_\theta(\mathbf{x}_t^{(i)}, t), \mathbf{x}_{t-1}^{(i)})$
  - 11 **Return:**  $f_{\theta^*}$
- Reverse Process. Denoising



Lab 9c

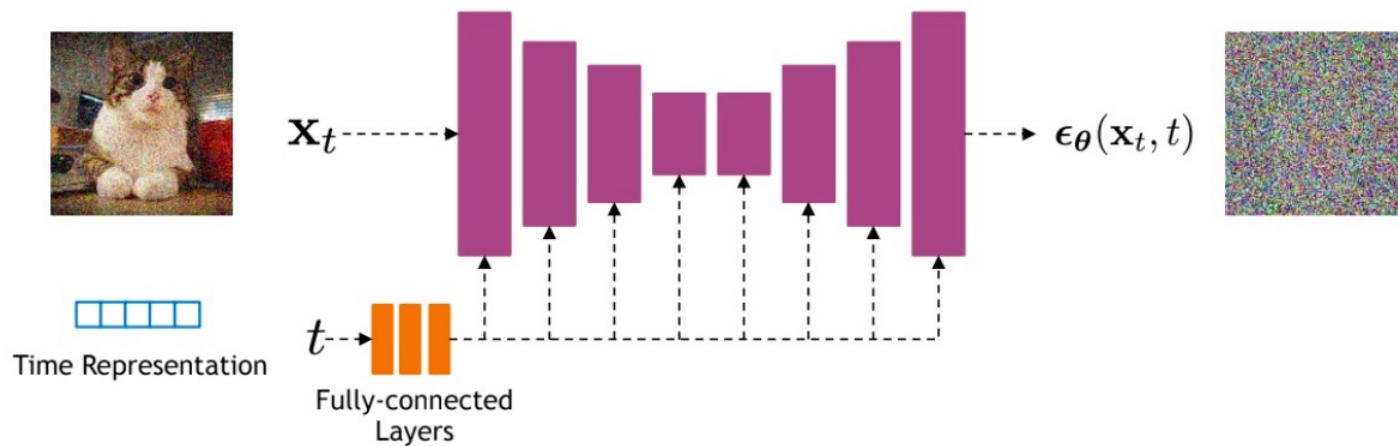
## Lab 9c: Diffusion Models

Duration: 10 min



# Diffusion Models: Training – Unet model transition

Diffusion models often use U-Net architectures with ResNet blocks and self-attention layers to represent  $\epsilon_{\theta}(\mathbf{x}_t, t)$

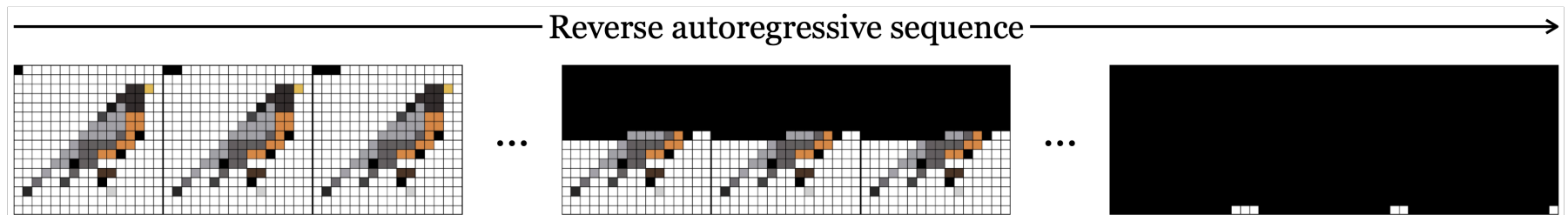
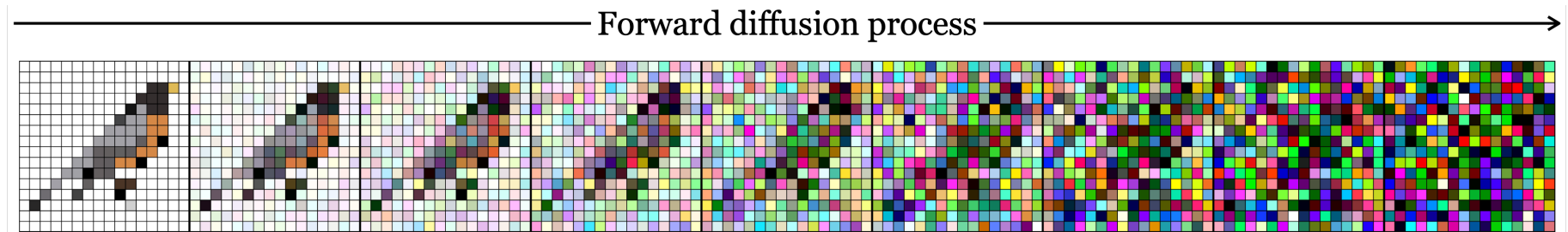


Time representation: sinusoidal positional embeddings or random Fourier features.

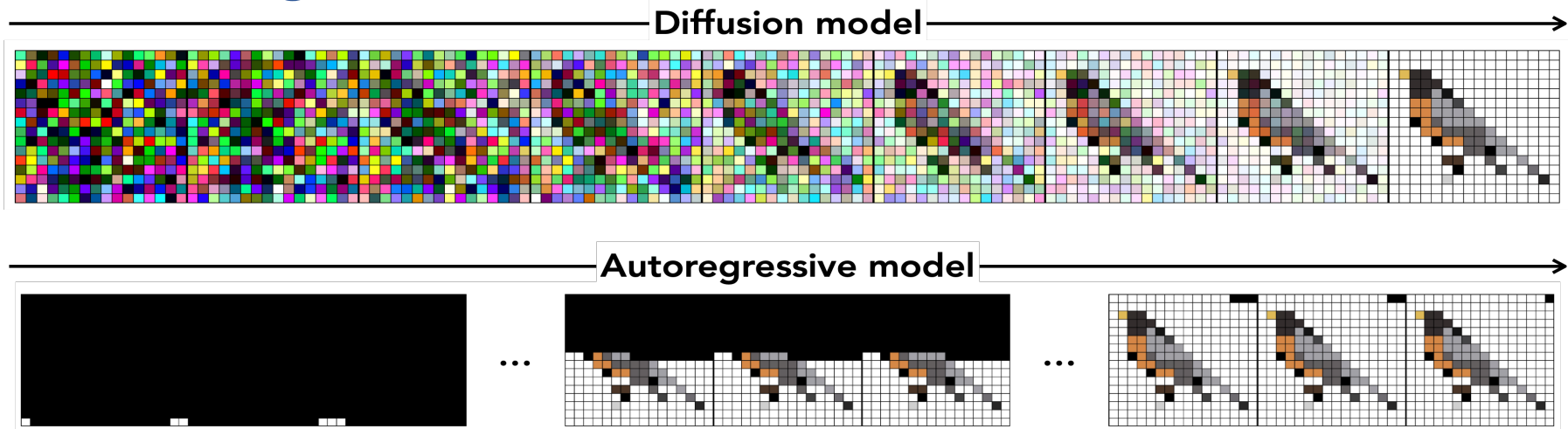
Time features are fed to the residual blocks using either simple spatial addition or using adaptive group normalization layers. (see [Dharivwal and Nichol NeurIPS 2021](#))

<https://cvpr2022-tutorial-diffusion-models.github.io/>

# Autoregressive Modes vs Diffusion Models



# Autoregressive Modes vs Diffusion Models



A common strategy is to turn generative modeling into a sequence of supervised learning problems

# Flow Matching

## Diffusion Model

Train a neural network to **predict/remove the noise** present from current noisy sample.

$$\hat{\epsilon}_t(x_t, \theta)$$

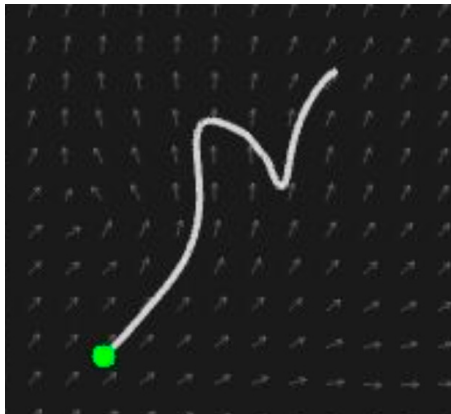
## Flow Matching

Train a neural network to predict **how noise moves**. And then use ODE solvers to find the noise.

$$\hat{v}_t(x_t, \theta)$$



# Flow Matching: ODE (Ordinary Differential Equation) Solvers?




---

## Algorithm 1 Simulating an ODE with the Euler method

---

**Require:** Vector field  $u_t$ , initial condition  $x_0$ , number of steps  $n$

- 1: Set  $t = 0$
  - 2: Set step size  $h = \frac{1}{n}$
  - 3: Set  $X_0 = x_0$
  - 4: **for**  $i = 1, \dots, n - 1$  **do**
  - 5:      $X_{t+h} = X_t + hu_t(X_t)$      *Small step into direction of vector field*
  - 6:     Update  $t \leftarrow t + h$
  - 7: **end for**
  - 8: **return**  $X_0, X_h, X_{2h}, \dots, X_1$      *Return trajectory*
-

# Flow Matching: ODE (Ordinary Differential Equation) Solvers?

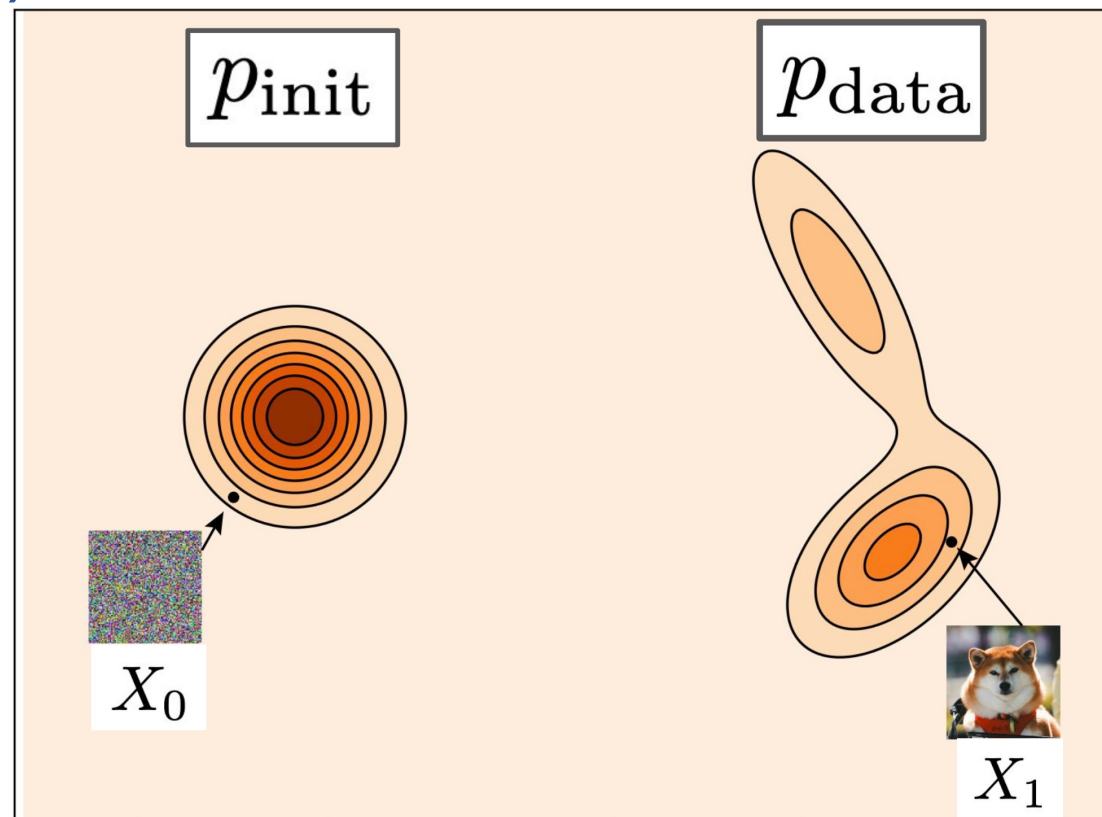
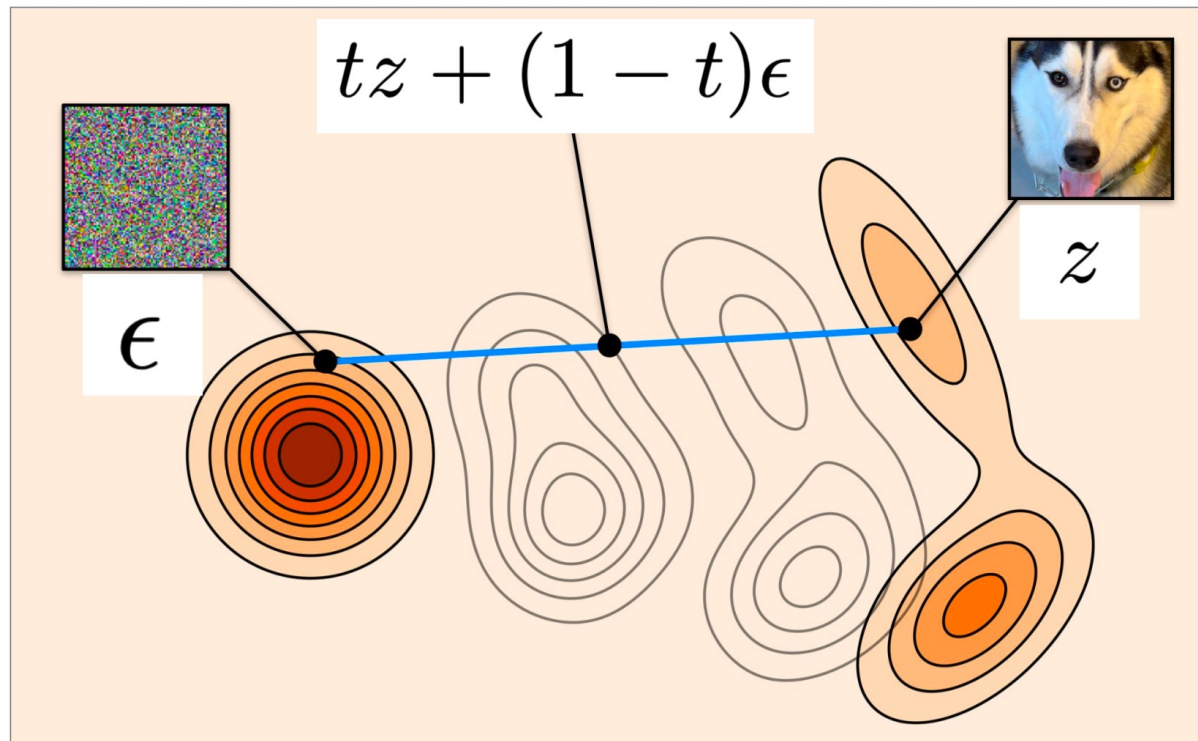


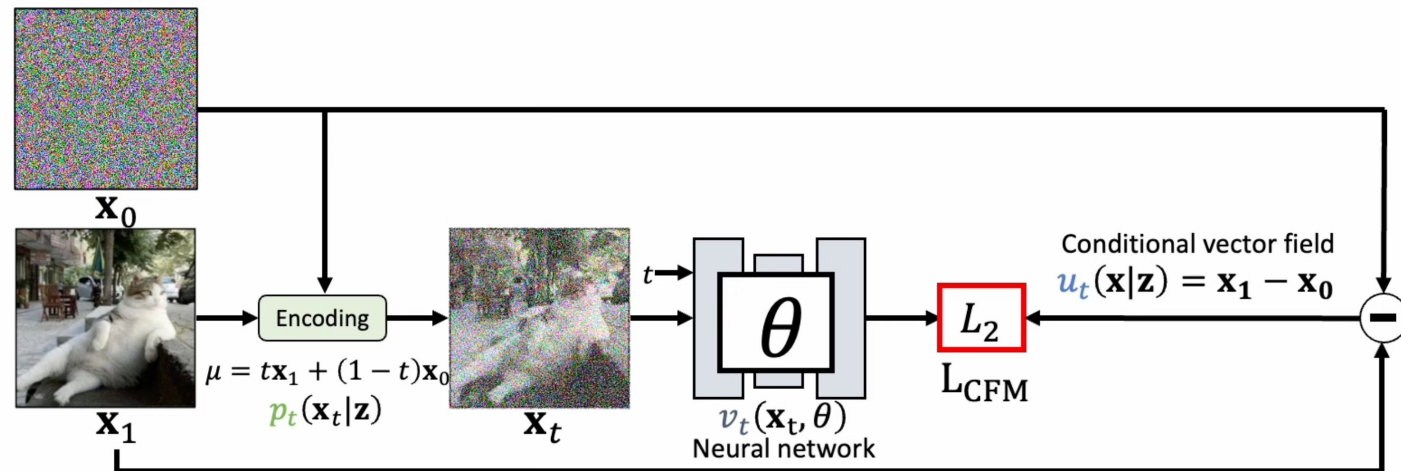
Figure credit: Yaron Lipman

# Flow Matching: ODE (Ordinary Differential Equation) Solvers?



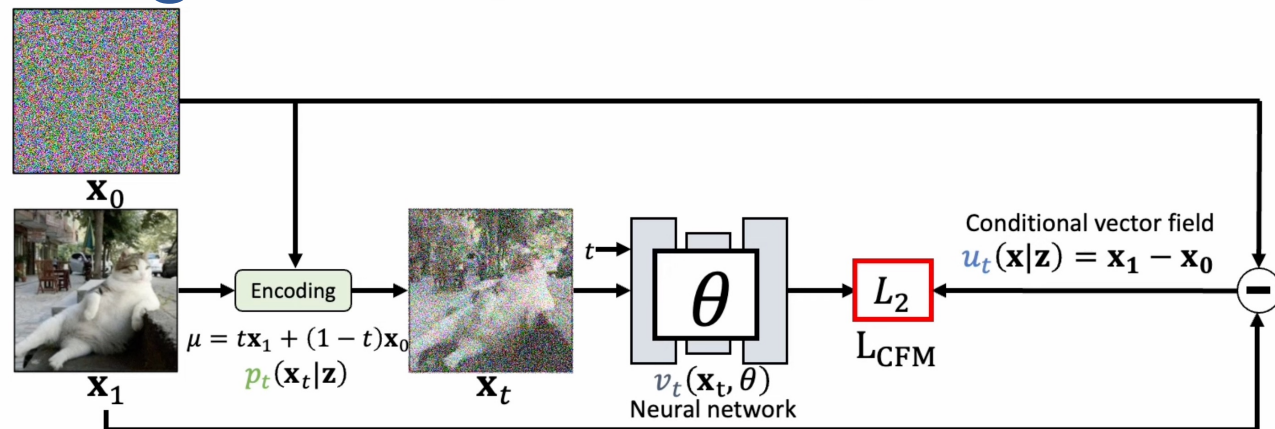
# Flow Matching

Flow matching

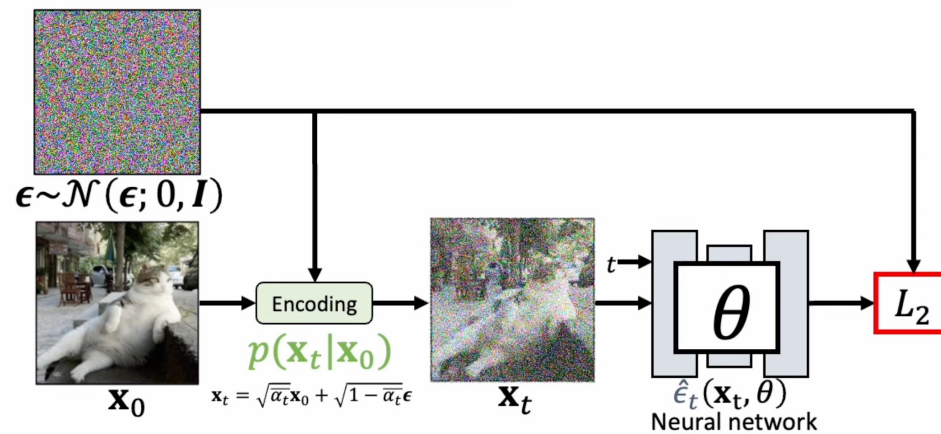


# Flow Matching

Flow matching



Diffusion



## Lab 9d: Flow Matching

Duration: 10 min



# Flow Matching: More resources

[Video](#) by Jia-Bin (Univ of Maryland):

- [Normalizing Flows](#)
- [Continuous Normalizing Flows](#)
- [Flow Matching](#)



MIT course: [Introduction to Flow Matching and Diffusion Models 2026](#)

- Jupyter Notebooks Available



<https://diffusion.csail.mit.edu/2026/>

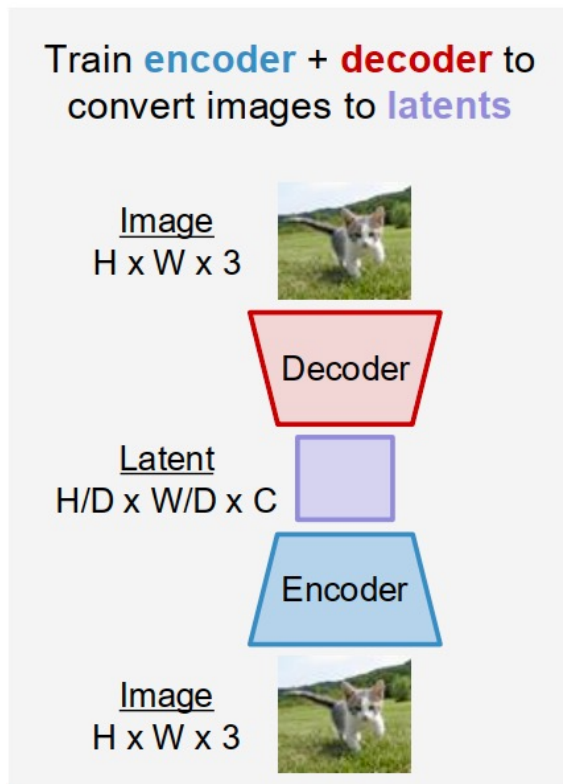
See Video by Jia-Bin (Univ of Maryland) - [https://www.youtube.com/watch?v=DDq\\_plfHqLs](https://www.youtube.com/watch?v=DDq_plfHqLs)

# Latent Diffusion Models: Motivation

- High-resolution image: 3 color channels, height 600, width 1000 → the resulting dimension is:  $3 \cdot 600 \cdot 1000 = 1.8 \text{ million!}$
- **This is a very high-dimensional space!!!!**



# Latent Diffusion Models (LDM)



Common setting:  $D=8, C=16$

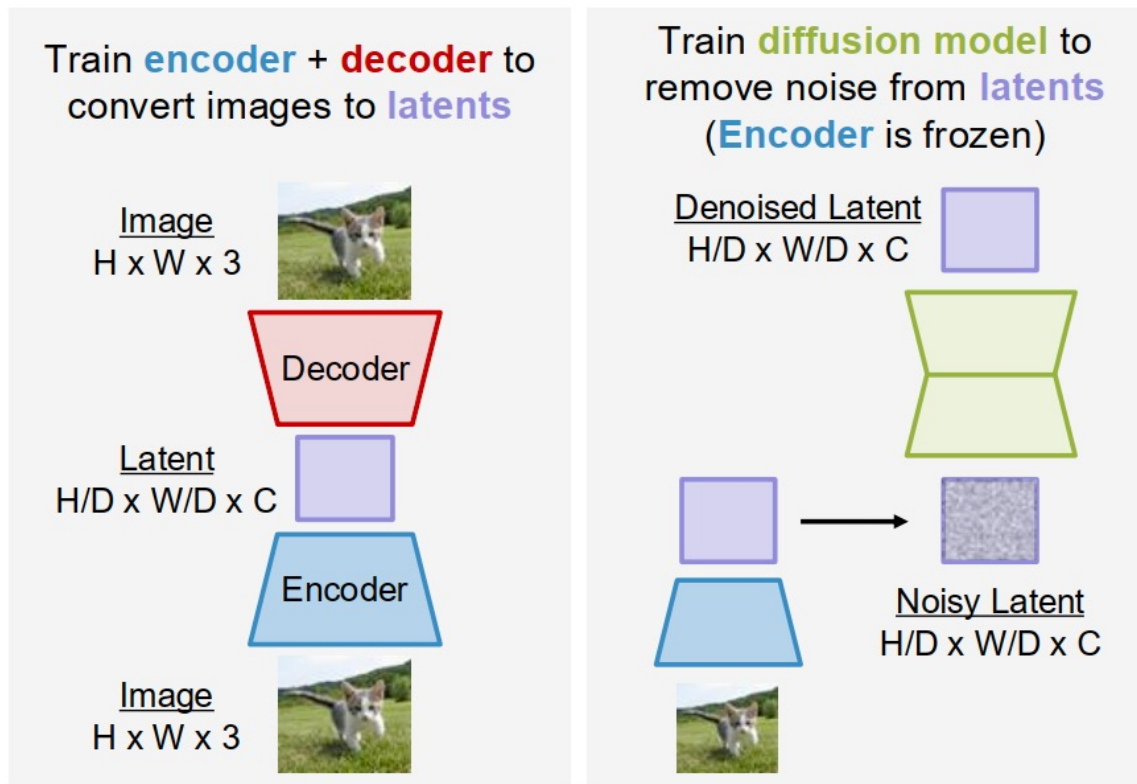
**Image:**  $256 \times 256 \times 3$

=> **Latent:**  $32 \times 32 \times 16$

Encoder / Decoder are CNNs with attention

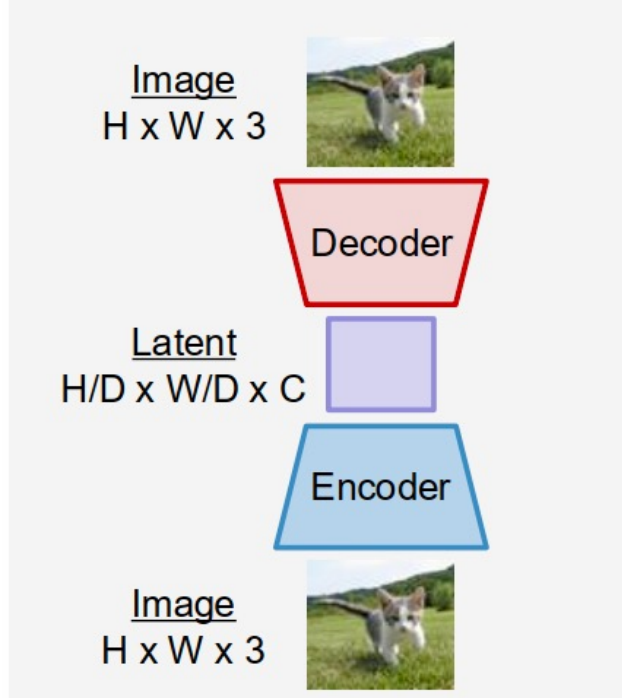
# Latent Diffusion Models (LDM)

Latent diffusion is the most common form today!



# Latent Diffusion Models (LDM)

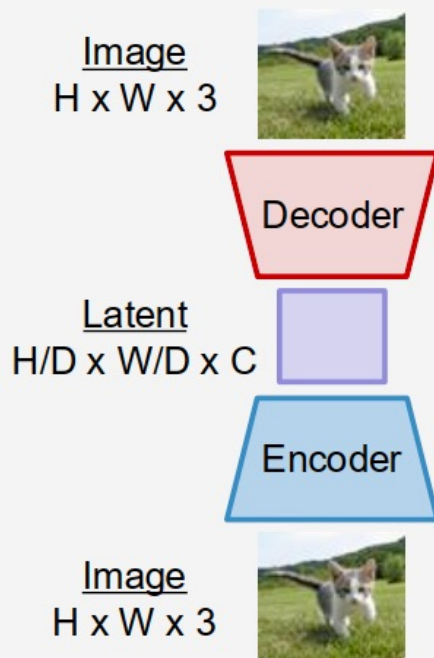
Train **encoder** + **decoder** to convert images to **latents**



How do we train the encoder+decoder?

# Latent Diffusion Models (LDM)

Train **encoder** + **decoder** to convert images to **latents**

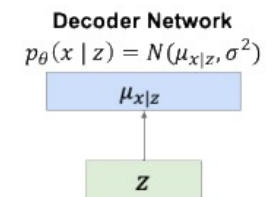
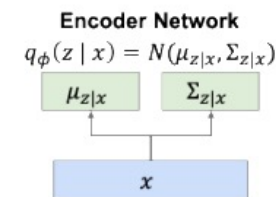


How do we train the encoder+decoder?

**Solution: It's a VAE!**  
Typically with very small KL prior weight

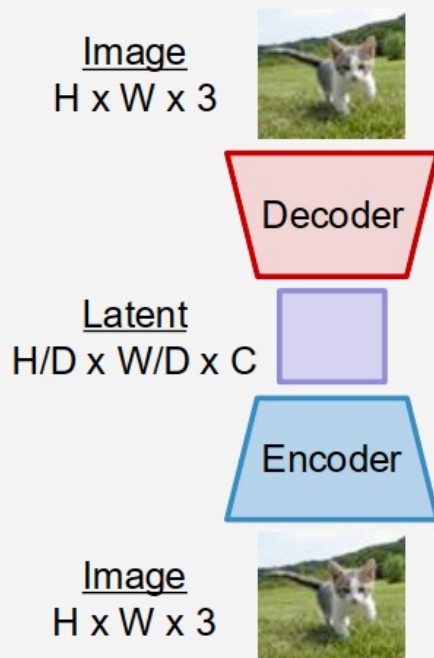
Recall: VAE

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



# Latent Diffusion Models (LDM)

Train **encoder** + **decoder** to convert images to **latents**



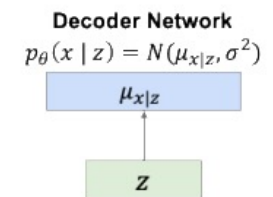
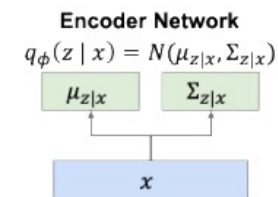
How do we train the encoder+decoder?

**Solution:** It's a VAE!  
Typically with very small KL prior weight

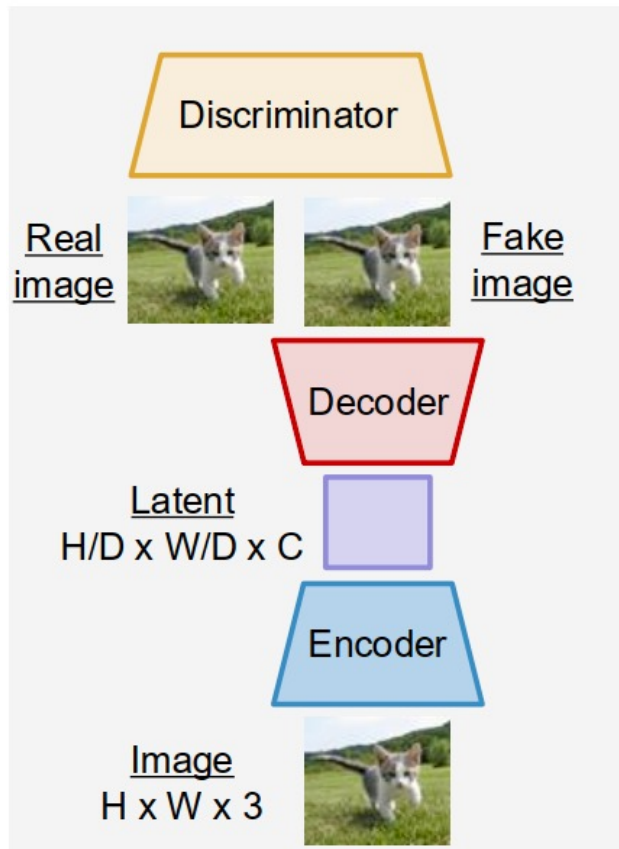
**Problem:** Decoder outputs often blurry

Recall: VAE

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



# Latent Diffusion Models (LDM)



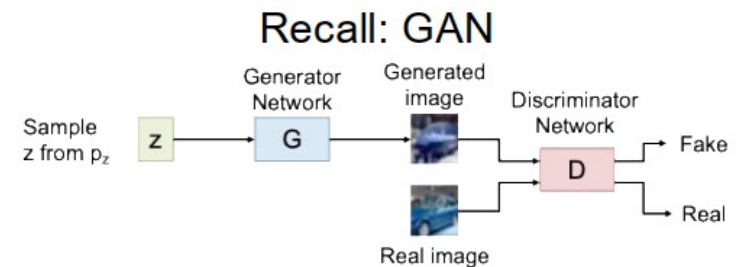
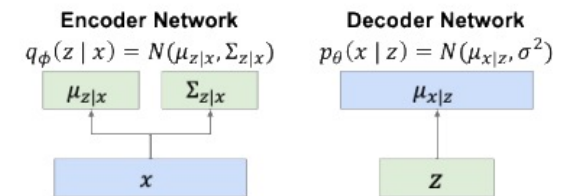
How do we train the encoder+decoder?

**Solution:** It's a VAE!  
Typically with very small KL prior weight

**Problem:** Decoder outputs often blurry

**Solution:** Add a discriminator!

Recall: VAE

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


## 2022 / 2023 : The year of diffusion and generative modeling?



**Parti**  
Pathways Autoregressive Text-to-Image Model

**DALL·E 2**

**Stable Diffusion**

slide from <https://cvpr2022-tutorial-diffusion-models.github.io/>  
Courtesy of Ruiqi Gao

# Useful Resources on Generative Models

CS 236: [Deep Generative Models](#) [[youtube](#)](Stanford)

CS 294-158 [Deep Unsupervised Learning](#) [[youtube](#)] (Berkeley)

[Introduction to Flow Matching and Diffusion Models](#) (MIT)

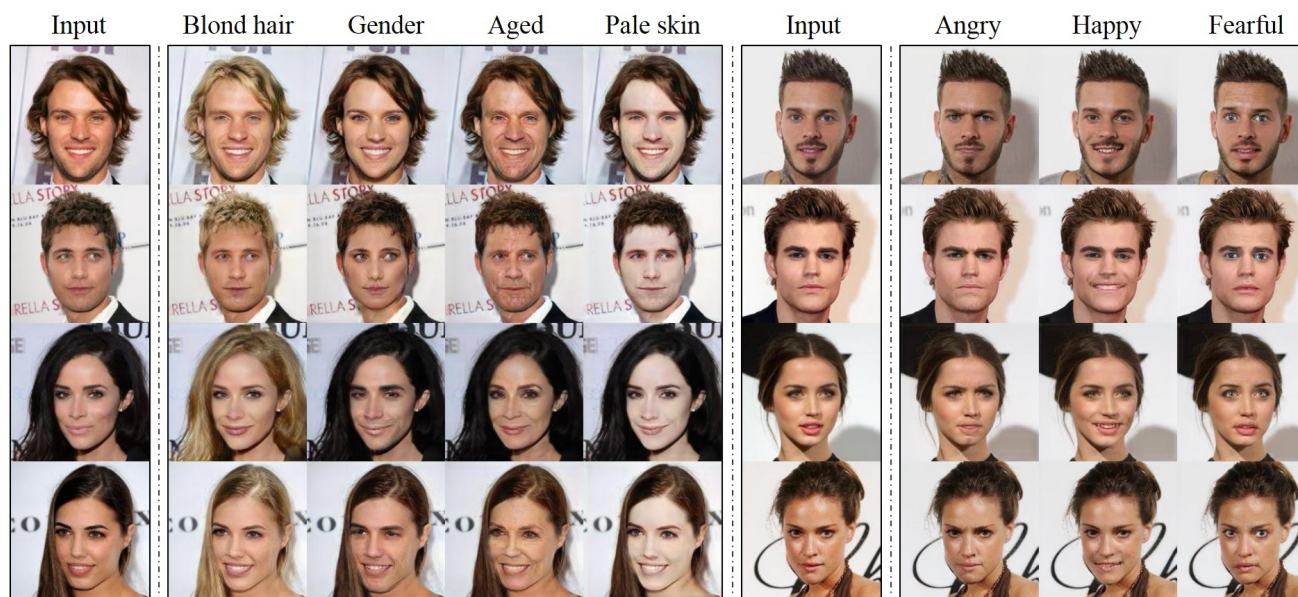


# Applications: Celebrities Who Never Existed



*Karras et al., "Progressive Growing of GANs for Improved Quality, Stability, and Variation", ICLR 2018*

# Applications: StarGAN



Choi et al., "[StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation](#)", CVPR 2018

Extra

# Training GANs: Two-player game

**Discriminator network:** try to distinguish between real and fake images

**Generator network:** try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\substack{\text{Discriminator output} \\ \text{for real data } x}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\substack{\text{Discriminator output for} \\ \text{generated fake data } G(z)}}) \right]$$

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

# Training GANs: Two-player game

**Discriminator network:** try to distinguish between real and fake images

**Generator network:** try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

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

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\substack{\text{Discriminator output} \\ \text{for real data } x}} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\substack{\text{Discriminator output for} \\ \text{generated fake data } G(z)}}) \right]$$

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

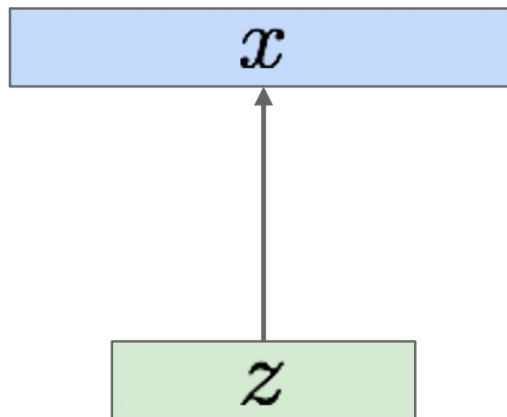
# Variational AutoEncoders

Sample from  
true conditional

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

Sample from  
true prior

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



We want to estimate the true parameters  $\theta^*$  of this generative model given training data  $x$ .

How to train the model?

Learn model parameters to maximize likelihood of training data

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

Q: What is the problem with this?



A: Intractable!

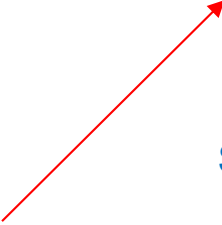

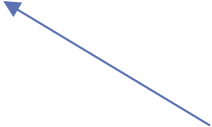
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Slide credit: Fei-Fei Li

# Variational AutoEncoders: Intractability

Data likelihood:  $p_{\theta}(x) = \int p_{\theta}(z) p_{\theta}(x|z) dz$

Simple Gaussian prior      Decoder neural network

Intractable to compute  $p(x|z)$  for every  $z$ !



$$\log p(x) \approx \log \frac{1}{k} \sum_{i=1}^k p(x|z^{(i)}), \text{ where } z^{(i)} \sim p(z)$$

Monte Carlo estimation is too high variance

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

Slide credit: Serena Young

# Variational AutoEncoders: Intractability

Data likelihood:  $p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$

Posterior density:  $p_{\theta}(z|x) = p_{\theta}(x|z)p_{\theta}(z)/p_{\theta}(x)$

Intractable data likelihood



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

Slide credit: Fei-Fei Li

# Variational AutoEncoders: Intractability

Data likelihood:  $p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$

Posterior density also intractable:  $p_{\theta}(z|x) = p_{\theta}(x|z)p_{\theta}(z)/p_{\theta}(x)$

**Solution:** In addition to modeling  $p_{\theta}(x|z)$ , learn  $q_{\phi}(z|x)$  that approximates the true posterior  $p_{\theta}(z|x)$ .

Will see that the approximate posterior allows us to derive a lower bound on the data likelihood that is tractable, which we can optimize.

Variational inference is to approximate the unknown posterior distribution from only the observed data  $x$

# Variational AutoEncoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

$$\begin{aligned}
 \log p_{\theta}(x^{(i)}) &= \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} \left[ \log p_{\theta}(x^{(i)}) \right] \quad (p_{\theta}(x^{(i)}) \text{ Does not depend on } z) \\
 &= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Bayes' Rule}) \\
 &= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z) q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)}) q_{\phi}(z | x^{(i)})} \right] \quad (\text{Multiply by constant}) \\
 &= \mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z)} \right] + \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Logarithms}) \\
 &= \mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z)) + D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z | x^{(i)}))
 \end{aligned}$$

Taking expectation wrt.  $z$   
(using encoder network) will  
come in handy later

The expectation wrt.  $z$  (using  
encoder network) let us write  
nice KL terms

Slide credit: Fei-Fei Li

# Variational AutoEncoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

$$\log p_{\theta}(x^{(i)}) = \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} \left[ \log p_{\theta}(x^{(i)}) \right] \quad (p_{\theta}(x^{(i)}) \text{ Does not depend on } z)$$

We want to maximize the Data likelihood

$$= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Bayes' Rule})$$

$$= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z) q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)}) q_{\phi}(z | x^{(i)})} \right] \quad (\text{Multiply by constant})$$

$$= \mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z)} \right] + \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Logarithms})$$

$$= \mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z)) + D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z | x^{(i)}))$$

Decoder network gives  $p_{\theta}(x|z)$ , can compute estimate of this term through sampling (need some trick to differentiate through sampling).

This KL term (between Gaussians for encoder and  $z$  prior) has nice closed-form solution!

$p_{\theta}(z|x)$  intractable (saw earlier), can't compute this KL term :( But we know KL divergence always  $\geq 0$ <sup>1</sup>

<sup>1</sup> <https://statproofbook.github.io/P/kl-nonneg.html>

# Variational AutoEncoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

$$\log p_{\theta}(x^{(i)}) = \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} \left[ \log p_{\theta}(x^{(i)}) \right] \quad (p_{\theta}(x^{(i)}) \text{ Does not depend on } z)$$

We want to maximize the Data likelihood

$$= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Bayes' Rule})$$

$$= \mathbf{E}_z \left[ \log \frac{p_{\theta}(x^{(i)} | z) p_{\theta}(z) q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)}) q_{\phi}(z | x^{(i)})} \right] \quad (\text{Multiply by constant})$$

Encoder:  
make approximate posterior distribution close to prior

Decoder:  
reconstruct the input data

$$= \mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right] - \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z)} \right] + \mathbf{E}_z \left[ \log \frac{q_{\phi}(z | x^{(i)})}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Logarithms})$$

$$= \underbrace{\mathbf{E}_z \left[ \log p_{\theta}(x^{(i)} | z) \right]}_{L(x^{(i)}, \theta, \phi)} - \underbrace{D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z))}_{\geq 0} + \underbrace{D_{KL}(q_{\phi}(z | x^{(i)}) || p_{\theta}(z | x^{(i)}))}_{\geq 0}$$

Tractable lower bound which we can take gradient of and optimize! ( $p_{\theta}(x|z)$  differentiable, KL term differentiable)