Lecture 19: Generative Models, Part 1

Reminder: Assignment 5

A5 released; due **Monday November 16, 11:59pm EST**

A5 covers object detection:
- Single-stage detectors
- Two-stage detectors

Midterm Grades Released

- Midterm grades released **on Gradescope**
- Mean score: 77.5 (std 12.3)

- If you think there was an error in grading your exam, submit a regrade request via Gradescope by **Tuesday, November 17**
- After all regrades are finalized, we'll copy the final exam grades over to **Canvas**

Last Time: Videos

Many video models:
Single-frame CNN (Try this first!)
Late fusion
Early fusion
3D CNN / C3D
Two-stream networks
CNN + RNN
Convolutional RNN
Spatio-temporal self-attention
SlowFast networks (current SoTA)

Today:
Generative Models, Part 1

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

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

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

Classification

Cat

Supervised vs Unsupervised Learning

Supervised Learning

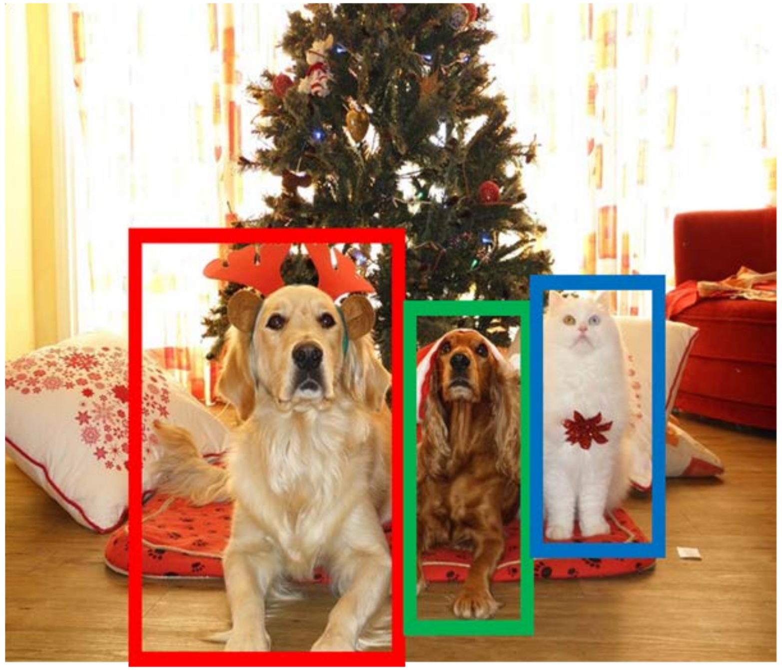
Data: (x, y)

x is data, y is label

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

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

Object Detection



Supervised vs Unsupervised Learning

Supervised Learning

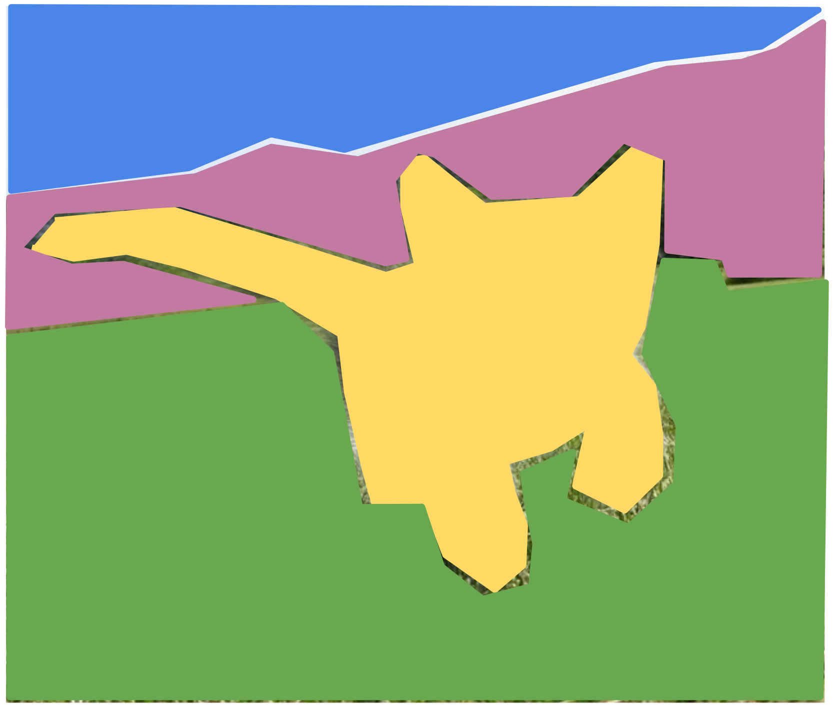
Data: (x, y)

x is data, y is label

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

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

Semantic Segmentation



GRASS, CAT, TREE, SKY

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

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

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

Image captioning

A cat sitting on a suitcase on the floor

Caption generated using neuraltalk2

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

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

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

Unsupervised Learning

Data: x

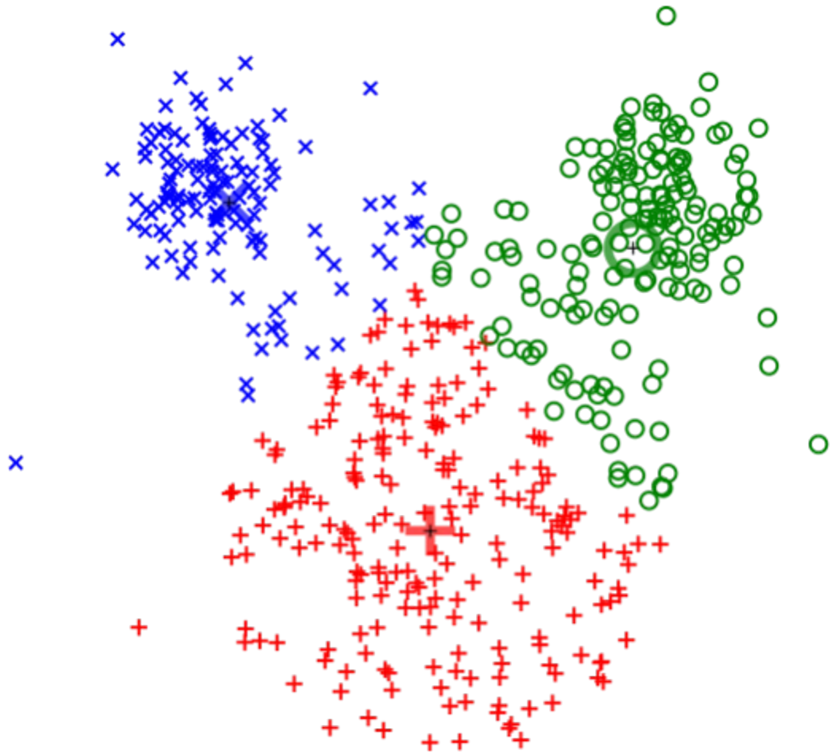
Just data, no labels!

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

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

Supervised vs Unsupervised Learning

Clustering
(e.g. K-Means)



Unsupervised Learning

Data: x

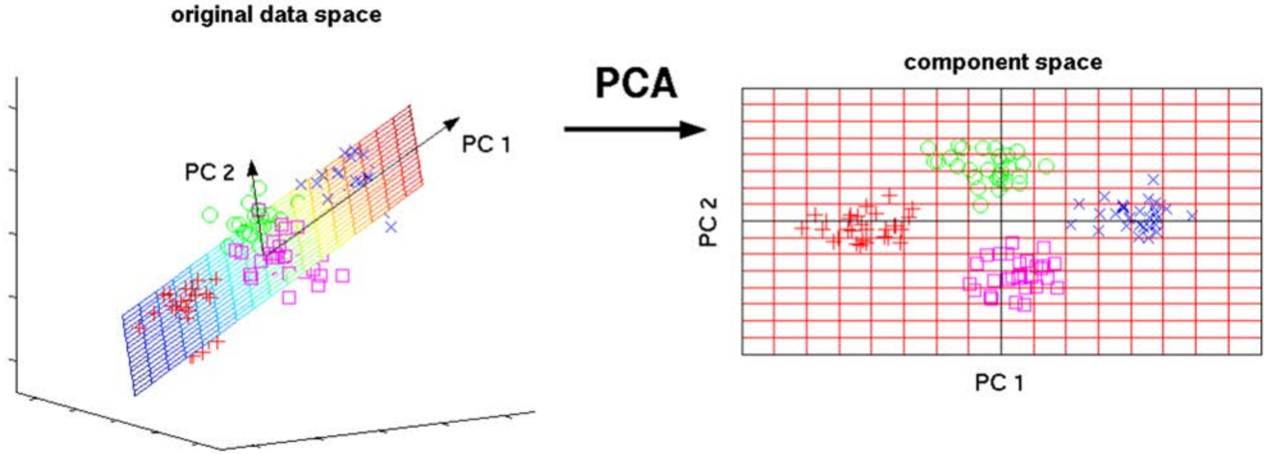
Just data, no labels!

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

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

Supervised vs Unsupervised Learning

Dimensionality Reduction
(e.g. Principal Components Analysis)



3D → 2D

Unsupervised Learning

Data: x

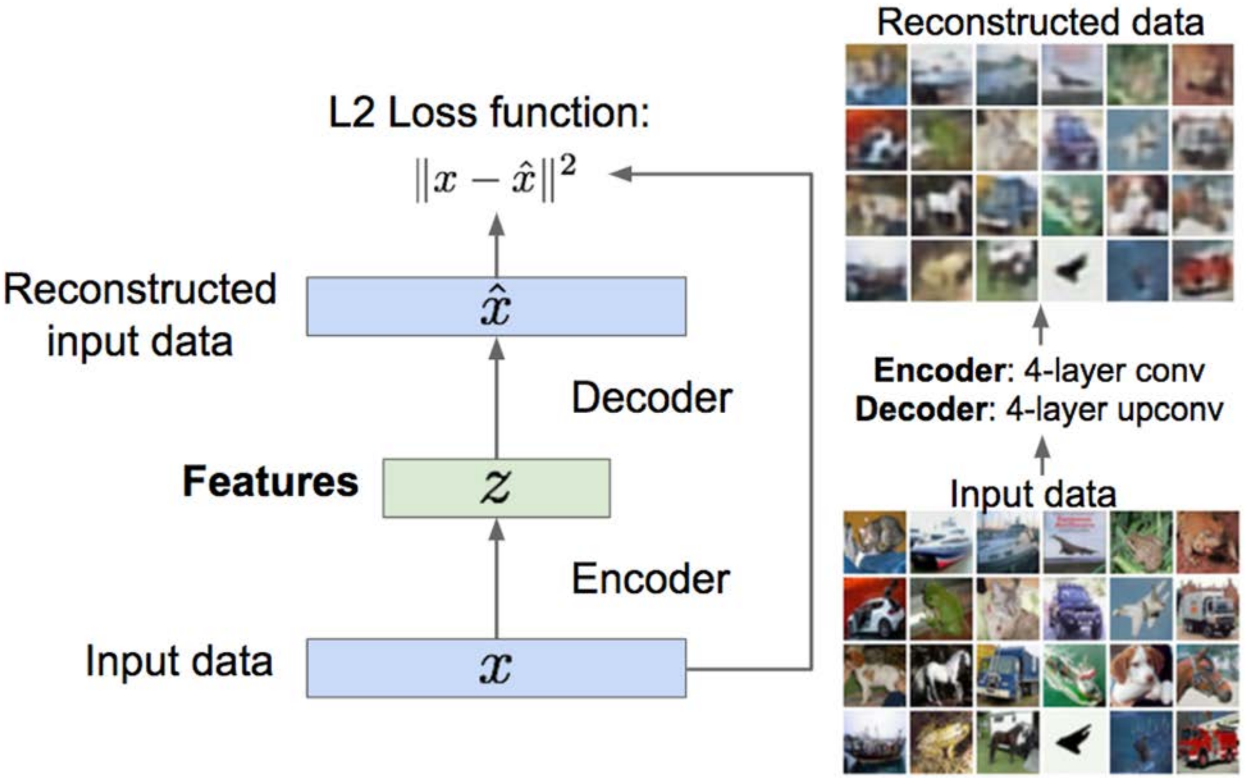
Just data, no labels!

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

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

Supervised vs Unsupervised Learning

Feature Learning
(e.g. autoencoders)



Unsupervised Learning

Data: x

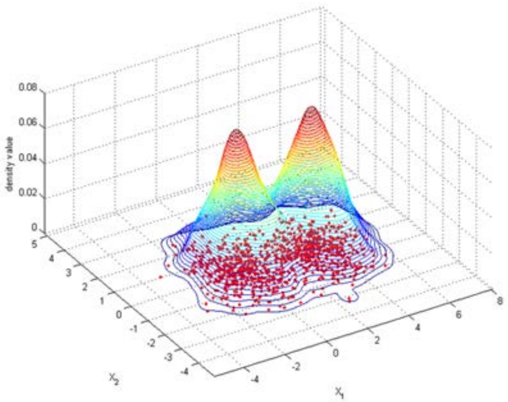
Just data, no labels!

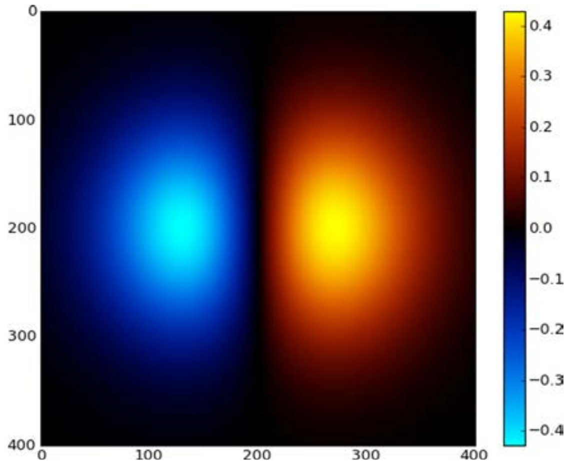
Goal: Learn some underlying hidden *structure* of the data

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

Supervised vs Unsupervised Learning

Density Estimation





Images left and right are CC0 public domain

Unsupervised Learning

Data: x

Just data, no labels!

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

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

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

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

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

Unsupervised Learning

Data: x

Just data, no labels!

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

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

Discriminative vs Generative Models

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

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

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

Data: x

Label: y

Cat

Discriminative vs Generative Models

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

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)

Data: x

Label: y

Cat

Probability Recap:

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

Density functions are **normalized**:

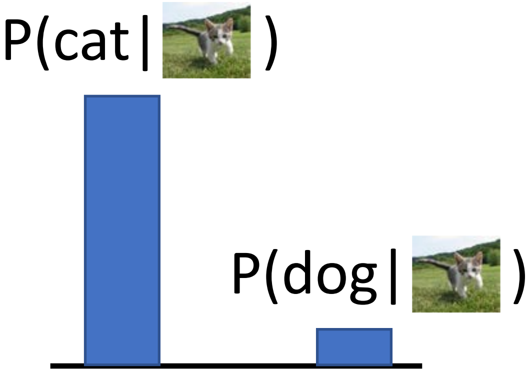
$$\int_X p(x)dx = 1$$

Different values of x **compete** for density

Discriminative vs Generative Models

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

Data: x



Generative Model: Learn a probability distribution p(x)

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

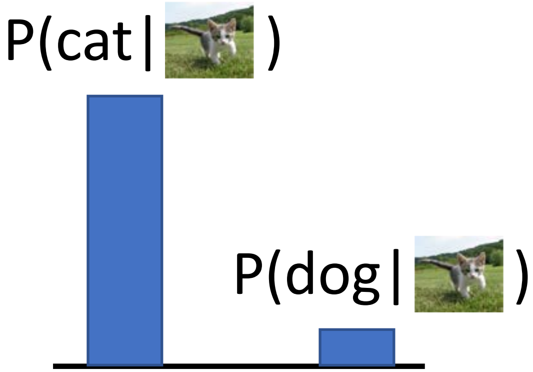
Conditional Generative Model: Learn p(x|y)

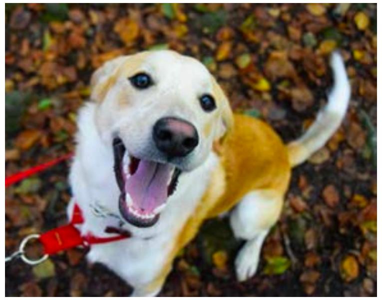
Discriminative vs Generative Models

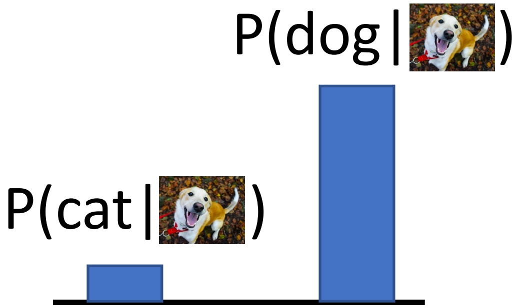
Discriminative Model: Learn a probability distribution p(y|x)

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)



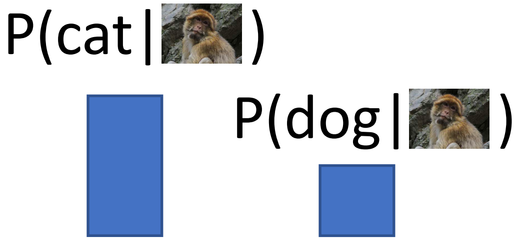




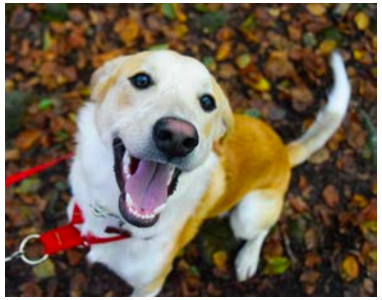
Discriminative model: the possible labels for each input "compete" for probability mass. But no competition between **images**

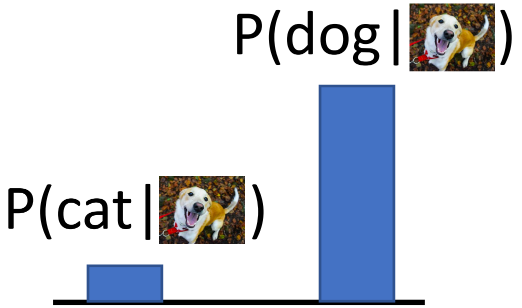
Discriminative vs Generative Models

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



Generative Model: Learn a probability distribution p(x)



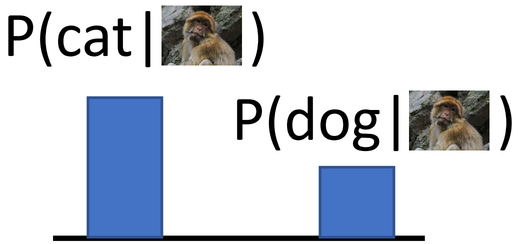


Conditional Generative Model: Learn p(x|y)

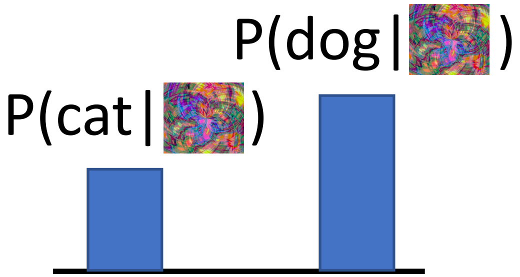
Discriminative model: No way for the model to handle unreasonable inputs; it must give label distributions for all images

Discriminative vs Generative Models

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



Generative Model: Learn a probability distribution p(x)



Conditional Generative Model: Learn p(x|y)

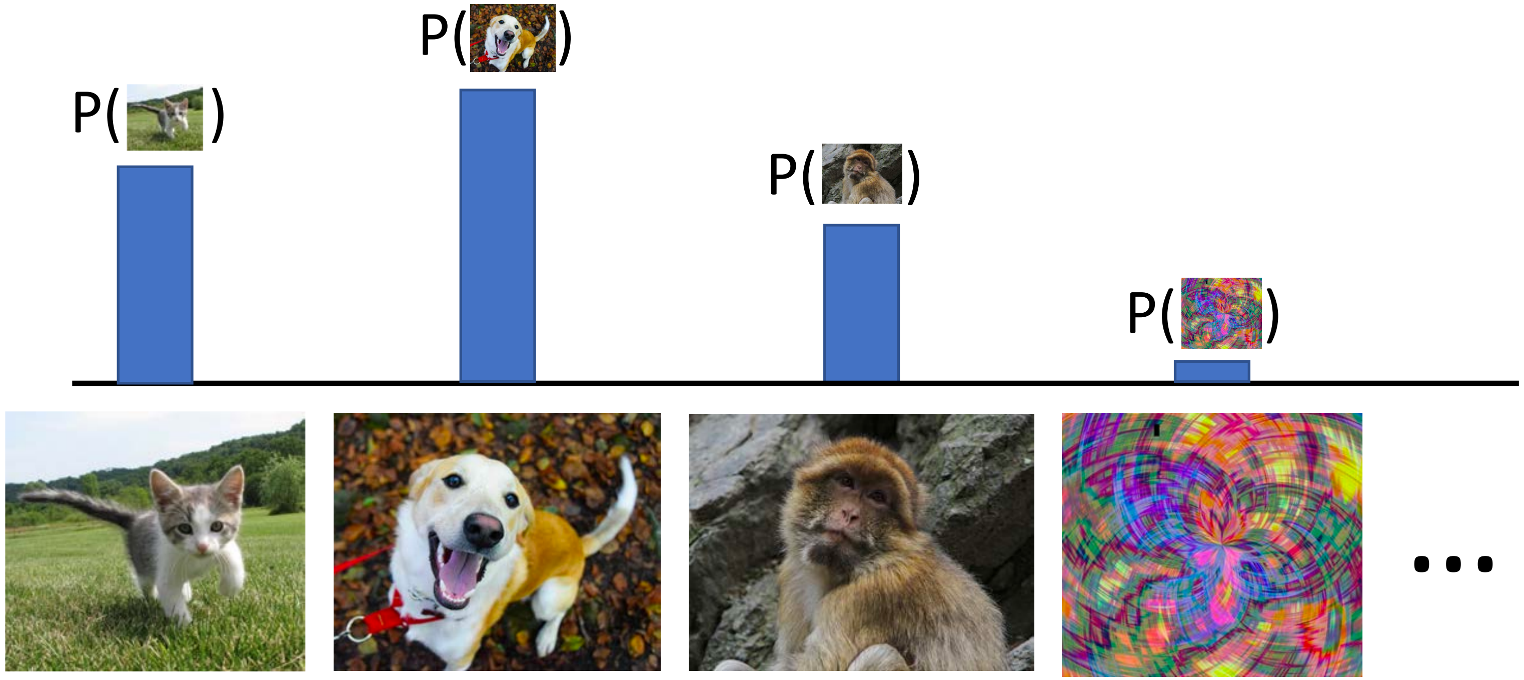
Discriminative model: No way for the model to handle unreasonable inputs; it must give label distributions for all images

Discriminative vs Generative Models

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

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)



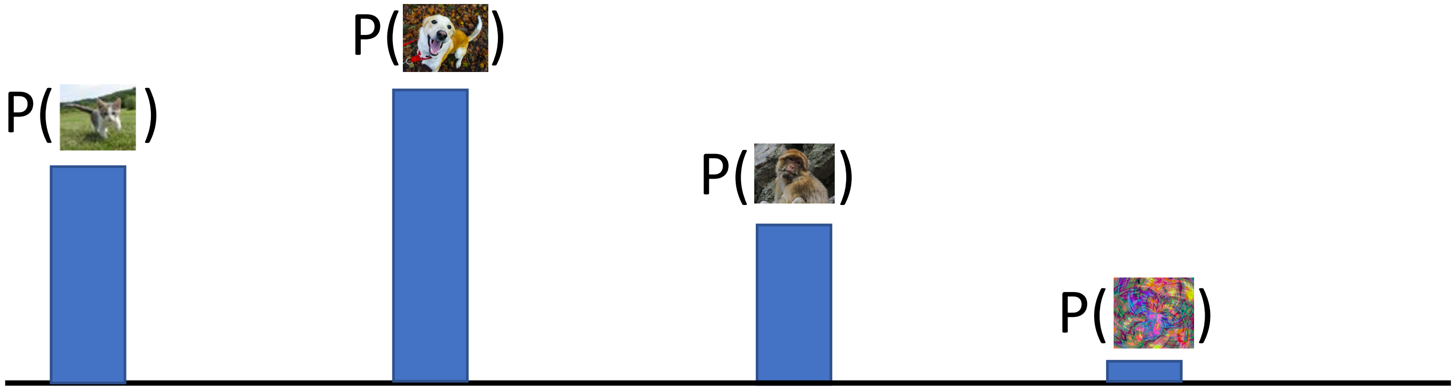
Generative model: All possible images compete with each other for probability mass

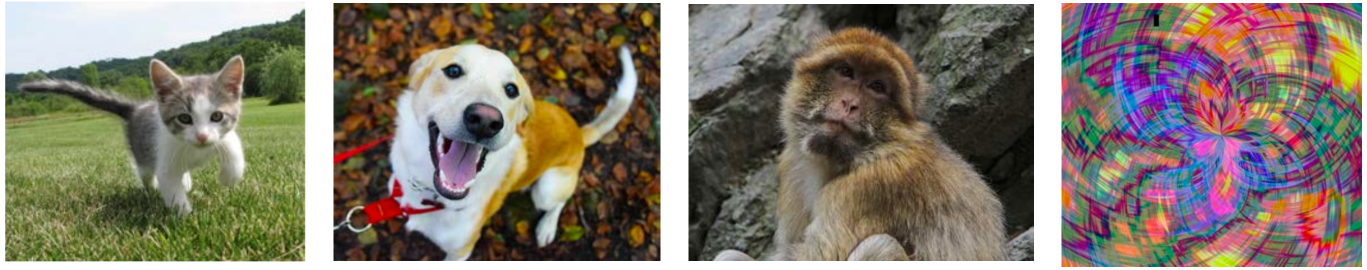
Discriminative vs Generative Models

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

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)





...

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

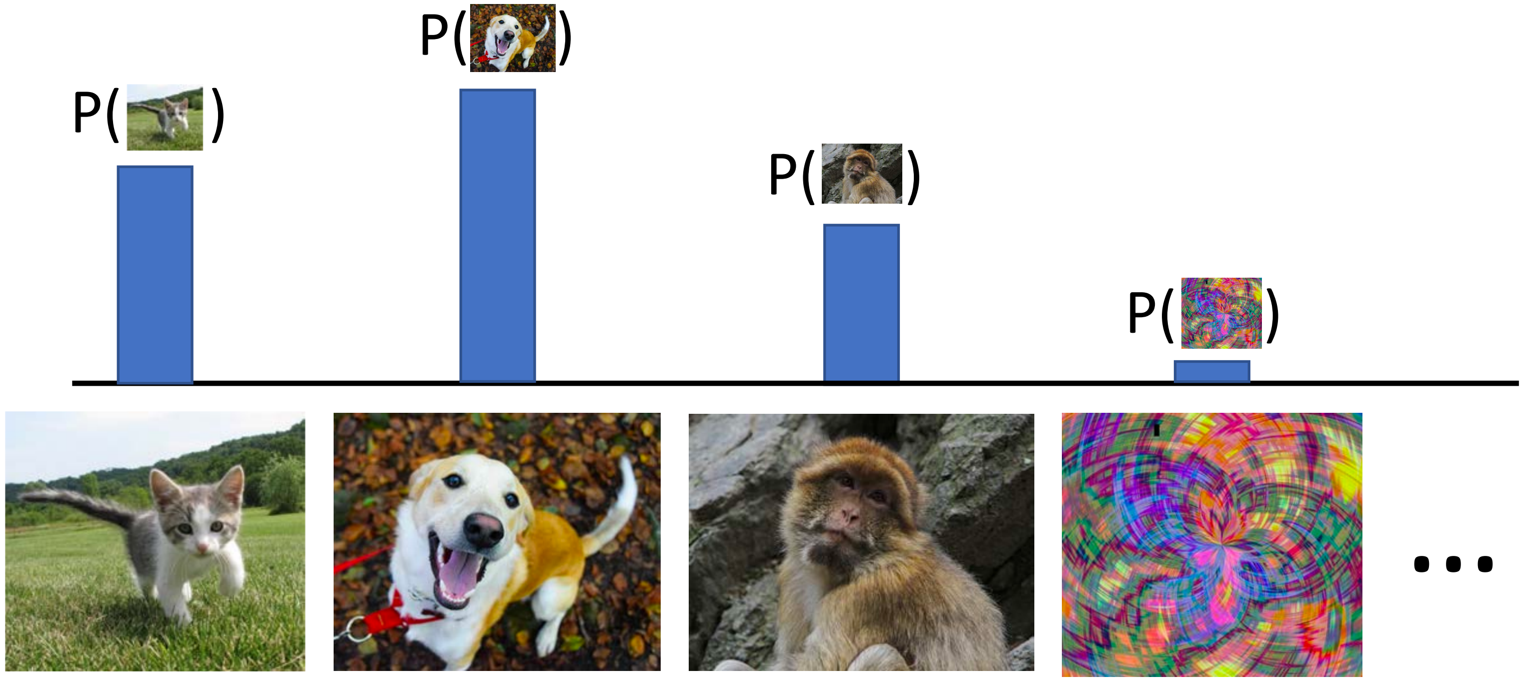
Requires deep image understanding! Is a dog more likely to sit or stand? How about 3-legged dog vs 3-armed monkey?

Discriminative vs Generative Models

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

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)



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

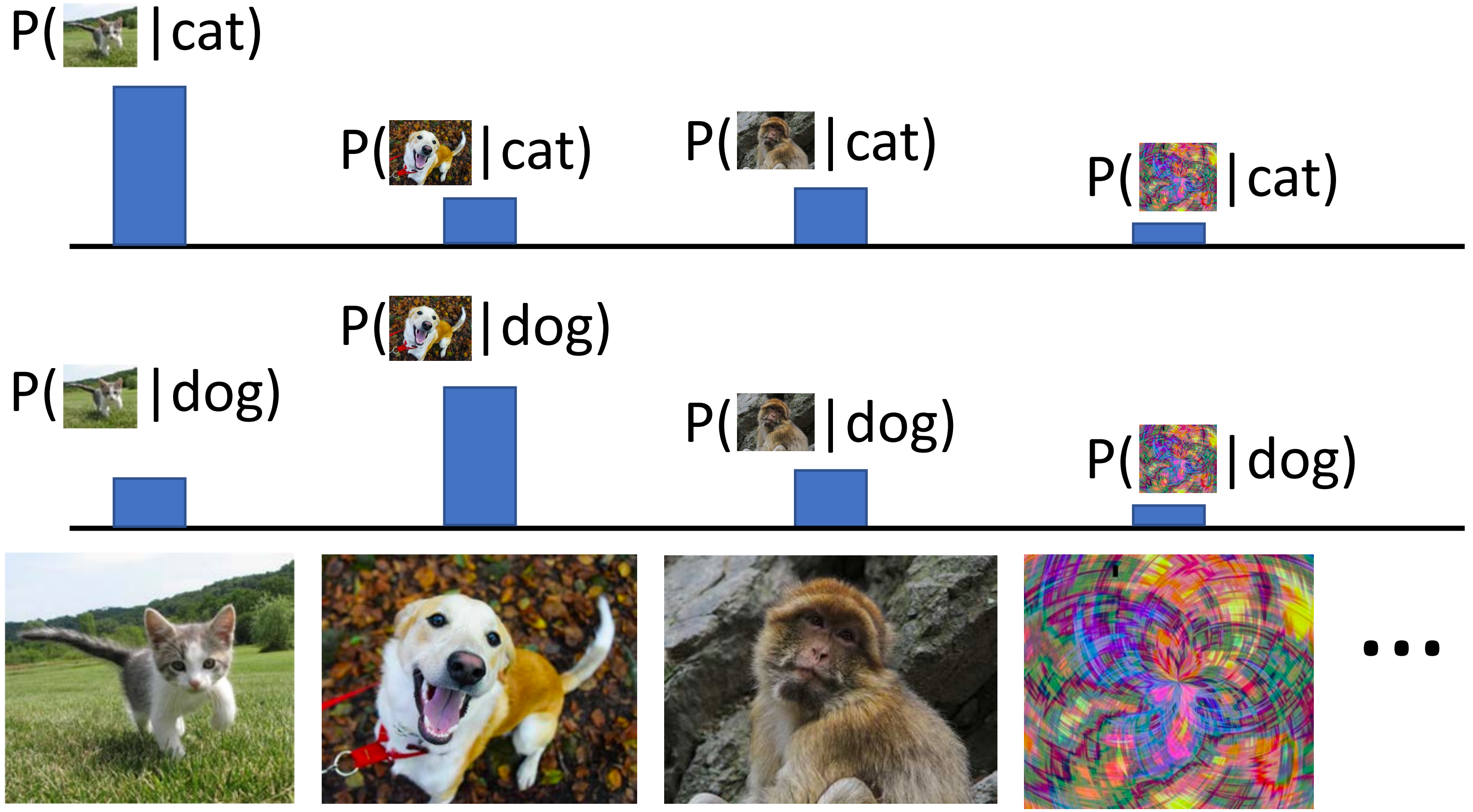
Model can “reject” unreasonable inputs by assigning them small values

Discriminative vs Generative Models

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

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)



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

Discriminative vs Generative Models

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

Generative Model:
Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)

Recall **Bayes' Rule:**

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

Discriminative vs Generative Models

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

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)

Recall **Bayes' Rule:**

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

We can build a conditional generative model from other components!

What can we do with a discriminative model?

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

→

Assign labels to data

Feature learning (with labels)

Generative Model:
Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)

What can we do with a generative model?

Discriminative Model:
Learn a probability distribution $p(y|x)$ → Assign labels to data
Feature learning (with labels)

Generative Model:
Learn a probability distribution $p(x)$ → Detect outliers
Feature learning (without labels)
Sample to **generate** new data

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

What can we do with a generative model?

Discriminative Model:
Learn a probability distribution $p(y|x)$ → Assign labels to data
Feature learning (with labels)

Generative Model:
Learn a probability distribution $p(x)$ → Detect outliers
Feature learning (without labels)
Sample to **generate** new data

Conditional Generative Model: Learn $p(x|y)$ → Assign labels, while rejecting outliers!
Generate new data conditioned on input labels

Taxonomy of Generative Models

Generative models

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

Taxonomy of Generative Models

Generative models

Model can compute p(x)

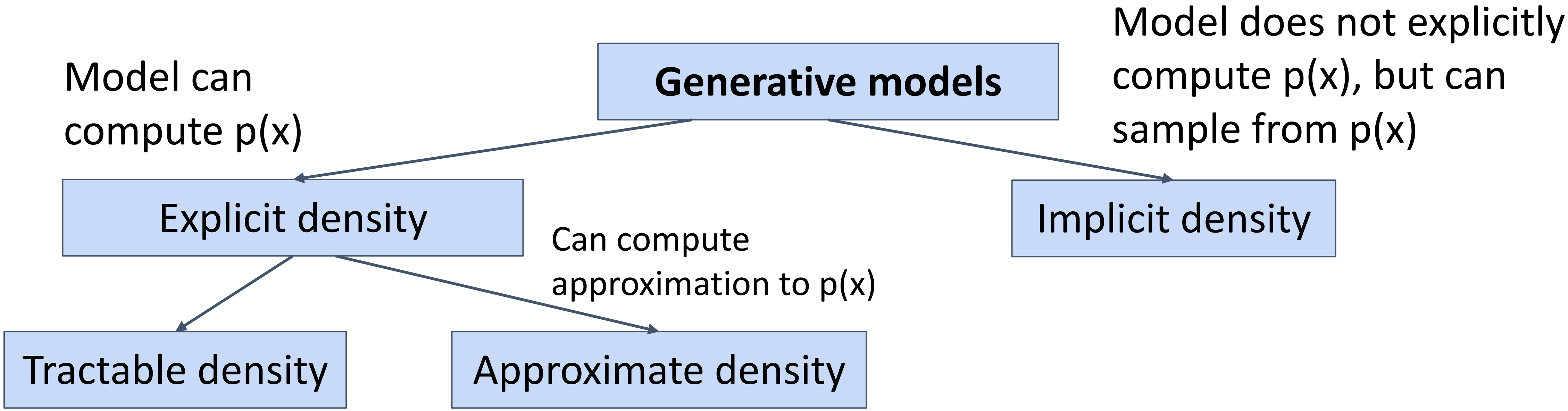
Explicit density

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

Implicit density

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

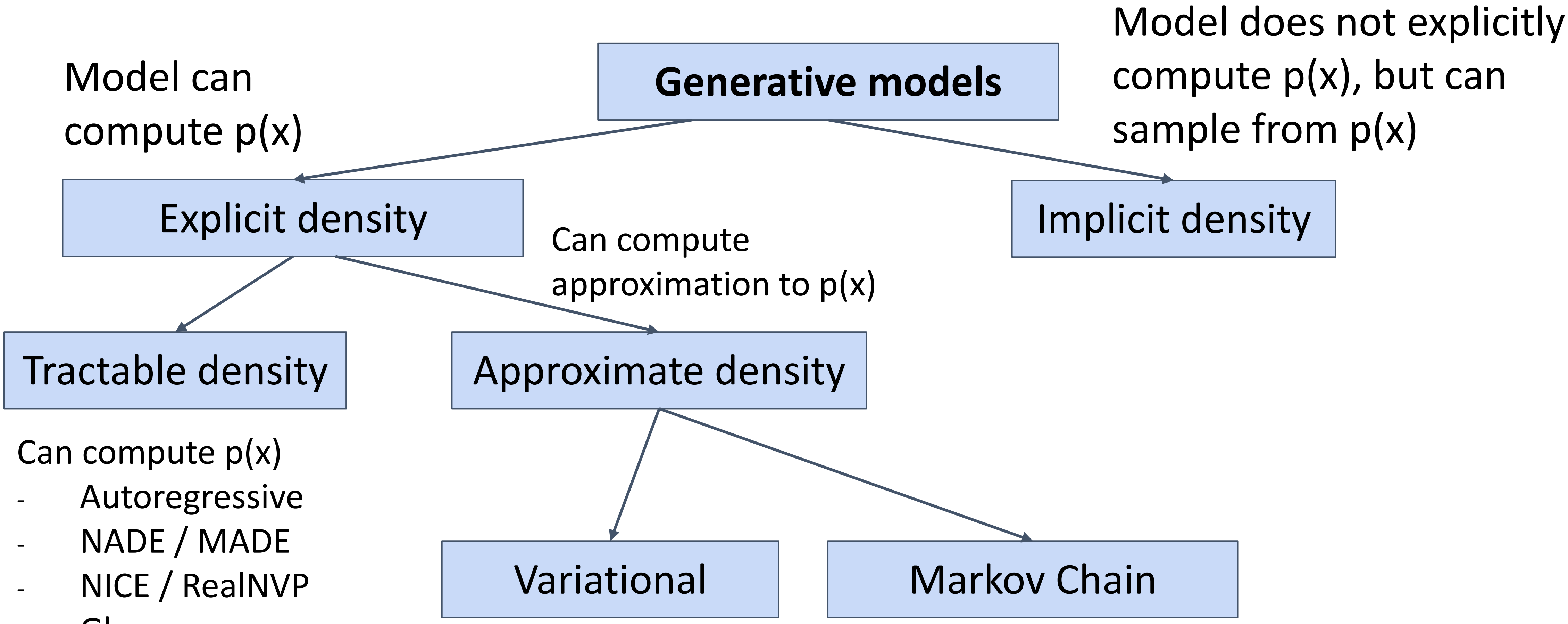
Taxonomy of Generative Models



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

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

Taxonomy of Generative Models



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

Variational Autoencoder

Boltzmann Machine

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

Taxonomy of Generative Models

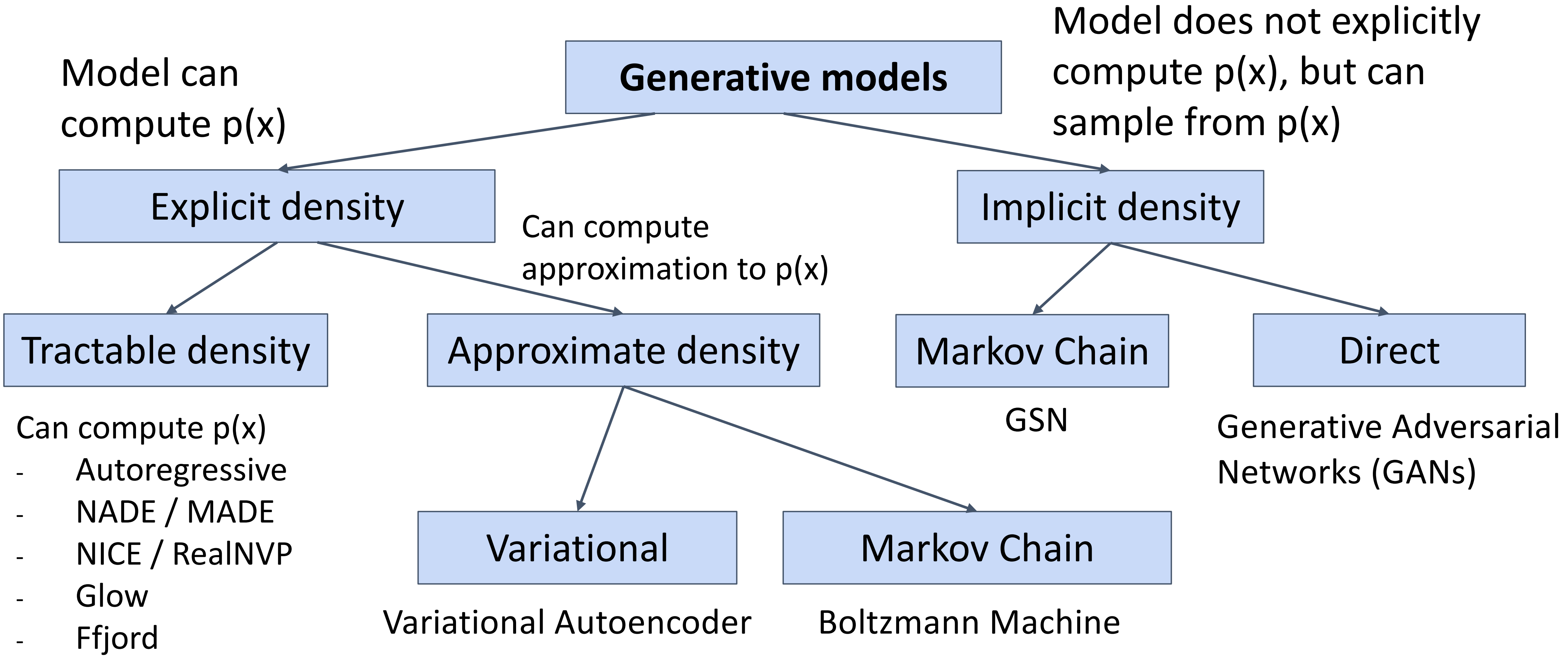


Figure adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Taxonomy of Generative Models

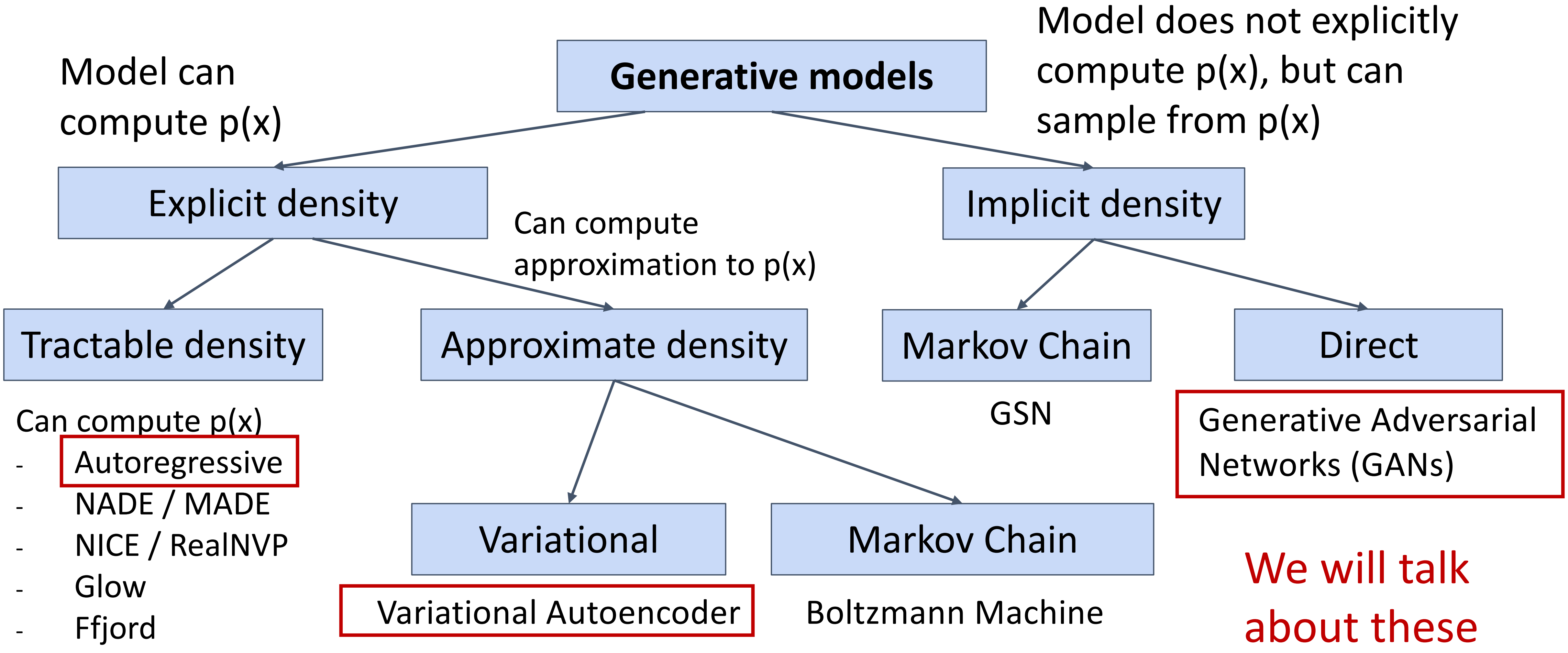


Figure adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Autoregressive models

Explicit Density Estimation

Goal: Write down an explicit function for $p(x) = f(x, W)$

Explicit Density Estimation

Goal: Write down an explicit function for $p(x) = f(x, W)$

Given dataset $x^{(1)}, x^{(2)}, \dots x^{(N)}$, train the model by solving:

$$W^* = \arg\max_{W} \prod_i p(x^{(i)})$$

Maximize probability of training data
(Maximum likelihood estimation)

Explicit Density Estimation

Goal: Write down an explicit function for $p(x) = f(x, W)$

Given dataset $x^{(1)}, x^{(2)}, \ldots x^{(N)}$, train the model by solving:

$$W^* = \arg\max_{W} \prod_i p(x^{(i)})$$

Maximize probability of training data
(Maximum likelihood estimation)

$$= \arg\max_{W} \sum_i \log p(x^{(i)})$$

Log trick to exchange product for sum

Explicit Density Estimation

Goal: Write down an explicit function for $p(x) = f(x, W)$

Given dataset $x^{(1)}, x^{(2)}, \ldots x^{(N)}$, train the model by solving:

$$W^* = \arg\max_{W} \prod_i p(x^{(i)})$$

Maximize probability of training data
(Maximum likelihood estimation)

$$= \arg\max_{W} \sum_i \log p(x^{(i)})$$

Log trick to exchange product for sum

$$= \arg\max_{W} \sum_i \log f(x^{(i)}, W)$$

This will be our loss function!
Train with gradient descent

Explicit Density: Autoregressive Models

Goal: Write down an explicit function for $p(x) = f(x, W)$

Assume x consists of multiple subparts:

$$x = (x_1, x_2, x_3, \ldots, x_T)$$

Explicit Density: Autoregressive Models

Goal: Write down an explicit function for $p(x) = f(x, W)$

Assume x consists of multiple subparts:

$$x = (x_1, x_2, x_3, \dots, x_T)$$

Break down probability using the chain rule:

$$\begin{aligned} p(x) &= p(x_1, x_2, x_3, \dots, x_T) \\ &= p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \dots \end{aligned}$$

Explicit Density: Autoregressive Models

Goal: Write down an explicit function for $p(x) = f(x, W)$

Assume x consists of multiple subparts:

$$x = (x_1, x_2, x_3, \ldots, x_T)$$

Break down probability using the chain rule:

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

Probability of the next subpart given all the previous subparts

Explicit Density: Autoregressive Models

Goal: Write down an explicit function for $p(x) = f(x, W)$

Assume x consists of multiple subparts:

$$x = (x_1, x_2, x_3, \dots, x_T)$$

Break down probability using the chain rule:

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

Probability of the next subpart given all the previous subparts

$$\begin{tikzcd} p(x_1) & p(x_2) & p(x_3) & p(x_4) \\ h_1 \arrow[u] \arrow[r] & h_2 \arrow[u] \arrow[r] & h_3 \arrow[u] \arrow[r] & h_4 \arrow[u] \\ x_0 \arrow[u] & x_1 \arrow[u] & x_2 \arrow[u] & x_3 \arrow[u] \end{tikzcd}$$

We've already seen this! Language modeling with an RNN!

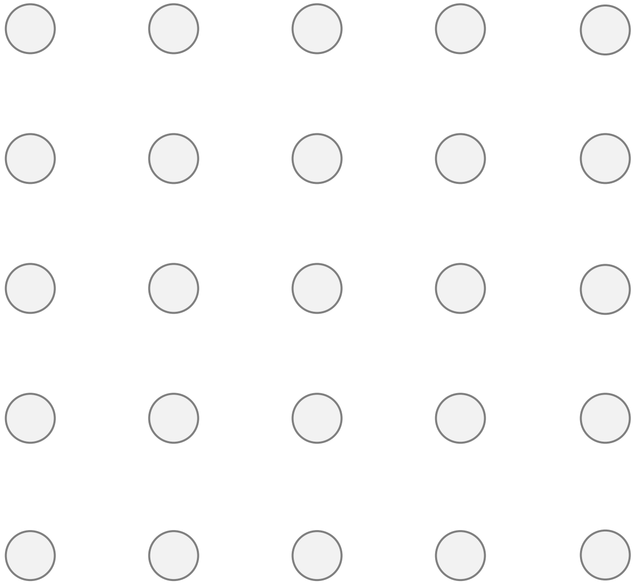
PixelRNN

Generate image pixels one at a time, starting at the upper left corner

Compute a hidden state for each pixel that depends on hidden states and RGB values from the left and from above (LSTM recurrence)

$$h_{x,y} = f(h_{x-1,y}, h_{x,y-1}, W)$$

At each pixel, predict red, then blue, then green: softmax over [0, 1, …, 255]



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

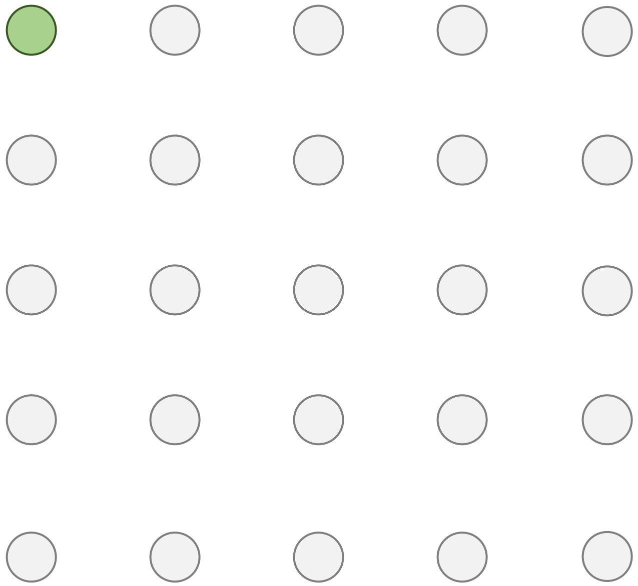
PixelRNN

Generate image pixels one at a time, starting at the upper left corner

Compute a hidden state for each pixel that depends on hidden states and RGB values from the left and from above (LSTM recurrence)

$$h_{x,y} = f(h_{x-1,y}, h_{x,y-1}, W)$$

At each pixel, predict red, then blue, then green: softmax over [0, 1, …, 255]



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

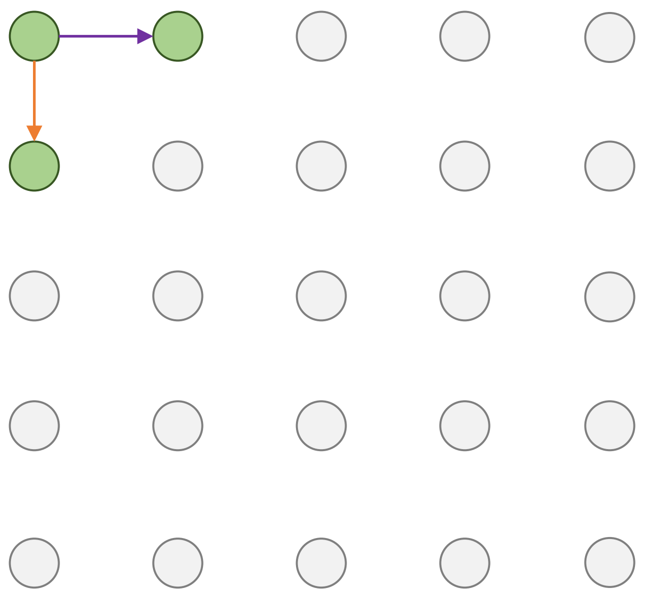
PixelRNN

Generate image pixels one at a time, starting at the upper left corner

Compute a hidden state for each pixel that depends on hidden states and RGB values from the left and from above (LSTM recurrence)

$$h_{x,y} = f(h_{x-1,y}, h_{x,y-1}, W)$$

At each pixel, predict red, then blue, then green: softmax over [0, 1, …, 255]



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

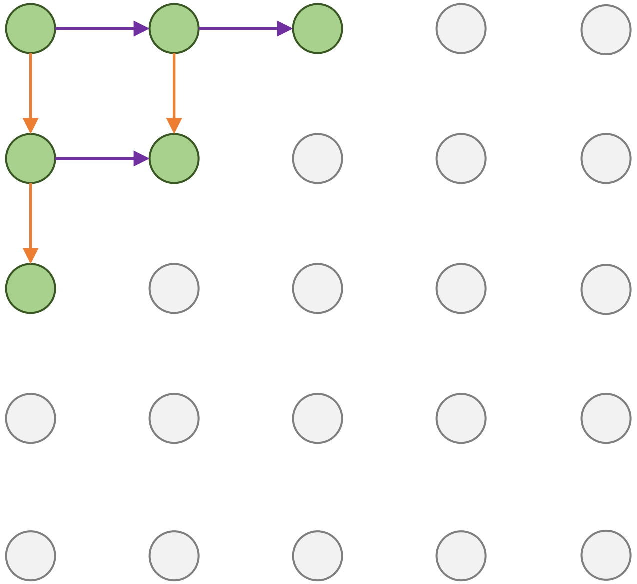
PixelRNN

Generate image pixels one at a time, starting at the upper left corner

Compute a hidden state for each pixel that depends on hidden states and RGB values from the left and from above (LSTM recurrence)

$$h_{x,y} = f(h_{x-1,y}, h_{x,y-1}, W)$$

At each pixel, predict red, then blue, then green: softmax over [0, 1, …, 255]



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

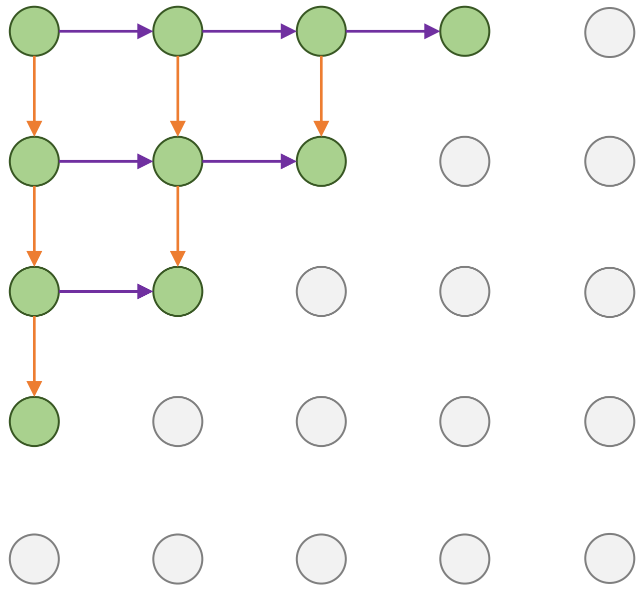
PixelRNN

Generate image pixels one at a time, starting at the upper left corner

Compute a hidden state for each pixel that depends on hidden states and RGB values from the left and from above (LSTM recurrence)

$$h_{x,y} = f(h_{x-1,y}, h_{x,y-1}, W)$$

At each pixel, predict red, then blue, then green: softmax over [0, 1, …, 255]



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

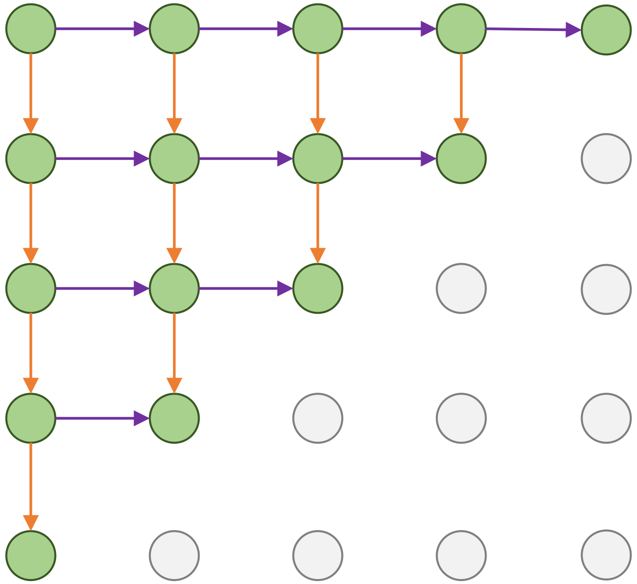
PixelRNN

Generate image pixels one at a time, starting at the upper left corner

Compute a hidden state for each pixel that depends on hidden states and RGB values from the left and from above (LSTM recurrence)

$$h_{x,y} = f(h_{x-1,y}, h_{x,y-1}, W)$$

At each pixel, predict red, then blue, then green: softmax over [0, 1, …, 255]



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

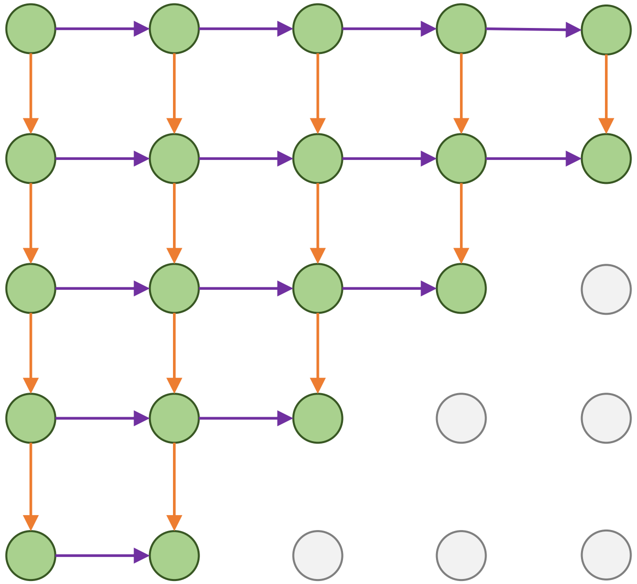
PixelRNN

Generate image pixels one at a time, starting at the upper left corner

Compute a hidden state for each pixel that depends on hidden states and RGB values from the left and from above (LSTM recurrence)

$$h_{x,y} = f(h_{x-1,y}, h_{x,y-1}, W)$$

At each pixel, predict red, then blue, then green: softmax over [0, 1, …, 255]



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

PixelRNN

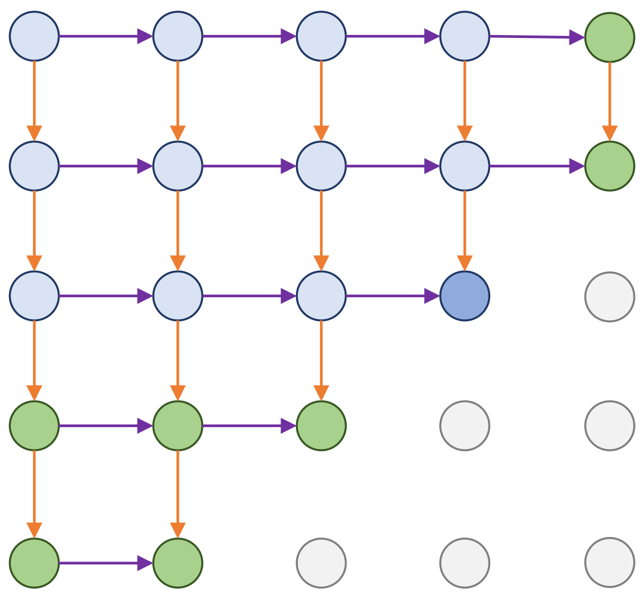
Generate image pixels one at a time, starting at the upper left corner

Compute a hidden state for each pixel that depends on hidden states and RGB values from the left and from above (LSTM recurrence)

$$h_{x,y} = f(h_{x-1,y}, h_{x,y-1}, W)$$

At each pixel, predict red, then blue, then green: softmax over [0, 1, …, 255]

Each pixel depends **implicity** on all pixels above and to the left:



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

PixelRNN

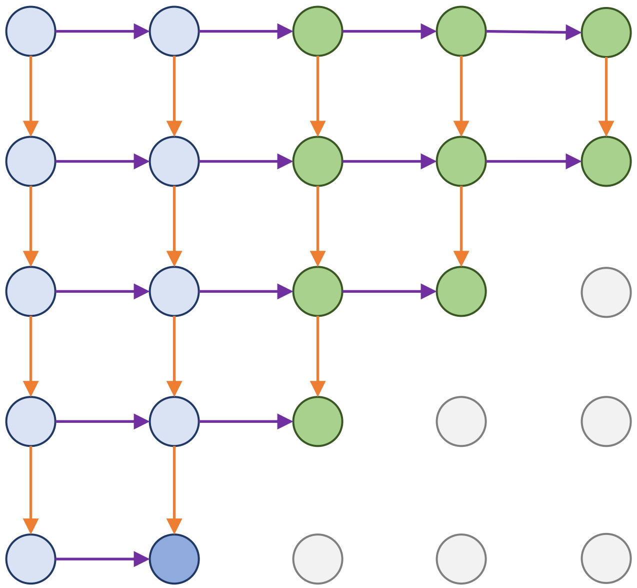
Generate image pixels one at a time, starting at the upper left corner

Compute a hidden state for each pixel that depends on hidden states and RGB values from the left and from above (LSTM recurrence)

$$h_{x,y} = f(h_{x-1,y}, h_{x,y-1}, W)$$

At each pixel, predict red, then blue, then green: softmax over [0, 1, …, 255]

Each pixel depends **implicity** on all pixels above and to the left:



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

PixelRNN

Generate image pixels one at a time, starting at the upper left corner

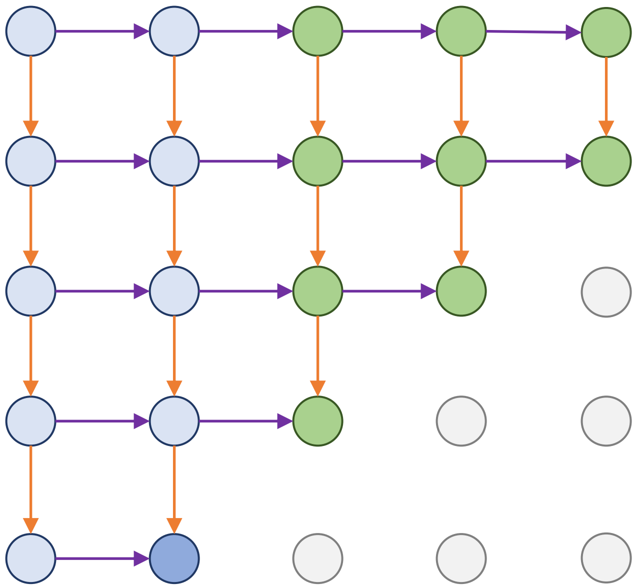
Compute a hidden state for each pixel that depends on hidden states and RGB values from the left and from above (LSTM recurrence)

$$h_{x,y} = f(h_{x-1,y}, h_{x,y-1}, W)$$

At each pixel, predict red, then blue, then green: softmax over [0, 1, …, 255]

Each pixel depends **implicity** on all pixels above and to the left:

Problem: Very slow during both training and testing; N x N image requires 2N-1 sequential steps

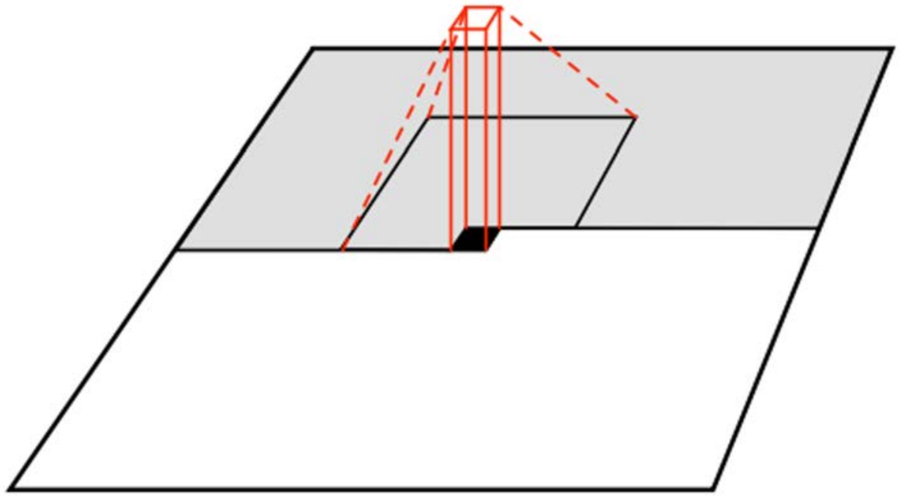


Van den Oord et al, "Pixel Recurrent Neural Networks", ICML 2016

PixelCNN

Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region



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

PixelCNN

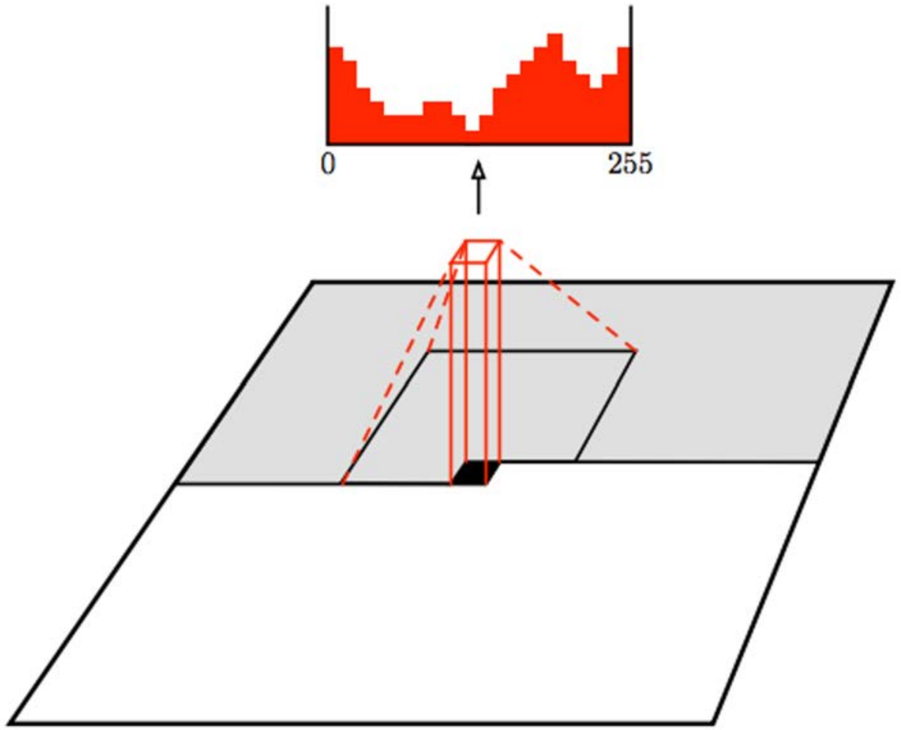
Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region

Training: maximize likelihood of training images

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

Softmax loss at each pixel



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

PixelCNN

Still generate image pixels starting from corner

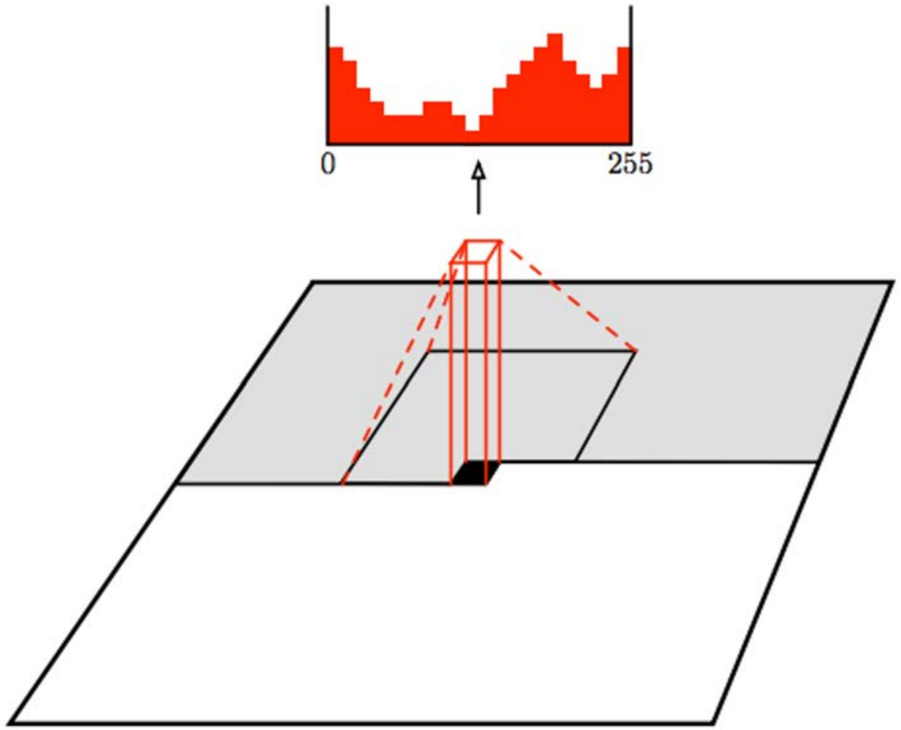
Dependency on previous pixels now modeled using a CNN over context region

Training: maximize likelihood of training images

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

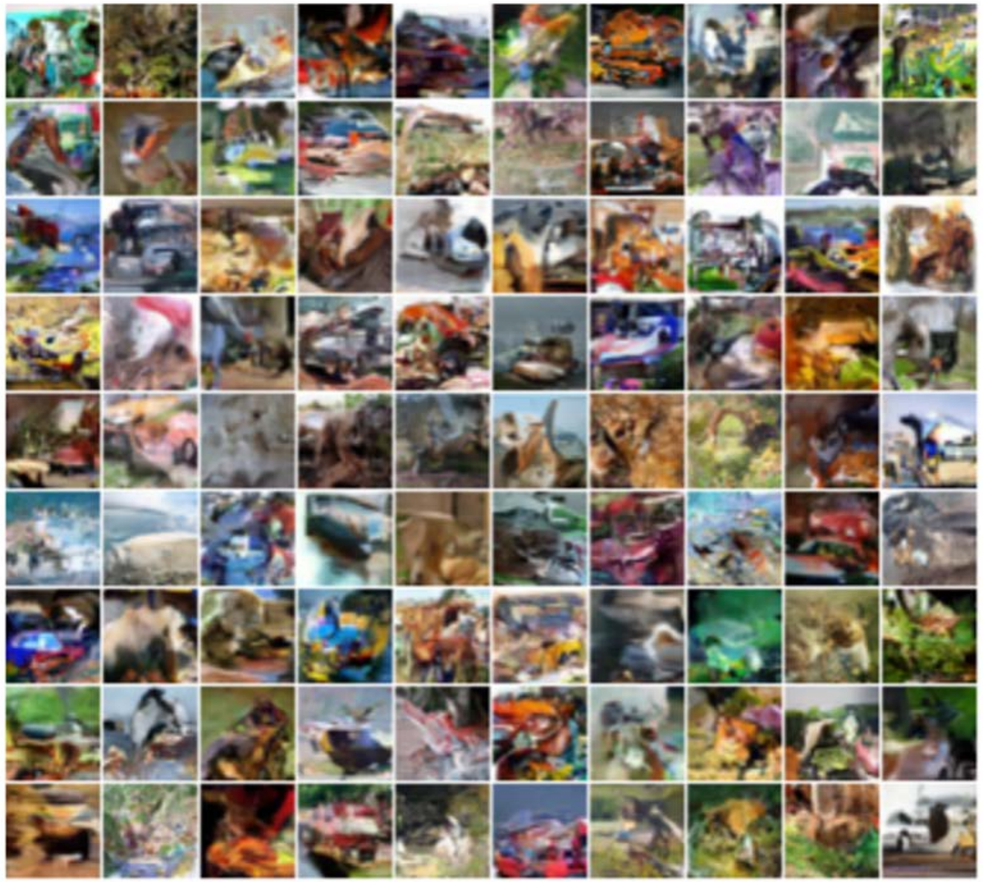
Generation must still proceed sequentially => still slow

Softmax loss at each pixel

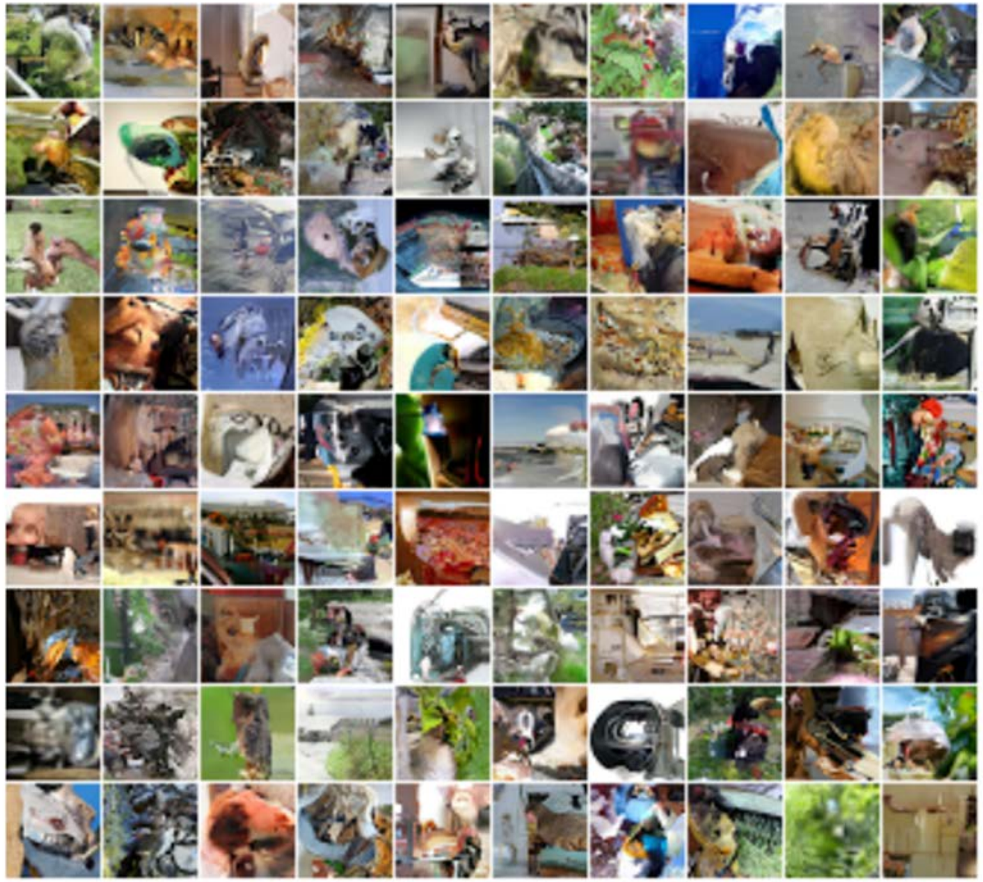


Van den Oord et al, “Conditional Image Generation with PixelCNN Decoders”, NeurIPS 2016

PixelRNN: Generated Samples



32x32 CIFAR-10



32x32 ImageNet

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

Autoregressive Models: PixelRNN and PixelCNN

Pros:

- Can explicitly compute likelihood p(x)
- Explicit likelihood of training data gives good evaluation metric
- Good samples

Con:

- 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++)

Variational Autoencoders

Variational Autoencoders

PixelRNN / PixelCNN explicitly parameterizes density function with a neural network, so we can train to maximize likelihood of training data:

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

Variational Autoencoders (VAE) define an **intractable density** that we cannot explicitly compute or optimize

But we will be able to directly optimize a **lower bound** on the density

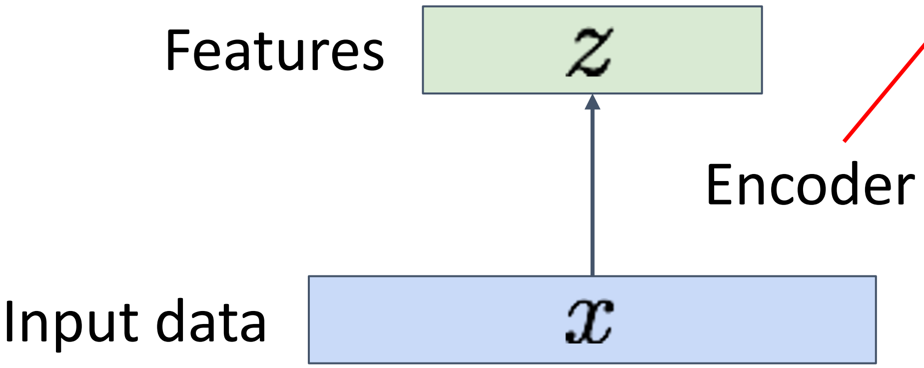
Variational <u>Autoencoders</u>

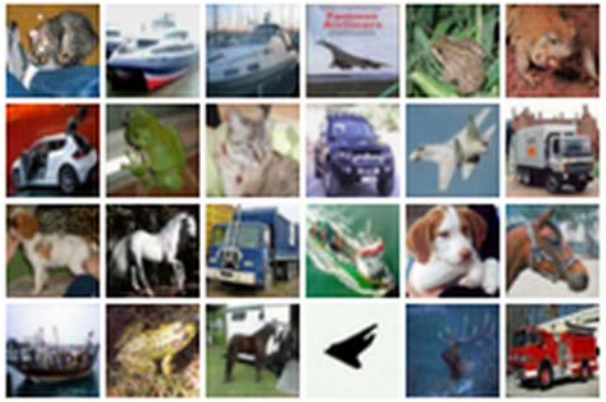
(Regular, non-variational) Autoencoders

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

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

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





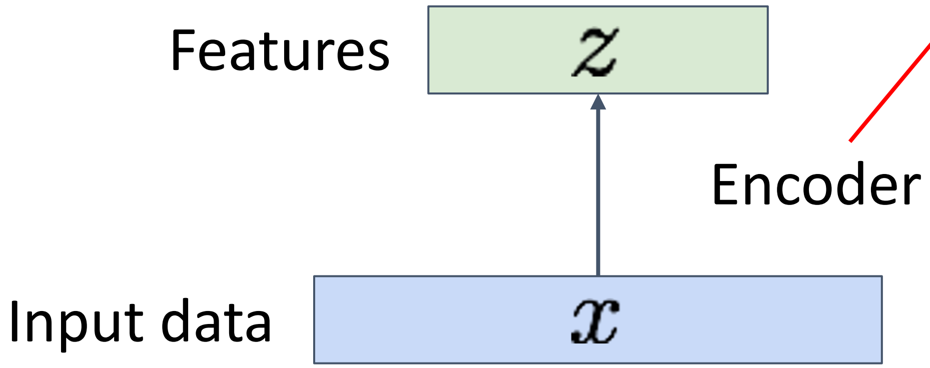
Input Data

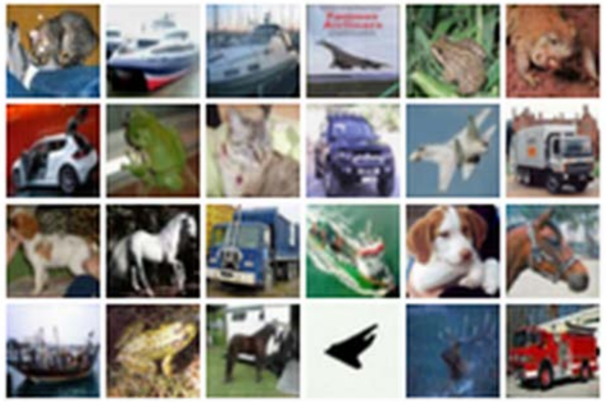
(Regular, non-variational) Autoencoders

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

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

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





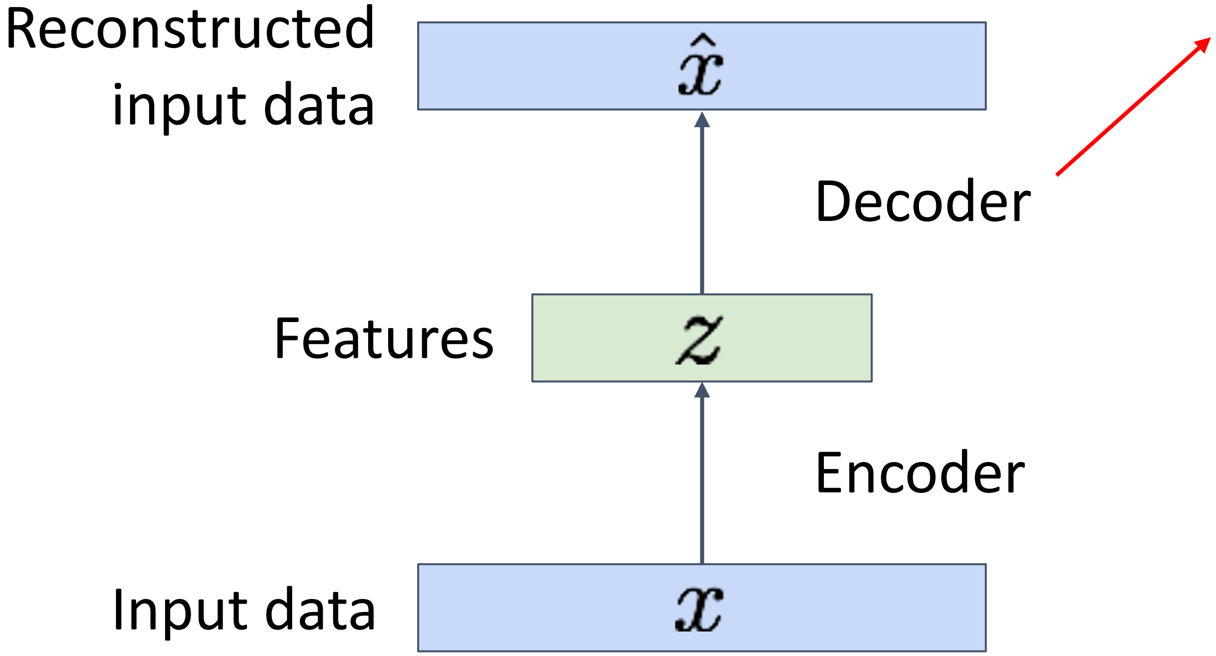
Input Data

(Regular, non-variational) Autoencoders

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

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

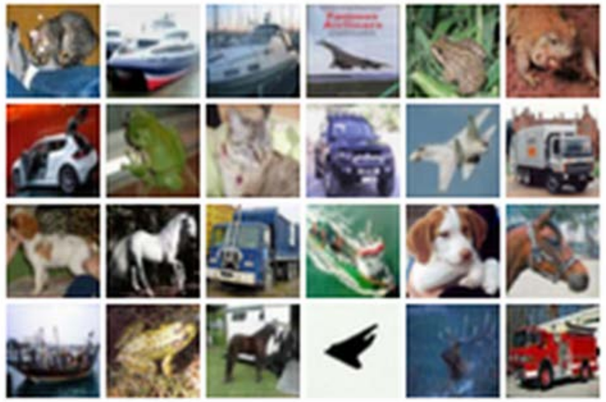
"Autoencoding" = encoding itself



Originally: Linear + nonlinearity (sigmoid)

Later: Deep, fully-connected

Later: ReLU CNN (upconv)

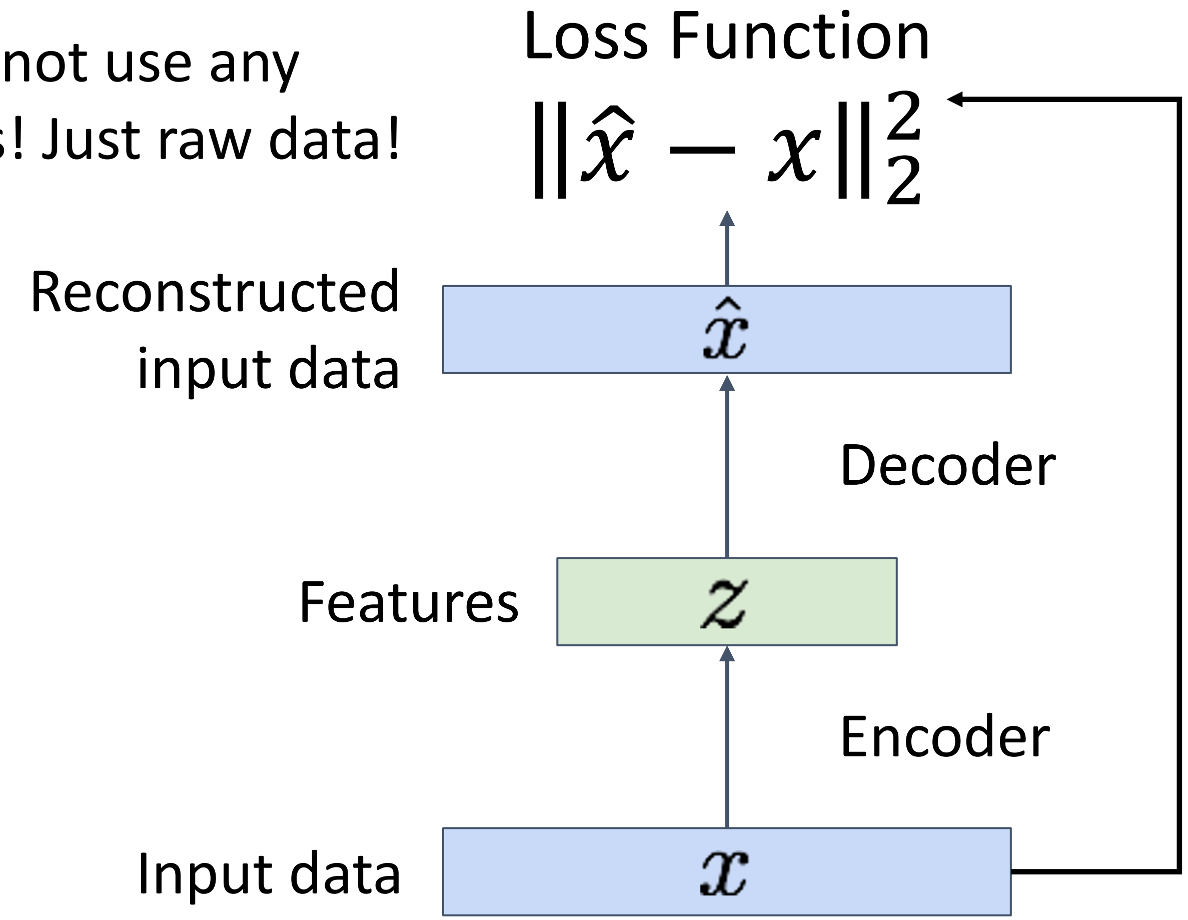


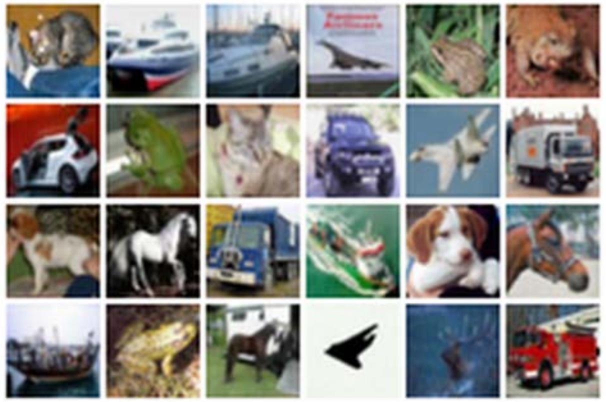
Input Data

(Regular, non-variational) Autoencoders

Loss: L2 distance between input and reconstructed data.

Does not use any labels! Just raw data!





Input Data

(Regular, non-variational) Autoencoders

Loss: L2 distance between input and reconstructed data.

Does not use any labels! Just raw data!

Loss Function

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

Reconstructed input data $\hat{x}$

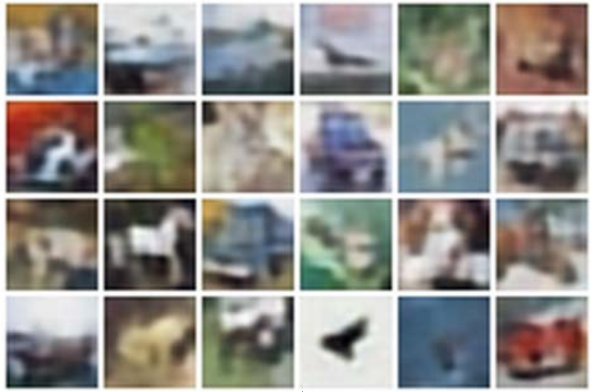
Decoder

Features z

Encoder

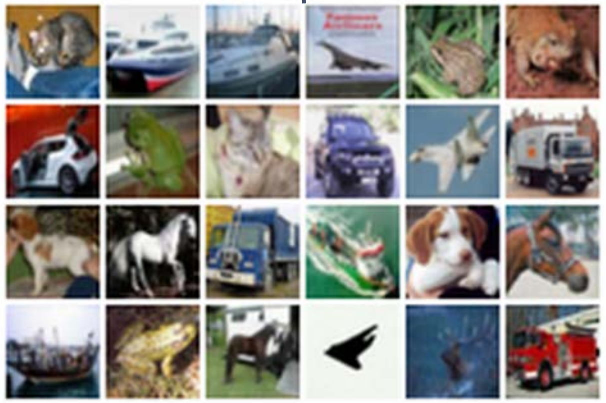
Input data x

Reconstructed data



Decoder:
4 tconv layers

Encoder:
4 conv layers



Input Data

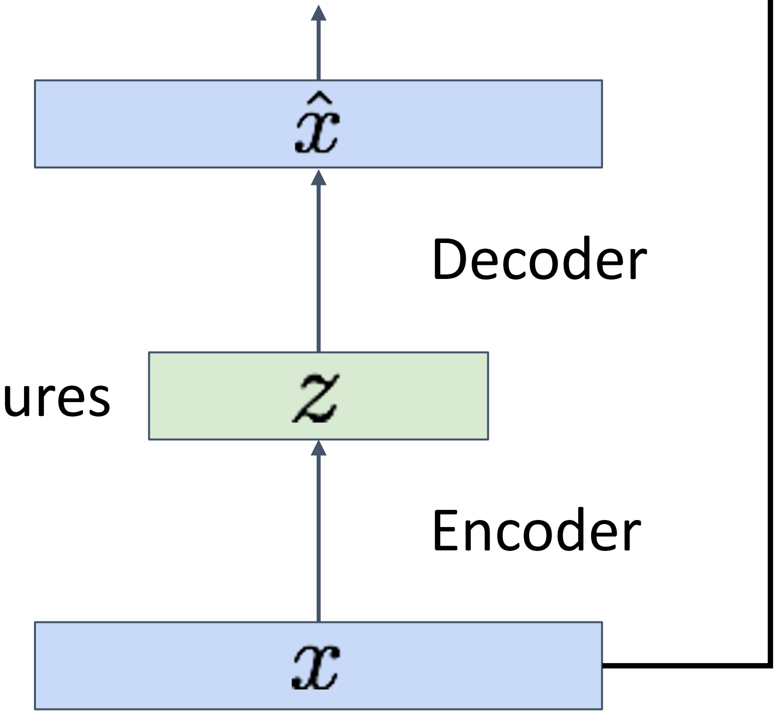
(Regular, non-variational) Autoencoders

Loss: L2 distance between input and reconstructed data.

Does not use any labels! Just raw data!

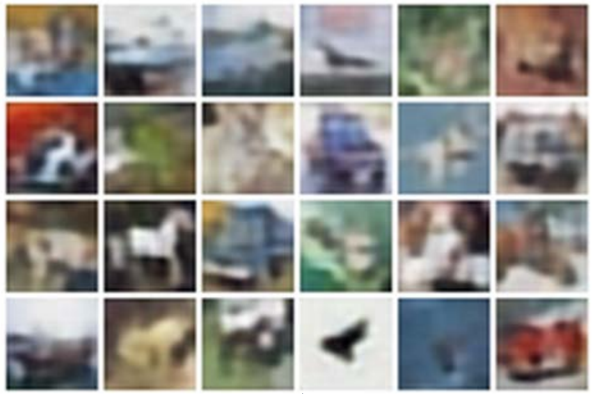
Loss Function

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

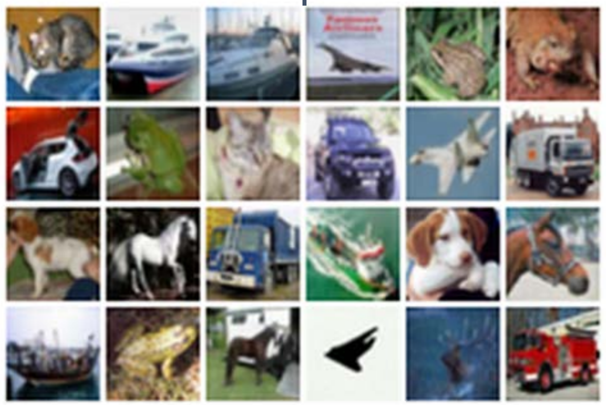


Features need to be **lower dimensional** than the data

Reconstructed data



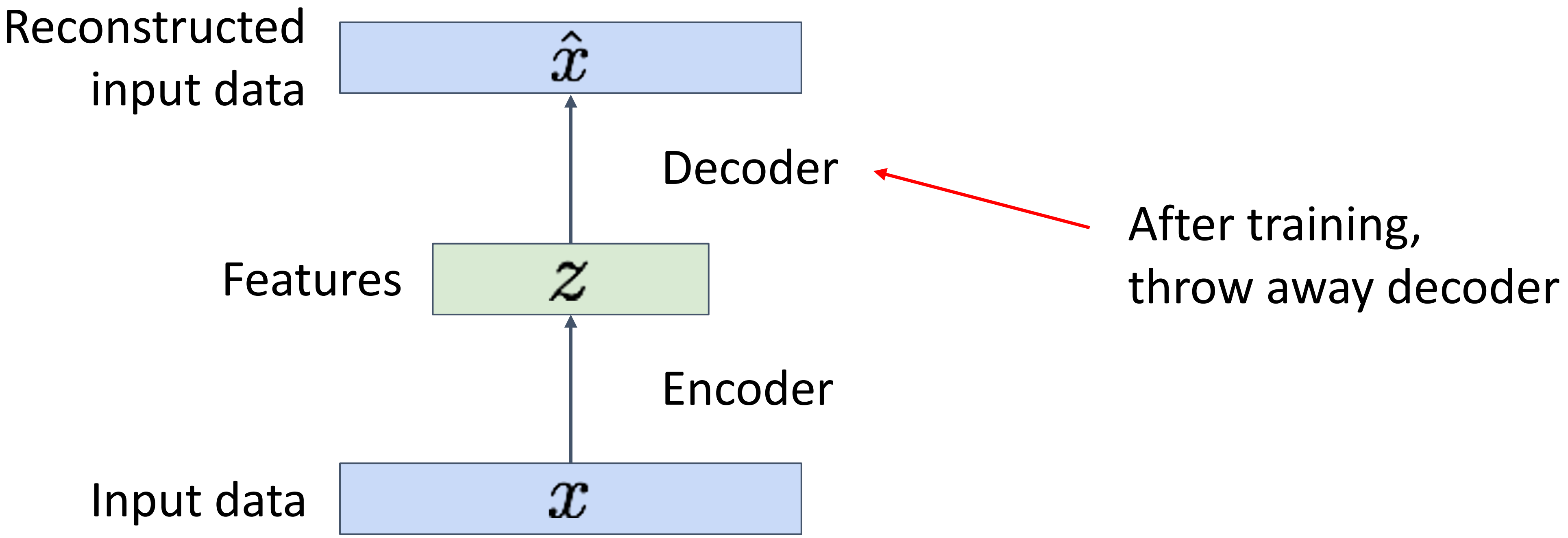
Decoder:
4 tconv layers
Encoder:
4 conv layers



Input Data

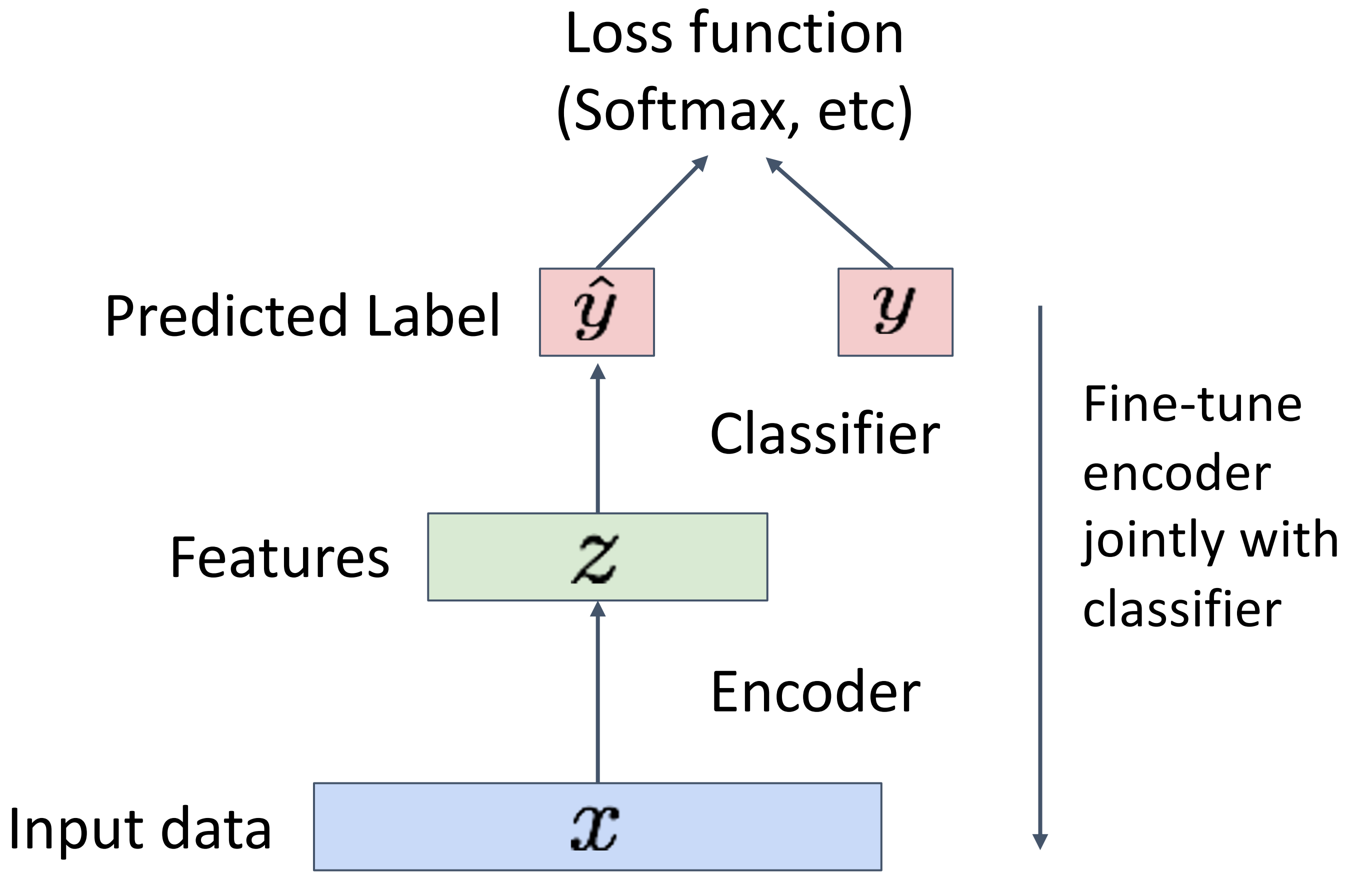
(Regular, non-variational) Autoencoders

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



(Regular, non-variational) Autoencoders

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



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

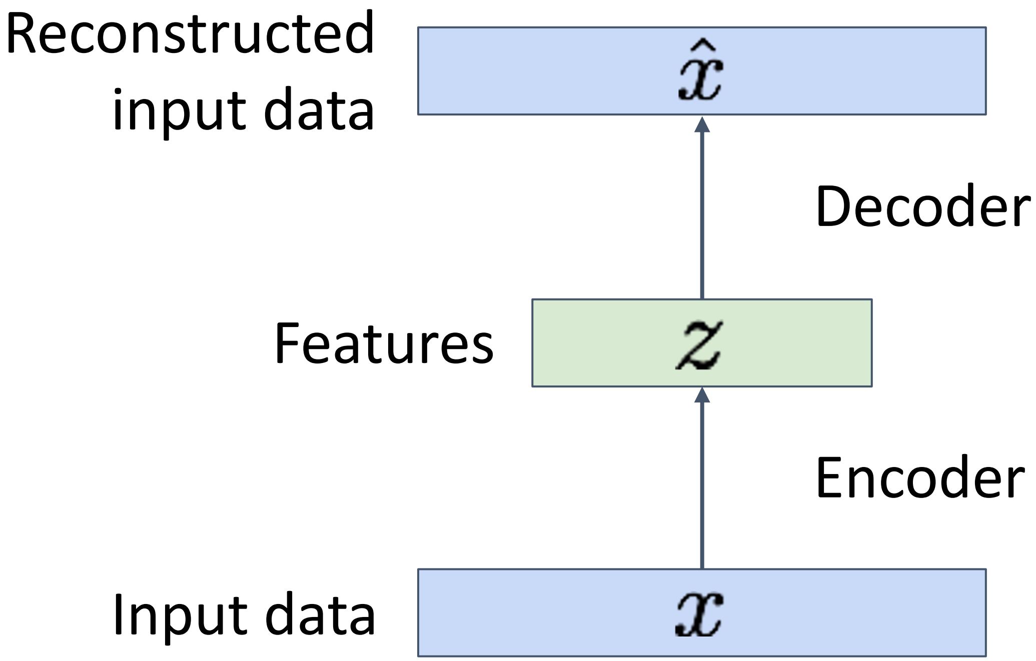
Train for final task (sometimes with small data)

(Regular, non-variational) Autoencoders

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

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

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



Variational Autoencoders

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

Variational Autoencoders

Probabilistic spin on autoencoders:

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

Variational Autoencoders

Probabilistic spin on autoencoders:
1. Learn latent features z from raw data
2. Sample from the model to generate new data

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

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

Variational Autoencoders

Probabilistic spin on autoencoders:

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

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

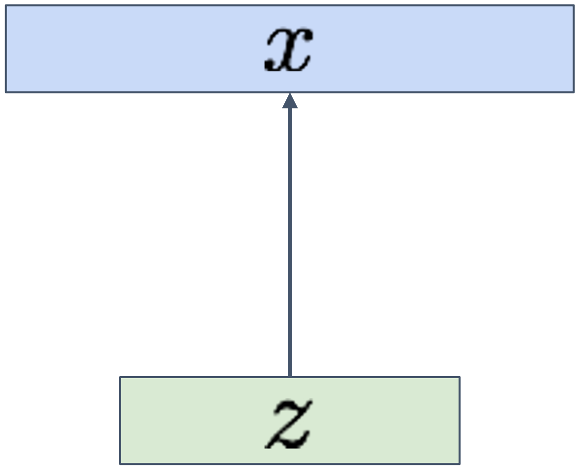
After training, sample new data like this:

Sample from conditional

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

Sample z from prior

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



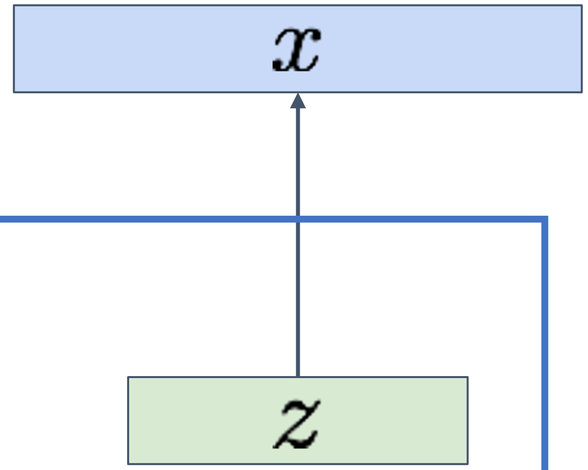
Intuition: x is an image, **z** is latent factors used to generate **x:** attributes, orientation, etc.

Variational Autoencoders

Probabilistic spin on autoencoders:
1. Learn latent features z from raw data
2. Sample from the model to generate new data

After training, sample new data like this:

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



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

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

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

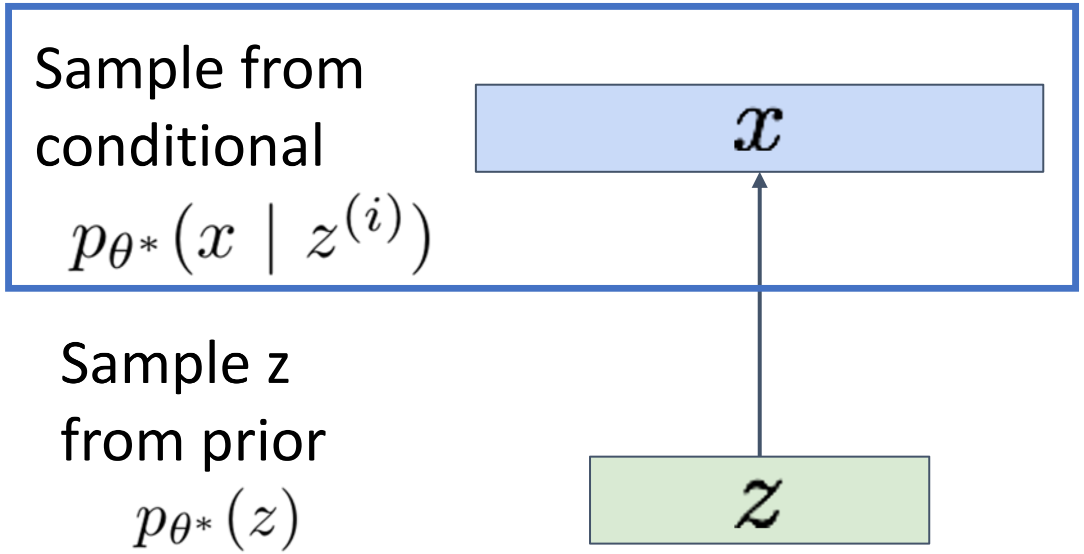
Assume simple prior p(z), e.g. Gaussian

Variational Autoencoders

Probabilistic spin on autoencoders:

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

After training, sample new data like this:



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

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

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

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

Variational Autoencoders

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

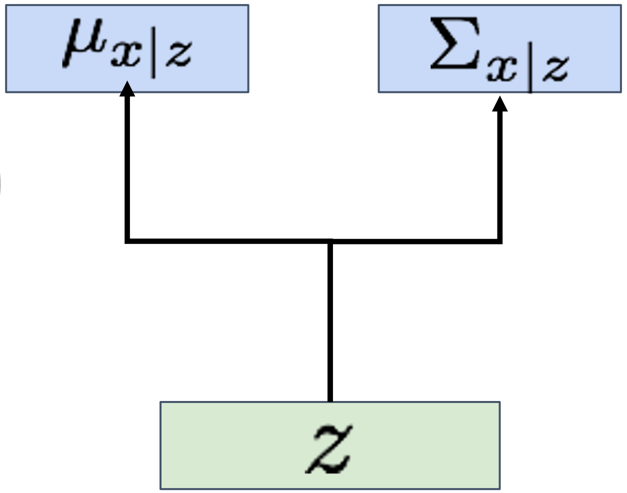
Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample from conditional

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

Sample z from prior

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



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

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

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

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

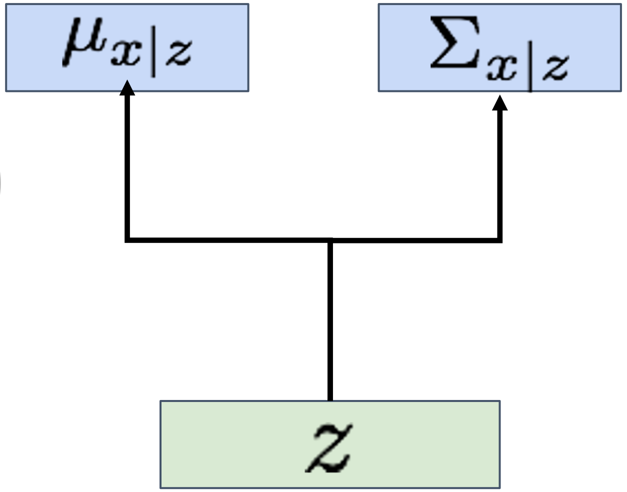
Variational Autoencoders

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

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

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

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



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

How to train this model?

Basic idea: **maximize likelihood of data**

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

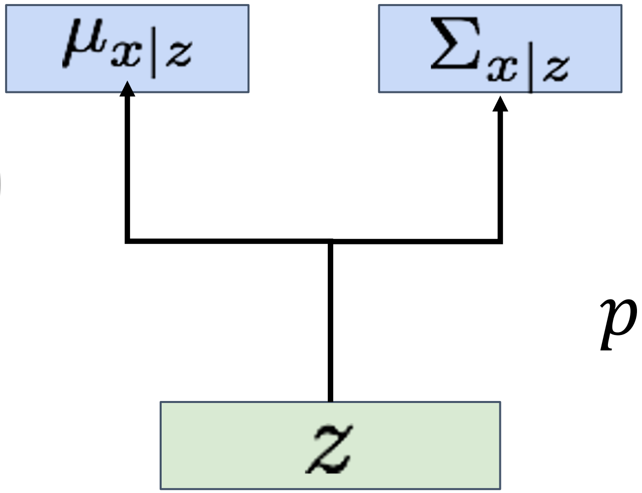
Variational Autoencoders

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

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

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

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



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

How to train this model?

Basic idea: **maximize likelihood of data**

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

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

Variational Autoencoders

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

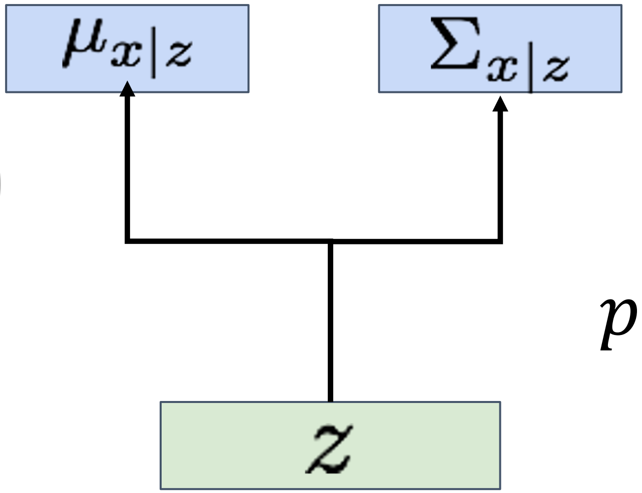
Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample from conditional

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

Sample z from prior

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



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

How to train this model?

Basic idea: **maximize likelihood of data**

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

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

Ok, can compute this with decoder network

Variational Autoencoders

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

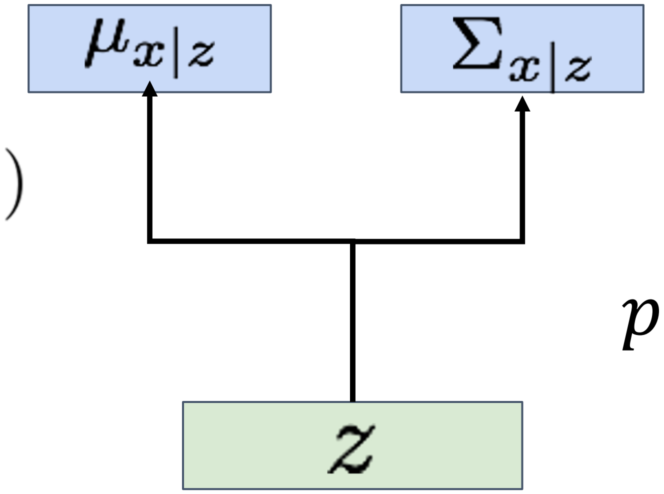
Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample from conditional

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

Sample z from prior

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



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

How to train this model?

Basic idea: **maximize likelihood of data**

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

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

Ok, we assumed Gaussian prior for z

Variational Autoencoders

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

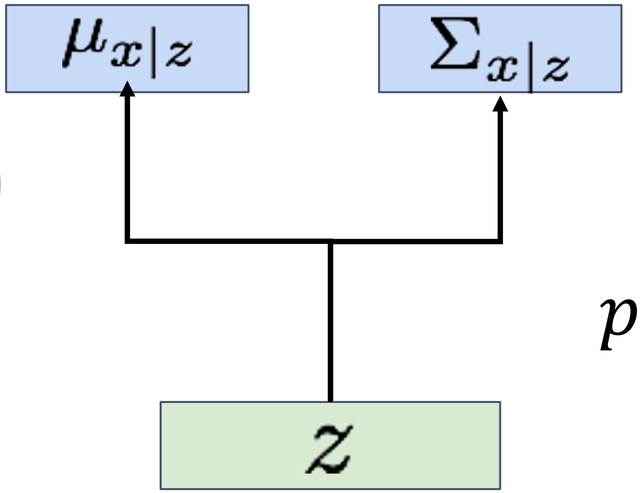
Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample from conditional

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

Sample z from prior

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



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

How to train this model?

Basic idea: **maximize likelihood of data**

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

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

Problem: Impossible to integrate over all z!

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

Variational Autoencoders

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

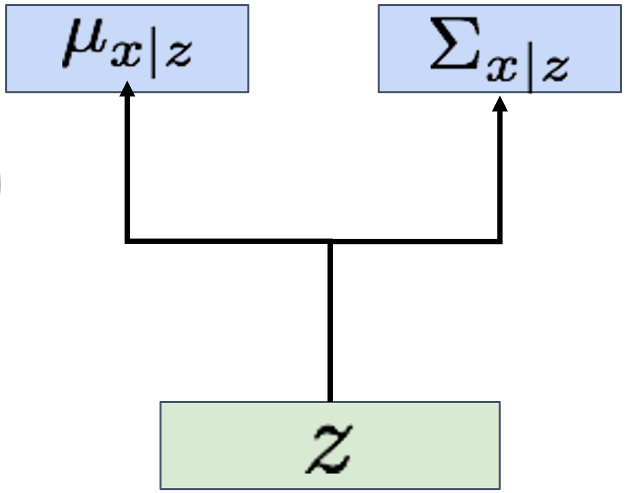
Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample from conditional

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

Sample z from prior

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



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

How to train this model?

Basic idea: **maximize likelihood of data**

Another idea: Try Bayes' Rule:

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

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

Variational Autoencoders

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

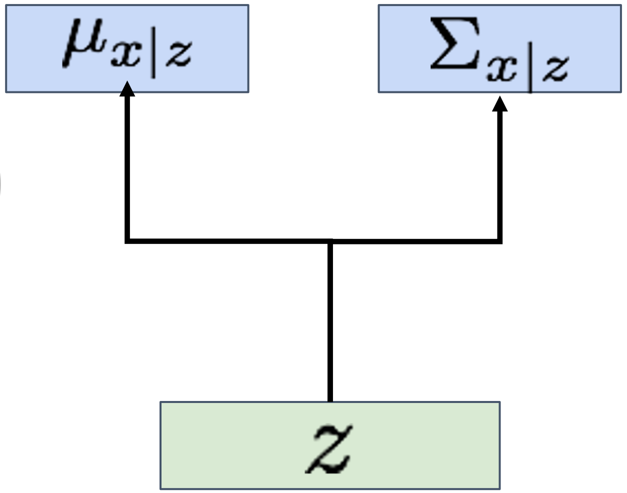
Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample from conditional

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

Sample z from prior

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



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

How to train this model?

Basic idea: **maximize likelihood of data**

Another idea: Try Bayes' Rule:

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

Ok, compute with decoder network

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

Variational Autoencoders

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

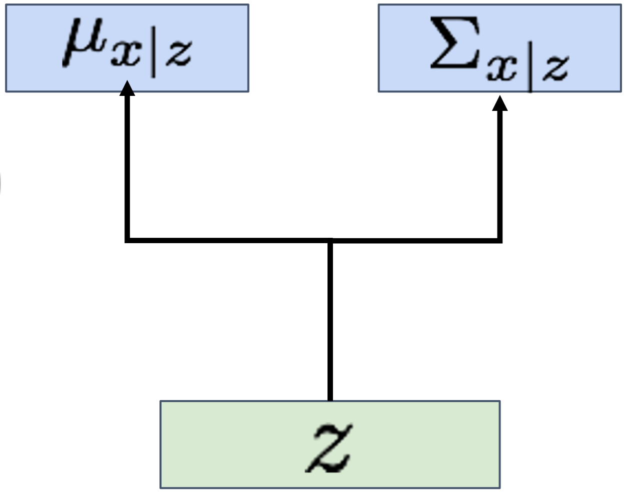
Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample from conditional

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

Sample z from prior

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



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

How to train this model?

Basic idea: **maximize likelihood of data**

Another idea: Try Bayes' Rule:

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

Ok, we assumed Gaussian prior

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

Variational Autoencoders

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

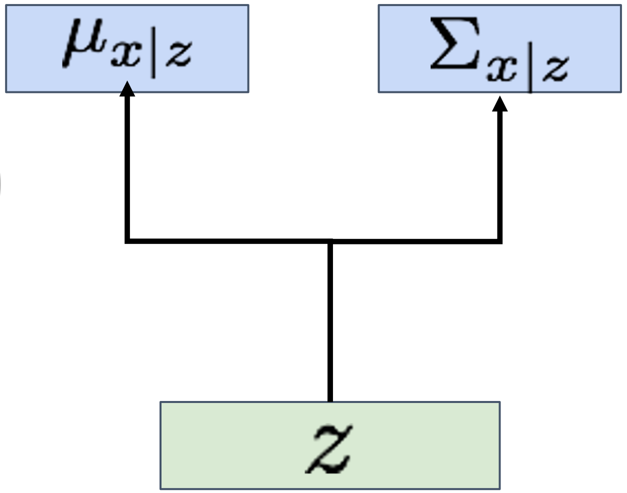
Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample from conditional

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

Sample z from prior

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



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

How to train this model?

Basic idea: **maximize likelihood of data**

Another idea: Try Bayes' Rule:

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

Problem: No way to compute this!

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

Variational Autoencoders

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

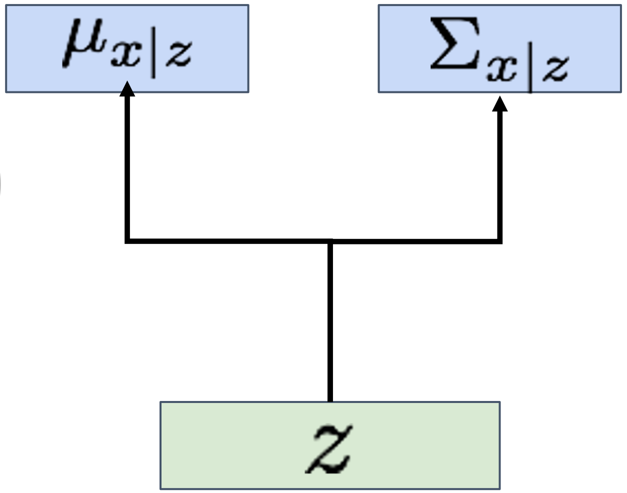
Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample from conditional

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

Sample z from prior

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



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

How to train this model?

Basic idea: **maximize likelihood of data**

Another idea: Try Bayes' Rule:

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

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

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

Variational Autoencoders

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

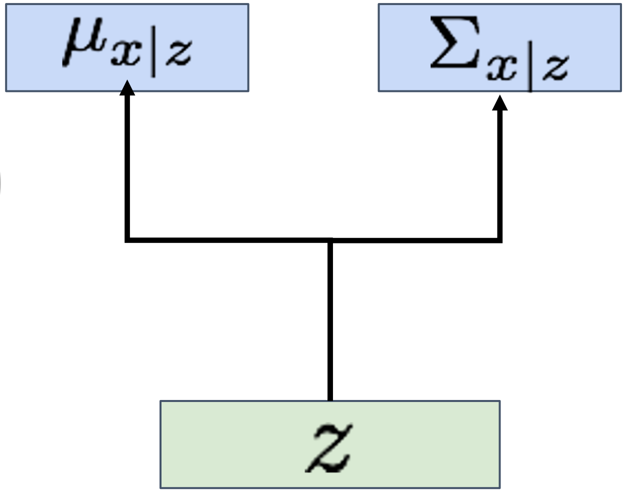
Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample from conditional

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

Sample z from prior

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



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

How to train this model?

Basic idea: **maximize likelihood of data**

Another idea: Try Bayes' Rule:

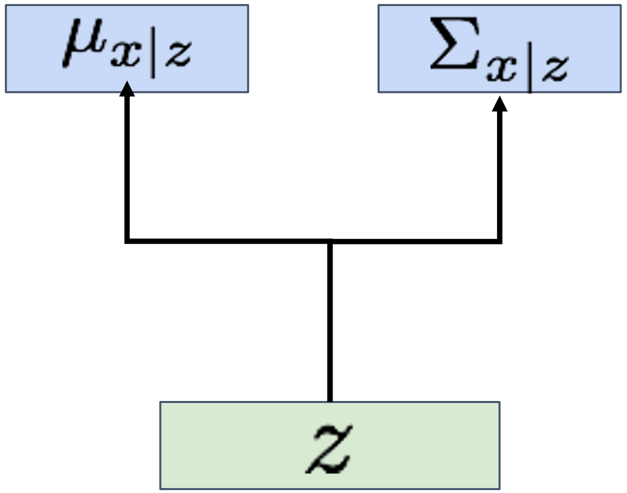
$$p_\theta(x) = \frac{p_\theta(x \mid z)p_\theta(z)}{p_\theta(z \mid x)} \approx \frac{p_\theta(x \mid z)p_\theta(z)}{\boxed{q_\phi(z \mid x)}}$$

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

Variational Autoencoders

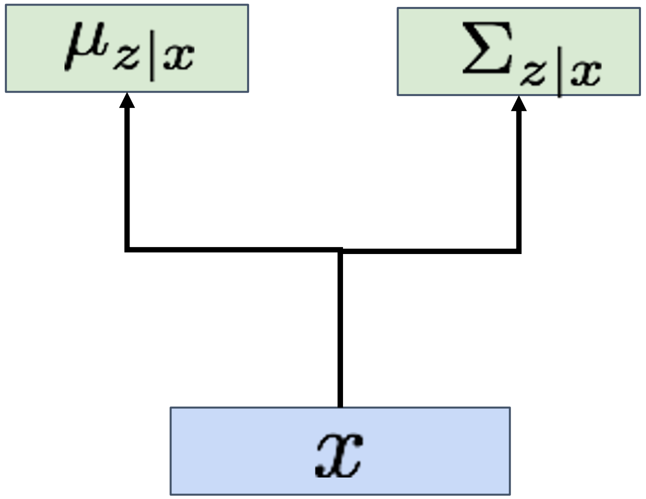
Decoder network inputs latent code z, gives distribution over data x

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



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

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



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

then we can approximate

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

Idea: Jointly train both encoder and decoder

Variational Autoencoders

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

Bayes' Rule

Variational Autoencoders

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

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

Variational Autoencoders

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

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

Split up using rules for logarithms

Variational Autoencoders

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

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

Split up using rules for logarithms

Variational Autoencoders

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

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

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

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

Variational Autoencoders

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

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

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

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

Variational Autoencoders

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

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

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

Data reconstruction

Variational Autoencoders

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

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

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

KL divergence between prior, and samples from the encoder network

Variational Autoencoders

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

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

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

KL divergence between encoder
and posterior of decoder

Variational Autoencoders

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

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

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

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

Variational Autoencoders

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

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

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

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

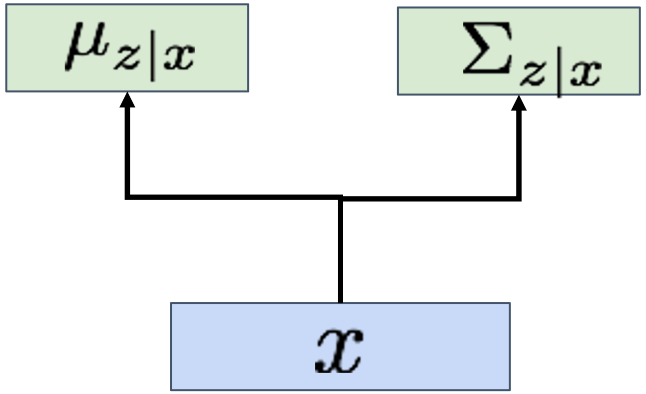
Variational Autoencoders

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

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

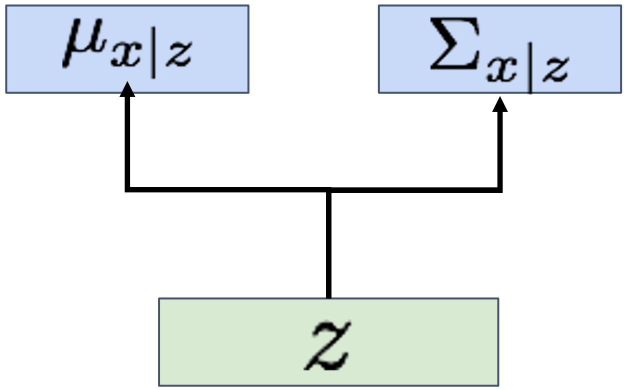
Encoder Network

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



Decoder Network

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



Next Time:
Generative Models, part 2

More Variational Autoencoders, Generative Adversarial Networks