# Lecture 17:
# Generative AI Part 1
## Autoregressive & VAEs

# Administrative

- A5 is out. It is the last assignment.
- A5 deadline changed to Friday March 8th 11:59pm
- Milestone is due tonight

- Almost done with the course :(

# Last time: Foundation Models

| Language | Classification | LM + Vision | And More! | Chaining |
| --- | --- | --- | --- | --- |
| ELMo | CLIP | Flamingo | Segment Anything | LMs + CLIP |
| BERT | CoCa | GPT-4V | Whisper | Visual Programming |
| GPT | | Gemini | Dall-E | |
| T5 | | | Stable Diffusion | |
| | | | Imagen | |

# Next 2 lectures:

| Language | Classification | LM + Vision | And More! | Chaining |
|---|---|---|---|---|
| ELMo | CLIP | Flamingo | Segment Anything | LMs + CLIP |
| BERT | CoCa | GPT-4V | Whisper | Visual Programming |
| GPT | | Gemini | Dall-E | |
| T5 | | | Stable Diffusion | |
| | | | Imagen | |

# 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.

# 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.

Cat

Classification

# 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.

*A cat sitting on a suitcase on the floor*

Image captioning

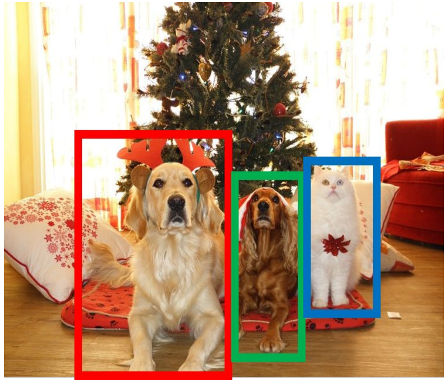
# Supervised vs Unsupervised Learning

**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.



DOG, DOG, CAT

Object Detection

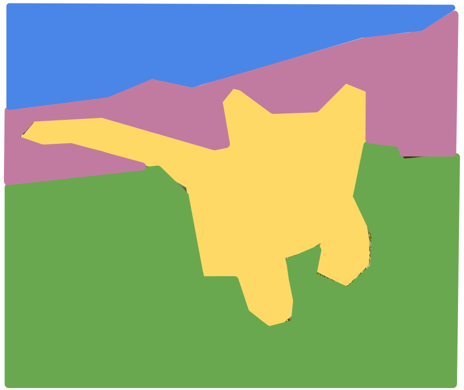
# Supervised vs Unsupervised Learning

**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.

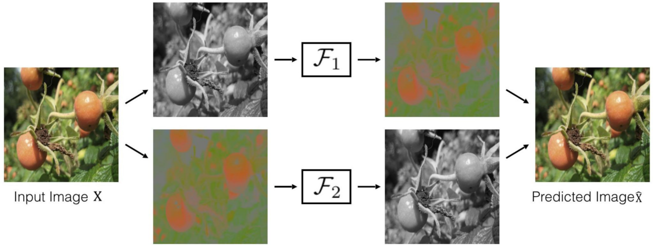


GRASS, CAT, TREE, SKY

Semantic Segmentation

# Supervised vs Unsupervised Learning

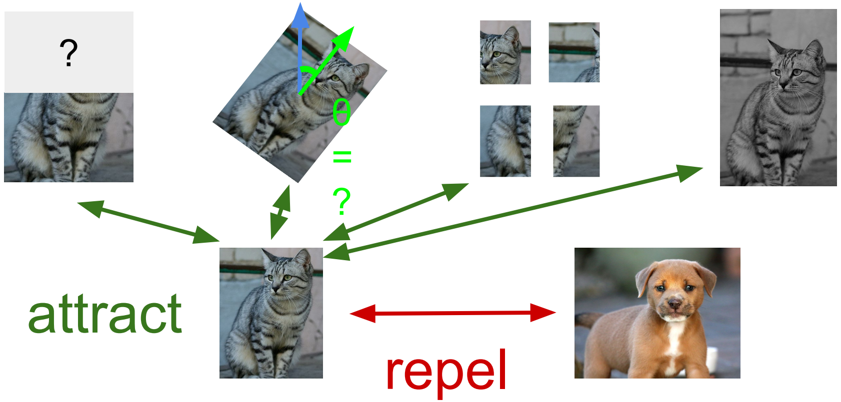
**Self-Supervised Learning**




**Data**: (x, y)
x is data, y is a proxy label

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

**Examples**: Inpainting, colorization, contrastive learning.



# 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, feature learning, density estimation, etc.

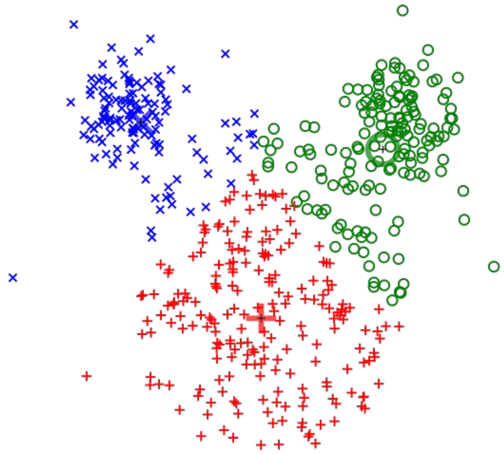
# Supervised vs Unsupervised Learning

**Unsupervised Learning**

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

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

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



K-means clustering

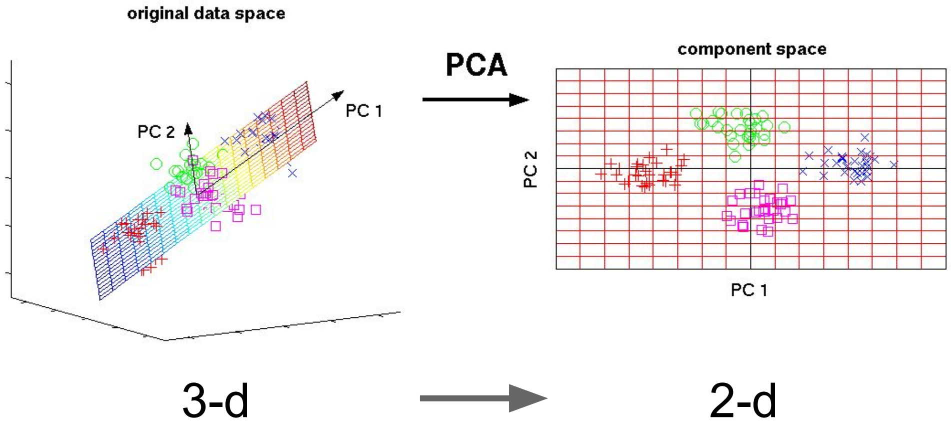
# Supervised vs Unsupervised Learning

**Unsupervised Learning**

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

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

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




Principal Component Analysis
(Dimensionality reduction)

# Supervised vs Unsupervised Learning

**Unsupervised Learning**

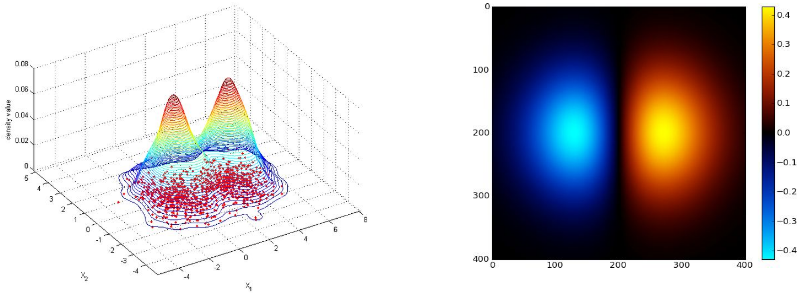
**Data**: x
Just data, no labels!

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

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



1-d density estimation



2-d density estimation

Modeling p(x)

# 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, density estimation, etc.

# A probabilistic interpretation of modeling

**Data: x, Label: y**

 **, cat**

**Density Function** p(x) assigns a positive number to each possible x; higher numbers mean x is more likely. $\int_X p(x)dx = 1$

Probabilities across all values of x sum up to 1

# A probabilistic interpretation of modeling

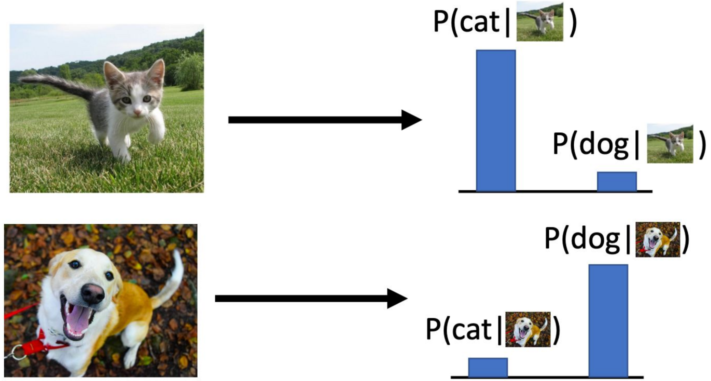
**Data: x, Label: y**

 **, cat**

**Density Function** p(x) assigns a positive number to each possible x; higher numbers mean x is more likely. $\int_X p(x)dx = 1$

Probabilities across all values of x sum up to 1

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




Sum of p(y | x) = 1 across C classes

# A probabilistic interpretation of modeling

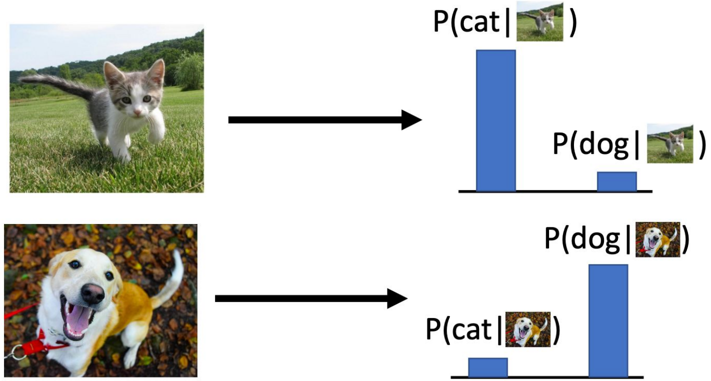
**Data: x, Label: y**


**, cat**

**Density Function** p(x) assigns a positive number to each possible x; higher numbers mean x is more likely. $\int_X p(x)dx = 1$

Probabilities across all values of x sum up to 1

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




Sum of p(y | x) = 1 across C classes

Bias term of last linear layer learns p(y)

# A probabilistic interpretation of modeling

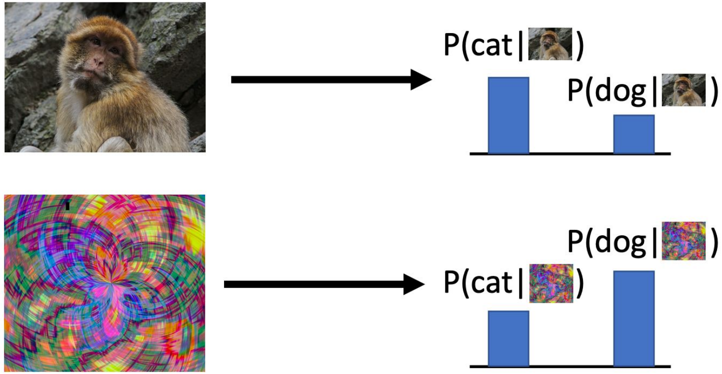
**Data: x, Label: y**

**, cat**

**Density Function** p(x) assigns a positive number to each possible x; higher numbers mean x is more likely. $\int_X p(x)dx = 1$

Probabilities across all values of x sum up to 1

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




If the images contain classes not part of the vocabulary, outputs are uninterpretable.

# A probabilistic interpretation of modeling

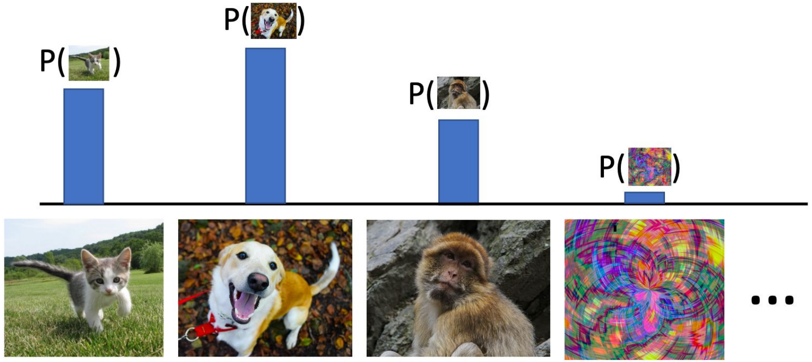
**Data: x, Label: y**

 **, cat**

**Density Function** p(x) assigns a positive number to each possible x; higher numbers mean x is more likely. $\int_X p(x)dx = 1$

Probabilities across all values of x sum up to 1

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




All possible images compete with each other for probability mass

Is a dog more likely to sit or stand? How about 3-legged dog vs 3-armed monkey?

# A probabilistic interpretation of modeling

**Data: x, Label: y**

, **cat**

**Density Function** p(x) assigns a positive number to each possible x; higher numbers mean x is more likely. $\int_X p(x)dx = 1$

Probabilities across all values of x sum up to 1

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

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

Recall Bayes' Rule:

# A probabilistic interpretation of modeling

**Data: x, Label: y**

 **, cat**

**Density Function** p(x) assigns a positive number to each possible x; higher numbers mean x is more likely. $\int_X p(x)dx = 1$

Probabilities across all values of x sum up to 1

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

Discriminative Model

(Unconditional) Generative Model

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

Conditional Generative Model

Prior over labels

We can build a conditional generative model from other components!

# Putting them together:

**Data: x, Label: y**

 **, cat**

**Density Function** p(x) assigns a positive number to each possible x; higher numbers mean x is more likely. $\int_X p(x)dx = 1$

Probabilities across all values of x sum up to 1

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

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

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

# Applications for Generative Models

1. Assign labels to data
2. Feature learning (with labels)

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

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

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

# Applications for Generative Models

1. Assign labels to data
2. Feature learning (with labels)

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

1. Detect outliers
2. Feature learning (without labels)
3. Sample to generate new data

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

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

# Applications for Generative Models

1. Assign labels to data
2. Feature learning (with labels)

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

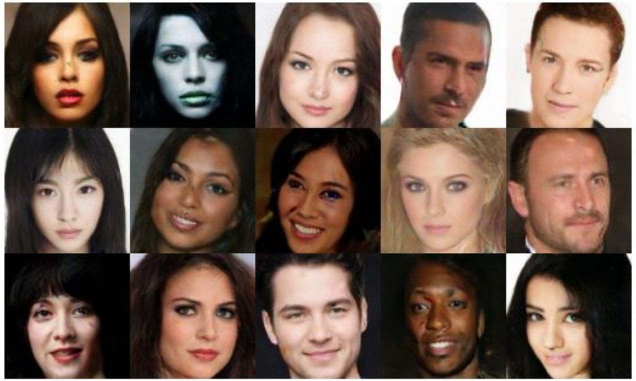
1. Detect outliers
2. Feature learning (without labels)
3. Sample to generate new data

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

1. Assign labels, rejecting outliers!
2. Generate new data conditioned on input labels

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

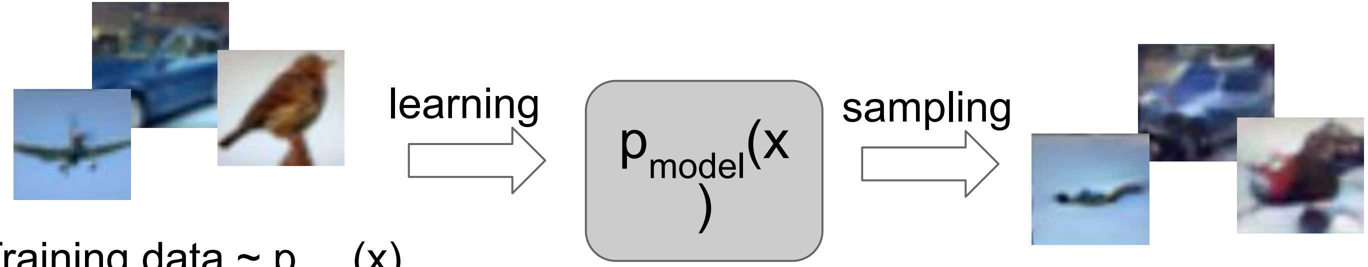
# Why Generative Models?

- 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 ...

FIgures from L-R are copyright: (1) Alec Radford et al. 2016; (2) Phillip Isola et al. 2017. Reproduced with authors permission (3) BAIR Blog.

# The two objectives of generative models

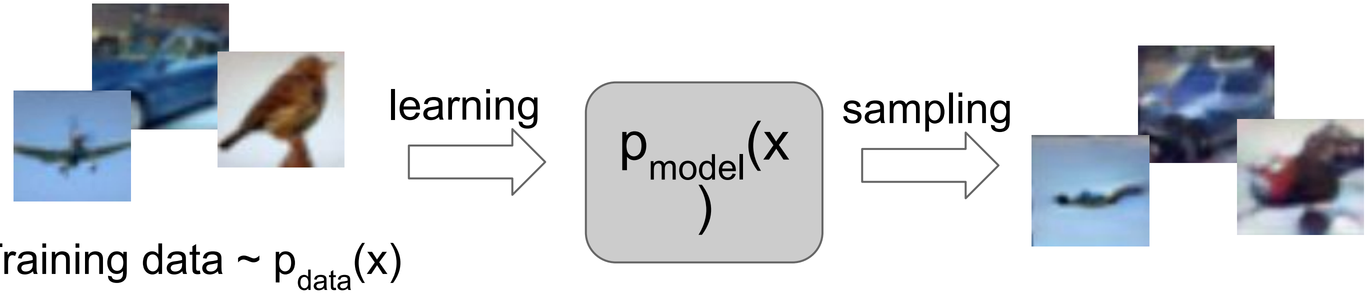



Training data ~ $p_{data}(x)$

**Objectives**:
1. Learn $p_{model}(x)$ that approximates $p_{data}(x)$
2. Sampling new x from $p_{model}(x)$

# Generative Modeling

Given training data, generate new samples from same distribution




Formulate as density estimation problems**:**

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

# Taxonomy of Generative Models

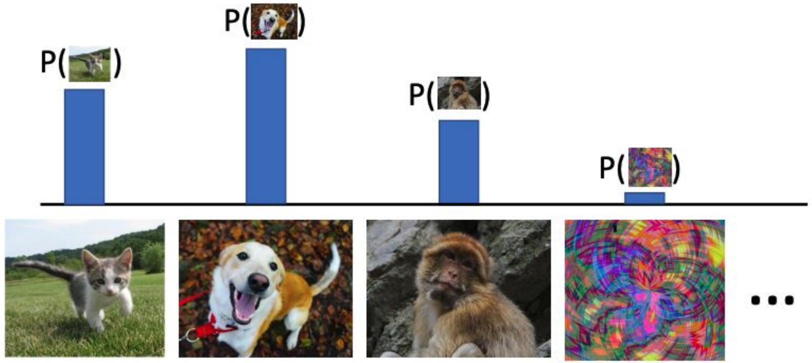
**Generative models**

Model can compute p(x)

Explicit density

Model does not compute p(x)
But can sample from p(x)

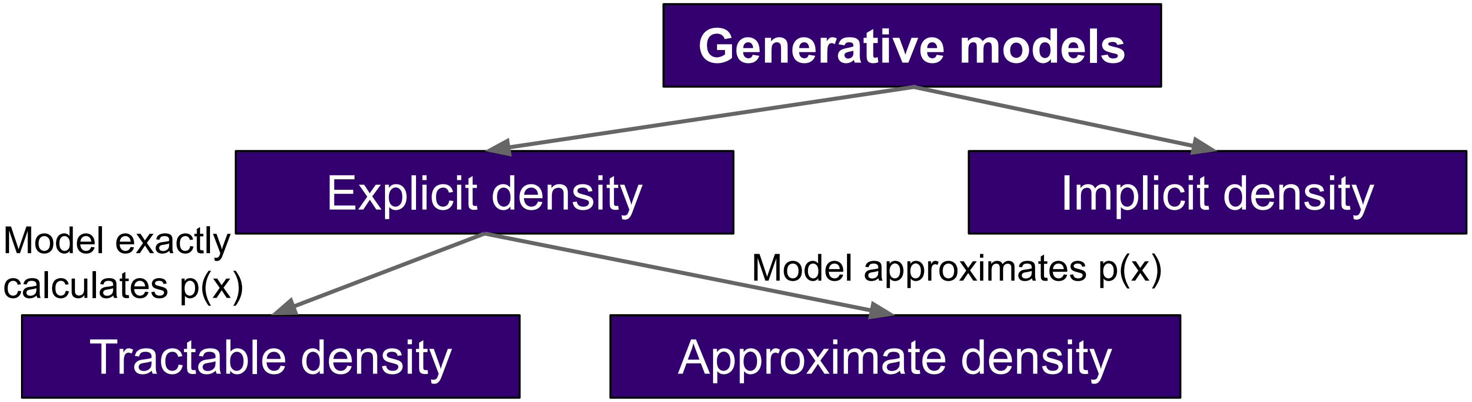
Implicit density




p(x) measures the likelihood of an image

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

# Taxonomy of Generative Models




Fully Visible Belief Nets

- Autoregressive
- NADE
- MADE
- NICE / RealNVP
- Glow
- Ffjord

# Taxonomy of Generative Models

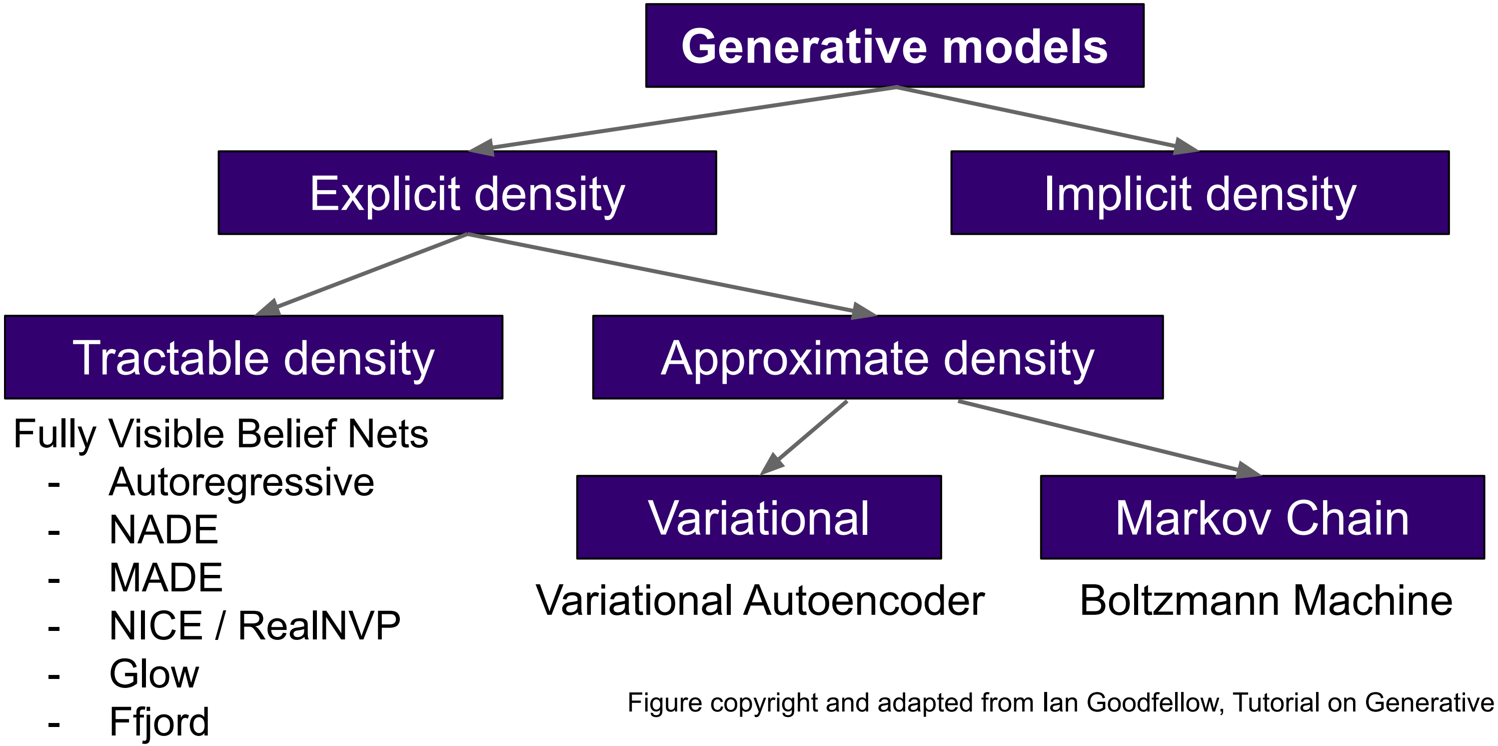



Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

# Taxonomy of Generative Models

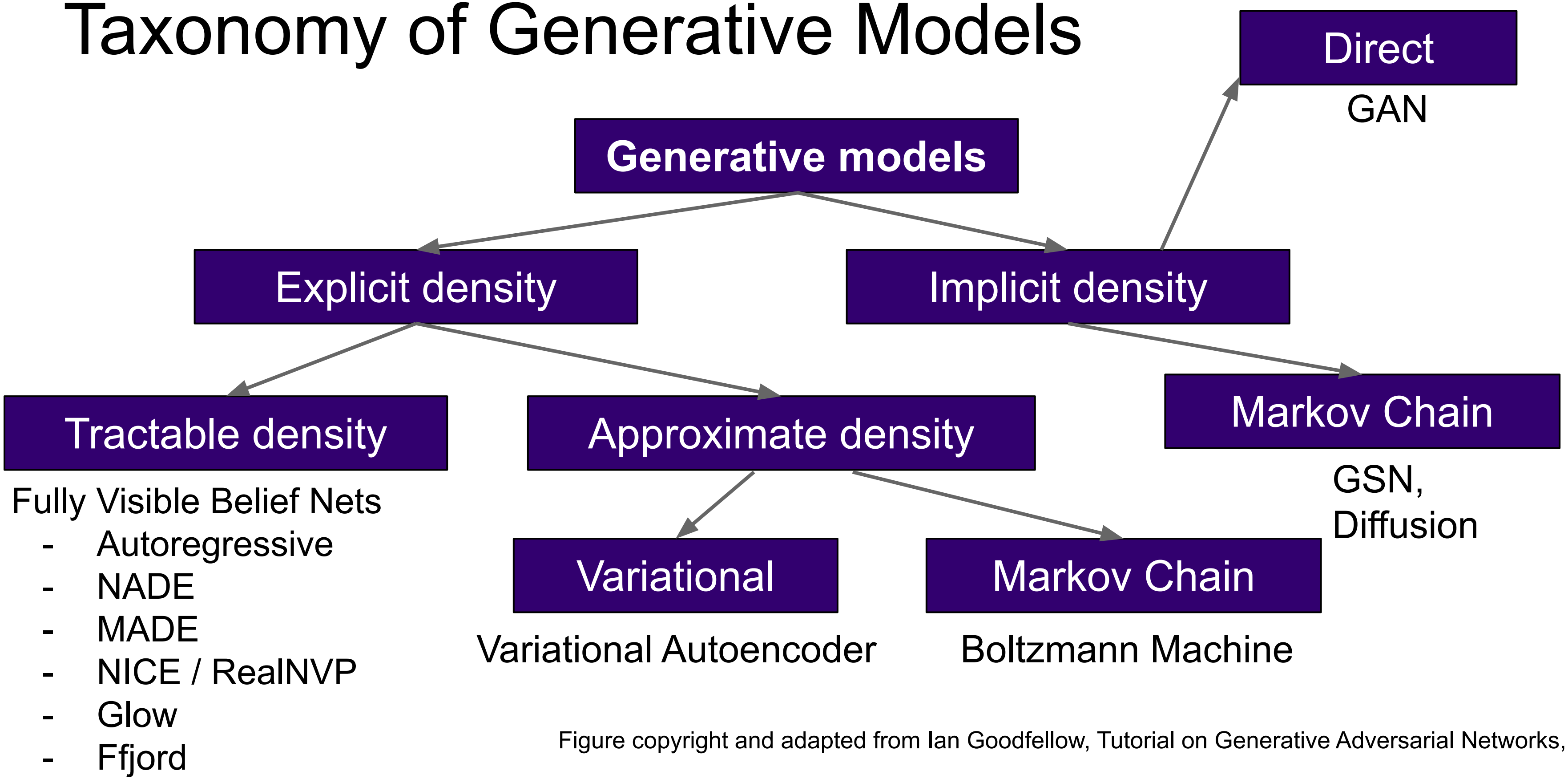



Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

# Q: Where would you place GPT4?

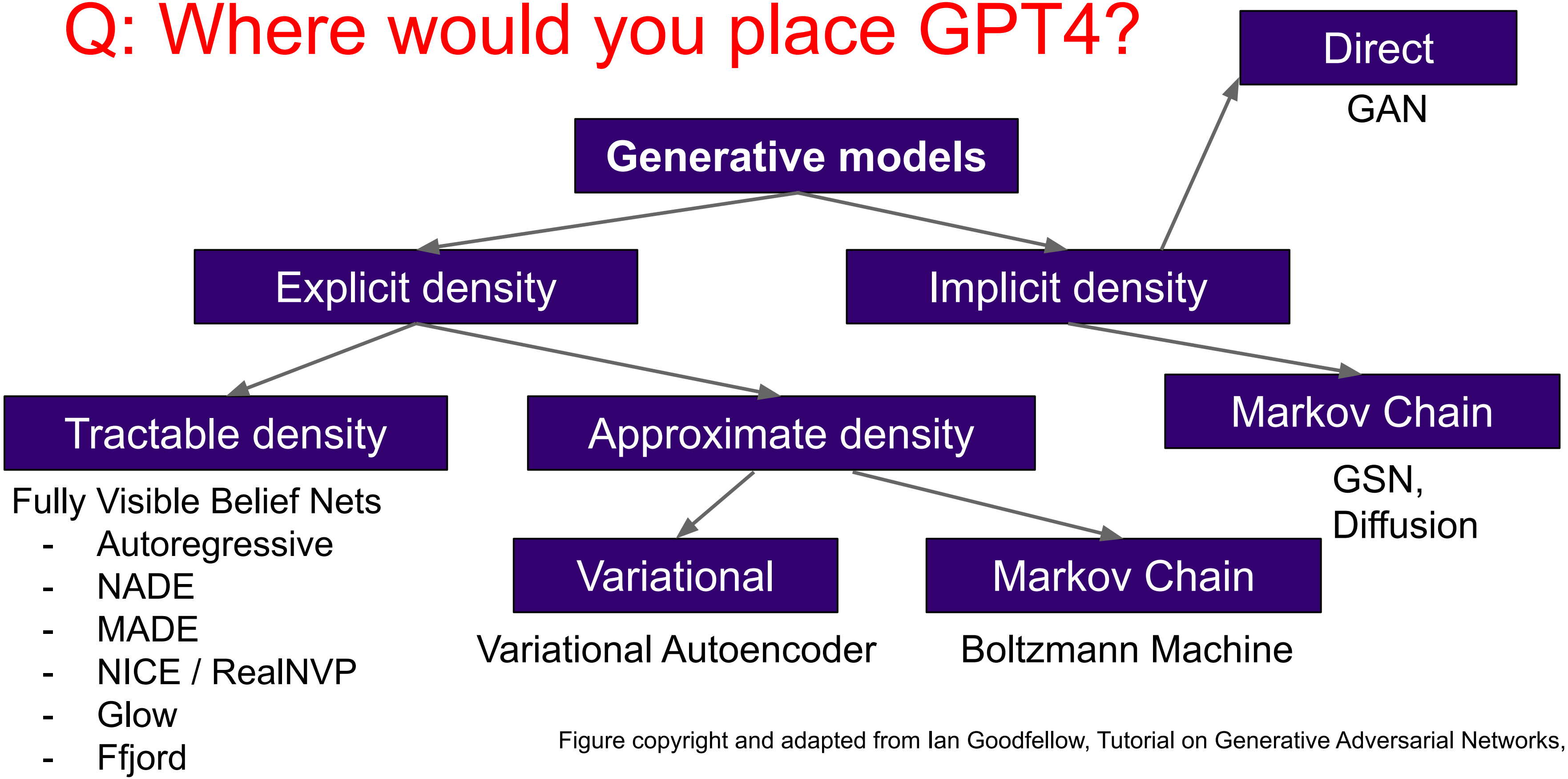



Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

# Taxonomy of Generative Models

Today: discuss 2 types of generative models today
More next lecture!

**Generative models**

- Explicit density
  - Tractable density
    - Fully Visible Belief Nets
      - Autoregressive
      - NADE
      - MADE
      - NICE / RealNVP
      - Glow
      - Ffjord
  - Approximate density
    - Variational
      - Variational Autoencoder
    - Markov Chain
      - Boltzmann Machine
- Implicit density
  - Direct
    - GAN
  - Markov Chain
    - GSN, Diffusion

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

# Explicit density 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 $x^{(1)}, x^{(2)}, \dots x^{(N)}$, train the model by solving:

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

Maximize probability of training data
(Maximum likelihood estimation)

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

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 (backprop)

# Autorgressive models

(PixelRNN and PixelCNN)

# Explicit density: autoregressive models

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

Assume that x is made up of multiple parts: $x = (x_1, x_2, x_3, \ldots, x_T)$

For example, images are made up of pixels, language is made up of words/characters/tokens

# Explicit density: autoregressive models

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

Assume that x is made up of multiple parts: $x = (x_1, x_2, x_3, \ldots, x_T)$

For example, images are made up of pixels, language is made up of words/characters/tokens

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

Likelihood of image x

Joint likelihood of each part in the data

# Explicit density: autoregressive models

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

Assume that x is made up of multiple parts: $x = (x_1, x_2, x_3, \ldots, x_T)$

For example, images are made up of pixels, language is made up of words/characters/tokens

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

Break down probability using the chain rule

# Explicit density: autoregressive models

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

Assume that x is made up of multiple parts: $x = (x_1, x_2, x_3, \ldots, x_T)$

For example, images are made up of pixels, language is made up of words/characters/tokens

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

Break down probability using the chain rule

Probability of the next subpart given all the previous subparts

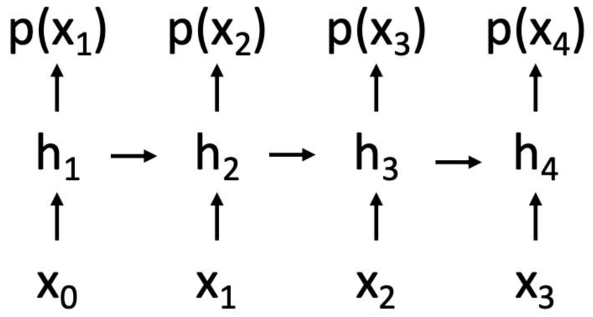
# Explicit density: autoregressive models

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

Assume that x is made up of multiple parts: $x = (x_1, x_2, x_3, \ldots, x_T)$

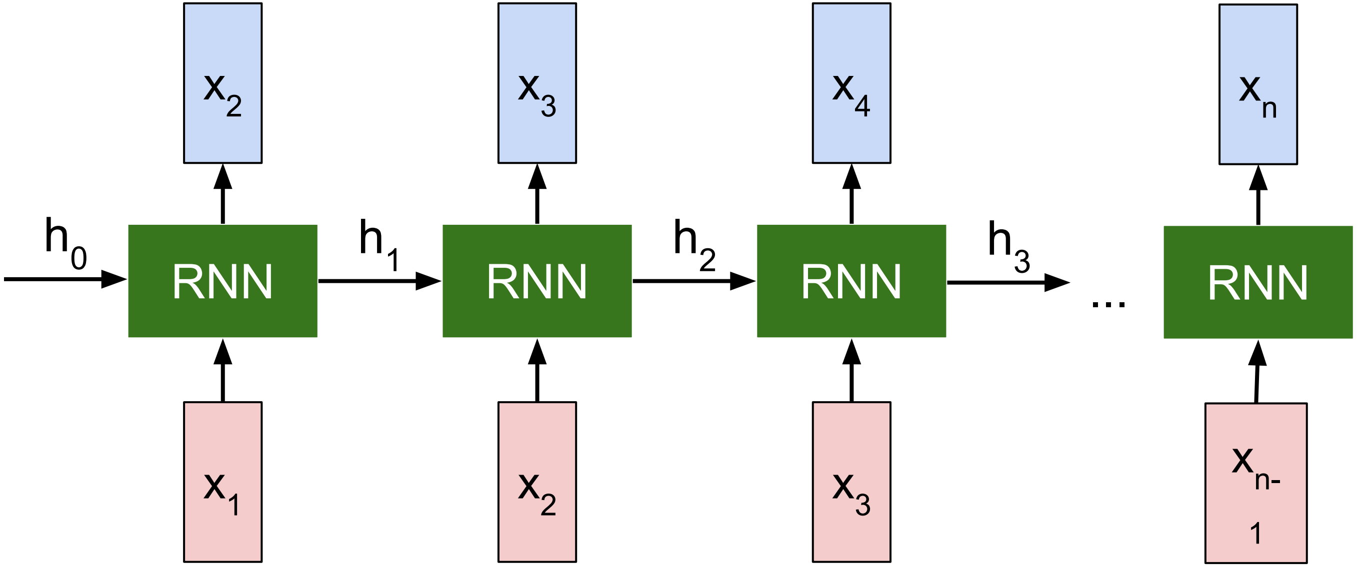
For example, images are made up of pixels, language is made up of words/characters/tokens

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




Language modeling with RNNs is an autoregressive model

# We assume hidden state encodes all prior information $x_0, \ldots, x_{t-1}$

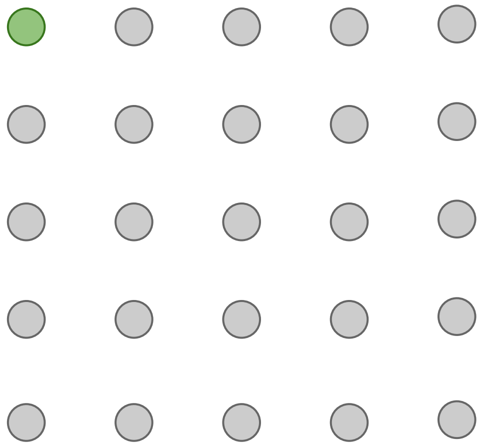



$$p(x_i | x_1, \ldots, x_{i-1})$$

# PixelRNN - autoregressive image generation

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

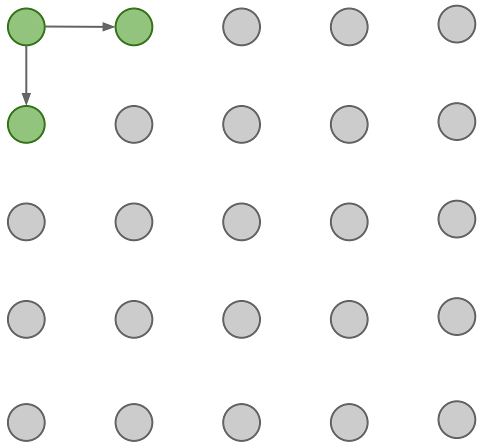


*[van der Oord et al. 2016]*

# PixelRNN - autoregressive image generation

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

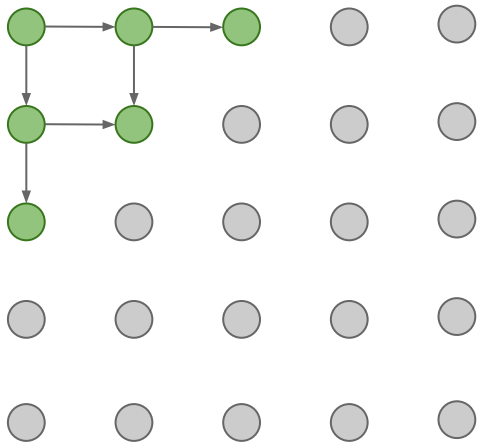


*[van der Oord et al. 2016]*

# PixelRNN - autoregressive image generation

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)



*[van der Oord et al. 2016]*

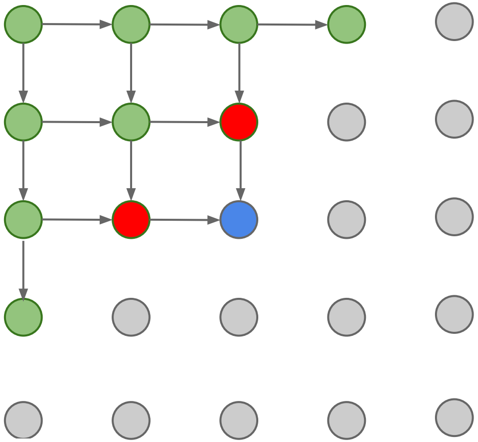
# PixelRNN - autoregressive image generation

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

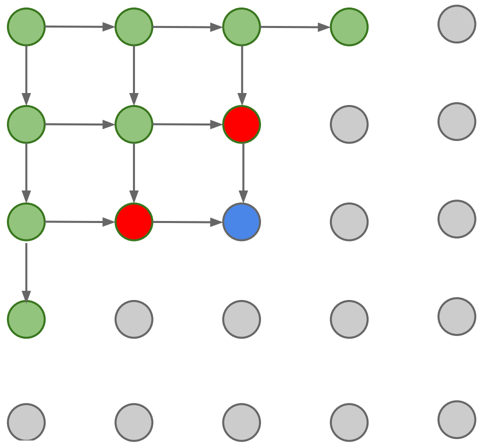
Hidden state for each pixel is conditioned on the hidden states and RGB values from the left and from above

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



*[van der Oord et al. 2016]*

# PixelRNN - autoregressive image generation



Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

Hidden state for each pixel is conditioned on the hidden states and RGB values from the left and from above

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

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

*[van der Oord et al. 2016]*

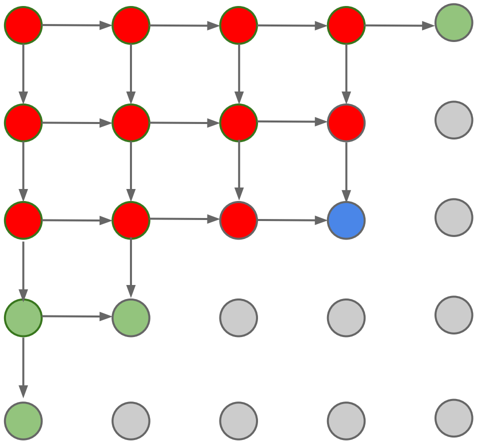
# PixelRNN - autoregressive image generation

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!

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

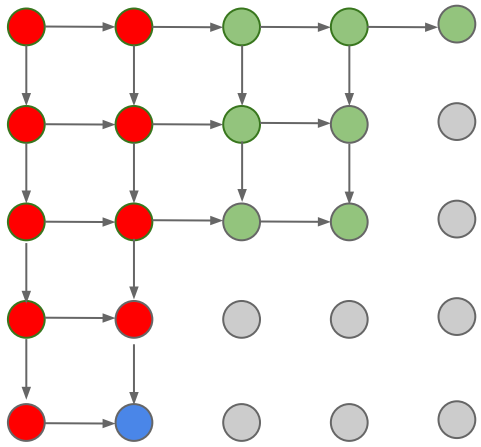


[van der Oord et al. 2016]

# PixelRNN - autoregressive image generation

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!

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

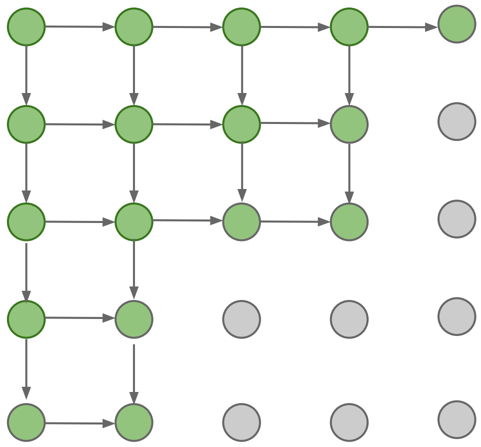
[van der Oord et al. 2016]

# PixelRNN - autoregressive image generation

Generate image pixels starting from corner

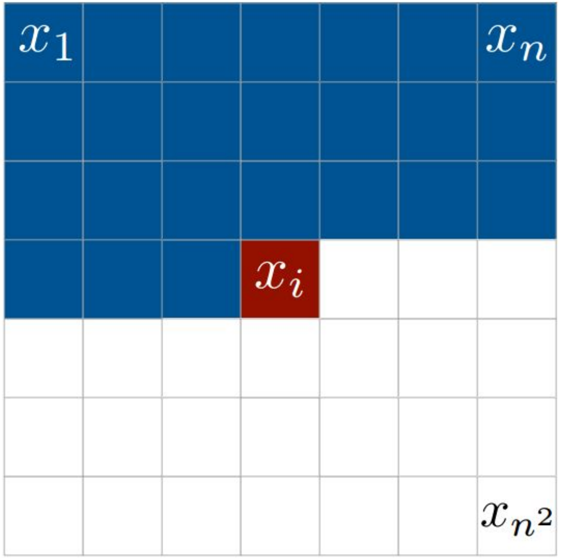
Dependency on previous pixels modeled using an RNN (LSTM)

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



[van der Oord et al. 2016]

Q: Where else have we seen a similar processing of input images by iterating over patches of the image?



# PixelCNN - improvements to training time

Still generate image pixels starting from corner

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

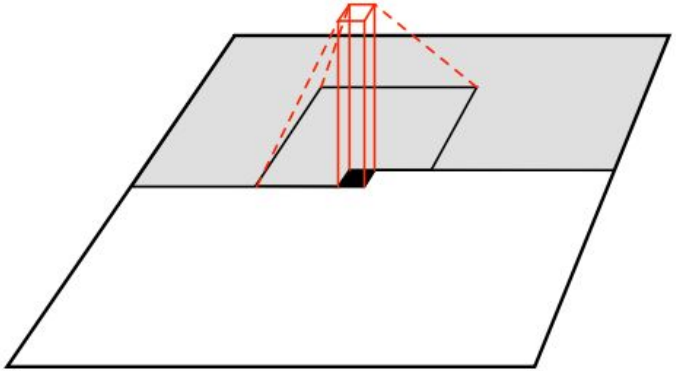


Figure copyright van der Oord et al., 2016. Reproduced with permission.

*[van der Oord et al. 2016]*

# PixelCNN *[van der Oord et al. 2016]*

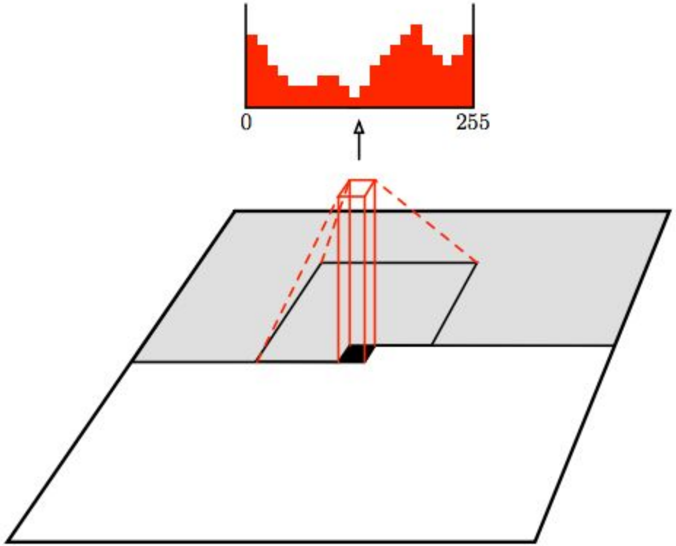
Still generate image pixels starting from corner

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

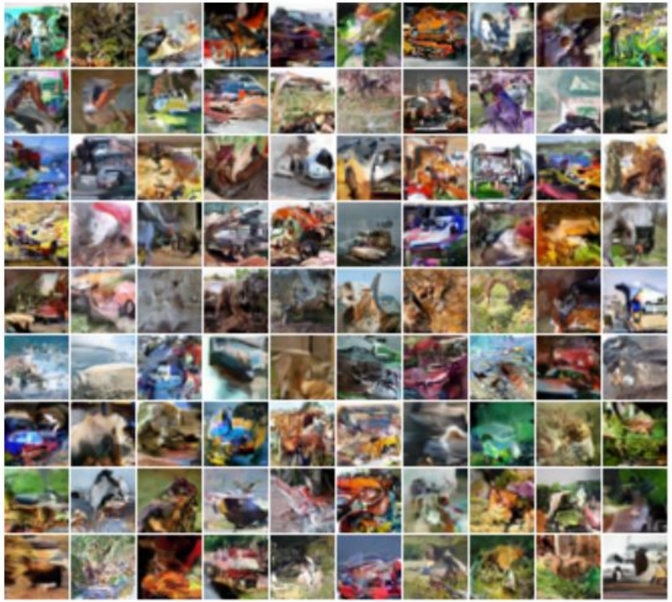
Training is faster than PixelRNN
(can parallelize convolutions since context region values known from training images)

Generation is still slow:
For a 32x32 image, we need to do forward passes of the network 1024 times for a single image

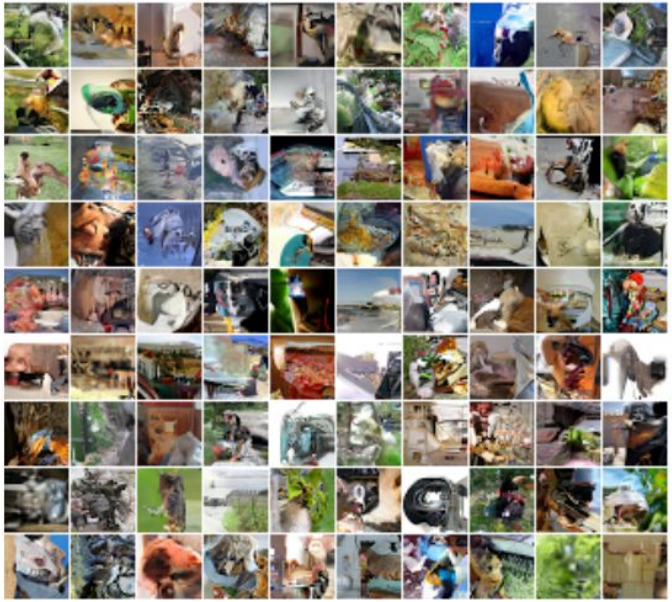
Softmax loss over pixel values at every location



# Generation Samples



32x32 CIFAR-10



32x32 ImageNet

# PixelRNN and PixelCNN

Pros:
- Can explicitly compute likelihood p(x)
- Easy to optimize
- 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++)

# Taxonomy of Generative Models

Direct

GAN

**Generative models**

Explicit density

Implicit density

Tractable density

Approximate density

Markov Chain

GSN

Fully Visible Belief Nets

- Autoregressive
- NADE
- MADE
- NICE / RealNVP
- Glow
- Ffjord

Variational

Variational Autoencoder

Markov Chain

Boltzmann Machine

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

# So far...

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

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

# So far...

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

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

Variational Autoencoders (VAEs) define an 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

# So far...

PixelCNNs define tractable density function, optimize likelihood of training data:

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

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

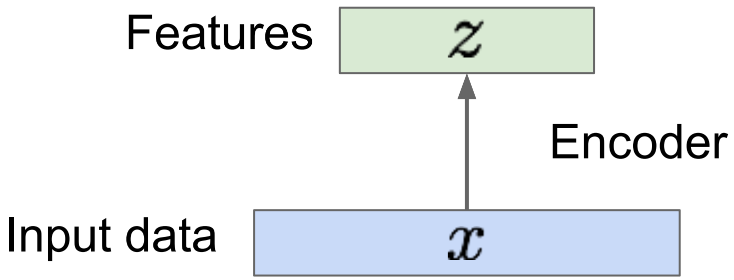
Why latent z?

# Variational Autoencoders (VAE)

# Some background first: Autoencoders

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

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

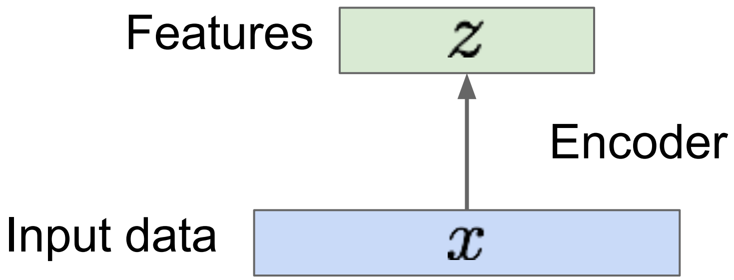


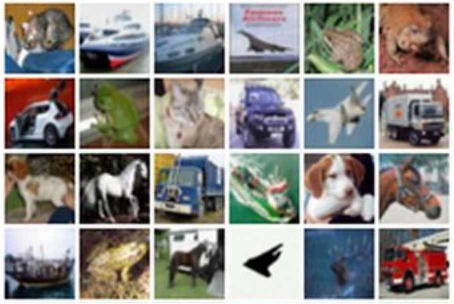

# Some background first: Autoencoders

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

**z** usually smaller than **x**
(dimensionality reduction)

Q: Why dimensionality reduction?



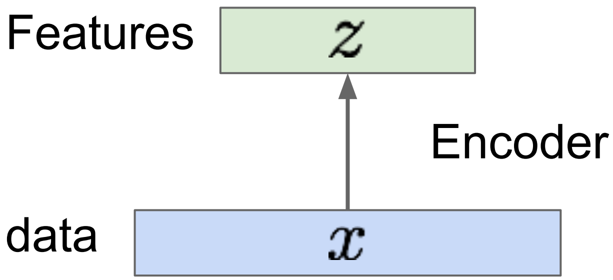
# Some background first: Autoencoders

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

**z** usually smaller than **x**
(dimensionality reduction)

Q: Why dimensionality reduction?

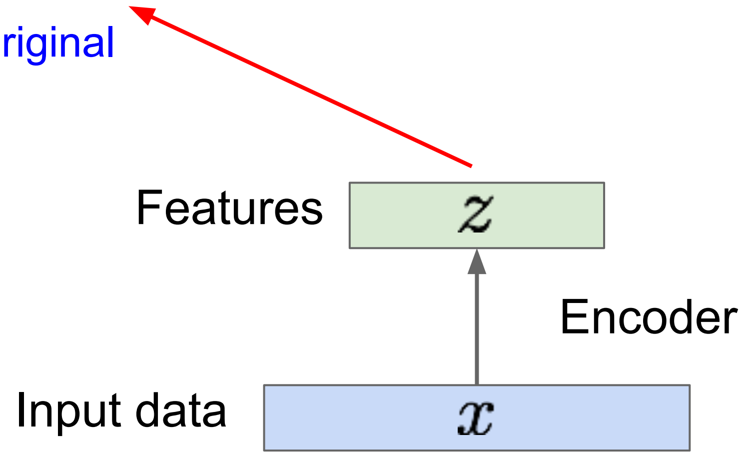
A: Want features to capture meaningful factors of variation in data




# Some background first: Autoencoders

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

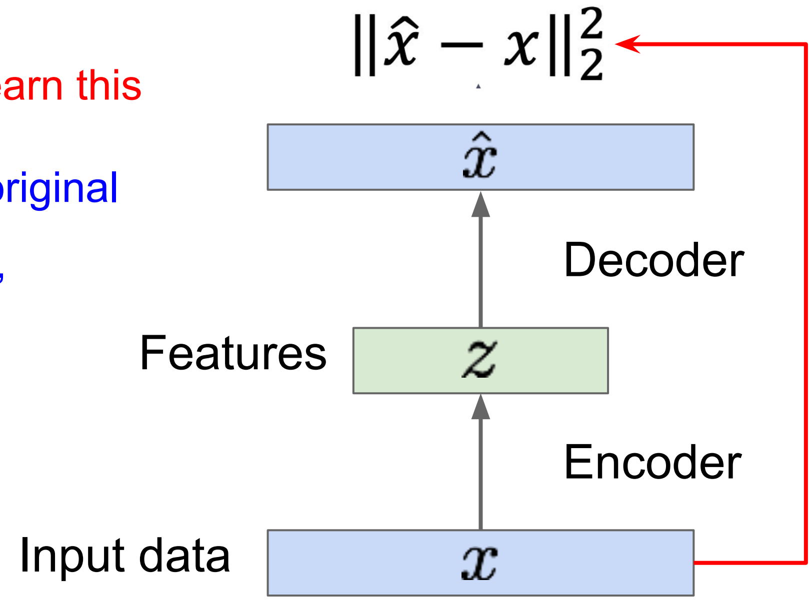
Q. How do we learn this z?
A. Reconstruct original input data: "**Autoencoding**"




# Some background first: Autoencoders

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

Q. How do we learn this z?
A. Reconstruct original input data: "**Autoencoding**"

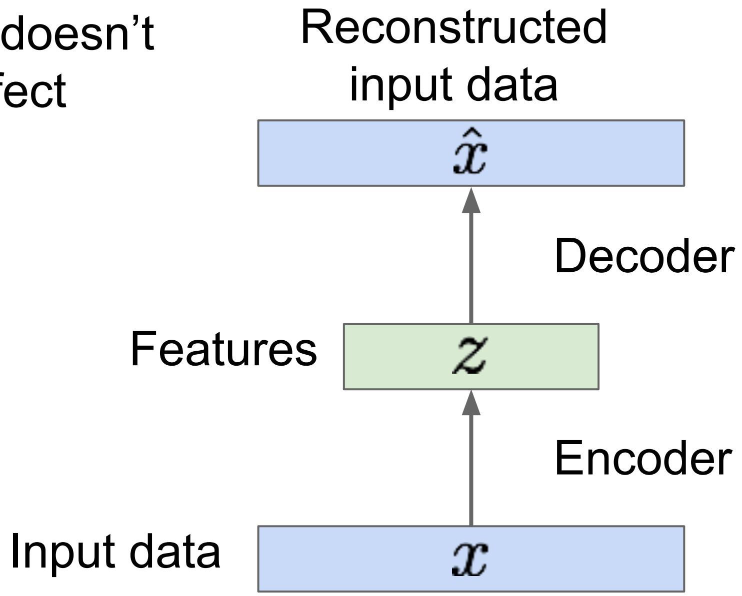


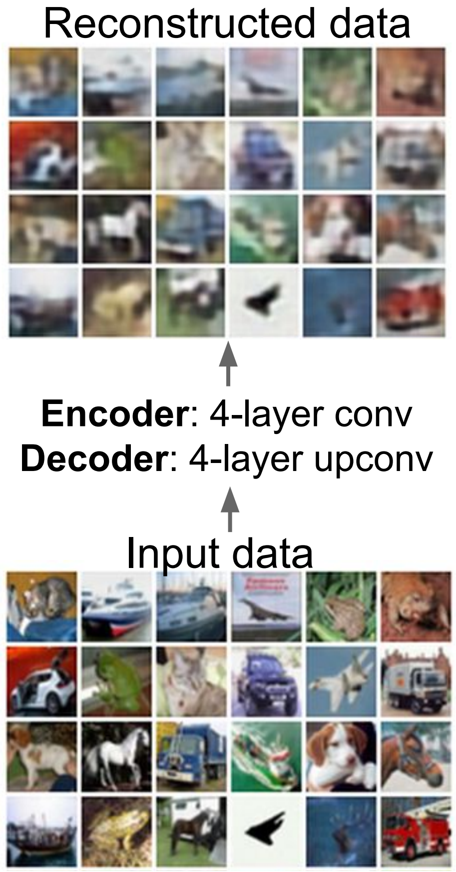

**Learning objective:** reconstruct the image and use l2 loss.

No labels are necessary!!

# Some background first: Autoencoders

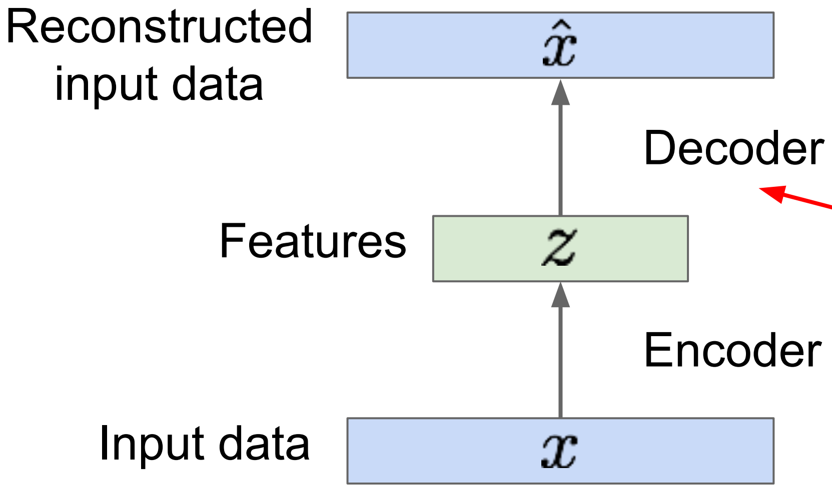
Images reconstructed are blurry because z is smaller and doesn't save pixel-perfect information



# Some background first: Autoencoders

Similar to the self-supervised feature learning + transfer to downstream tasks

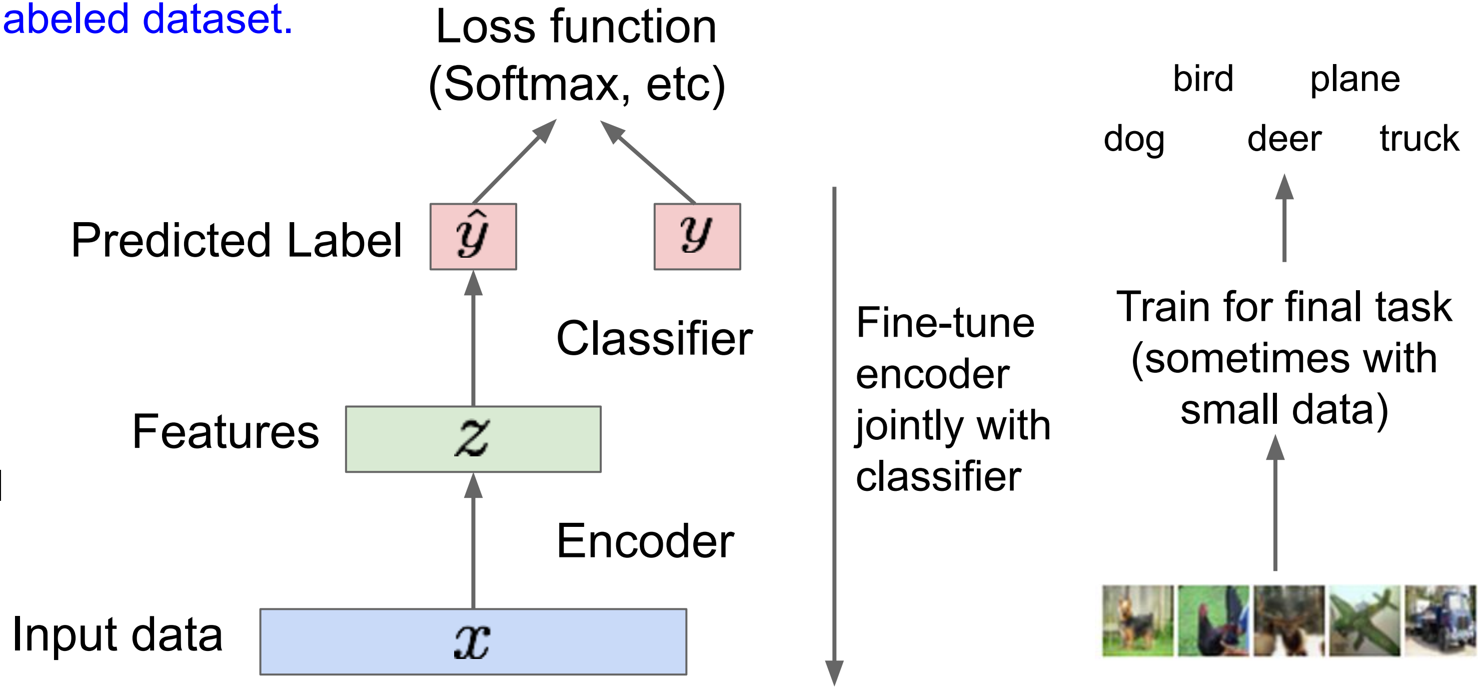



After training, throw away decoder

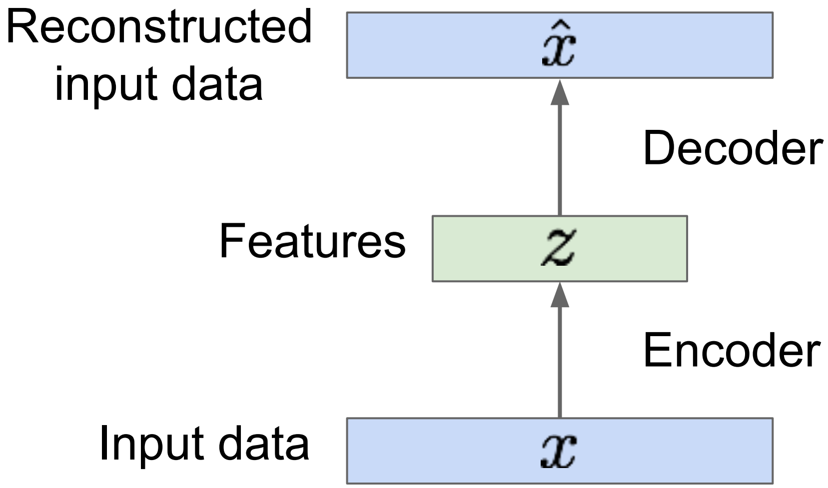
# Some background first: Autoencoders

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

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



# Some background first: 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!

# 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**

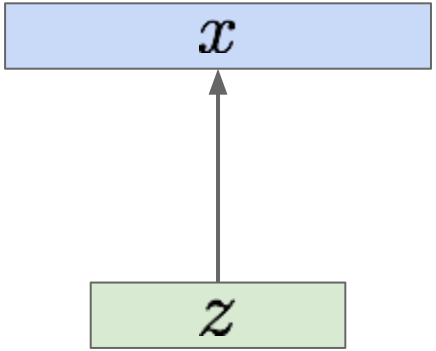
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

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




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

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

# Variational Autoencoders

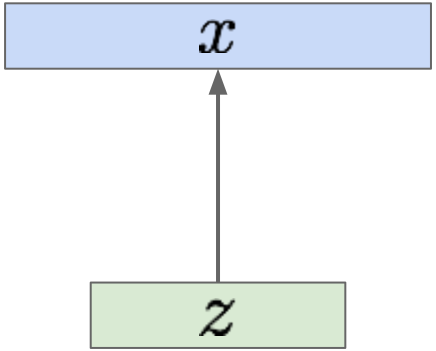
Probabilistic spin on autoencoders - will let us sample from the model to generate data!

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

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

Sample from true conditional

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




Sample from true prior

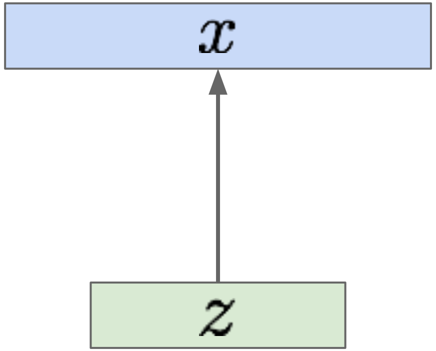
$z^{(i)} \sim p_{\theta^*}(z)$

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

# Variational Autoencoders

We want to estimate the parameters $\theta^*$ given training real data x.

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




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

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

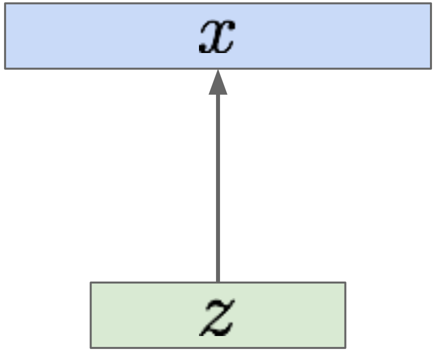
# Variational Autoencoders

Sample from
true conditional

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

Sample from
true prior

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




We want to estimate the parameters $\theta^*$ given training real data x.

How should we represent this model?

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

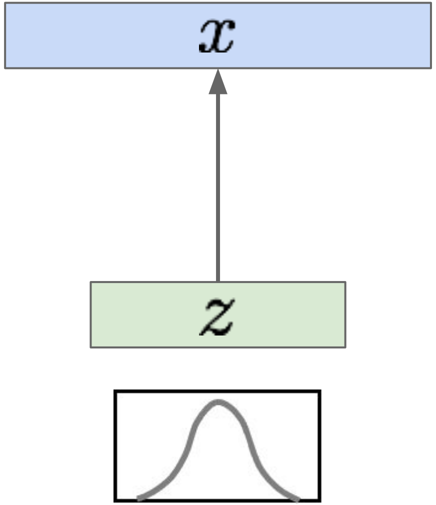
# Variational Autoencoders

Sample from true conditional

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

Sample from true prior

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




We want to estimate the parameters $\theta^*$ given training real data x.

How should we represent this model?

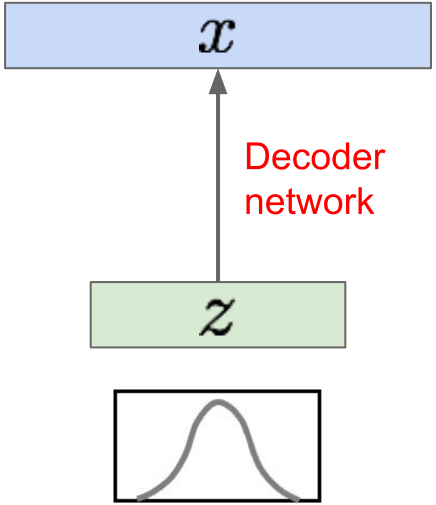
Choose prior p(z) to be simple, e.g. Gaussian. Reasonable for latent attributes, e.g. pose, how much smile.

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

# Variational Autoencoders

Sample from true conditional

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




Sample from true prior

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

We want to estimate the parameters $\theta^*$ given training real data x.

How should we represent this model?

Choose prior p(z) to be simple, e.g. Gaussian. Reasonable for latent attributes, e.g. pose, how much smile.

Conditional p(x|z) is complex (generates image) => represent with neural network

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

# Variational Autoencoders

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

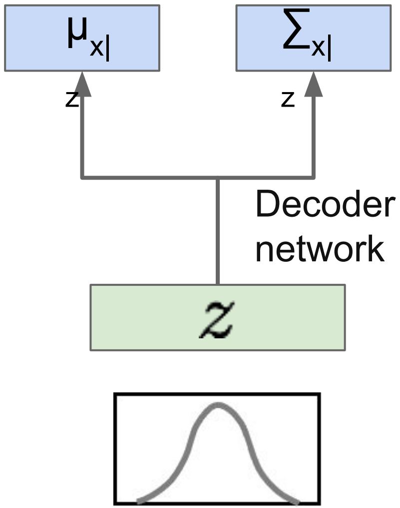
We want to estimate the parameters $\theta^*$ given training real data x.

Sample from true conditional

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

Sample from true prior

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

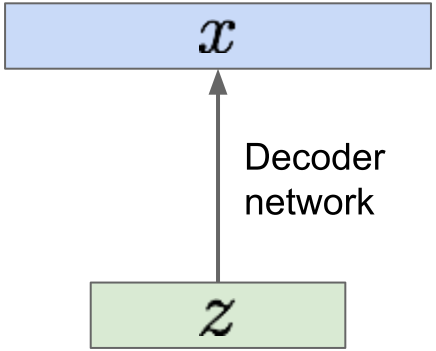



Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

# Variational Autoencoders

Sample from true conditional

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




Sample from true prior

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

We want to estimate the parameters $\theta^*$ given training real data x.

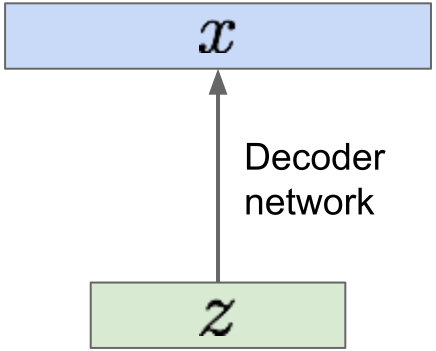
How to train the model?

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

# Variational Autoencoders

Sample from true conditional

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




Sample from true prior

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

We want to estimate the parameters $\theta^*$ given training real data x.

How to train the model?

Learn model parameters to maximize likelihood of training data

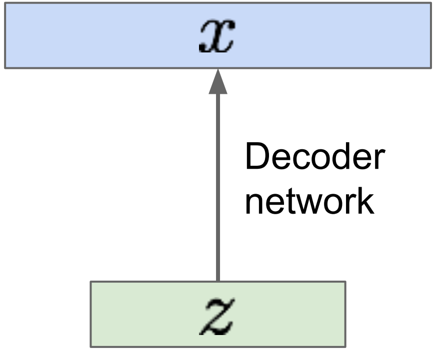
$$p_\theta(x) = \int p_\theta(z) p_\theta(x|z) dz$$

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

# Variational Autoencoders

Sample from true conditional

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




Sample from true prior

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

We want to estimate the parameters $\theta^*$ given training real 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?

Intractable! Impossible to iterate over all z

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

# Variational Autoencoders: Intractability

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

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

# Variational Autoencoders: Intractability

✔

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

Simple Gaussian prior

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

# Variational Autoencoders: Intractability

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

Decoder neural network

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

# Variational Autoencoders: Intractability

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

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

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

# Variational Autoencoders: Intractability

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

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

# Variational Autoencoders: Intractability

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

Another idea: $p_\theta(x) = \dfrac{p_\theta(x \mid z) p_\theta(z)}{p_\theta(z \mid x)}$ ← Use Bayes rule

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

# Variational Autoencoders: Intractability

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

Another idea: $p_\theta(x) = \dfrac{\boxed{p_\theta(x \mid z)p_\theta(z)}}{p_\theta(z \mid x)}$

We know how to calculate these

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

# Variational Autoencoders: Intractability

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

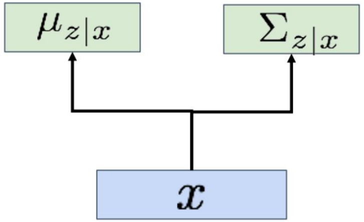
Another idea: $p_\theta(x) = \dfrac{p_\theta(x \mid z) p_\theta(z)}{\boxed{p_\theta(z \mid x)}}$

But how do you calculate this?

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

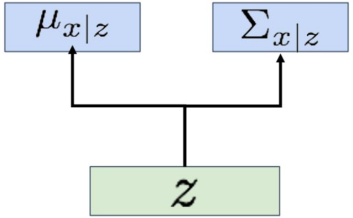
**Encoder Network**

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




**Decoder Network**

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




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

# Variational Autoencoders: Intractability

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

Another idea: $p_\theta(x) = \dfrac{p_\theta(x \mid z) p_\theta(z)}{\boxed{p_\theta(z \mid x)}}$

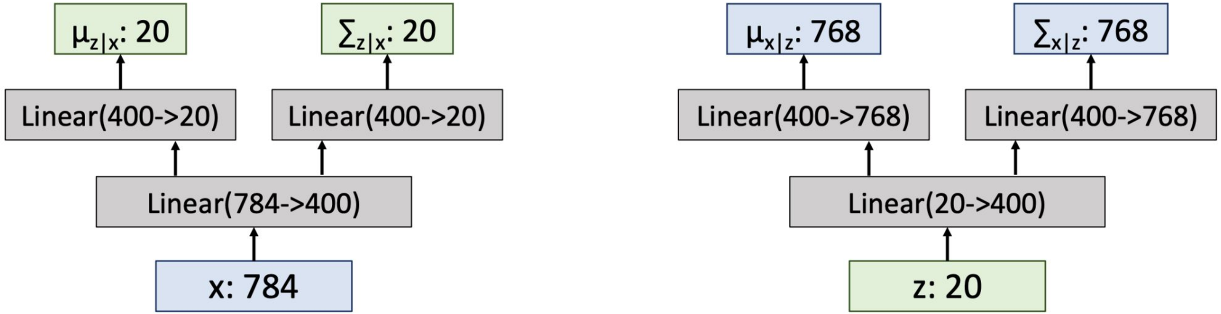
**x:** 28x28 image = 784-dim vector
**z:** 20-dim vector

**Encoder Network**

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

**Decoder Network**

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




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

# Variational Autoencoders

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

Using this approximation, we can derive a lower bound on the data likelihood p(x), making it tractable, therefore, possible to optimize.

# Variational Autoencoders

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

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

# Variational Autoencoders

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

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

# Variational Autoencoders

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

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

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

# Variational Autoencoders

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

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

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

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

# Variational Autoencoders

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

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

# Variational Autoencoders

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

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

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

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

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

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

# Variational Autoencoders

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

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!

# Variational Autoencoders

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

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 >= 0.

# Variational Autoencoders

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

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

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

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

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

We want to maximize the data likelihood

Decoder network gives $p_\theta(x|z)$, can compute estimate of this term 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 >= 0.

# Variational Autoencoders

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

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

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

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

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

We want to maximize the data likelihood

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

# Variational Autoencoders

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

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

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

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

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

Decoder:
reconstruct
the input data

Encoder:
make approximate
posterior distribution
close to prior

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

# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

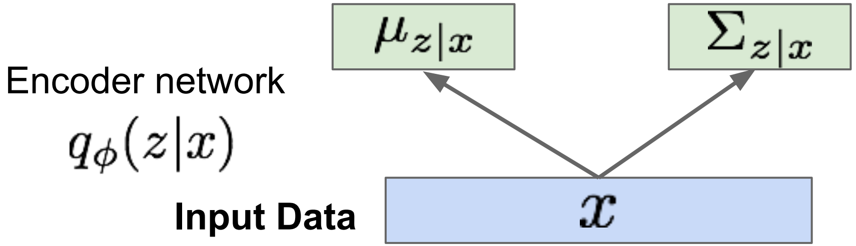
Let's look at computing the KL divergence between the estimated posterior and the prior given some data

**Input Data** $x$

# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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



# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

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

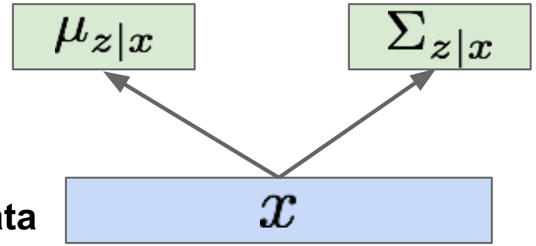
This equation has an analytical solution

Make approximate posterior distribution close to prior

Encoder network

$q_\phi(z|x)$

**Input Data**



# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

Make approximate posterior distribution close to prior

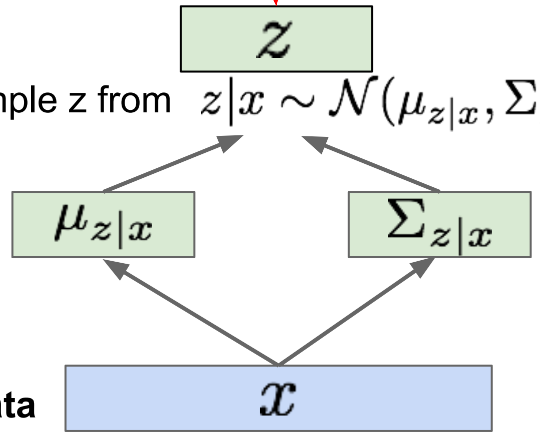
Not part of the computation graph!

Sample z from $z|x \sim \mathcal{N}(\mu_{z|x}, \Sigma_{z|x})$

Encoder network

$q_\phi(z|x)$

**Input Data**



# Variational Autoencoders

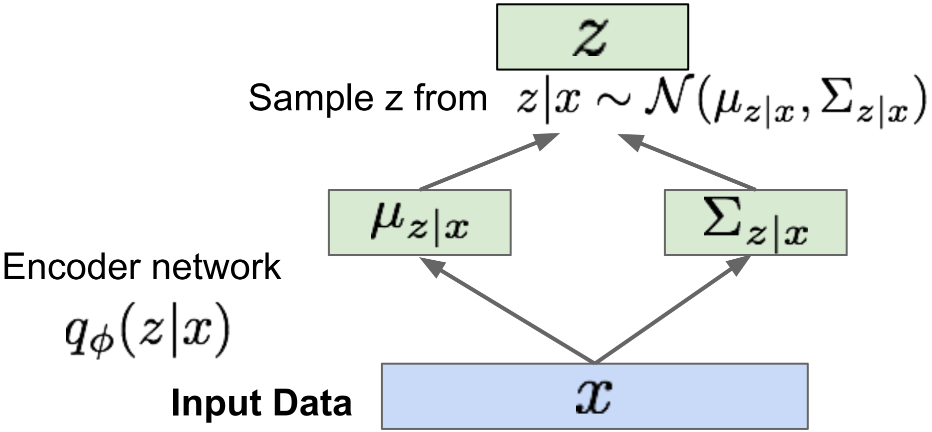
Putting it all together: maximizing the likelihood lower bound

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

Reparameterization trick to make sampling differentiable:

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

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



# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

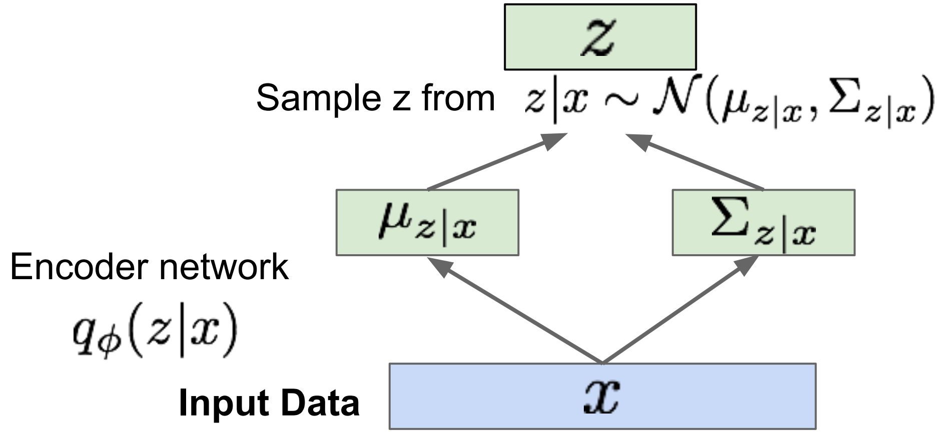
Reparameterization trick to make sampling differentiable:

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

Input to the graph

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

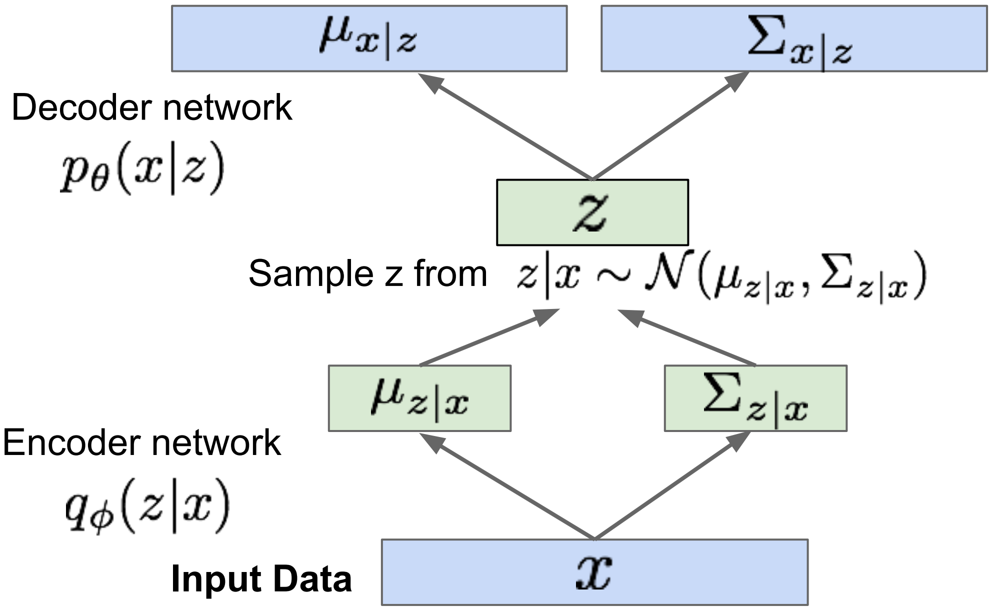
Part of computation graph



# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

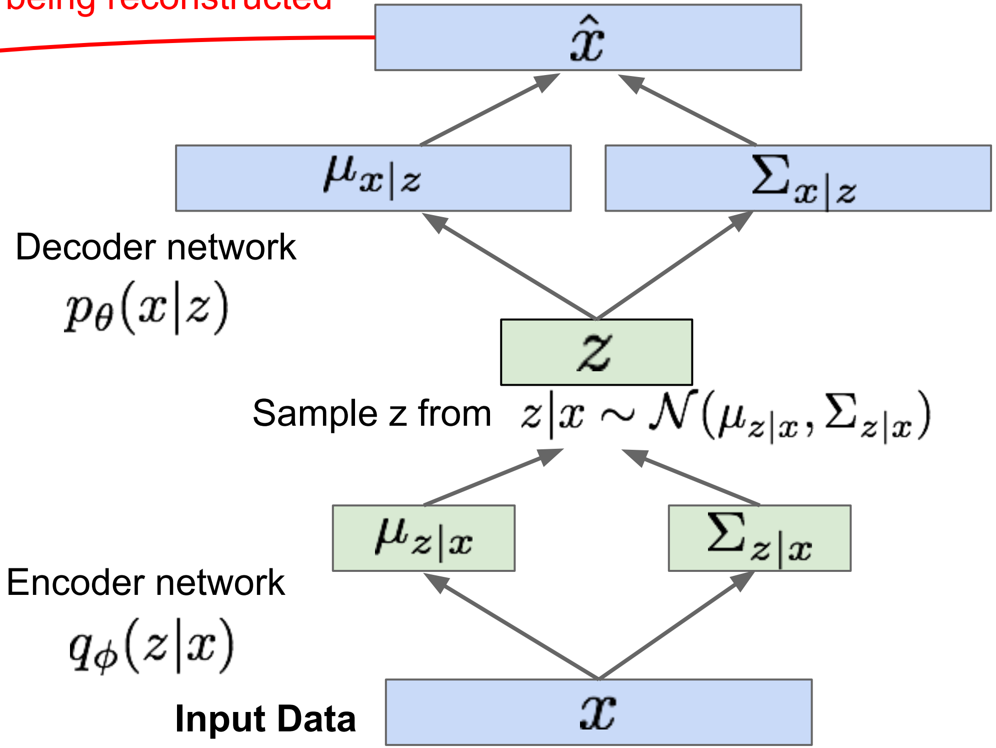


# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

Maximize likelihood of original input being reconstructed

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

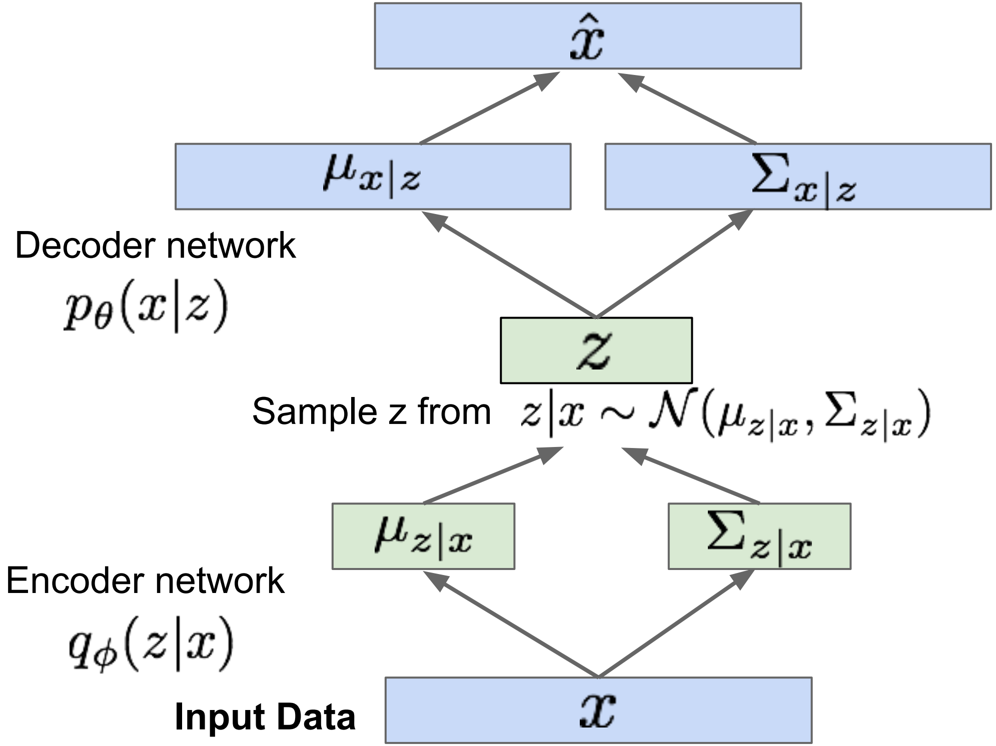


# Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

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

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

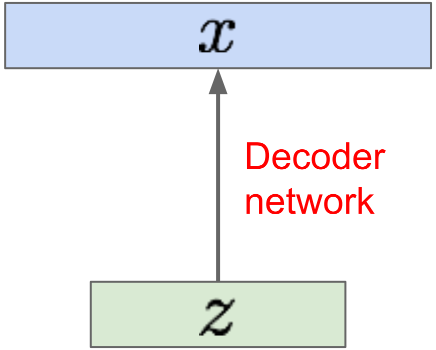


# Variational Autoencoders: Generating Data!

Our assumption about data generation process

Sample from true conditional

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




Sample from true prior

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

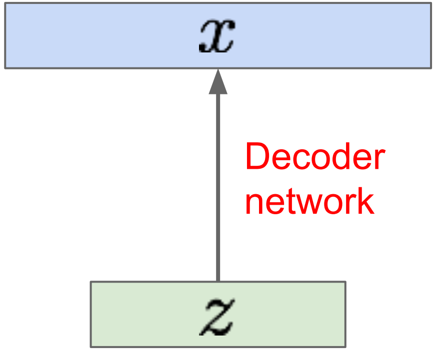
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

# Variational Autoencoders: Generating Data!

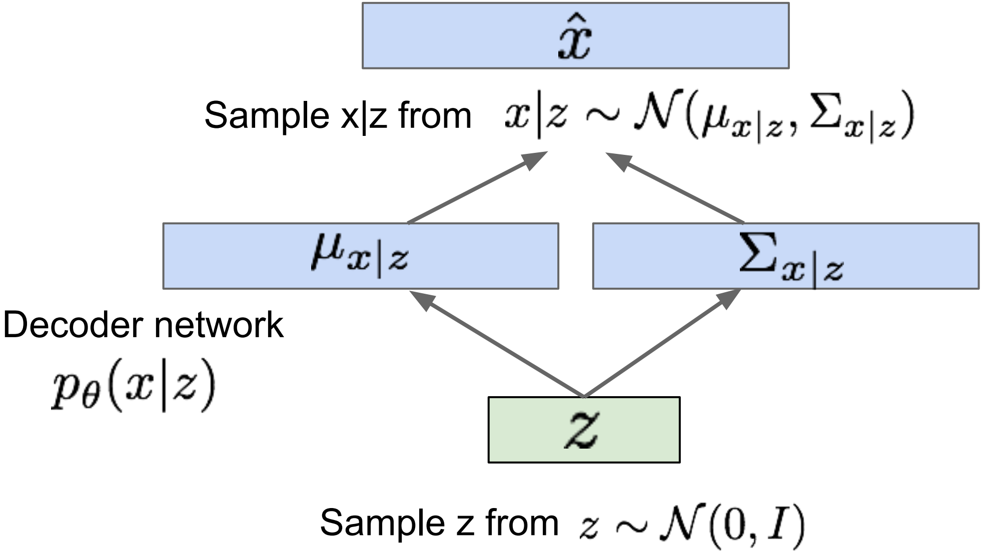
Our assumption about data generation process

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

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



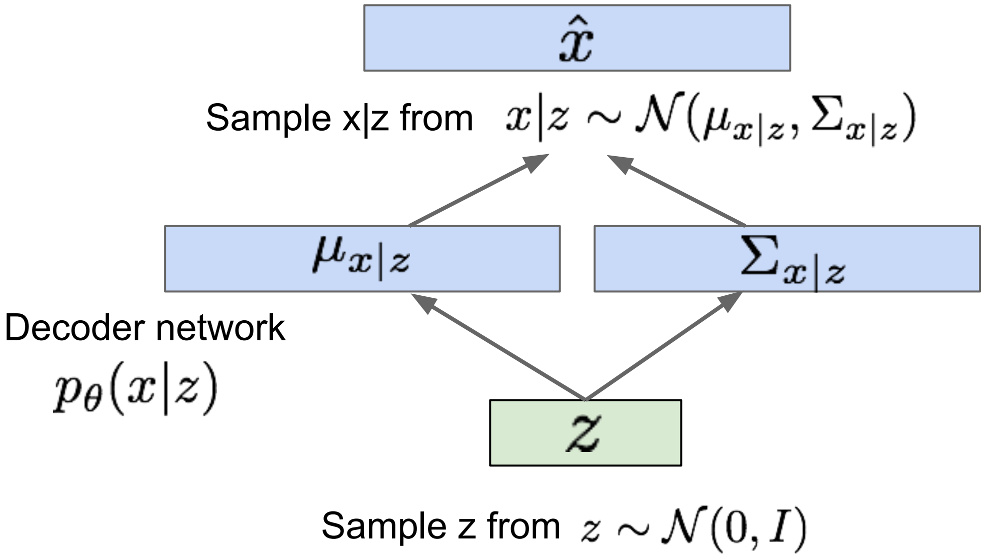

Now given a trained VAE:
use decoder network & sample z from prior!

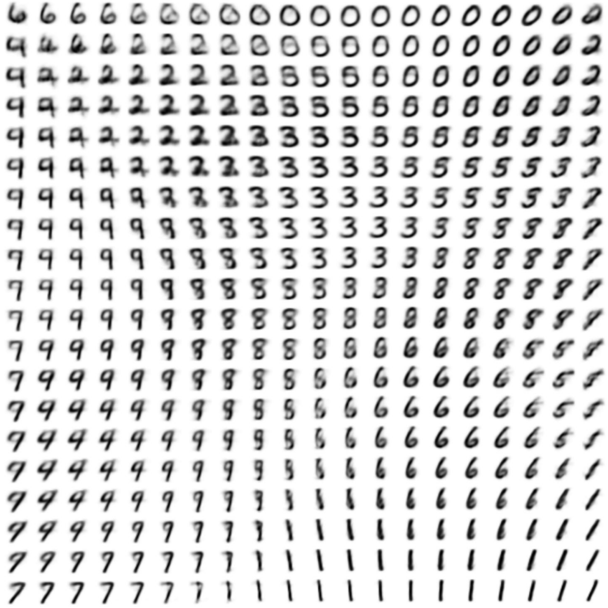



Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

# Variational Autoencoders: Generating Data!

Use decoder network. Now sample z from prior!

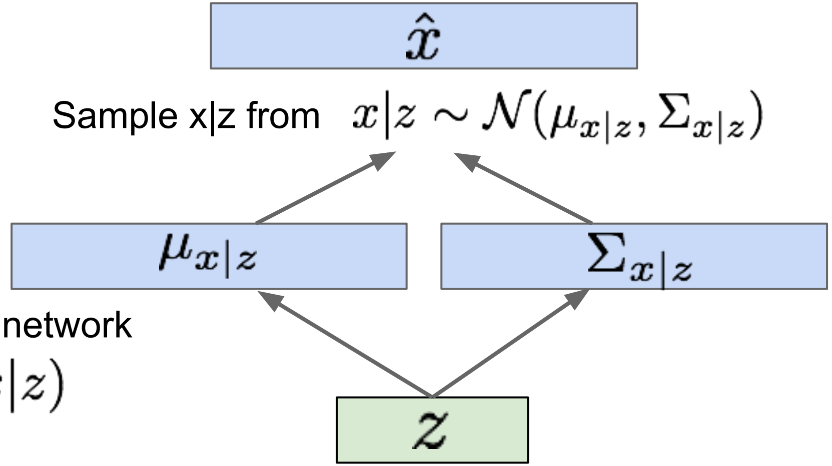


Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

# Variational Autoencoders: Generating Data!

Use decoder network. Now sample z from prior!



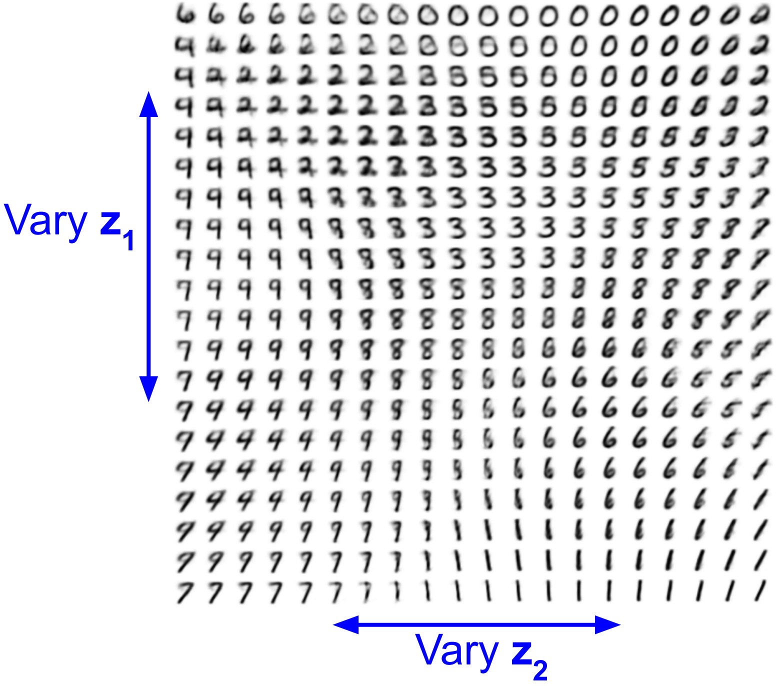

Decoder network

$p_\theta(x|z)$

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

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

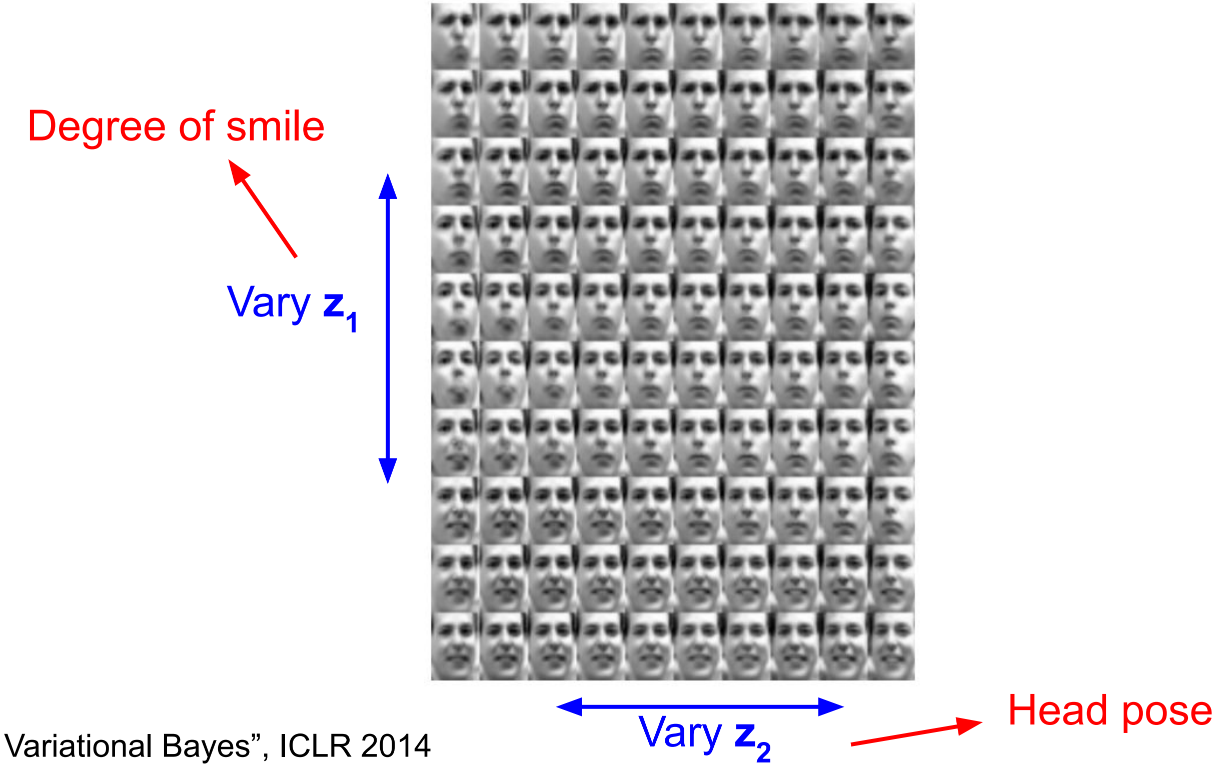
Data manifold for 2-d **z**



# Variational Autoencoders: Generating Data!

Diagonal prior on **z**
=> independent latent variables

Different dimensions of **z** encode interpretable factors of variation




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

# Variational Autoencoders: Generating Data!

Diagonal prior on **z**
=> independent
latent variables

Different
dimensions of **z**
encode
interpretable factors
of variation

Degree of smile

Vary $\mathbf{z}_1$

Vary $\mathbf{z}_2$

Head pose

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

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

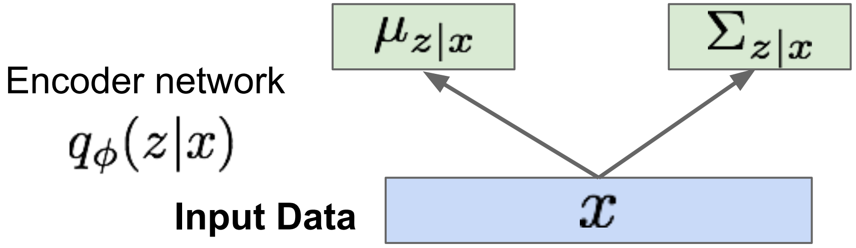
# Variational Autoencoders: Generating Data!

32x32 CIFAR-10

Labeled Faces in the Wild

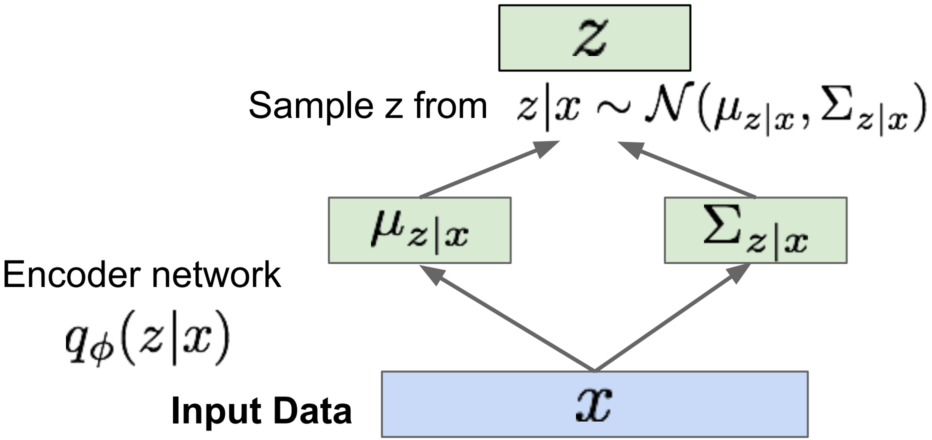
# Editing images with VAEs

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



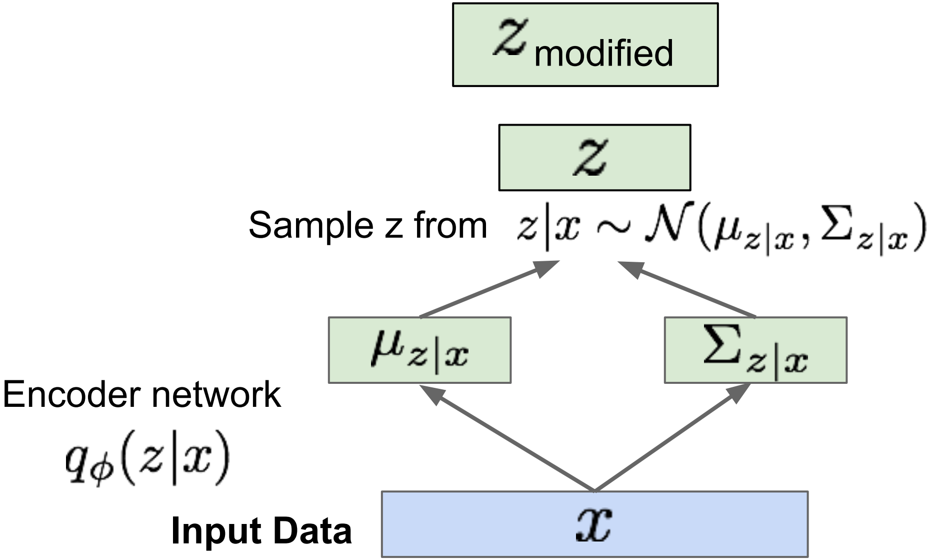
# Editing images with VAEs

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



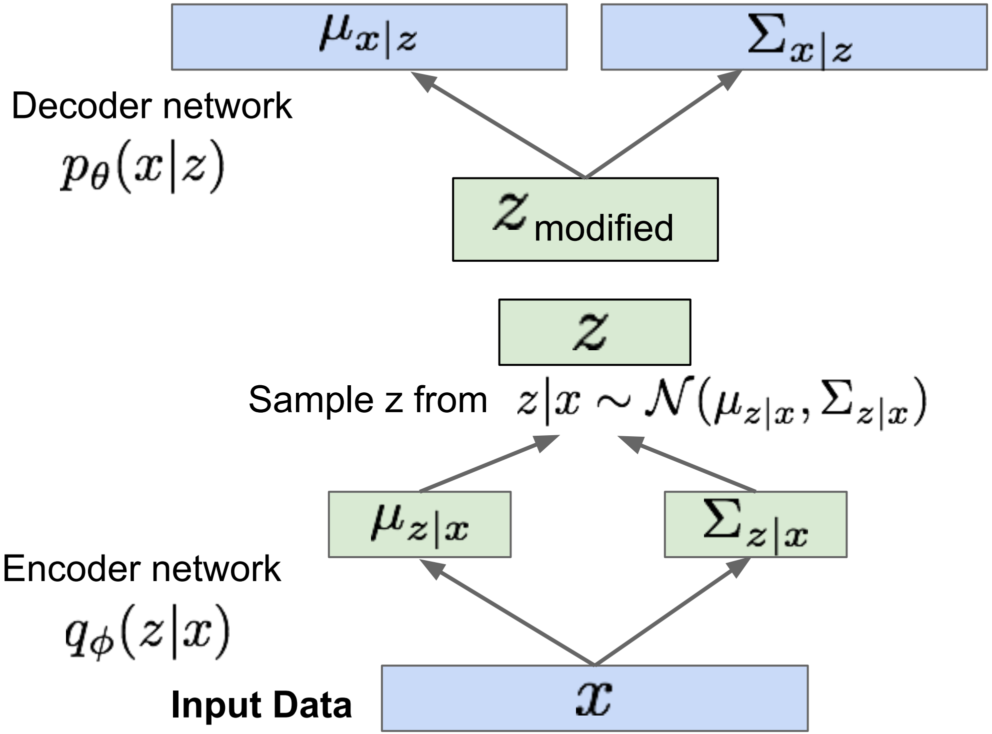
# Editing images with VAEs

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



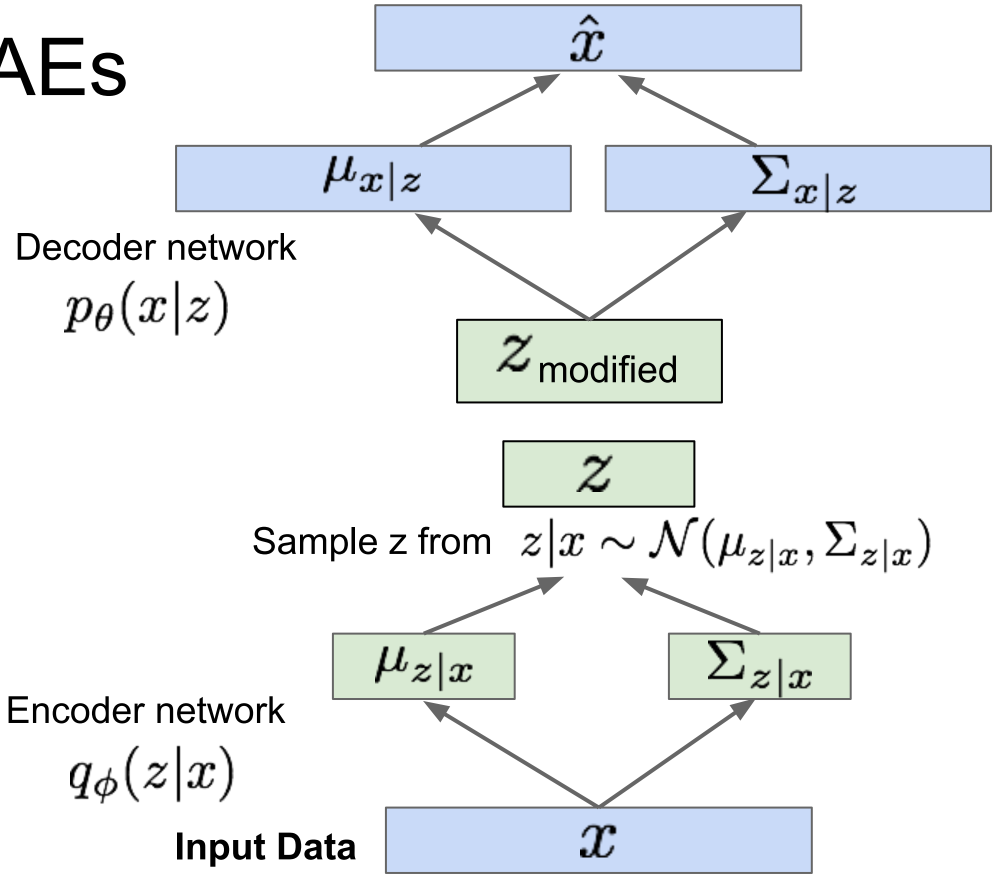
# Editing images with VAEs

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

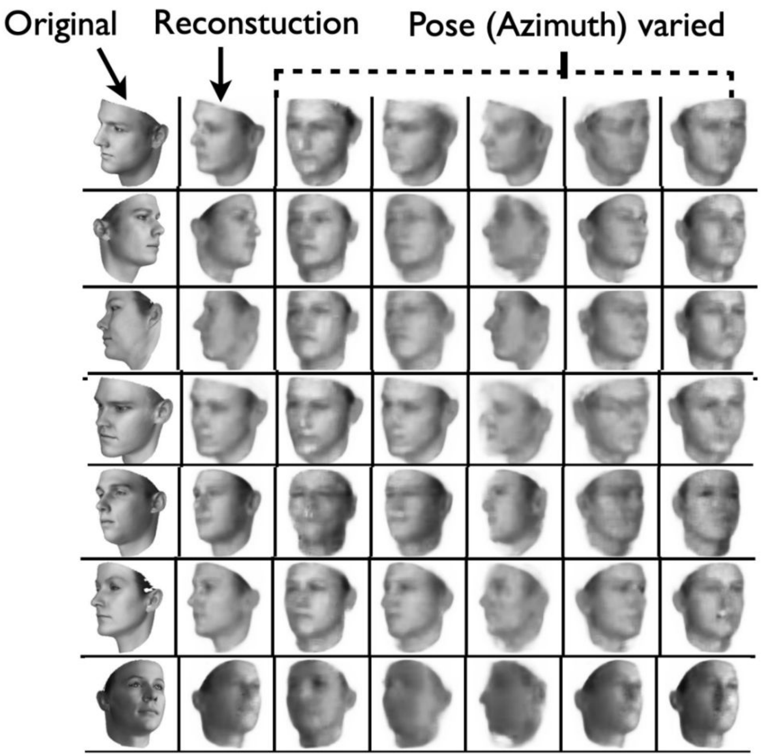


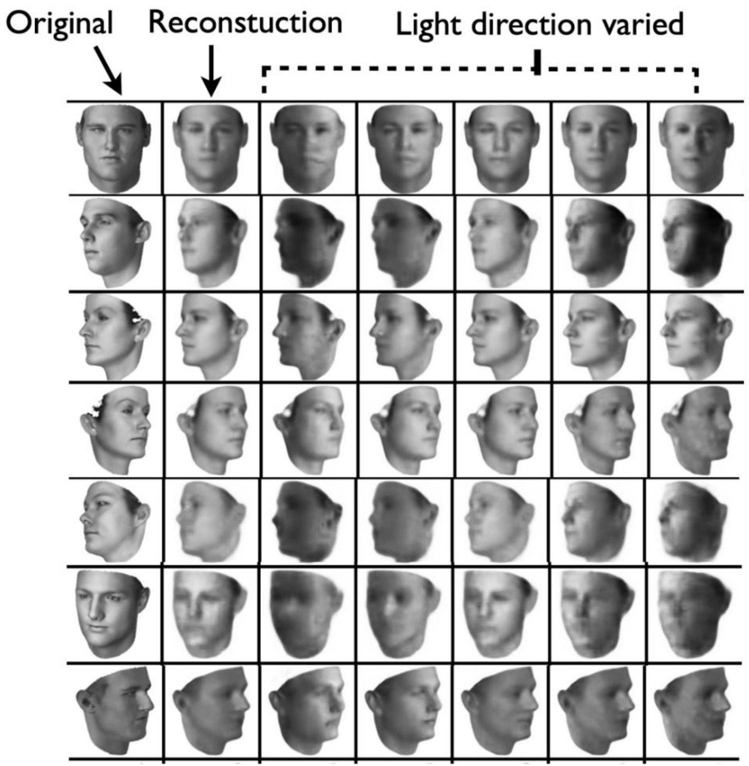
# Editing images with VAEs

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



# Editing images with VAEs



# Variational Autoencoders

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 state-of-the-art (GANs)

**Active areas of research:**

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

# Comparing the two methods so far

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

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

# Next time: GANs and diffusion