CS60010: Deep Learning
Spring 2023

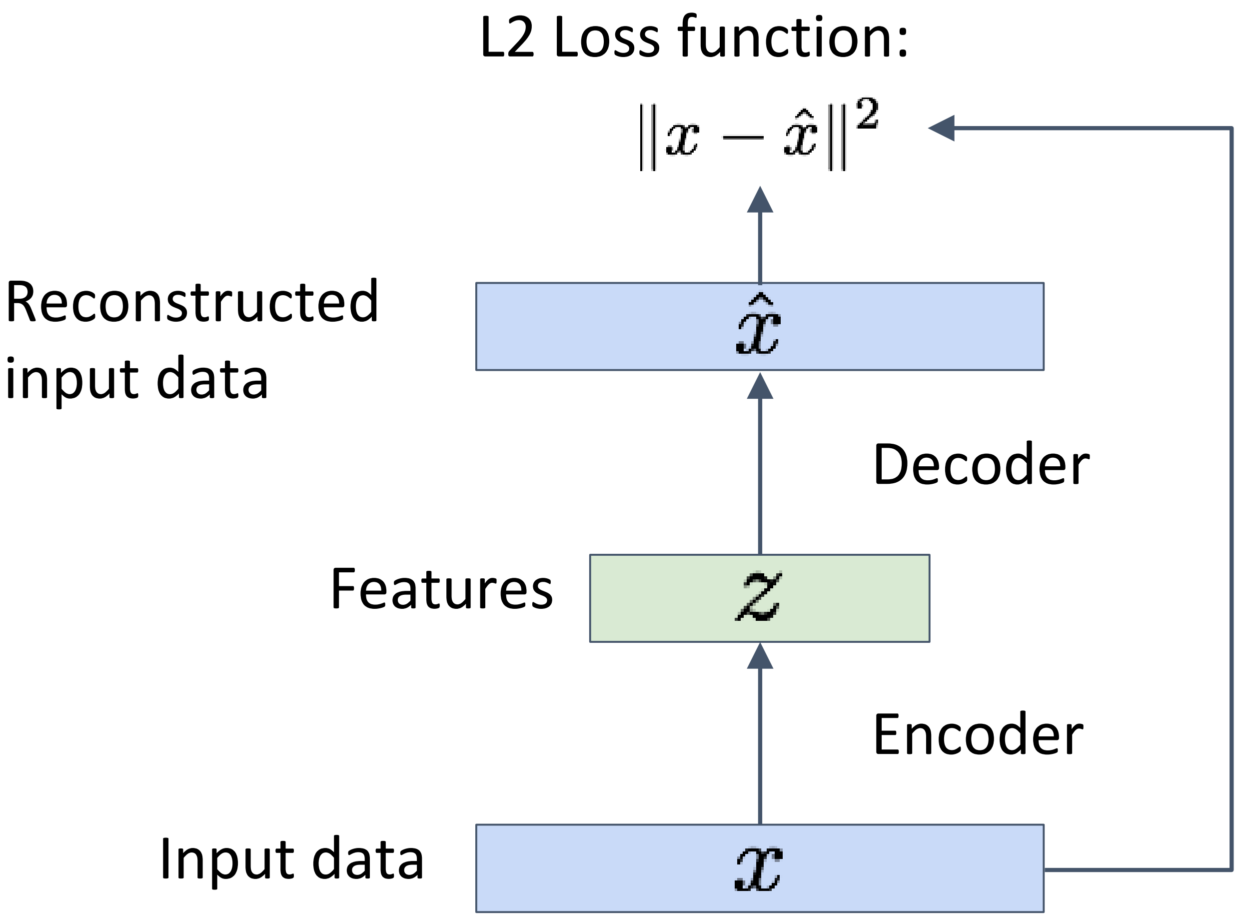
Sudeshna Sarkar

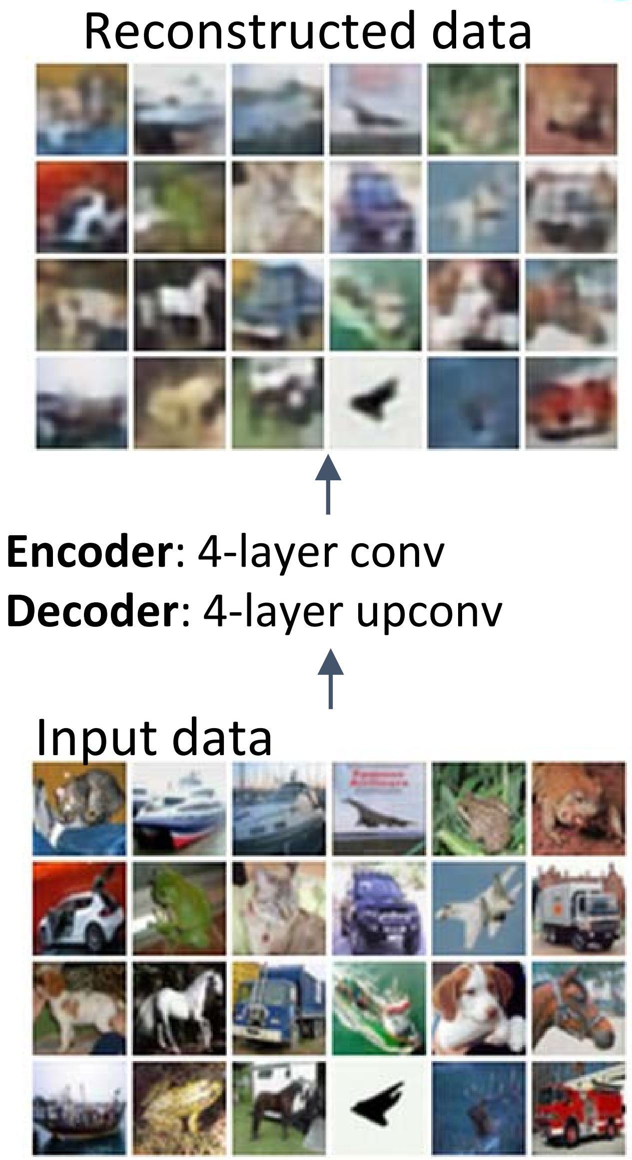
Variational AutoEncoder

23 Mar 2023

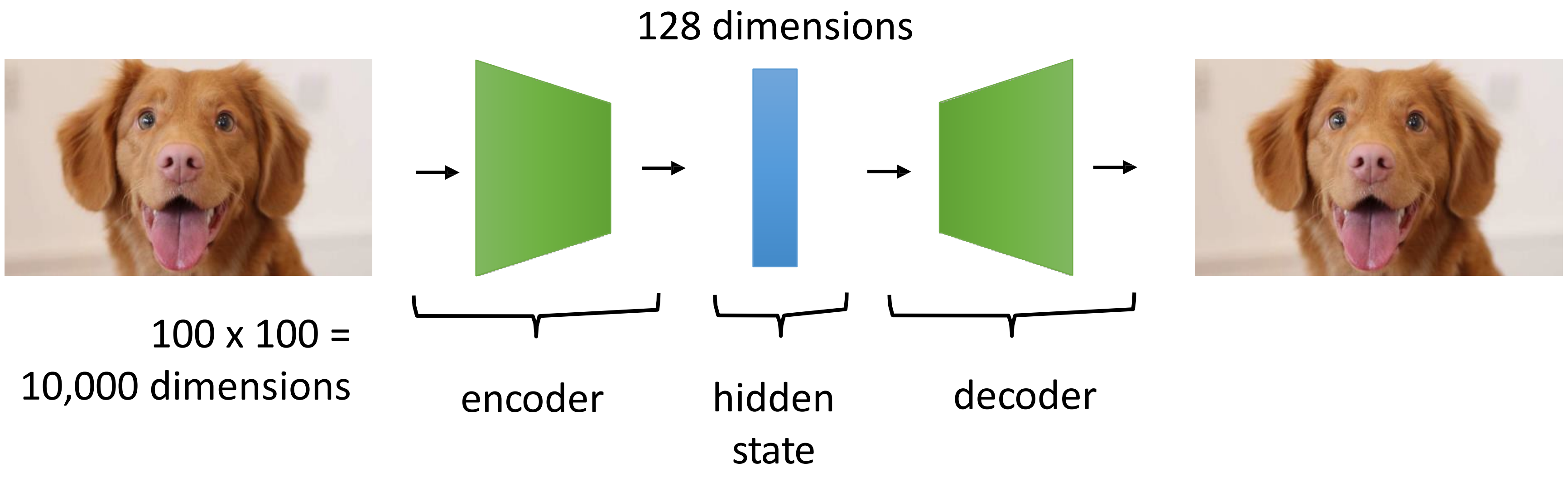
Autoencoder

Autoencoders





Bottleneck autoencoder



- This has some interesting properties:
  - If both encoder and decoder are linear, this exactly recovers PCA
  - Can be viewed as “non-linear dimensionality reduction” – could be useful simply because dimensionality is lower and we can use various algorithms that are only tractable in low-dimensional spaces (e.g., discretization)

Sparse autoencoder

Idea: can we describe the input with a small set of "attributes"?

This might be a more **compressed** and **structured** representation

Pixel (0,0): #FE057D

Pixel (0,1): #FD0263

Pixel (0,2): #E1065F

"dense representation": most values non-zero
Not structured

Idea: "sparse" representations are going to be more structured!

has_ears: 1

has_wings: 0

has_wheels: 0

very structured!

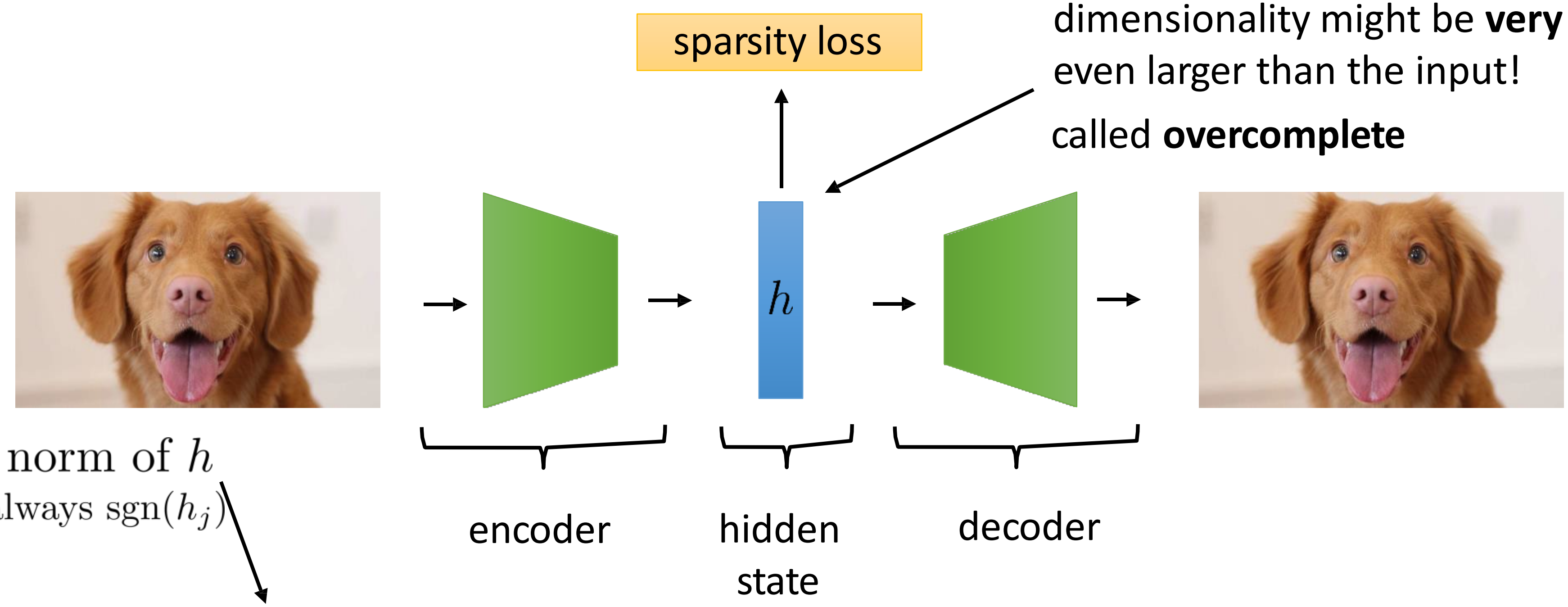
"sparse": most values are zero

there are many possible "attributes," and most images don't have most of the attributes

Sparse autoencoder

dimensionality might be **very** large, even larger than the input!

called **overcomplete**



ℓ_1 norm of h

gradient is always $\mathrm{sgn}(h_j)$

simple sparsity loss: $\sum_{j=1}^{D} |h_j|$

Sparse Autoencoder

- Regularize outputs of hidden layer to enforce sparsity:

$$\tilde{J}(x) = J(x, g(f(x))) + \alpha\, \Omega(h)$$

J : loss function, f : encoder, g : decoder, h=f(x), Ω penalizes non-sparsity of h

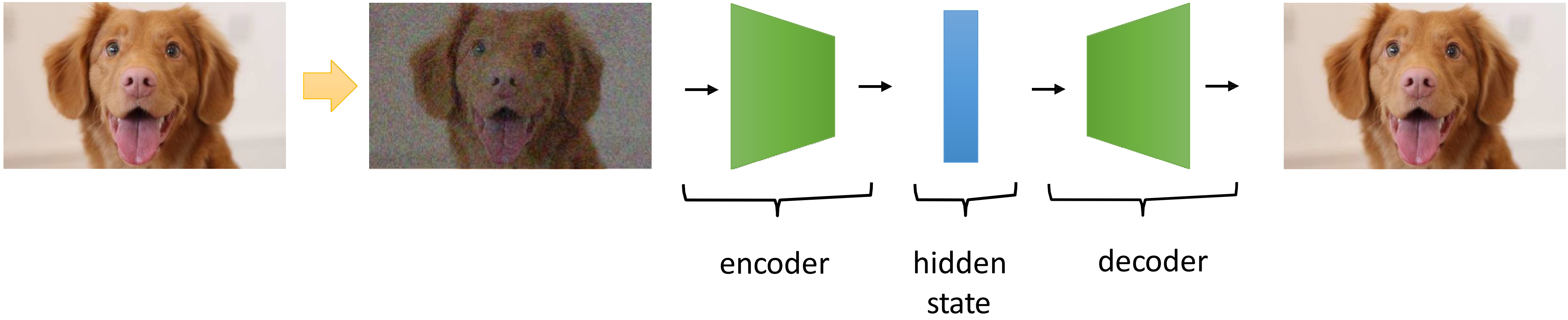
- E.g., can use $\Omega(h) = \sum_i |h_i|$ and ReLU activation to force many zero outputs in hidden layer
- Can also measure average activation of h_i across mini-batch and compare it to user-specified **target sparsity** value ρ (e.g., 0.1) via square error or **Kullback-Leibler divergence**:

$$p \log \frac{p}{q} + (1 - p) \log \frac{1 - p}{1 - q}$$

q is average activation of h_i over mini-batch

Denoising autoencoder

Idea: a good model that has learned meaningful structure should "fill in the blanks"

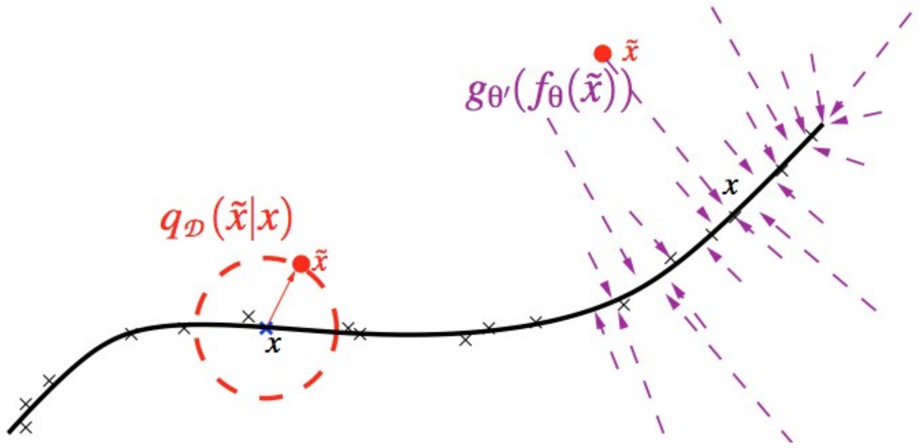


Can train an autoencoder to learn to **denoise** input by giving input **corrupted** instance $\tilde{\boldsymbol{x}}$ and targeting **uncorrupted** instance $\boldsymbol{x}$

There are **many variants** on this basic idea, and this is one of the most widely used simple autoencoder designs

Denoising Autoencoder

- How does it work?
- Even though, e.g., MNIST data are in a 784-dimensional space, they lie on a low-dimensional **manifold** that captures their most important featur
- Corruption process moves instance ***x*** off of manifol
- Encoder f_θ and decoder $g_{\theta'}$ are trained to project $\tilde{x}$ back onto manifold



The types of autoencoders: Forcing Structure

1. Dimensionality: make the **hidden state** smaller than the **input/output**, so that the network must **compress** it

+ very simple to implement

- simply reducing dimensionality often does not provide the structure we want

2. Sparsity: force the **hidden state** to be sparse (most entries are 0), so that the network must **compress** input

+ principled approach that can provide a “disentangled” representation

- **harder in practice, requires choosing the regularizer and adjusting hyperparameters**

3. Denoising: corrupt the **input** with **noise**, forcing the autoencoder to learn to distinguish **noise from signal**

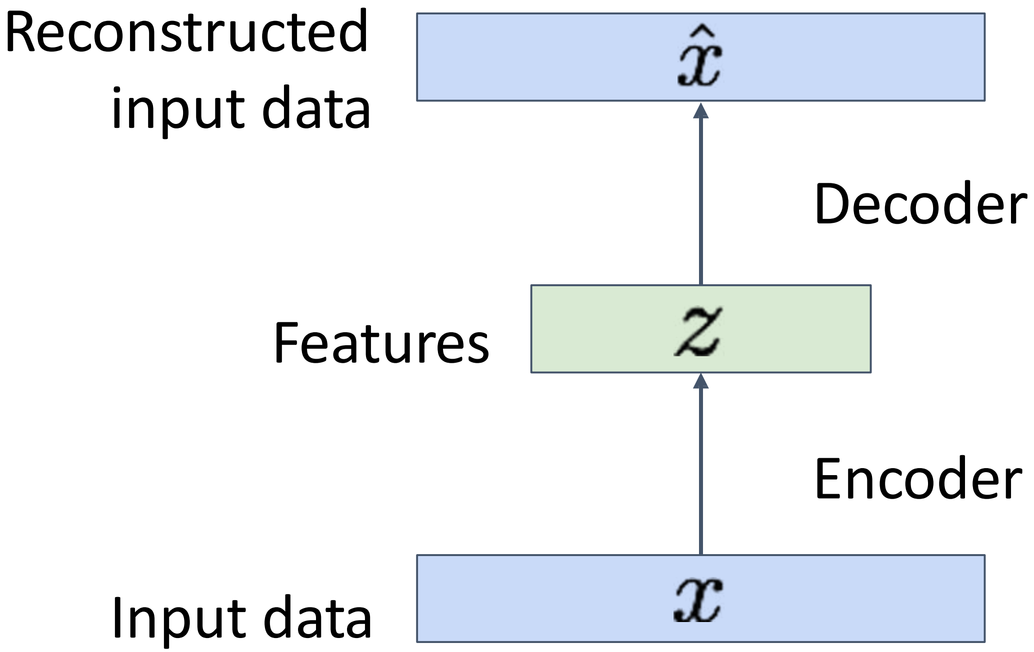
+ very simple to implement

- **not clear which layer to choose for the bottleneck, adhoc choicers (e.g., how much noise to add)**

4. Probabilistic modeling: force the **hidden state** to agree with a **prior distribution**

Autoencoders

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



Variational Autoencoders

Variational Autoencoders

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

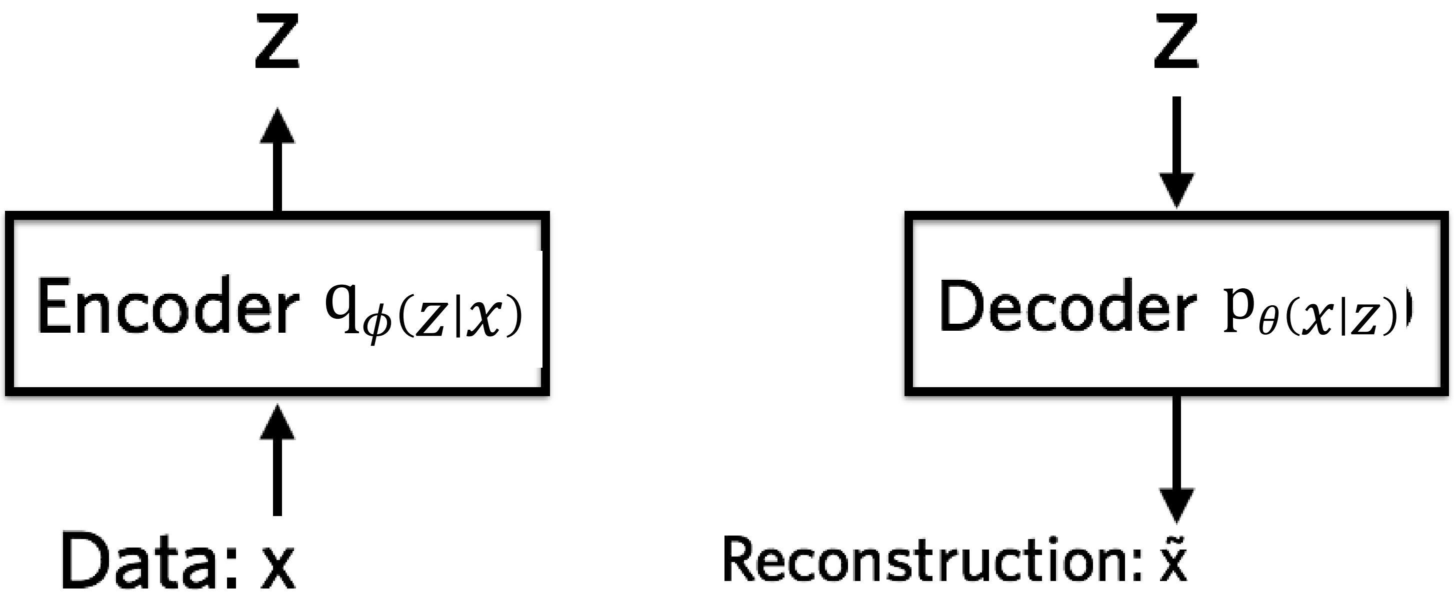
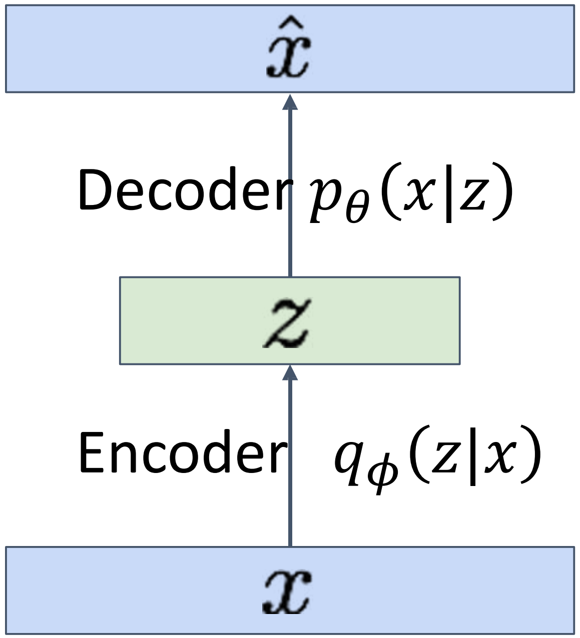


Image Credit: https://jaan.io/what-is-variational-autoencoder-vae-tutorial/

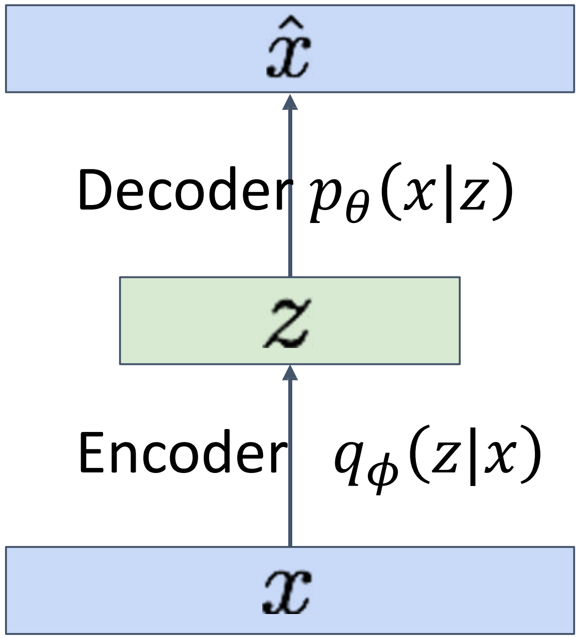
Training the model



How does one train an AE to ensure that the hidden representation z has a specific distribution e.g., N(0,1)

- Minimize the error between x and $\hat{x}$
- Minimize the KL divergence between the distribution of $\boldsymbol{z}$ and *N(0,1)*
  - Minimize the negative log likelihood of $\boldsymbol{z}$ as computed from a standard Gaussian

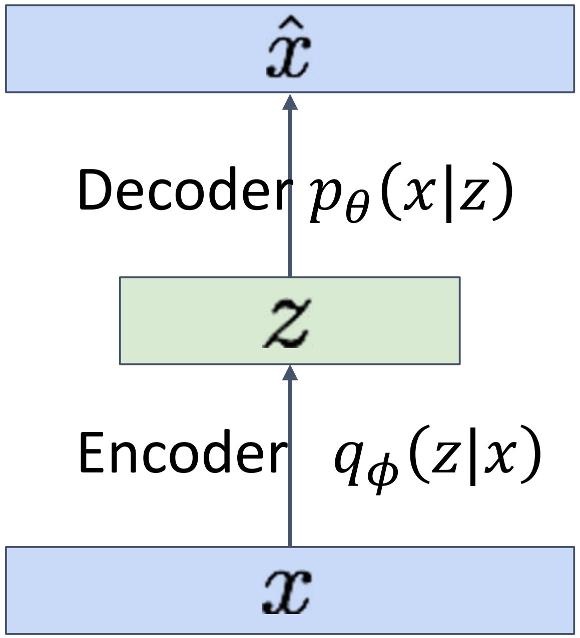
Training the model



Instead of finding the unique z for any x, we will find the distribution $P(z|x)$

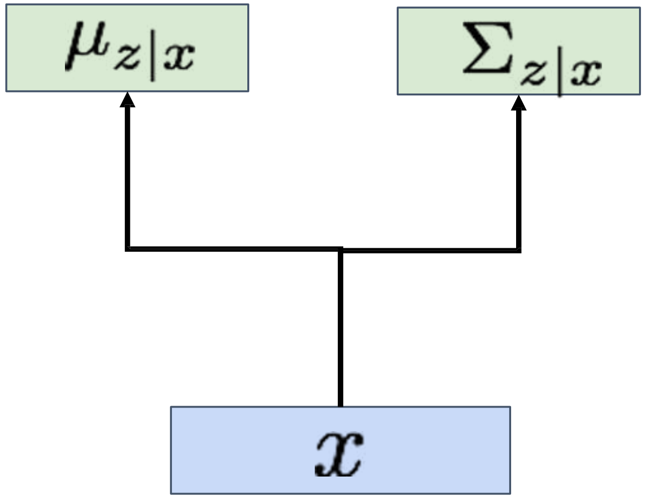
- The encoder computes the distribution $P(z|x)$ for any x
- The decoder tries to convert a randomly sampled z from $P(z|x)$ to x
- Constraint on z: Make $P(z|x)$ as close to the standard Gaussian as possible

Encoder

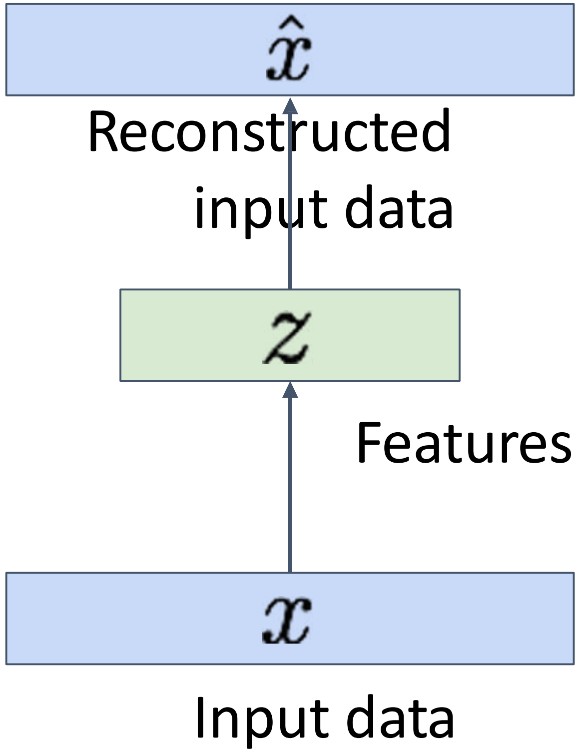


The encoder computes the distribution $P(z|x)$ for any x

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



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



Decoder

Encoder

Variational Autoencoders

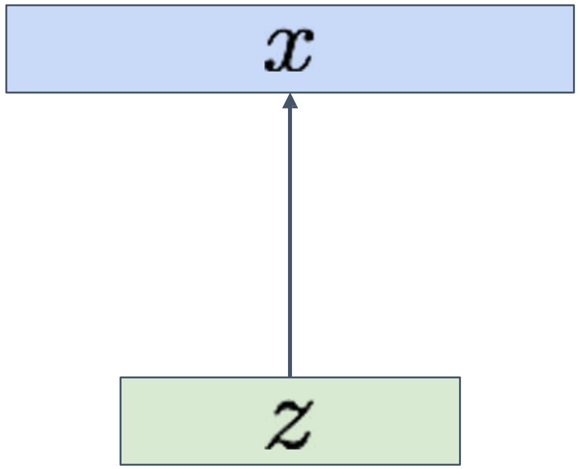
Probabilistic spin on autoencoders:

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

What we want at test time:

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

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



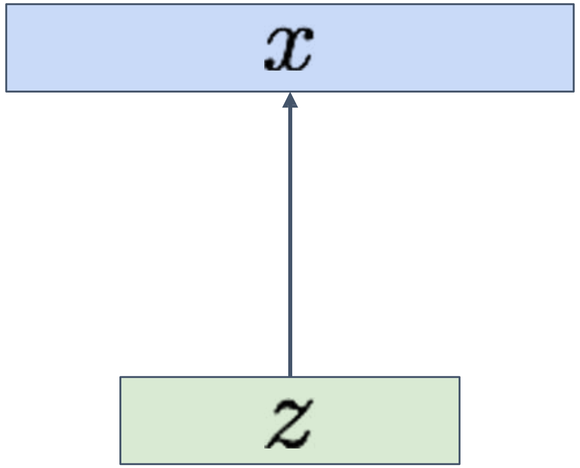
Intuition: x is an image, **z** is latent factors used to generate **x:** attributes, orientation, etc.

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

- p(x|z) : **decoder** as in autencoder

Variational Autoencoders

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



$p_{\theta*}(z)$

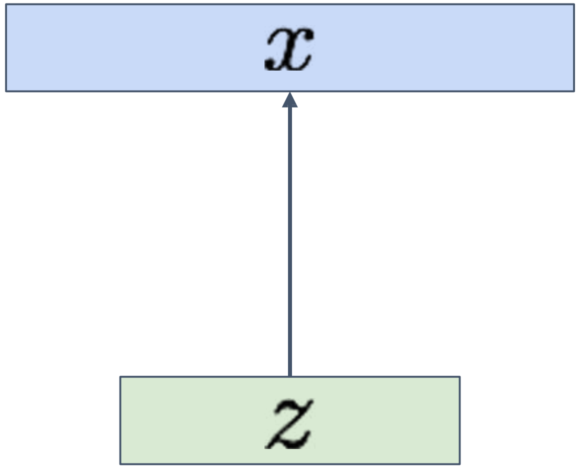
How to train this model?

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

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



$p_{\theta*}(z)$

How to train this model?

Basic idea: **maximize likelihood of data**

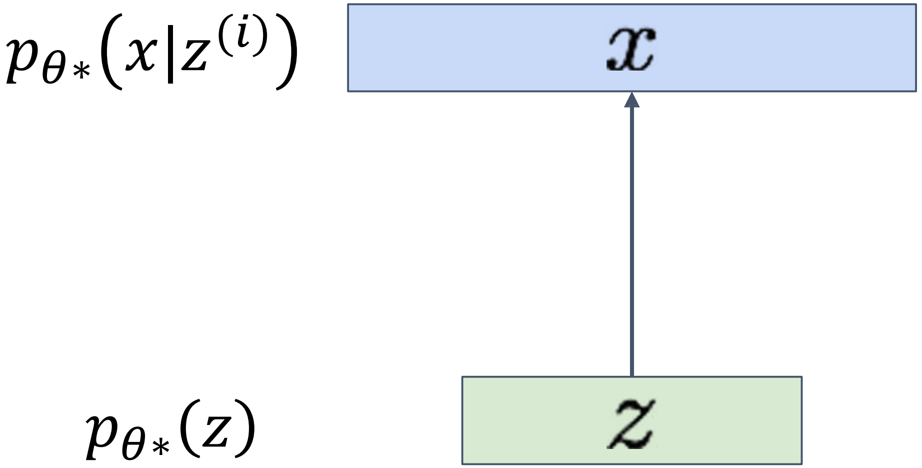
We don't observe z, so need to marginalize:

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

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

Ok, can compute this with decoder network

Variational Autoencoders



How to train this model?

Basic idea: **maximize likelihood of data**

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

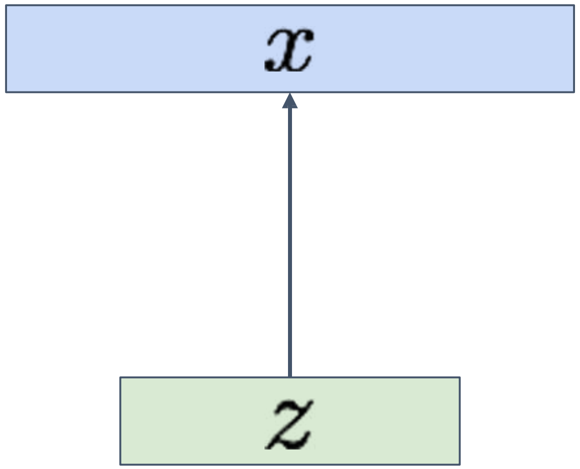
$$p_\theta(x) = \int p_\theta(x, z) dz$$

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

Ok, we assumed Gaussian prior for z

Variational Autoencoders

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



$p_{\theta*}(z)$

How to train this model?

Basic idea: **maximize likelihood of data**

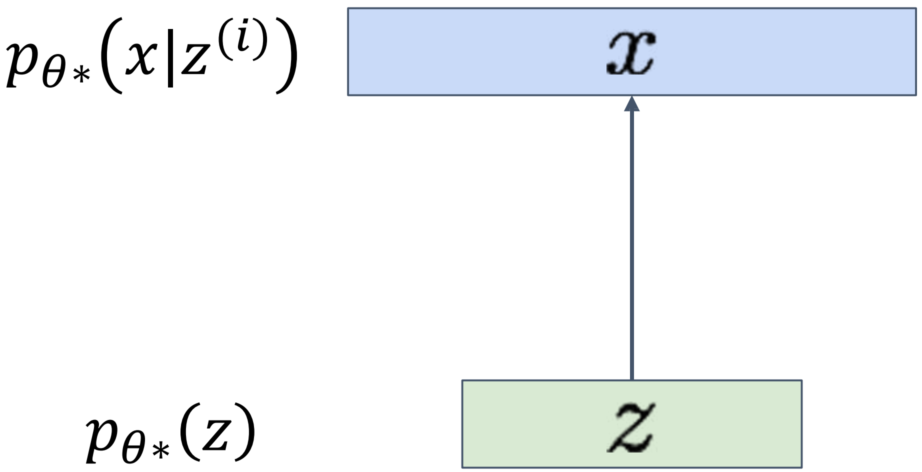
We don’t observe z, so need to marginalize:

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

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

Problem: Impossible to integrate over all z!

Variational Autoencoders



How to train this model?

Basic idea: **maximize likelihood of data**

Another idea: Try Bayes' Rule:

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

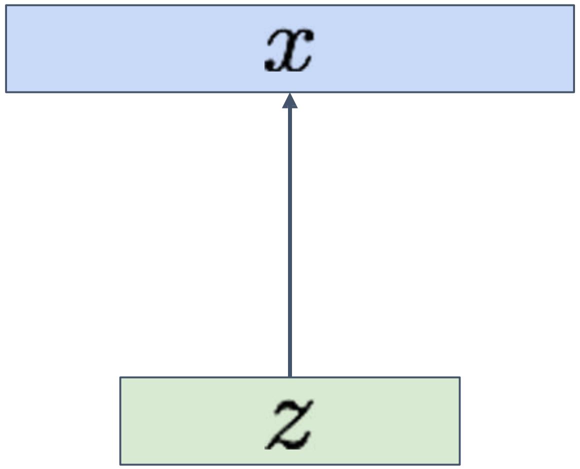
Ok, compute with decoder network

Ok, we assumed Gaussian prior

Problem: No way to compute this!

Variational Autoencoders

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



$p_{\theta*}(z)$

How to train this model?

Basic idea: **maximize likelihood of data**

Another idea: Try Bayes' Rule:

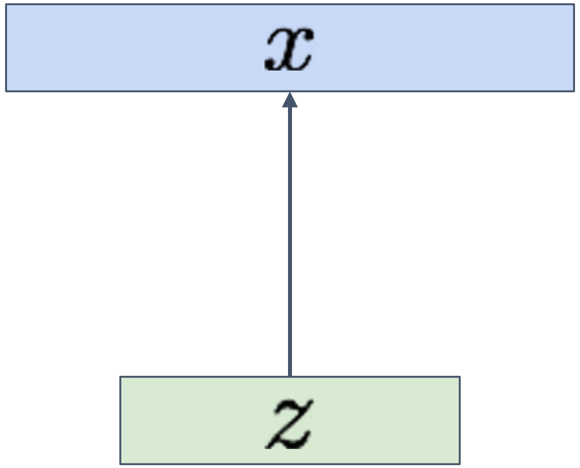
$$p_\theta(x) = \frac{p_\theta(x|z)\ p_\theta(z)}{p_\theta(z|x)}$$

Solution: Train another network **(encoder)** that learns

$$q_\phi(z|x) \approx p_\theta(z|x)$$

Variational Autoencoders

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



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

How to train this model?

Basic idea: **maximize likelihood of data**

Another idea: Try Bayes' Rule:

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

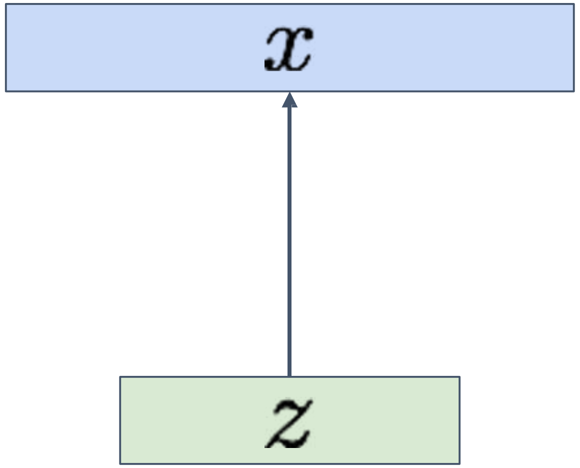
Use **encoder** to compute

$q_\phi(z|x) \approx p_\theta(z|x)$

Variational Autoencoders

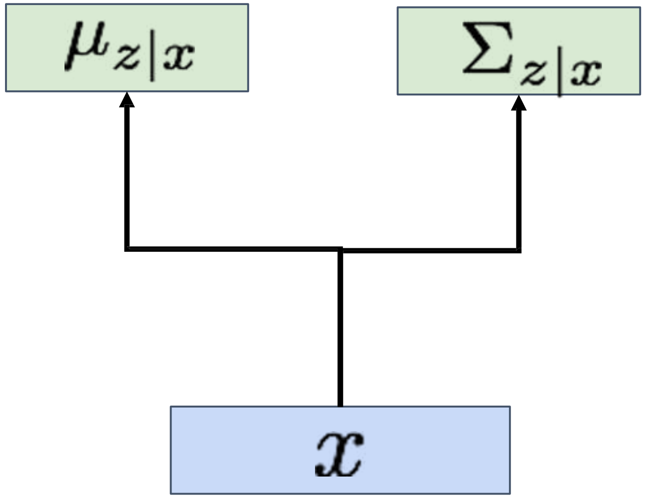
Decoder network inputs latent code z, gives distribution over data x

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



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

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



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

then we can approximate

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

Idea: Jointly train both encoder and decoder

Variational Autoencoders

Bayes' Rule

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

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

Variational Autoencoders

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

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

Split up using rules for logarithms

Variational Autoencoders

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

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

Split up using rules for logarithms

Variational Autoencoders

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

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

Note: We can wrap LHS in an expectation without any loss of generality since it doesn't depend on z

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

Variational Autoencoders

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

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

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

Note: We can wrap LHS in an expectation without any loss of generality since it doesn’t depend on z

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

Variational Autoencoders

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

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

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

Data reconstruction

KL divergence between prior, and samples from the encoder network

KL divergence between encoder and posterior of decoder

Variational Autoencoders

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

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

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

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

Variational Autoencoders

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

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

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

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

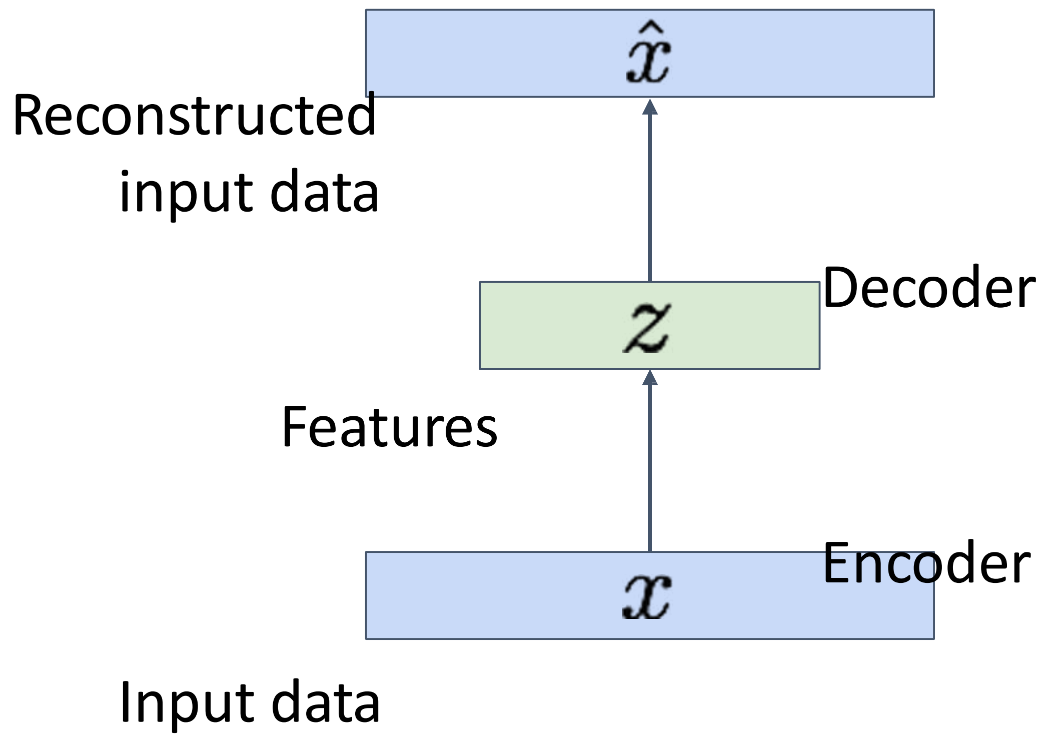
Variational Autoencoders

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

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

Recall: Vanilla Autoencoders

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



Variational Autoencoders

Train by maximizing the
variational lower bound

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

Input Data x

Variational Autoencoders

Train by maximizing the **variational lower bound**

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

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

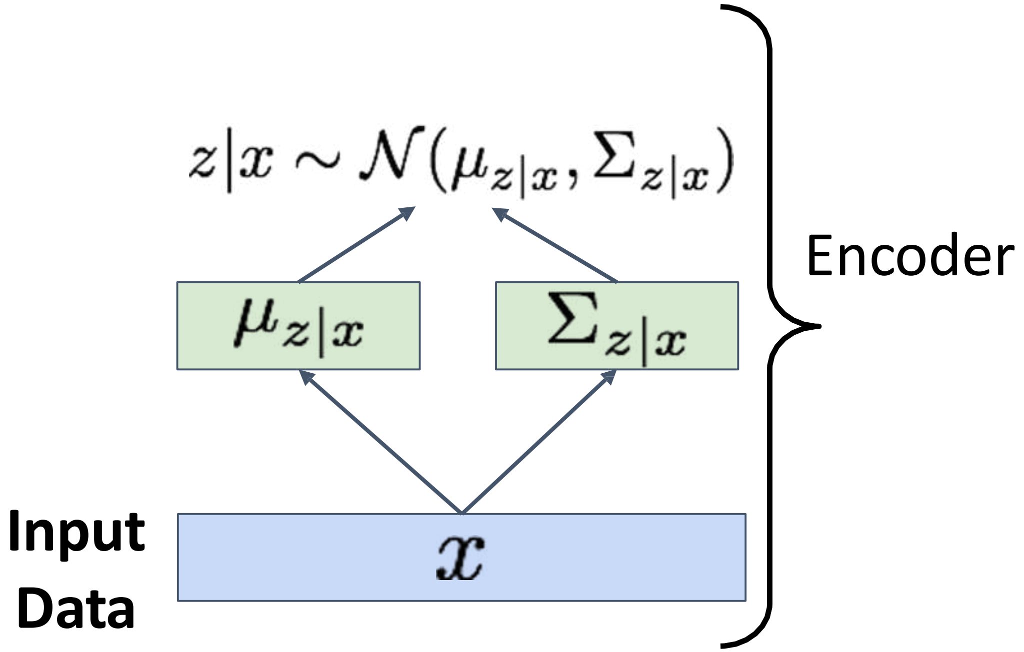
Input Data x

Variational Autoencoders

- Train by maximizing the **variational lower bound**

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

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



Variational Autoencoders

Train by maximizing the **variational lower bound**

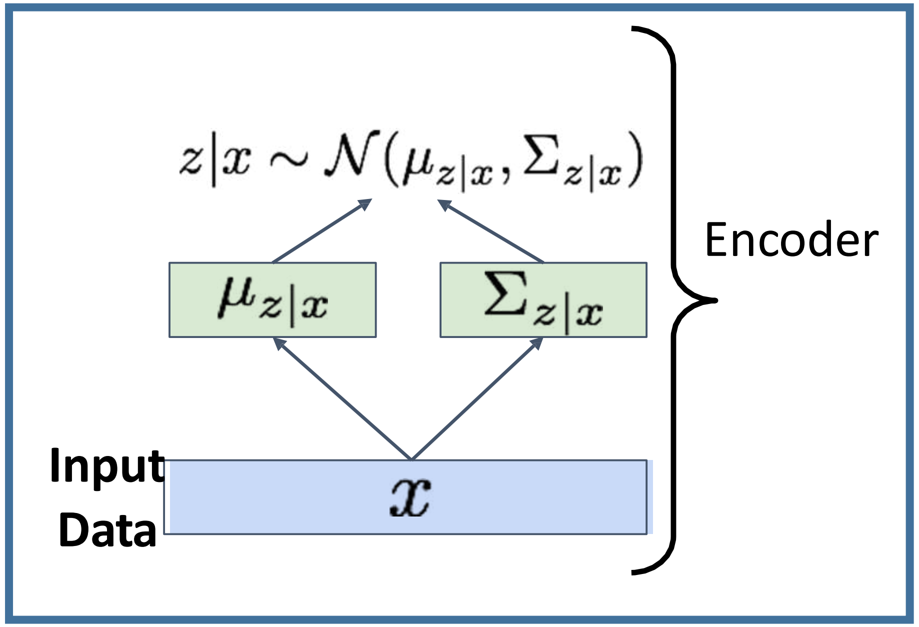
$$E_{z\sim\, q_\phi(z|x)}[\log p_\theta(x|z)] - \boxed{D_{KL}\left(q_\phi(z|x), p(z)\right)}$$

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

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

$$= \int_Z N\left(z; \mu_{z|x}, \Sigma_{z|x}\right) \log \frac{N(z; 0,1)}{N\left(z; \mu_{z|x}, \Sigma_{z|x}\right)} dz$$

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

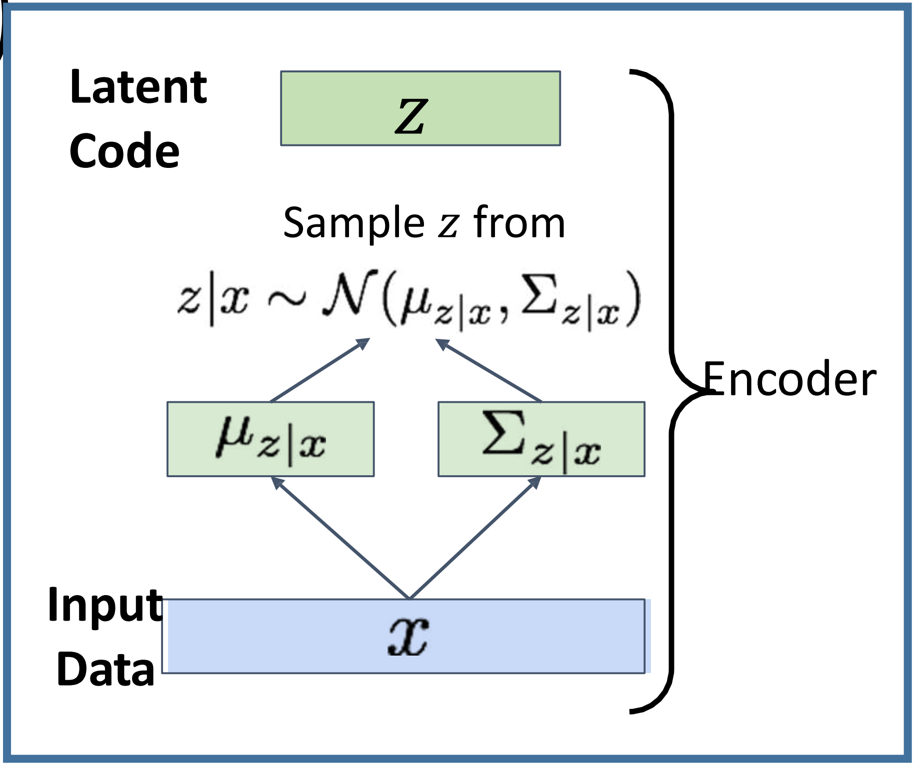


Variational Autoencoders

Train by maximizing the **variational lower bound**

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

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

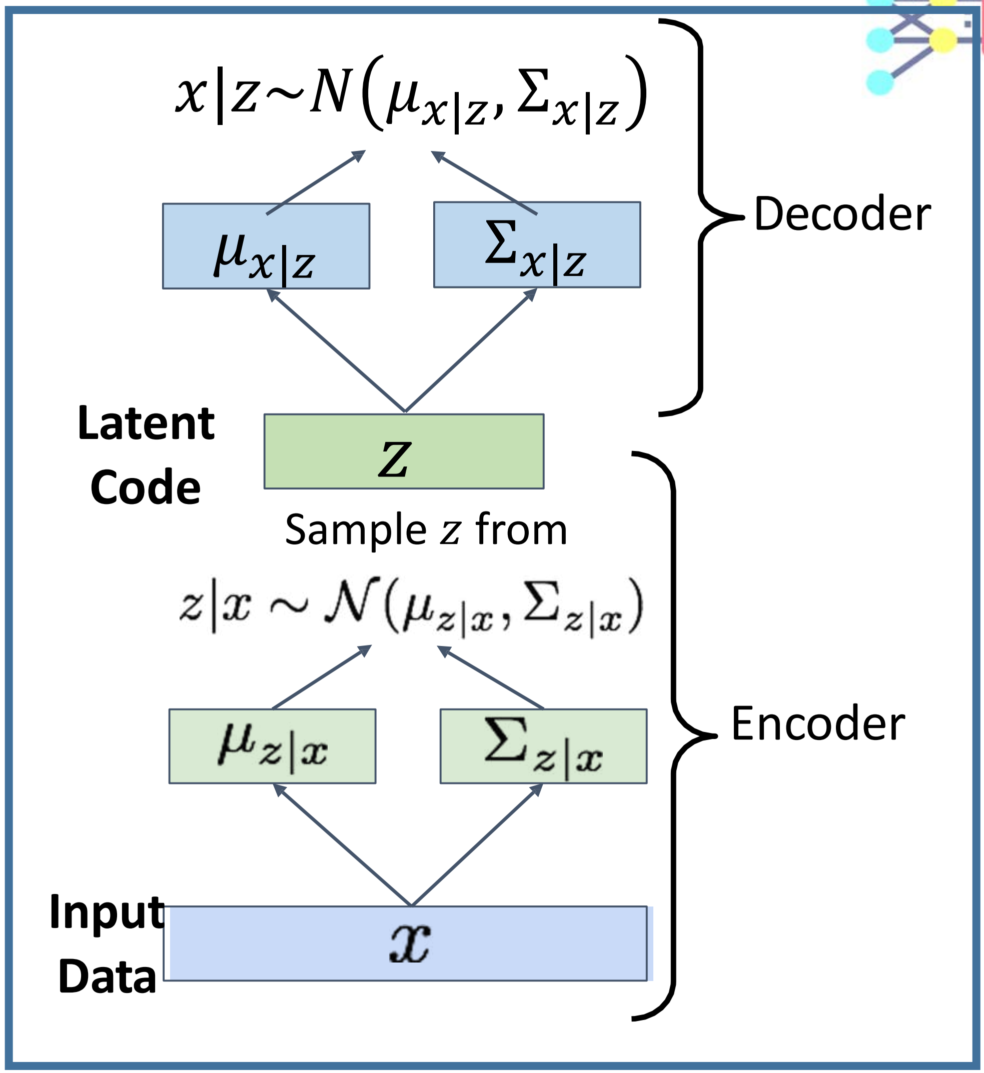


Variational Autoencoders

Train by maximizing the **variational lower bound**

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

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

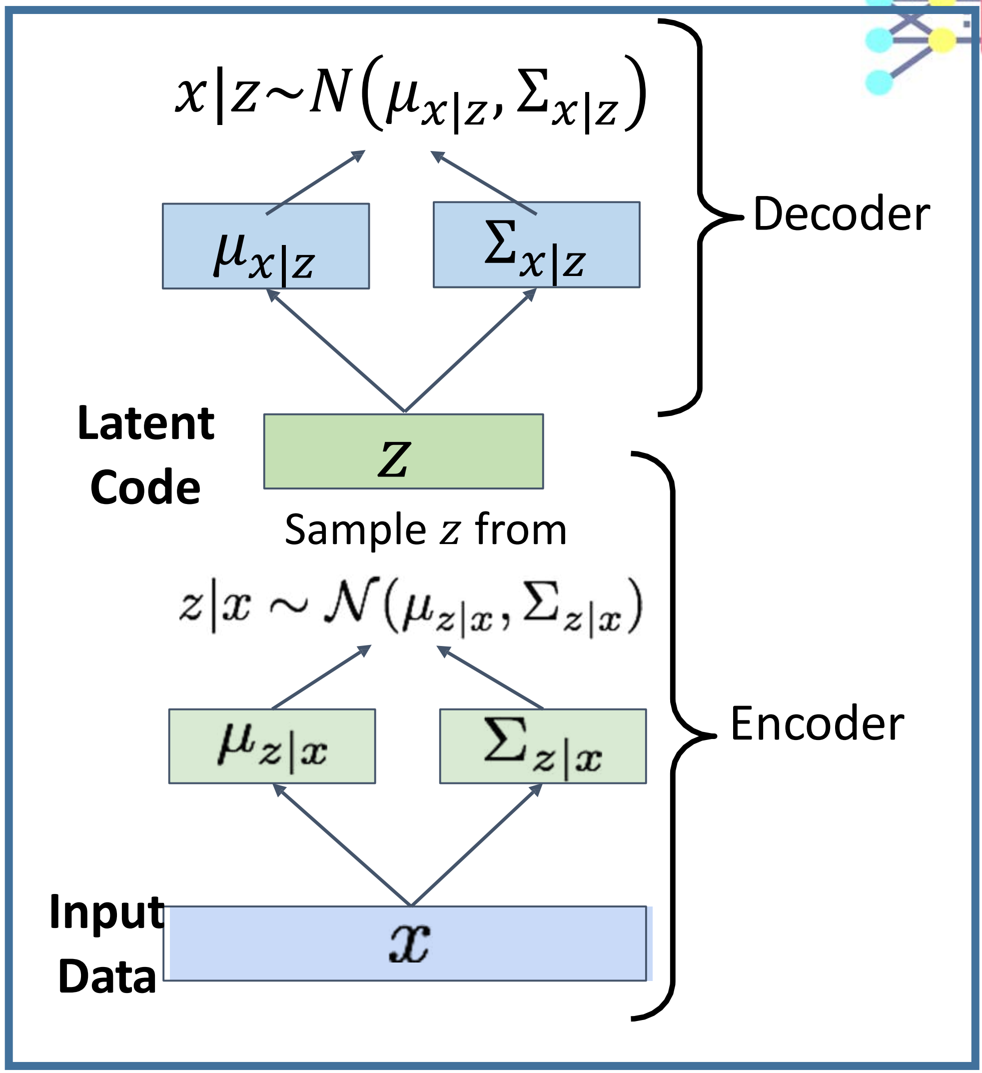


Variational Autoencoders

Train by maximizing the **variational lower bound**

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

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

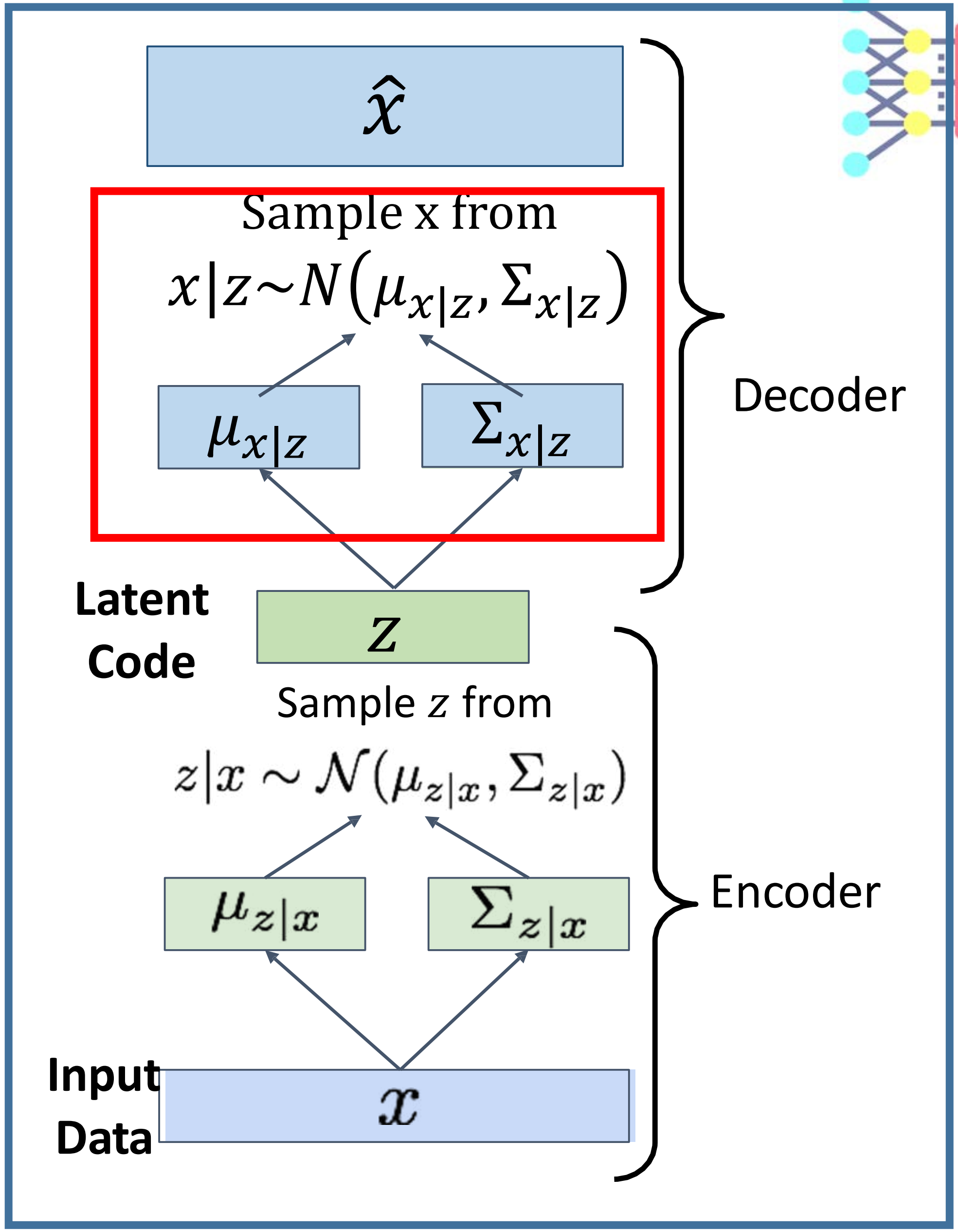


Variational Autoencoders

Train by maximizing the **variational lower bound**

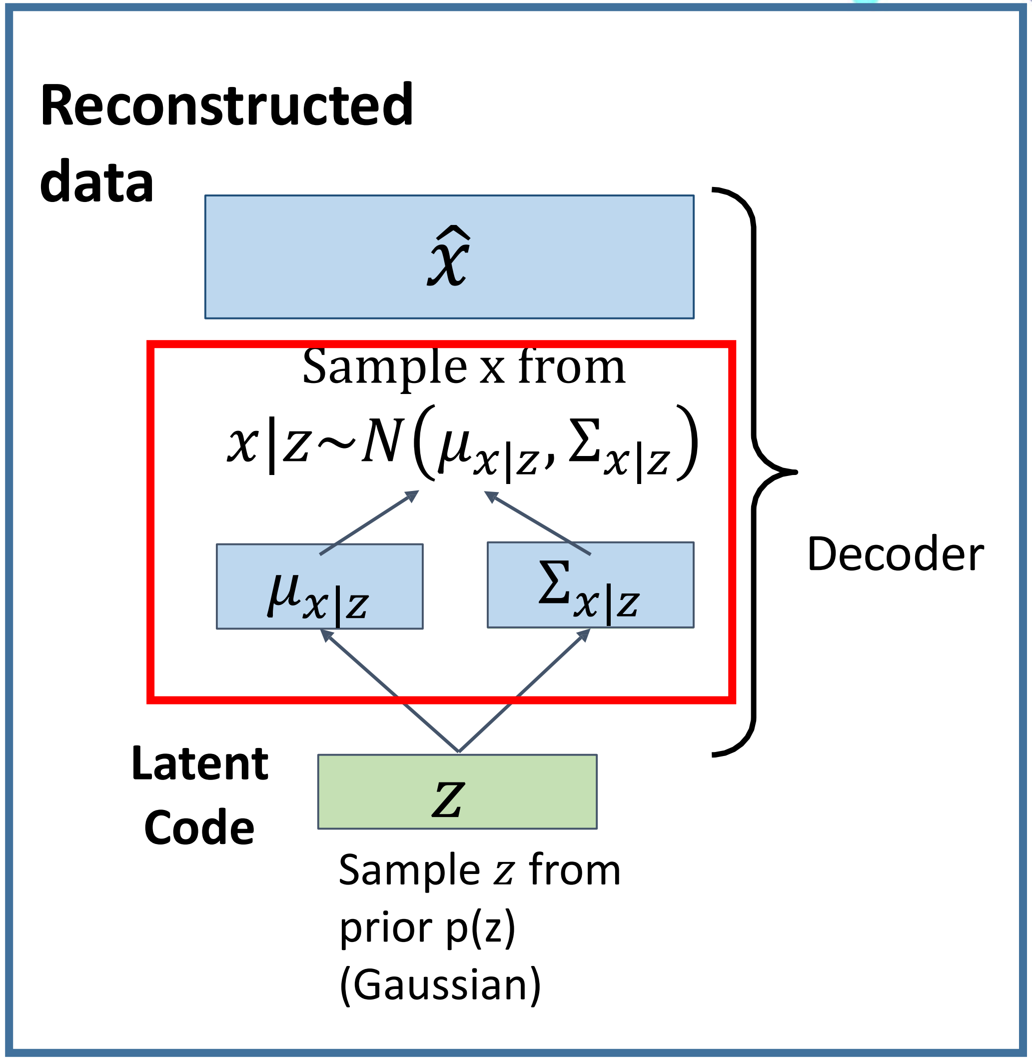
$$\boxed{E_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)]} - \boxed{D_{KL}\left(q_\phi(z|x), p(z)\right)}$$

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



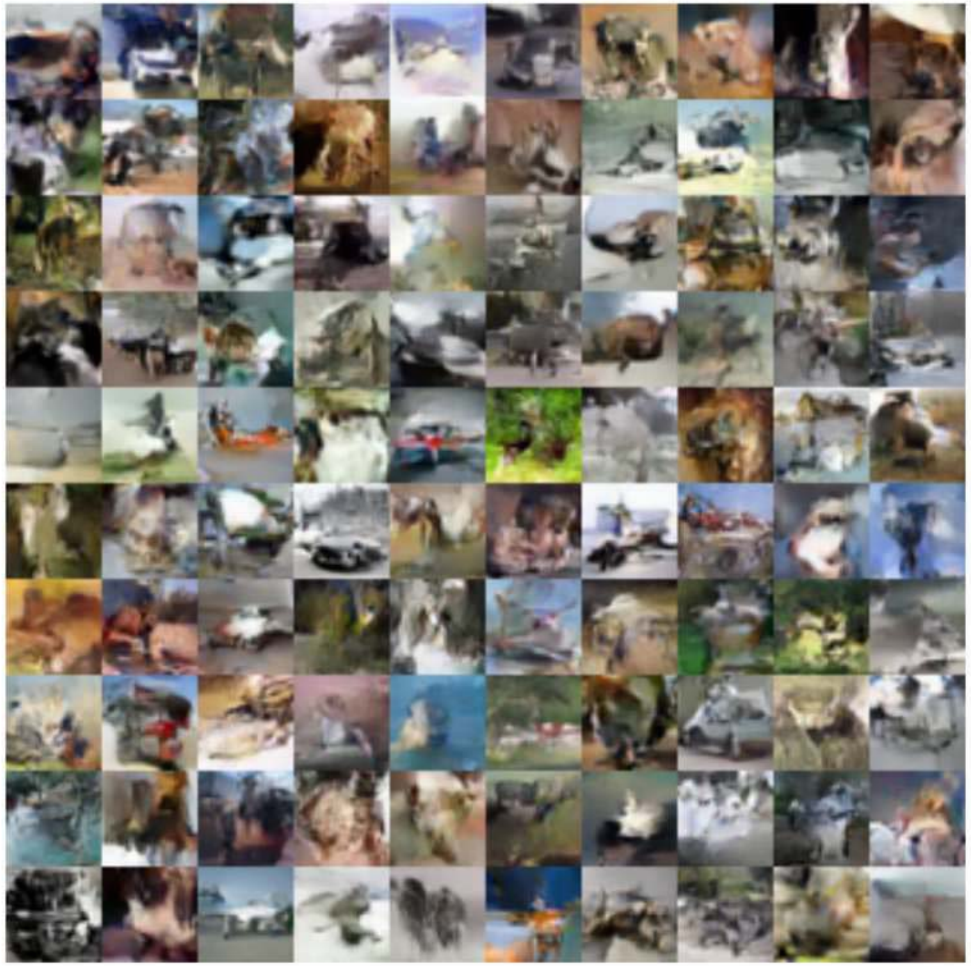
Variational Autoencoders

Test Time



Variational Autoencoders: Generating Data

32x32 CIFAR-10



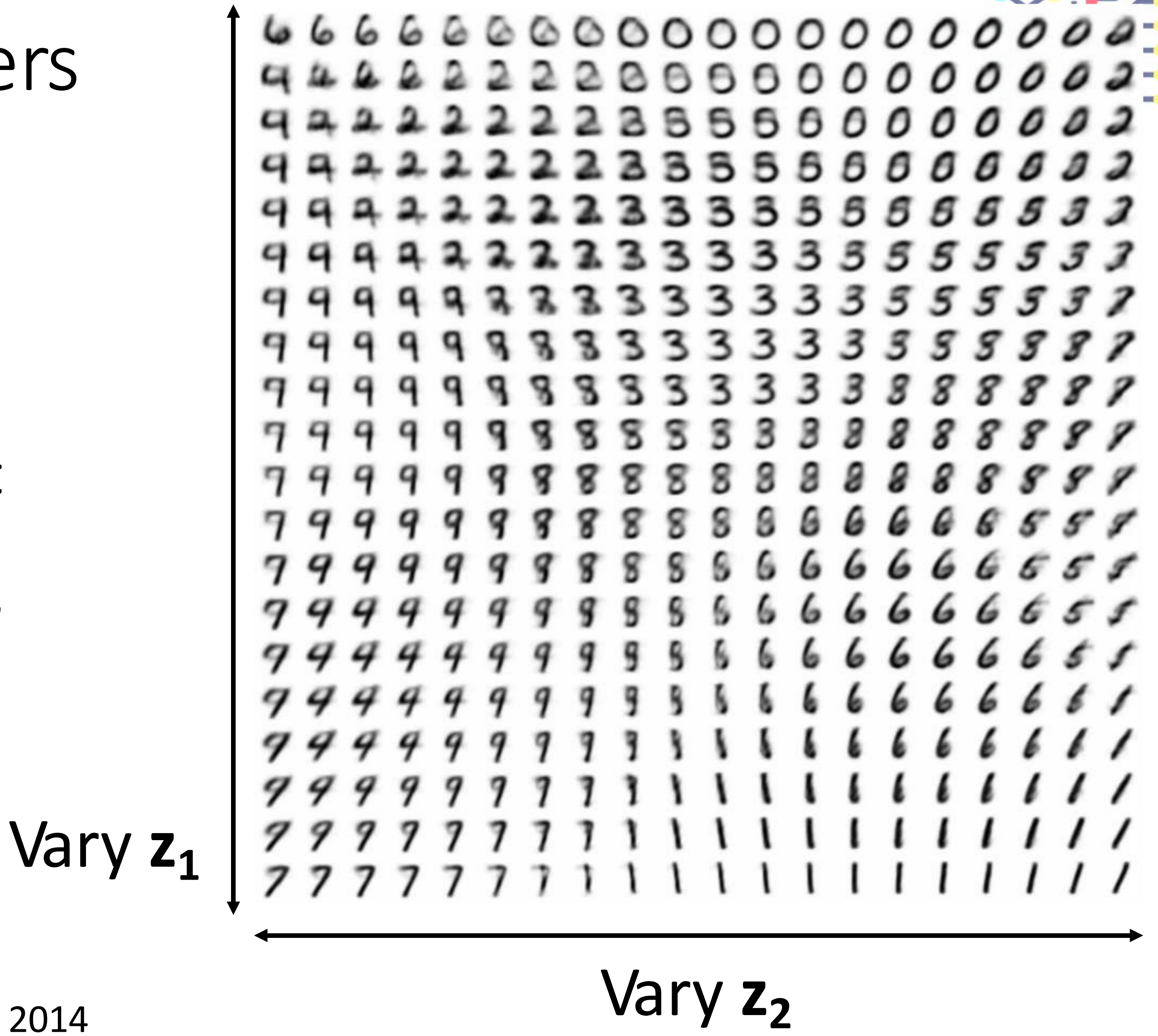
Labeled Faces in the Wild

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

Variational Autoencoders

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

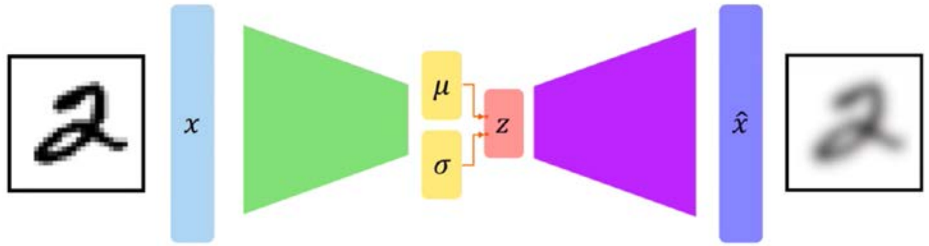
"Disentangling factors of variation"



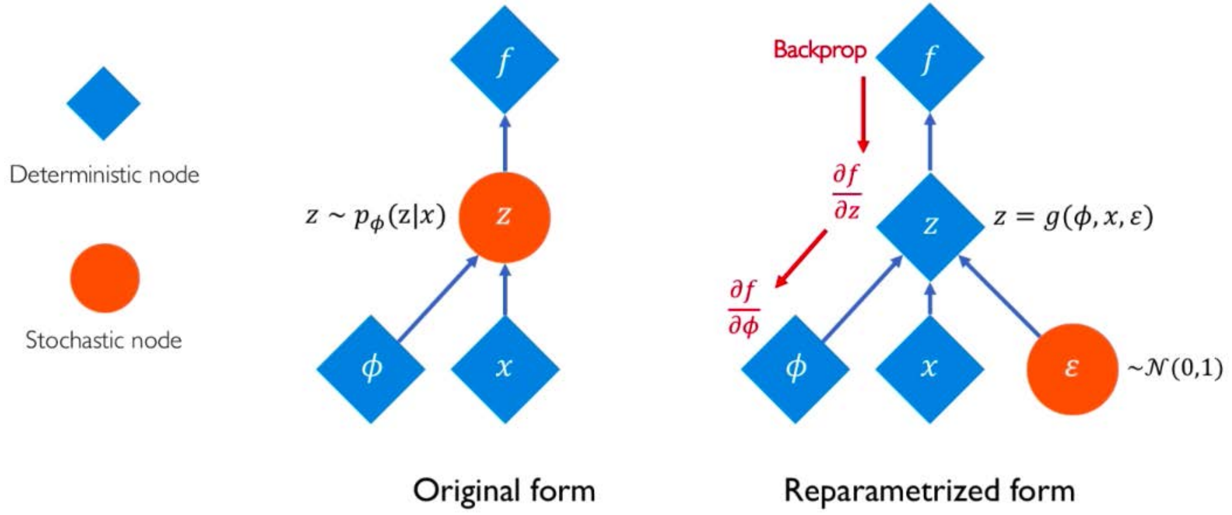
Kingma and Welling, Auto-Encoding Variational Beyes, ICLR 2014

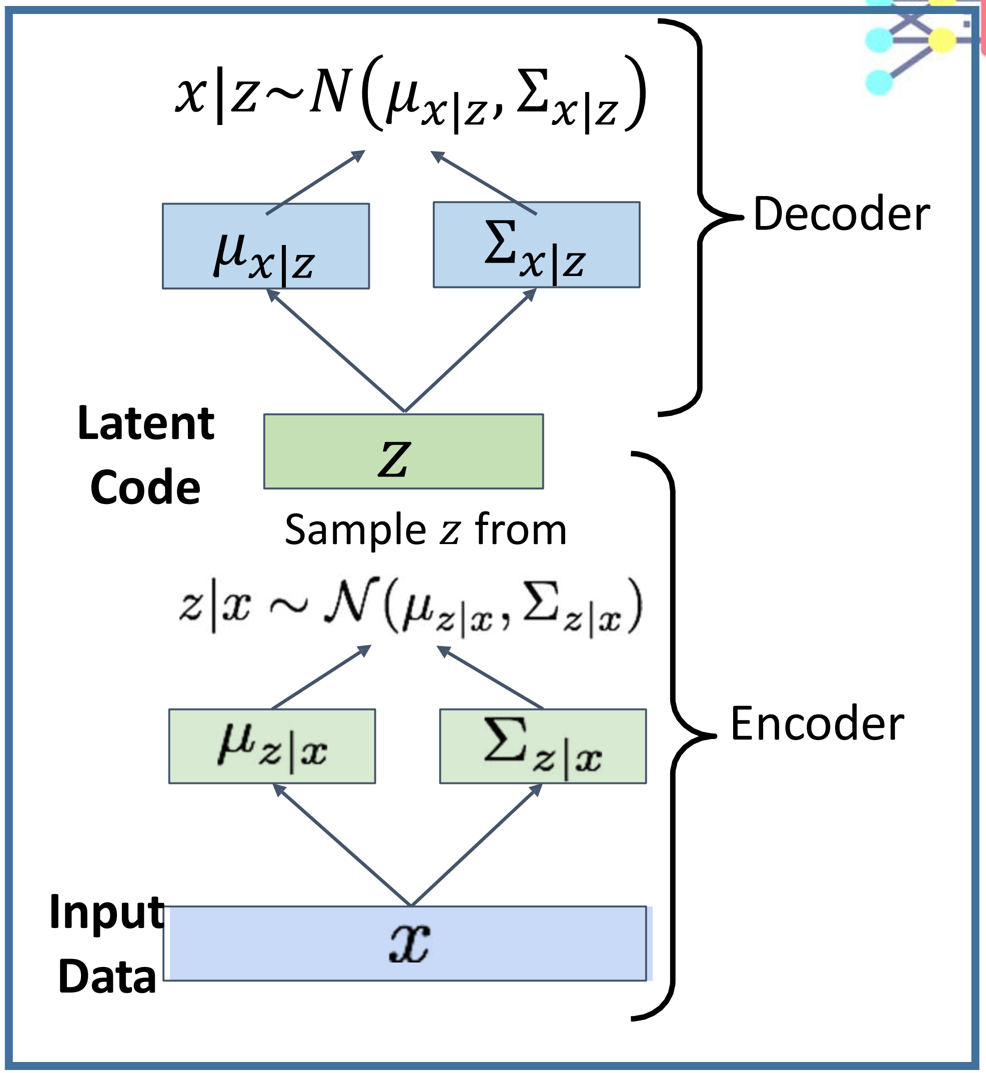
Variational Autoencoders

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

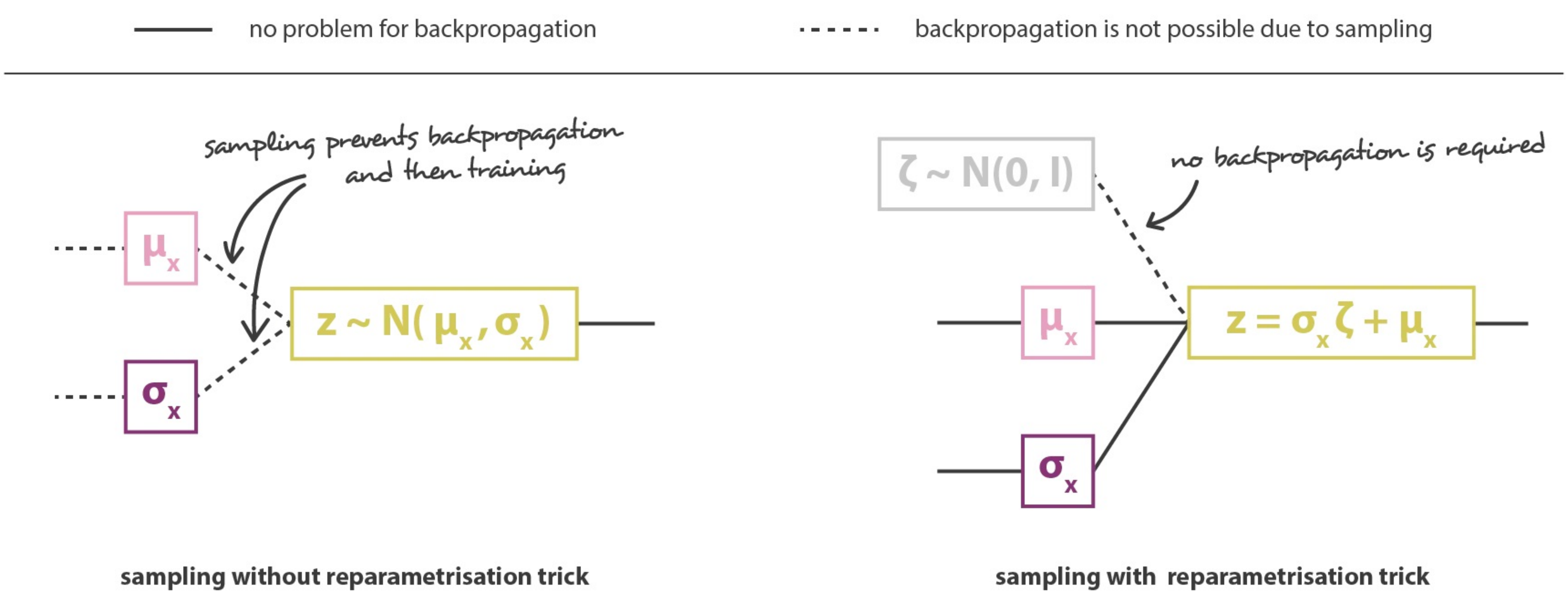


Problem: Cannot backpropagate gradients through sampling layers





Reparameterization

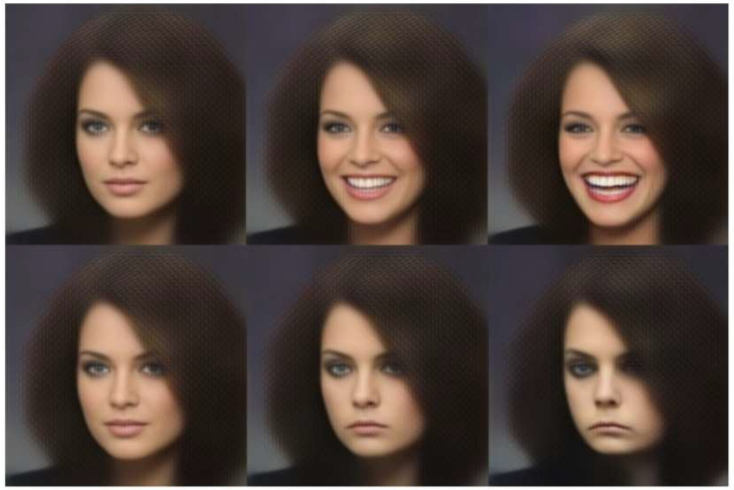


VAE Takeaways

- An autoencoder with statistical constraints on the hidden representation
  - The encoder is a statistical model that computes the parameters of a Gaussian
  - The decoder converts samples from the Gaussian back to the input
- The decoder is a generative model that, when excited by standard Gaussian inputs, generates samples similar to the training data

VAE and latent spaces

- The latent space often captures underlying structure in the data x in a smooth manner
    - Varying z continuously in different directions can result in plausible variations in the drawn output
- Reproductions of an input x can be manipulated by wiggling z around its expected value $\mu(x)$



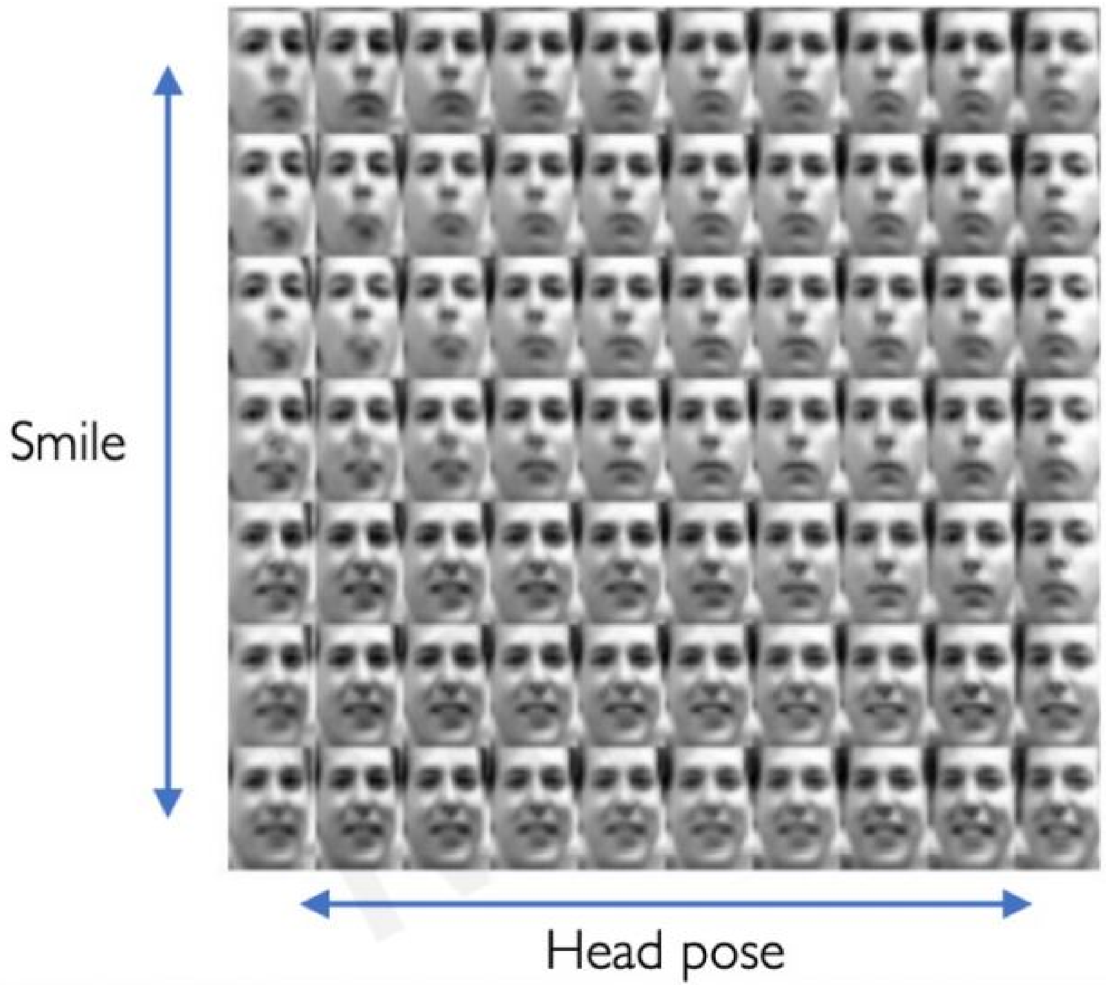
VAEs : Latent perturbation

Slowly increase or decrease a **single latent variable**
Keep all other variables fixed

Head pose

Different dimensions of z encodes **different interpretable latent features**

VAEs : Latent perturbation



Ideally, we want latent variables that are uncorrelated with each other

Enforce diagonal prior on the latent variables to encourage independence

Disentanglement

Variational Autoencoders
Latent Variables

- Independence of z dimensions makes it easy to generate instances wrt complex distributions via decoder g
- Latent variables can be thought of as values of attributes describing inputs
  - E.g., for MNIST, latent variables might represent "thickness", "slant", "loop closure"