Introduction to Probabilistic Deep Learning and Generative modeling

CSE 849 Deep Learning
Spring 2025

Zijun Cui

Slides are revised based on 598 University of Michigan by Justin Johnson and Probabilistic Deep Learning RPI by Qiang Ji

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

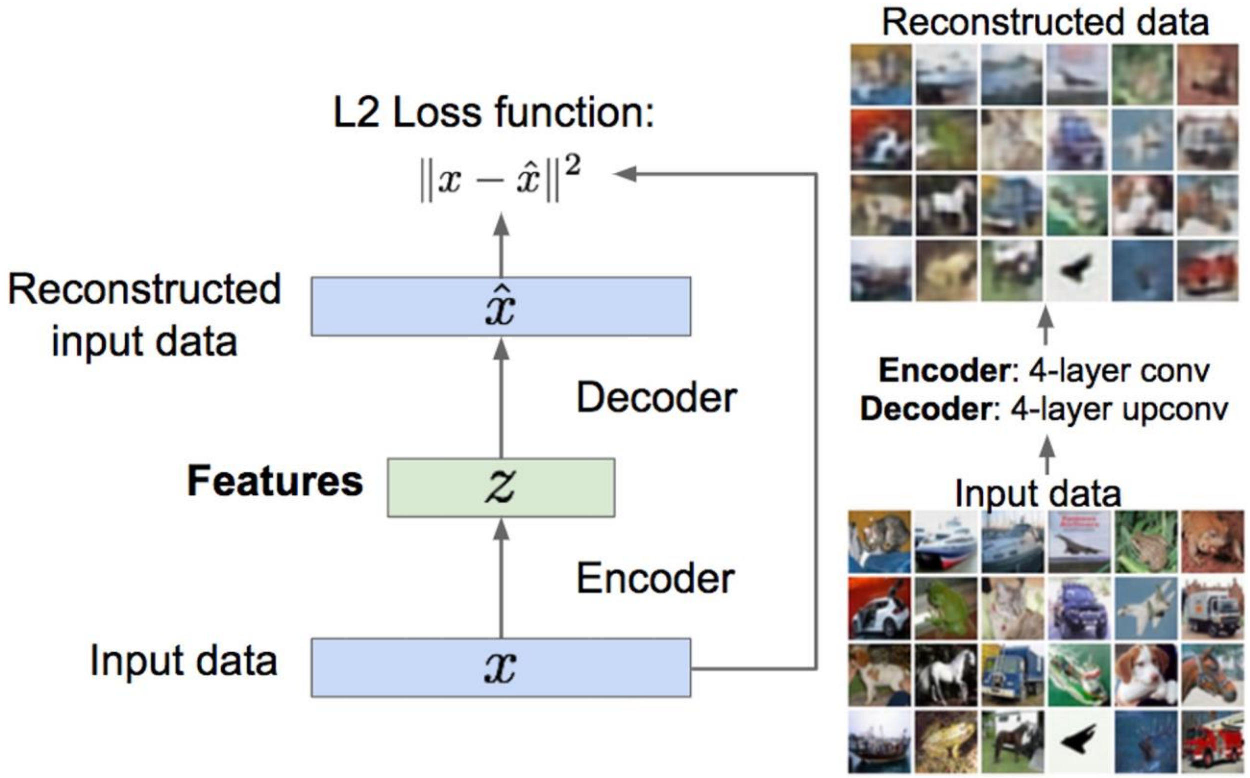
Just data, no labels!

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

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

Supervised vs Unsupervised Learning

Feature Learning
(e.g. autoencoders)



Unsupervised Learning

Data: x

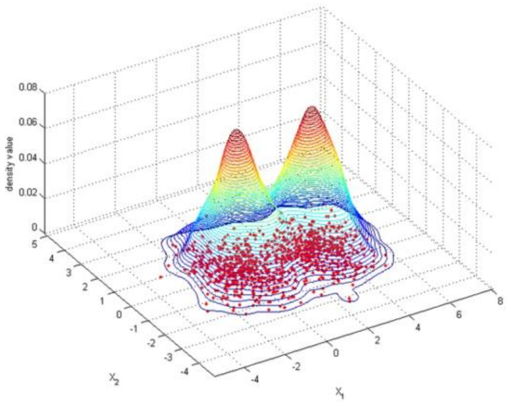
Just data, no labels!

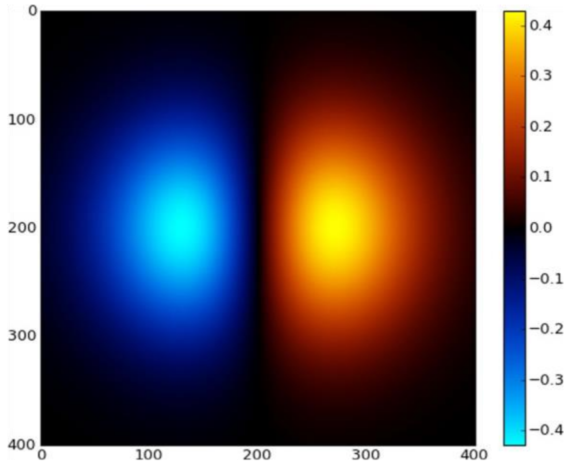
Goal: Learn some underlying hidden *structure* of the data

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

Supervised vs Unsupervised Learning

Density Estimation





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.

Introduction to Probabilistic Machine Learning

- Deterministic machine learning
- Probabilistic machine learning
- Bayesian machine learning

**By saying machine learning, the techniques include but not limited to deep learning*

Deterministic Machine Learning

$$\mathbf{x} \longrightarrow \boxed{f(\mathbf{x},\boldsymbol{\theta},\boldsymbol{\alpha})} \longrightarrow \mathbf{y}$$

- $\mathbf{x} \ni \mathcal{X}$-input, $\mathbf{y} \ni \mathcal{Y}$-output
- f() the mapping function that maps input to output
- $\boldsymbol{\theta} \ni \Theta$ and $\alpha \ni \mathrm{A}$ are the parameters and hyper-parameters of the mapping function
- The goal is to learn the mapping function parameter $\boldsymbol{\theta}$ and tune the hyper-parameter α to map input **x** to output **y**
- After training, an optimal set of parameters Θ^* are obtained

Issues:

- It cannot effectively capture the uncertainties in input data (X), in the model parameters (Θ), and in the output (y). Hence, it cannot quantify the output confidence.
- Its prediction is based on point estimation of the parameters (Θ^*) only and it hence tends to overfit.

Probabilistic Machine Learning

Probabilistic machine learning constructs the probability distribution of input and output $P_\Theta(x, y)$, where $\Theta \ni Q$,$\mathbf{X} \ni X$ and $\mathbf{Y} \ni Y$ and use P_Θ to perform the prediction.

- Generative approach – $p(\mathbf{X},\mathbf{Y}|\ \Theta)$

  Learn the parameters $\boldsymbol{\Theta}^*$ that characterizes the joint probability of **X** and **Y**, and performs the classification/regression using

  $$\mathbf{Y}^*=\text{argmax}_{\mathbf{Y}}\ p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\Theta}^*)$$

  Or only $p(\mathbf{X}|\ \Theta)$ if no labels, i.e., Y

- Discriminative approach – $p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\Theta})$

  Learn the parameters $\boldsymbol{\Theta}^*$ that characterizes the conditional joint probability of **Y** given **X** , and performs the classification/regression using

  $$\mathbf{Y}^*=\text{argmax}_{\mathbf{Y}}\ p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\Theta}^*)$$

- Θ are parameters that characterize the probability distributions, and it is learnt through a training process.

Probabilistic Machine Learning
-- Pros and Cons

- Pros:
  - Capable of capturing the input and output data uncertainties
  - Can quantify the output confidence and predict output accuracy

- Cons:
  - Computationally more expensive during both training and prediction; more difficult to train the probabilistic loss function
  - Still require a training process to learn the parameters
  - Prediction is still based on point-estimation
  - Cannot quantify the model uncertainty, i.e., that of Θ.

Bayesian Machine Learning

Bayesian Machine models the posterior distribution of the model parameters Θ, P(Θ|**D**, **a**), where **D** are the training data and **α** are the hyper-parameters that specify the prior probability of Θ, p(Θ | **α**).

Given an input **X**, prediction of output **Y** can be done through empirical or full Bayesian inference

- Empirical inference

$$\mathbf{Y}^* = \arg\max_{\mathbf{Y}} p(\mathbf{Y} \mid \mathbf{X}, \mathbf{D}, \boldsymbol{\alpha}^*) = \arg\max_{\mathbf{Y}} \int_{\boldsymbol{\Theta}} p(\mathbf{Y} \mid \mathbf{X}, \boldsymbol{\Theta})\, p(\boldsymbol{\Theta} \mid \mathbf{D}, \boldsymbol{\alpha}^*)\, d\boldsymbol{\Theta}$$

- Full Bayesian inference

$$\mathbf{Y}^* = \operatorname*{argmax}_{\mathbf{Y}} p(\mathbf{Y} \mid \mathbf{X}, \mathbf{D}) = \operatorname*{argmax}_{\mathbf{Y}} \int_{\boldsymbol{\Theta}} \int_{\boldsymbol{\alpha}} p(\mathbf{Y} \mid \mathbf{X}, \boldsymbol{\Theta})\, p(\boldsymbol{\Theta} \mid \mathbf{D}, \boldsymbol{\alpha})\, p(\boldsymbol{\alpha} \mid \mathbf{D})\, d\boldsymbol{\Theta}\, d\boldsymbol{\alpha}$$

Bayesian Machine Learning (cont’d)

- **Compared to probabilistic machine learning, Bayesian learning has the following advantages:**

  – Bayesian inference does not need learn parameters Θ
  – Instead of performing point-based prediction, Bayesian inference performs output prediction using all parameters through parameter integration, hence avoiding the overfitting problem.
  – Bayesian inference produces not only outputs but also generates their distribution, based on which we can derive both data and model uncertainties.

- **Bayesian learning has the following disadvantages:**

  – It needs either manually specify or learn the hyper-parameters. Manual specification of the hyper-parameters is inaccurate, while automatic hyper-parameter learning is computationally complex.
  – Bayesian inference requires integration over all parameters as well as the hyper-parameters, which is computationally intractable and cannot scale up well for a large number of parameters.
  – Approximated and inaccurate solutions are often used to approximate the parameter integration, hence leading to non-optimal /inaccurate solutions.

Introduction to Probabilistic Machine Learning

- Deterministic machine learning

- **Probabilistic machine learning**
  - **Generative**
  - **Discriminative**

- Bayesian machine learning

Discriminative vs Generative Models

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

Generative Model: Learn a probability distribution p(x)

*assume no label Y

Conditional Generative Model: Learn p(x|y)

Data: x

Label: y

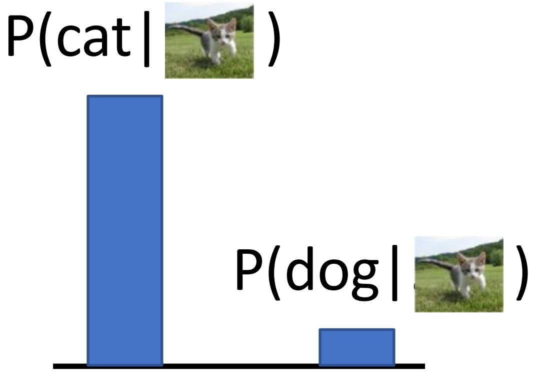
Cat

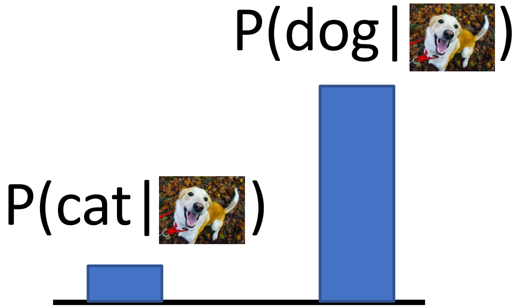
Discriminative vs Generative Models

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

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)

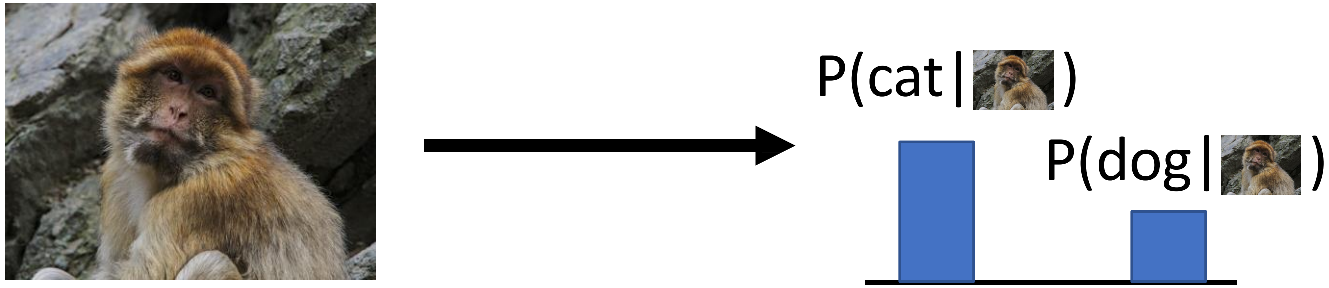




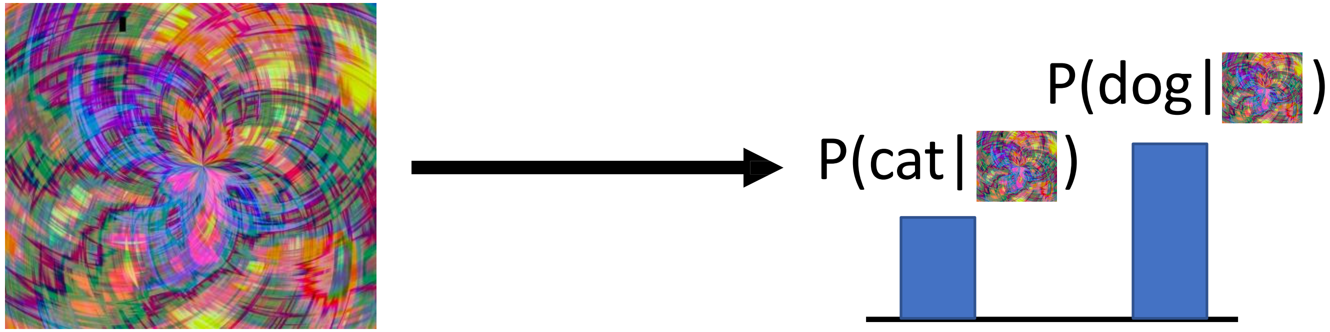
Discriminative model: estimate the distribution over the possible labels y.
But no probabilistic relationships between **images** x

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)

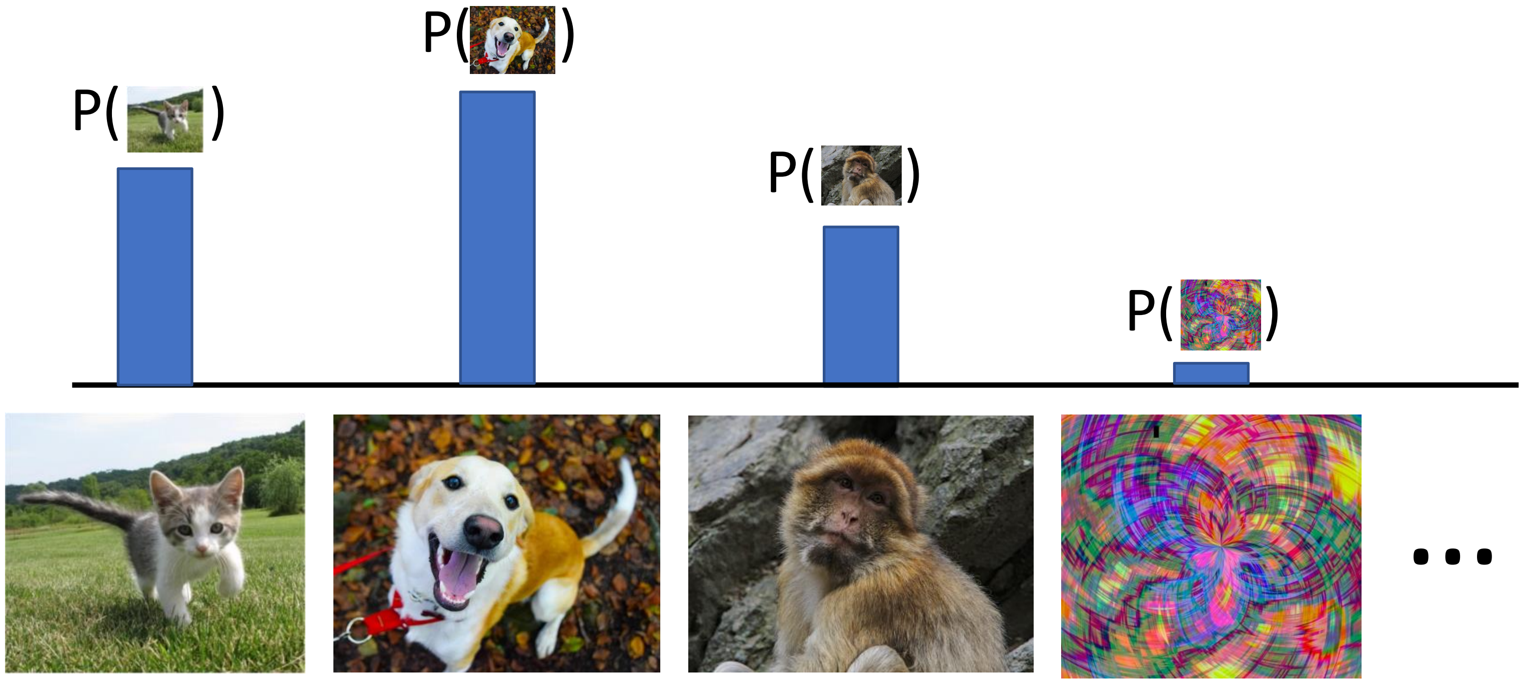
Discriminative model: No way for the model to handle unreasonable inputs; it must give label distributions for all images

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)



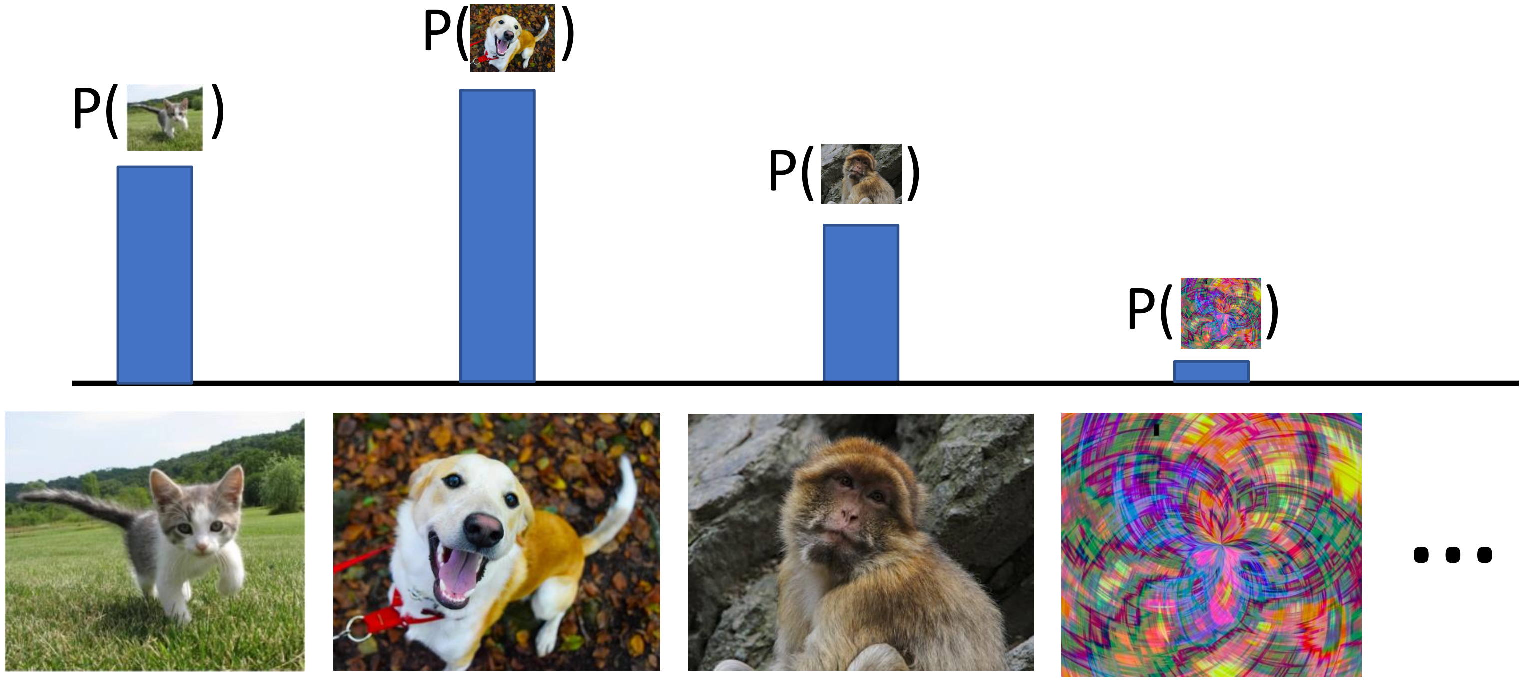
Generative model: estimate a distribution over all images. All possible images compete with each other for probability mass

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)



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

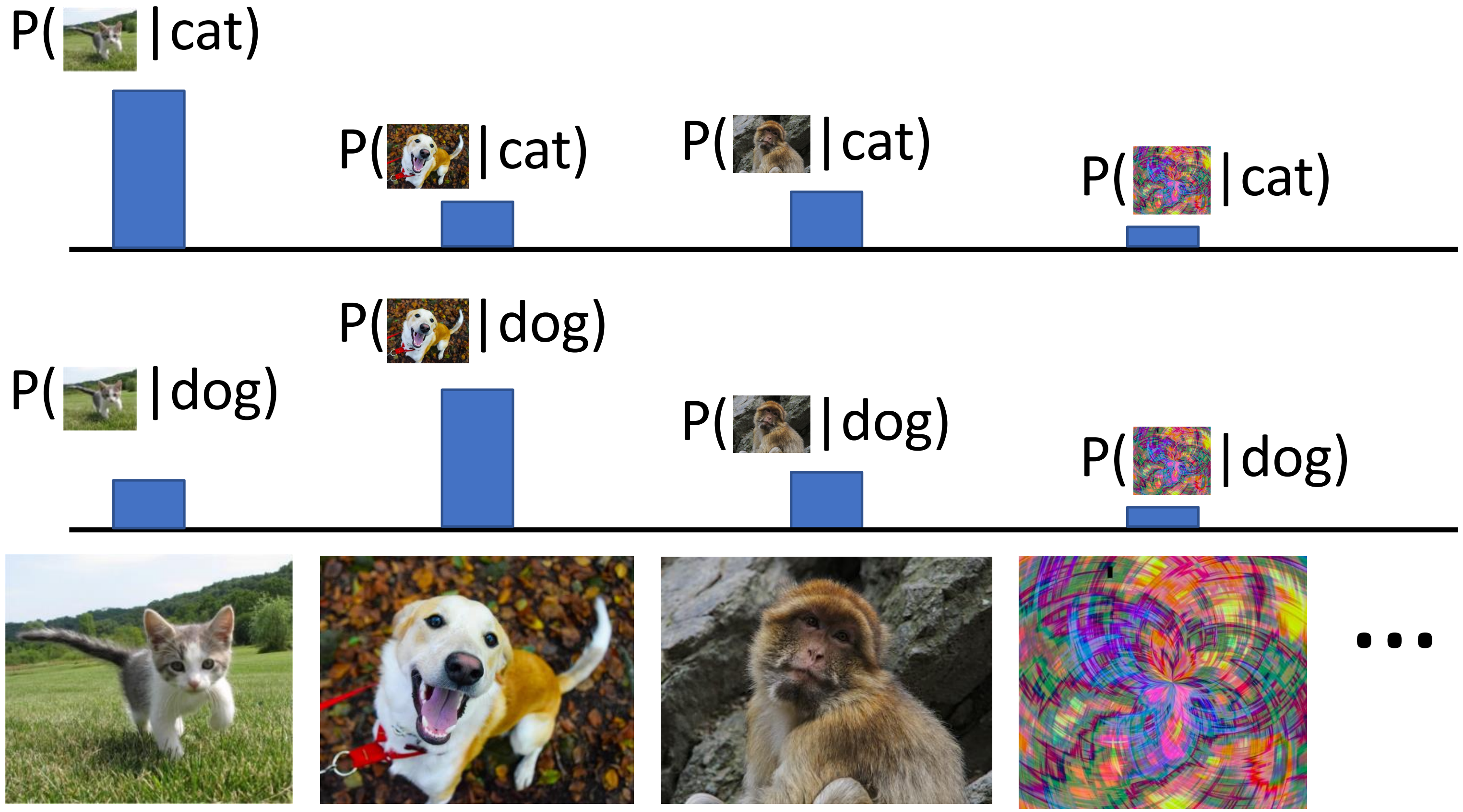
Model can “reject” unreasonable inputs by assigning them small values

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:**

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

We can build a conditional generative model from other components

What can we do with a 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 is revised based on Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Taxonomy of Generative Models

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

Model can compute p(x)

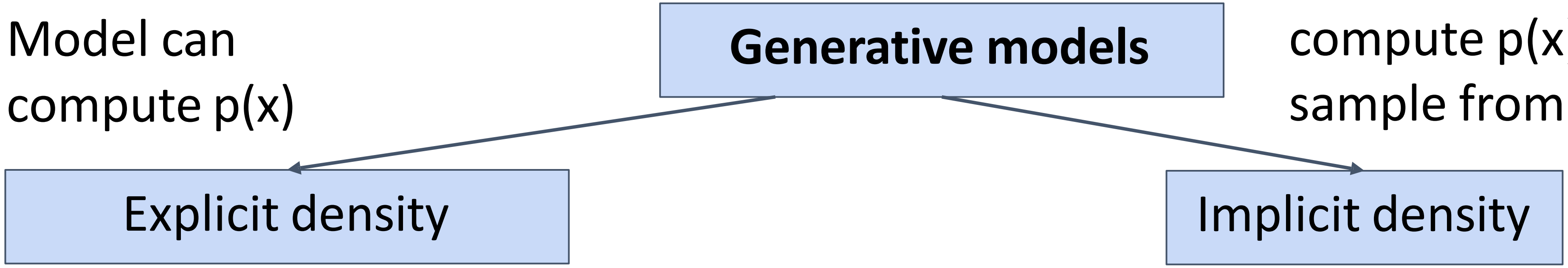


Figure is revised based on Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Taxonomy of Generative Models

Model can compute p(x)

Generative models

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

Explicit density

Implicit density

Can compute approximation to p(x)

Tractable density

Approximate density

Can compute p(x)
- Autoregressive
- Normalizing flows

Figure is revised based on Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Taxonomy of Generative Models

Model can compute p(x)

Generative models

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

Explicit density

Implicit density

Can compute approximation to p(x)

Tractable density

Approximate density

Can compute p(x)

- Autoregressive
- Normalizing flows

Variational Autoencoder

Energy-based models

Diffusion models

Figure is revised based on Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Taxonomy of Generative Models

Model can compute p(x)

Generative models

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

Explicit density

Can compute approximation to p(x)

Implicit density

Generative Adversarial Networks (GANs)

Tractable density

Approximate density

Can compute p(x)

- Autoregressive
- Normalizing flows

Variational Autoencoder

Energy-based models

Diffusion models

Figure is revised based on Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Taxonomy of Generative Models

Model can compute p(x)

Generative models

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

Explicit density

Can compute approximation to p(x)

Implicit density

Generative Adversarial Networks (GANs)

Tractable density

Approximate density

Can compute p(x)
- Autoregressive
- Normalizing flows

Variational Autoencoder

Energy-based models

Diffusion models

We will talk about these three models

Figure is revised based on Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

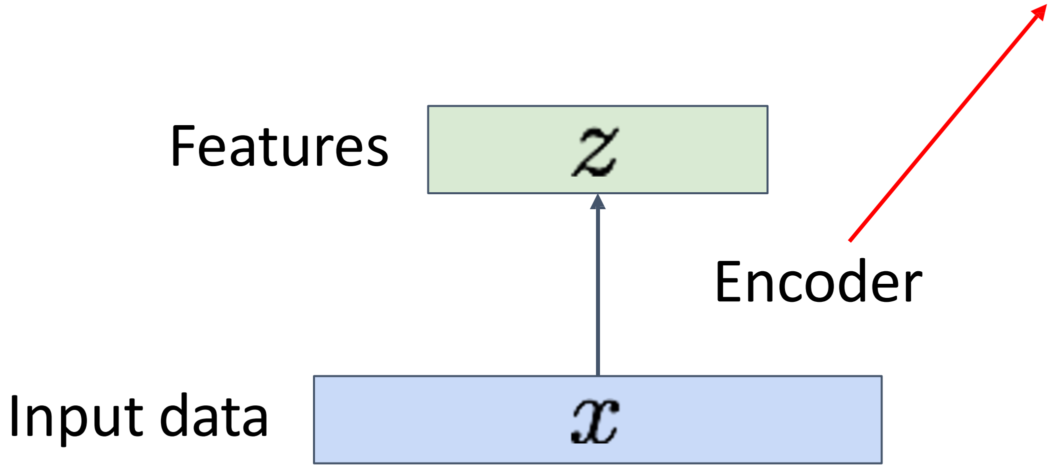
Variational Autoencoders

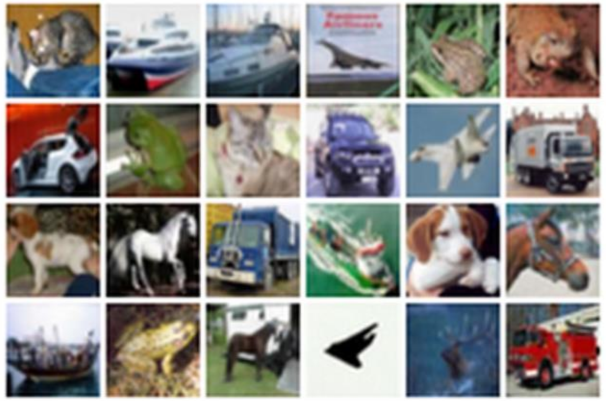
(Regular, non-variational) Autoencoders

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

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

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





Input Data

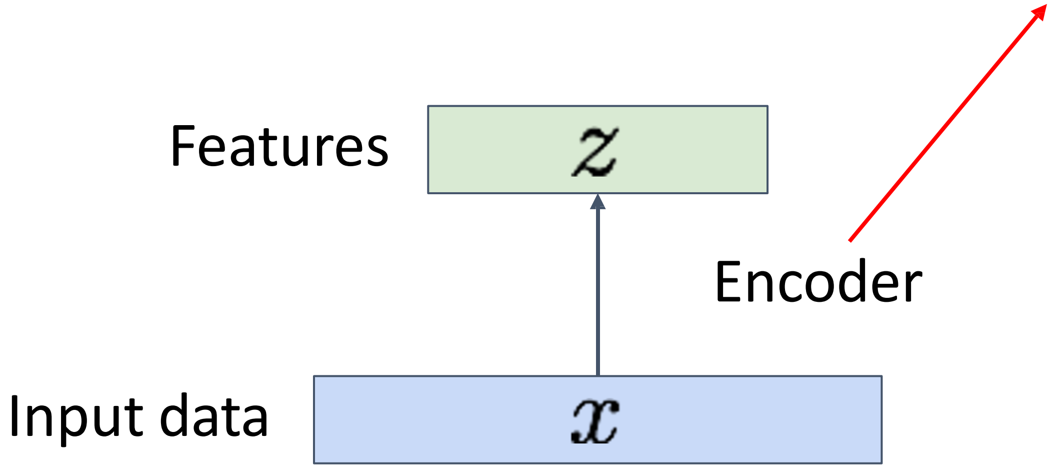
(Regular, non-variational) Autoencoders

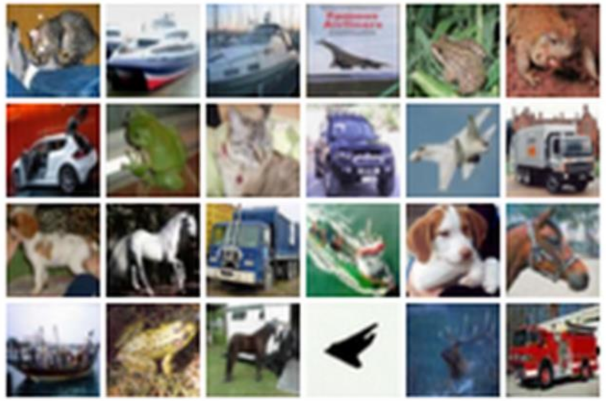
Problem: How can we learn this feature transform from raw data?

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

But we can't observe features!

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





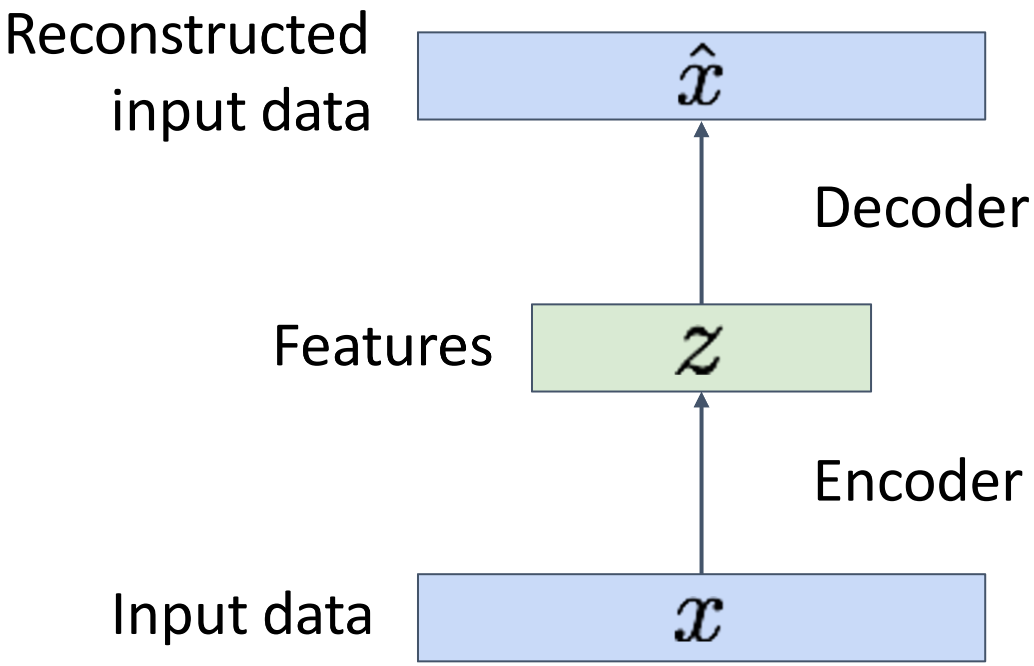
Input Data

(Regular, non-variational) Autoencoders

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

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

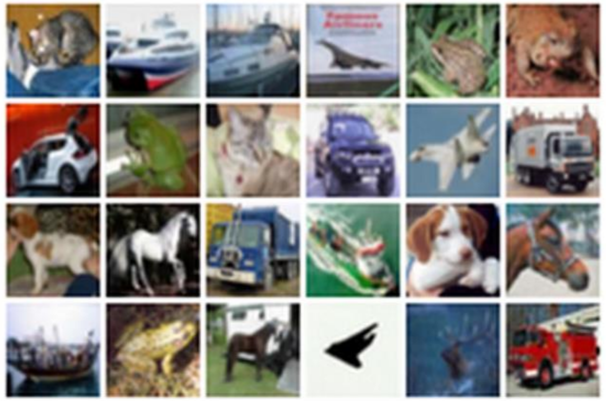
"Autoencoding" = encoding itself



Originally: Linear + nonlinearity (sigmoid)

Later: Deep, fully-connected

Later: ReLU CNN (upconv)



Input Data

(Regular, non-variational) Autoencoders

Loss: L2 distance between input and reconstructed data.

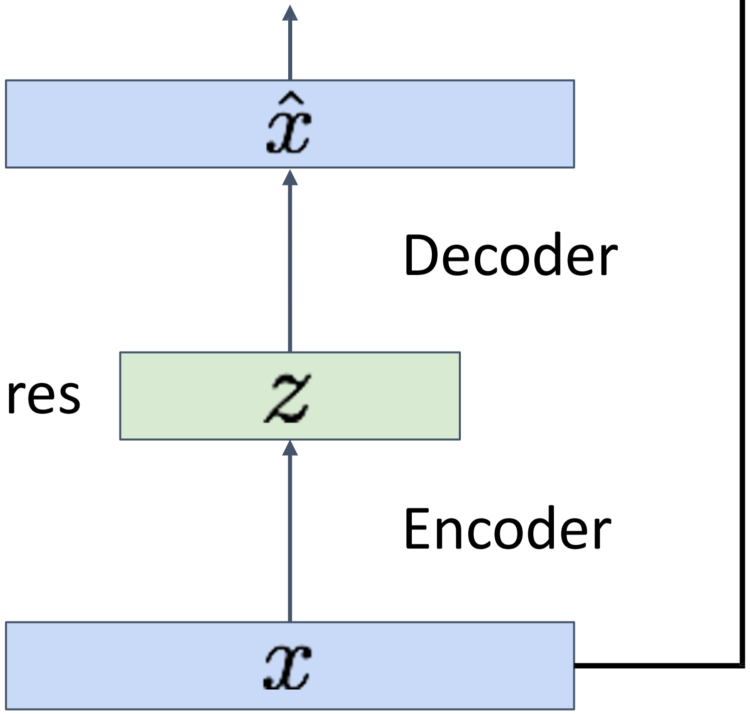
Loss Function

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

Does not use any labels

Just raw data

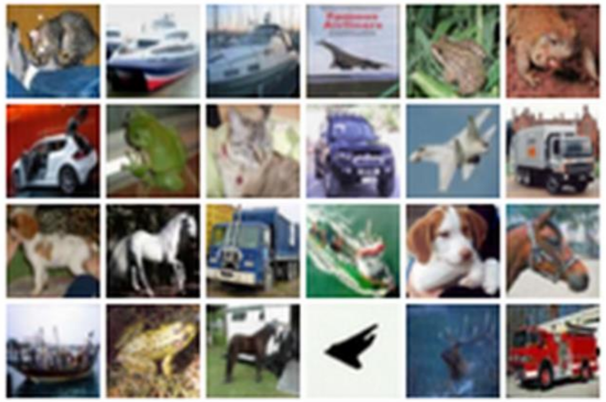
Reconstructed input data



Features

Input data

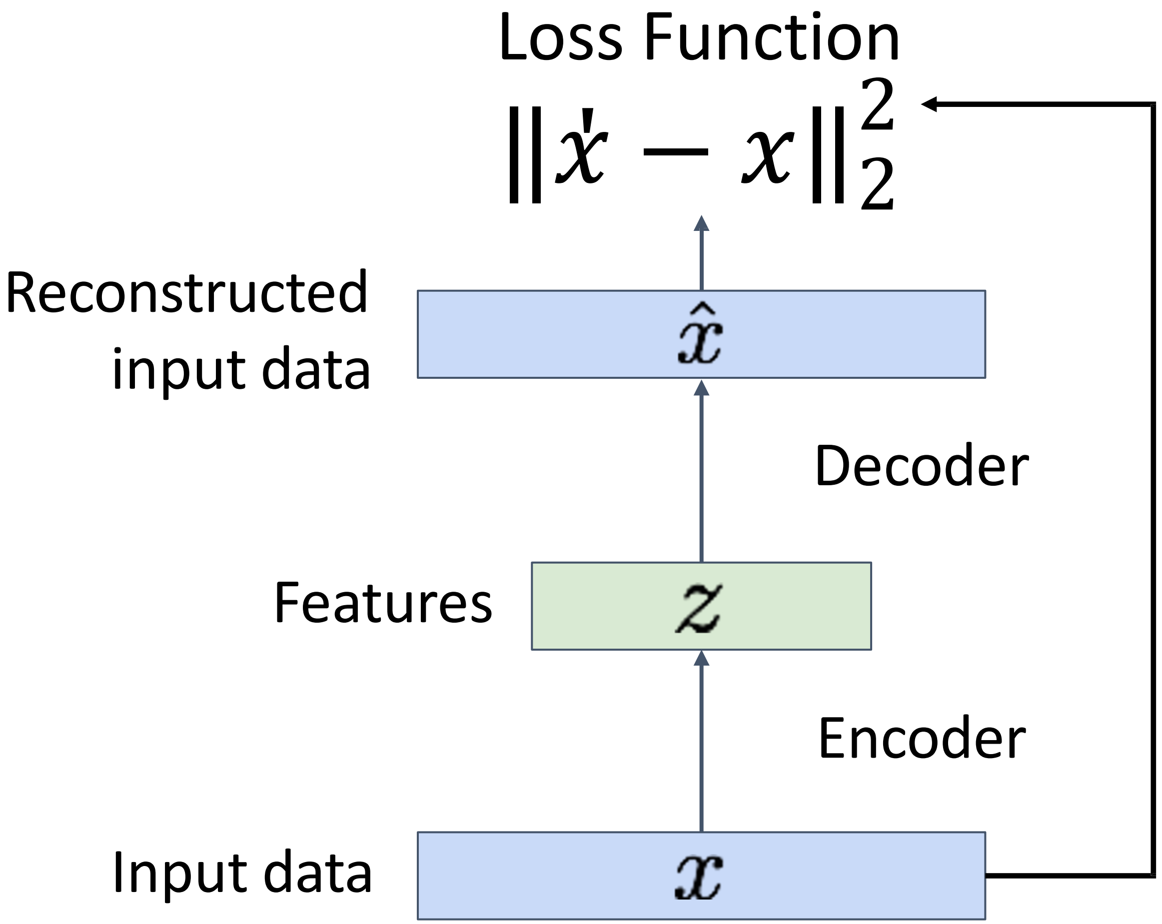
Features need to be **lower dimensional** than the data

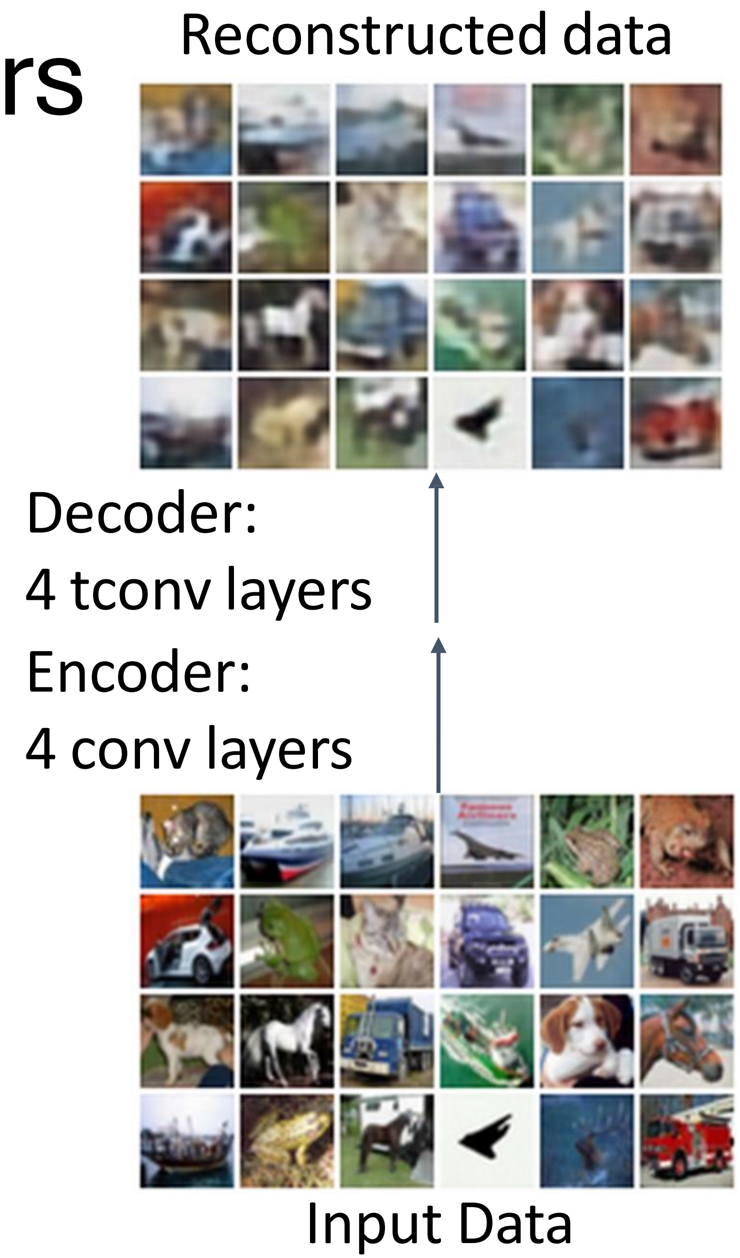


Input Data

(Regular, non-variational) Autoencoders

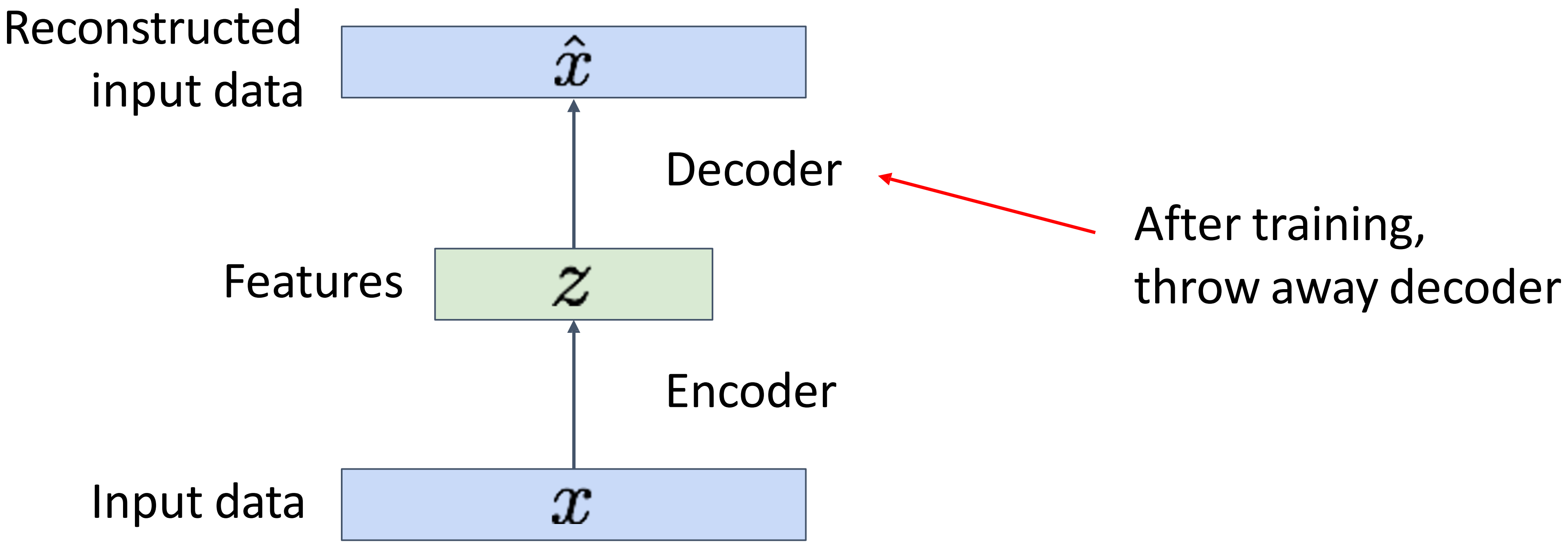
Loss: L2 distance between input and reconstructed data.





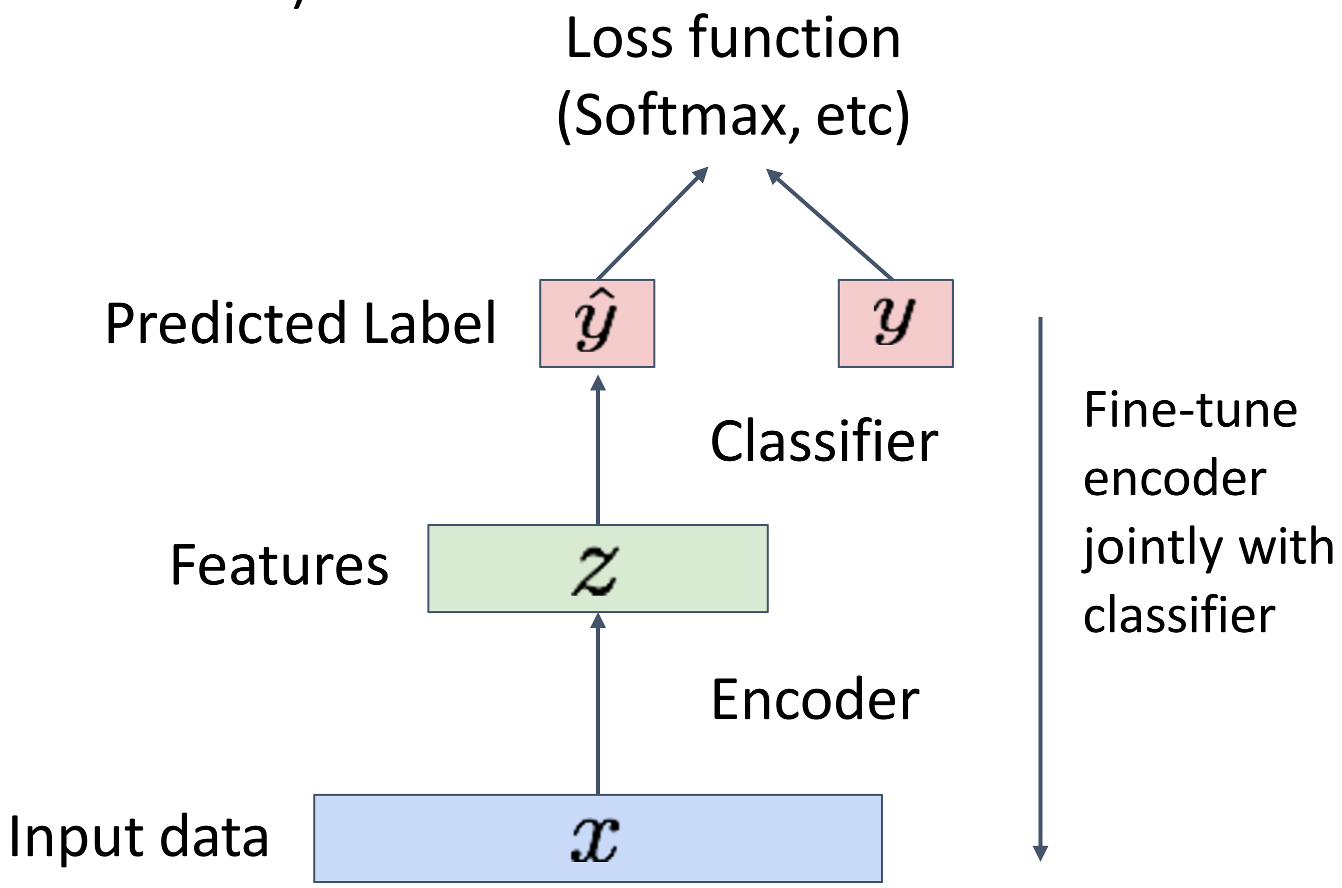
(Regular, non-variational) Autoencoders

- 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)

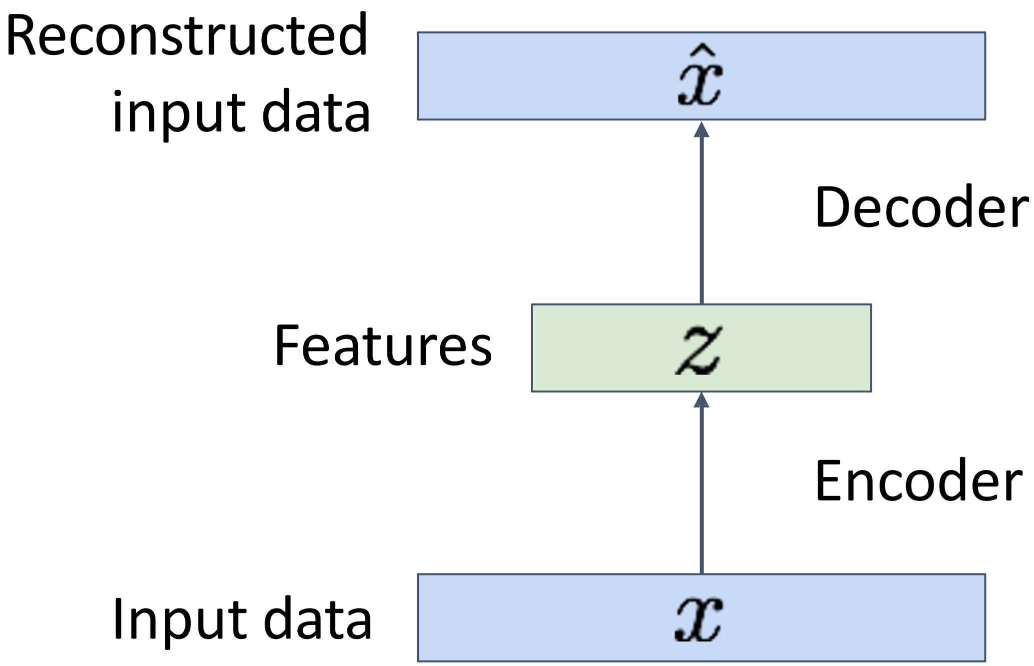


(Regular, non-variational) Autoencoders

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

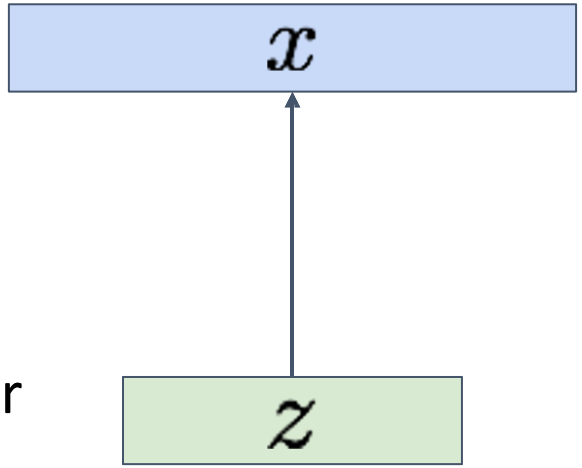
Kingma and Welling, Auto-Encoding Variational Beyes, ICLR 2014

Variational Autoencoders

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

After training, sample new data like this:

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



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

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

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

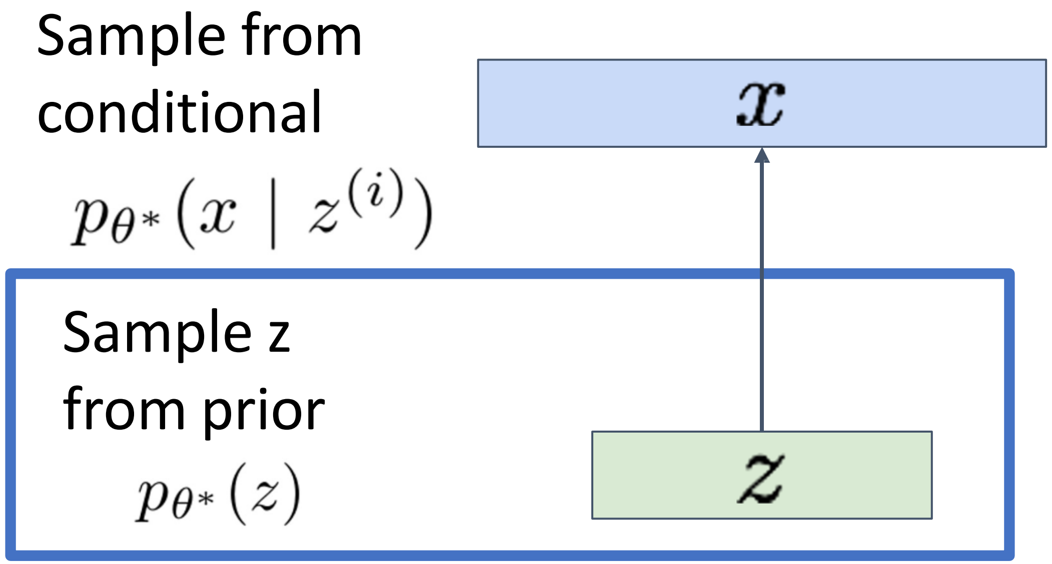
Variational Autoencoders

Probabilistic spin on autoencoders:

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

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

After training, sample new data like this:



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

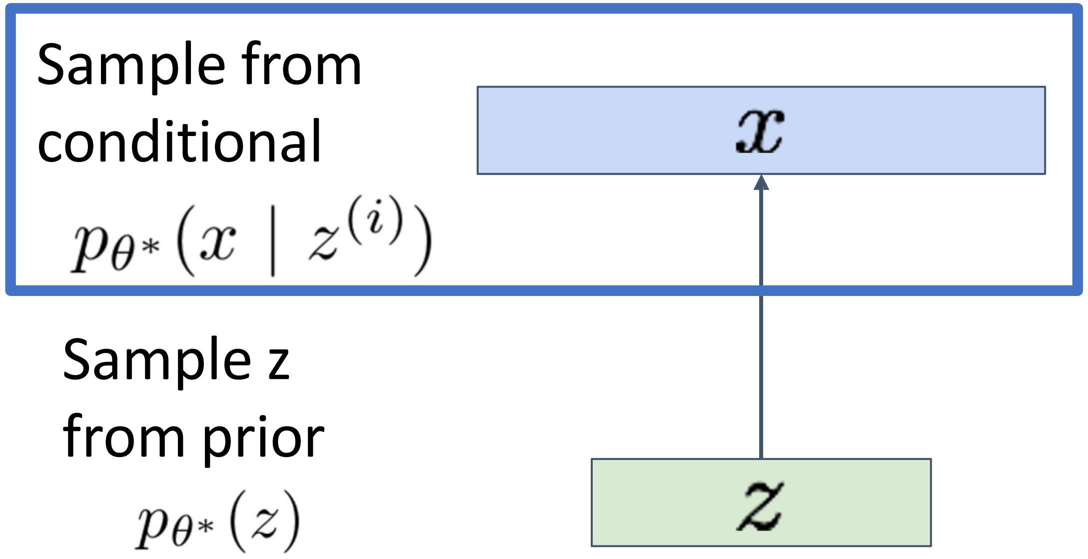
Assume simple prior p(z), e.g. Gaussian

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.

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

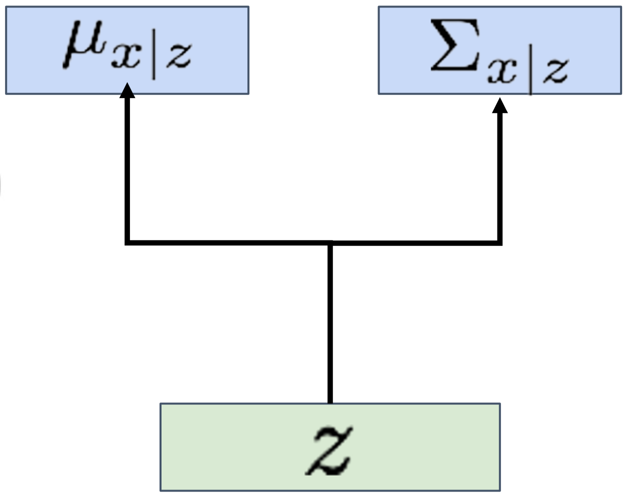
Variational Autoencoders

Sample from conditional

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

Sample z from prior

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



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 $\Sigma_{x|z}$

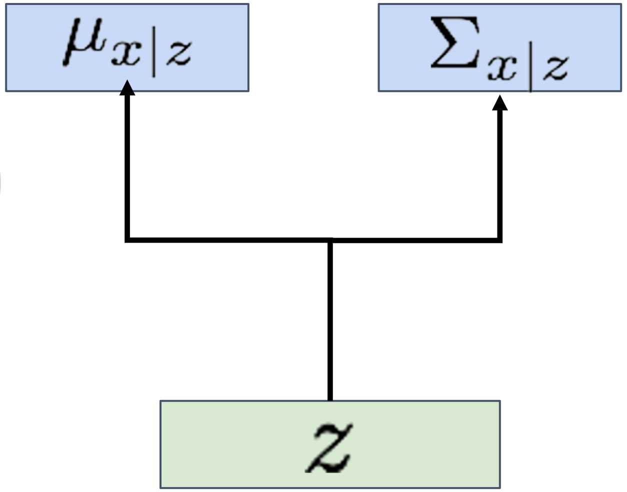
Variational Autoencoders

Sample from conditional

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

Sample z from prior

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



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)

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

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

can compute this with decoder network

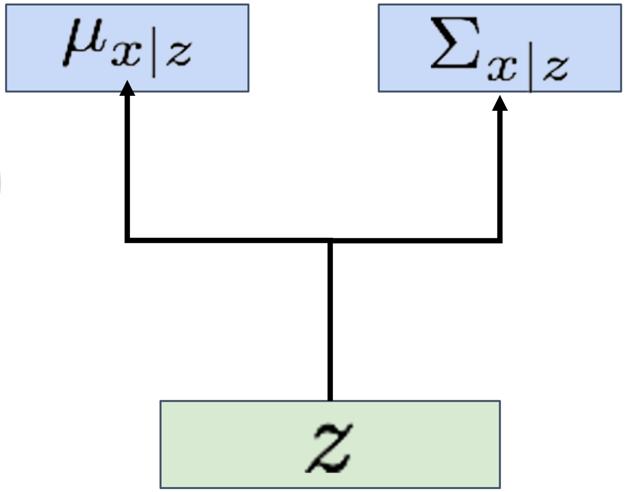
Variational Autoencoders

Sample from conditional

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

Sample z from prior

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



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)

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

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

we assumed Gaussian prior for z

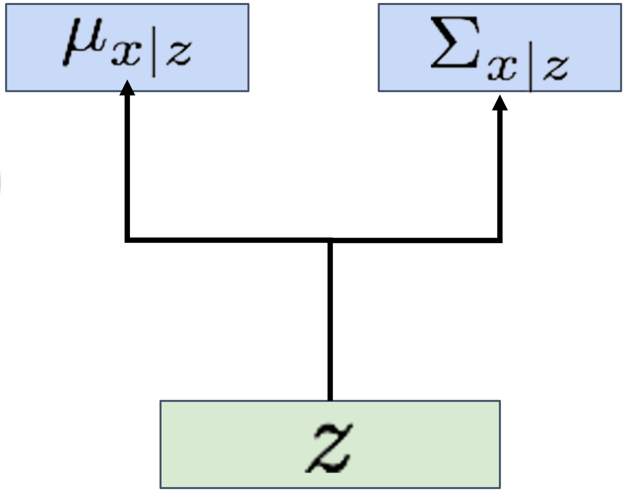
Variational Autoencoders

Sample from conditional

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

Sample z from prior

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



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)

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!

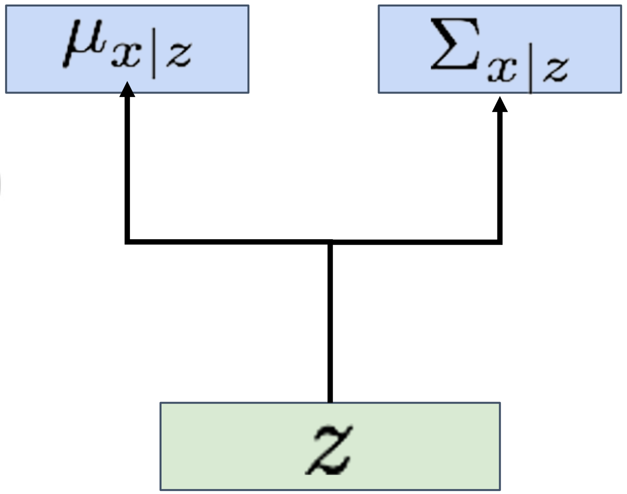
Variational Autoencoders

Sample from conditional

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

Sample z from prior

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



How to train this model?

Basic idea: **maximize likelihood of data**

Another idea: Try Bayes' Rule:

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

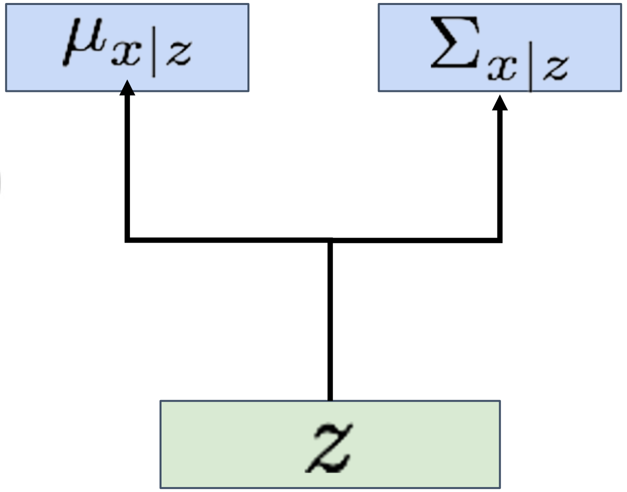
Variational Autoencoders

Sample from conditional

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

Sample z from prior

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



How to train this model?

Basic idea: **maximize likelihood of data**

Another idea: Try Bayes' Rule:

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

compute with decoder network

we assumed Gaussian prior

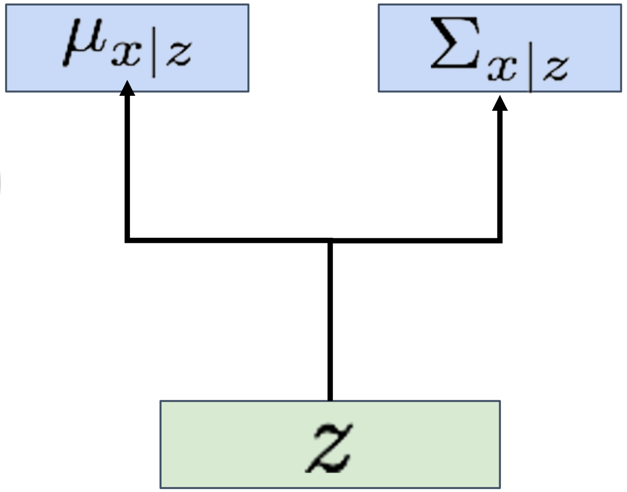
Variational Autoencoders

Sample from conditional

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

Sample z from prior

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



How to train this model?

Basic idea: **maximize likelihood of data**

Another idea: Try Bayes' Rule:

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

Problem: No way to compute this!

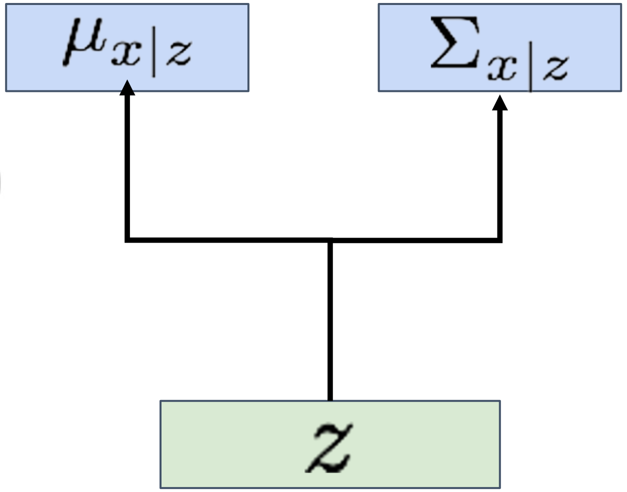
Solution: Train another network **(encoder)** that learns

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

Variational Autoencoders

Sample from conditional

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



Sample z from prior

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

How to train this model?

Basic idea: **maximize likelihood of data**

Another idea: Try Bayes' Rule:

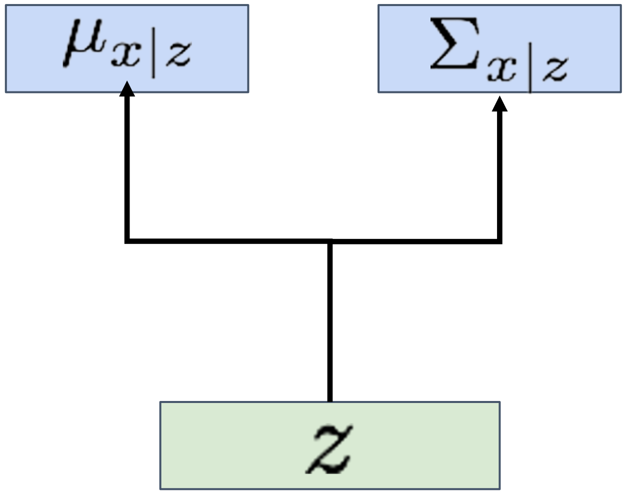
$$p_\theta(x) = \frac{p_\theta(x \mid z)p_\theta(z)}{p_\theta(z \mid x)} \approx \frac{p_\theta(x \mid z)p_\theta(z)}{\boxed{q_\phi(z \mid x)}}$$

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

Variational Autoencoders

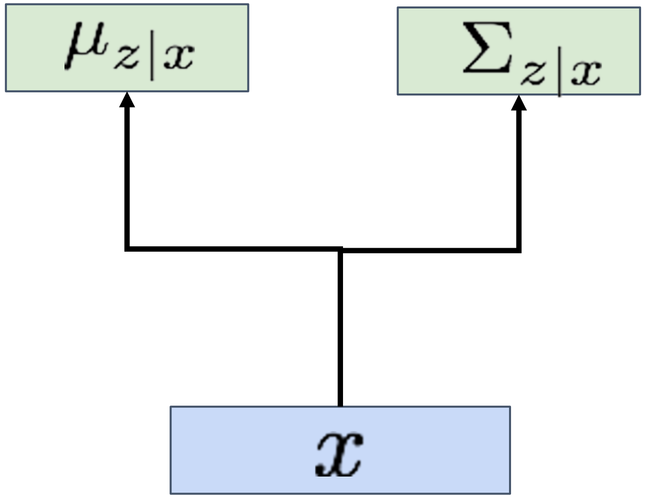
Decoder network inputs latent code z, gives distribution over data x

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



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

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

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

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

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

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

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

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

Variational Autoencoders

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

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

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

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

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

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

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

KL divergence between prior, and
samples from the encoder network

Variational Autoencoders

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

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

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

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

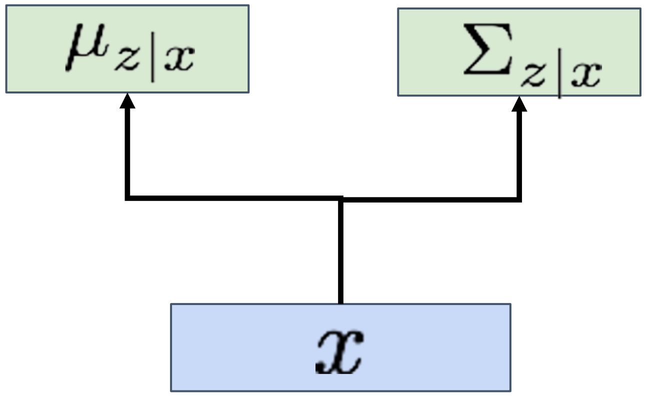
Jointly train **encoder** q and **decoder** p to maximize the **variational lower bound** on the data likelihood

Also called **Evidence Lower Bound (ELBO)**

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

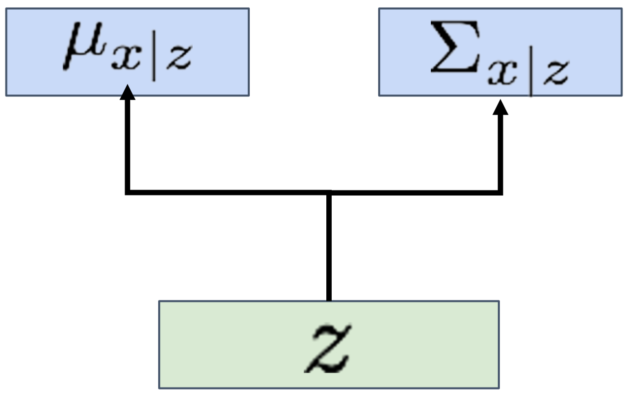
Encoder Network

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



Decoder Network

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



Example: Fully-Connected VAE

x: 28x28 image, flattened to 784-dim vector
z: 20-dim vector

Encoder Network

Decoder Network

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

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