CS 7643: Deep Learning

Topics:

- Generative Models (PixelRNNs, VAEs, GANs)
- Key Ideas
  - AE, Reparameterization, Variational Inference

Dhruv Batra

Georgia Tech

Invited Talk

- Peter Anderson, ANU
  - *Visual Understanding in Natural Language*
  - Co-located as ML@GT Seminar, Nov 27 11am, Nano 1117

PhD student in Computer Vision / Deep Learning

Sydney / Canberra

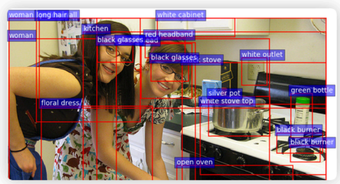
Email

Twitter

LinkedIn

Google Scholar

Publications



Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering

Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, Lei Zhang

preprint arXiv:1707.07998, 2017.

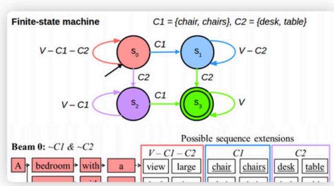
Project | PDF | Code

Tips and Tricks for Visual Question Answering: Learnings from the 2017 Challenge

Damien Teney, Peter Anderson, Xiaodong He, Anton van den Hengel

preprint arXiv:1708.02711, 2017.

PDF | Slides

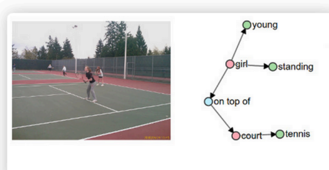


Guided Open Vocabulary Image Captioning with Constrained Beam Search

Peter Anderson, Basura Fernando, Mark Johnson and Stephen Gould

In *Conference on Empirical Methods for Natural Language Processing (EMNLP)*, 2017.

PDF



SPICE: Semantic Propositional Image Caption Evaluation

Peter Anderson, Basura Fernando, Mark Johnson and Stephen Gould

In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016.

Project | PDF | Code

Administrativia

- Poster Presentation: Best Project Award!
  - Wed 11/29, 2-4pm
    - In two sessions
  - Klaus Auditorium
  - Less text, more pictures.

Overview

- Unsupervised Learning
- Generative Models
  - PixelRNN and PixelCNN
  - Variational Autoencoders (VAE)
  - Generative Adversarial Networks (GAN)

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

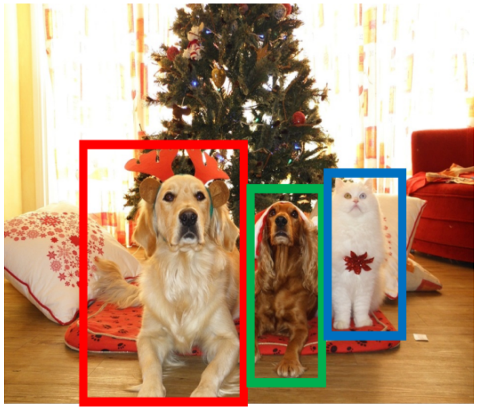
Supervised vs Unsupervised Learning

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.



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.

Semantic Segmentation

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.

x

y *A cat sitting on a suitcase on the floor*

Image captioning

Caption generated using neuraltalk2
Image is CC0 Public domain.

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.

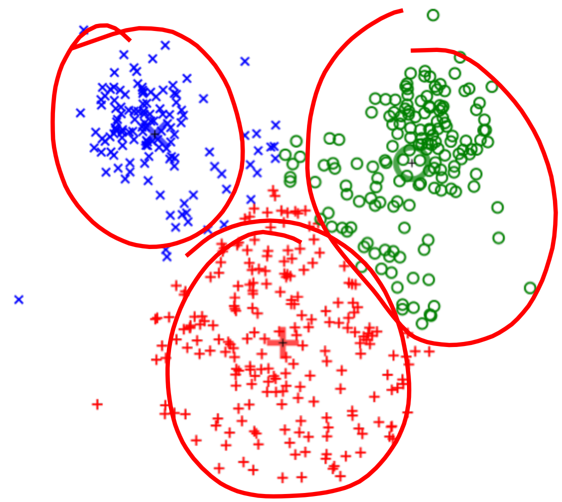
Supervised vs Unsupervised Learning

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

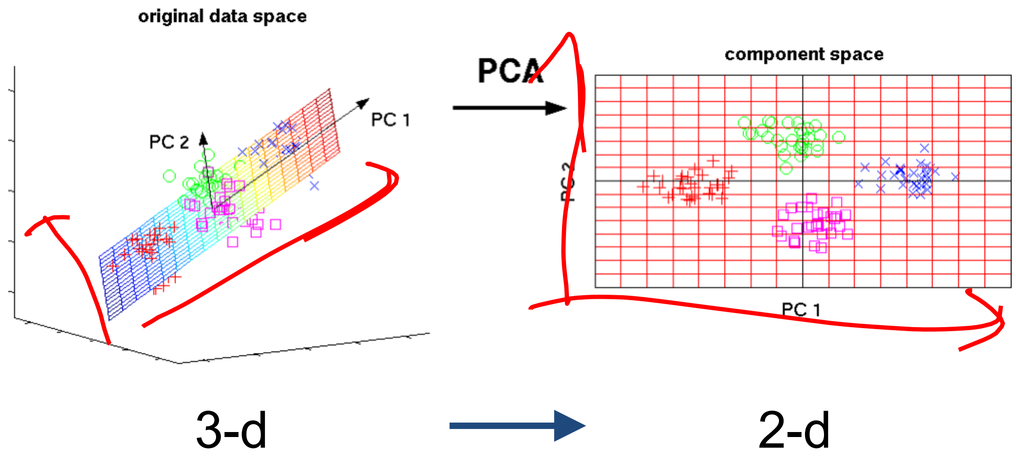
Supervised vs Unsupervised Learning

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)

$P_{model}(x)$

Supervised vs Unsupervised Learning

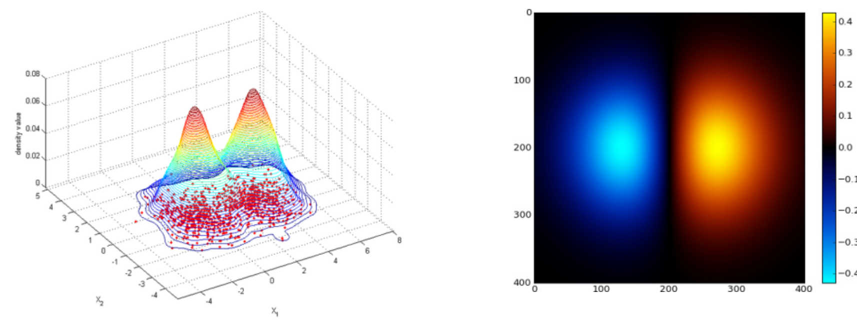
Unsupervised Learning

Data: x
Just data, no labels!

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

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

1-d density estimation



2-d density estimation

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

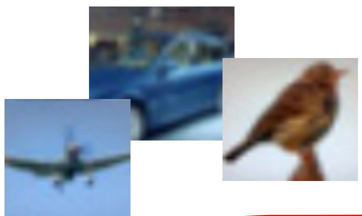
Data: x
Just data, no labels!

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

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

Generative Models

Given training data, generate new samples from same distribution



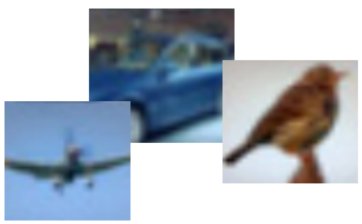
Training data ~ $p_{data}(x)$

Generated samples ~ $p_{model}(x)$

Want to learn $p_{model}(x)$ similar to $p_{data}(x)$

Generative Models

Given training data, generate new samples from same distribution



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

Generated samples ~ $p_{model}(x)$

Want to learn $p_{model}(x)$ similar to $p_{data}(x)$

Addresses density estimation, a core problem in unsupervised learning

Several flavors:

- Explicit density estimation: explicitly define and solve for $p_{model}(x)$
- Implicit density estimation: learn model that can sample from $p_{model}(x)$ w/o explicitly defining it

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Taxonomy of Generative Models

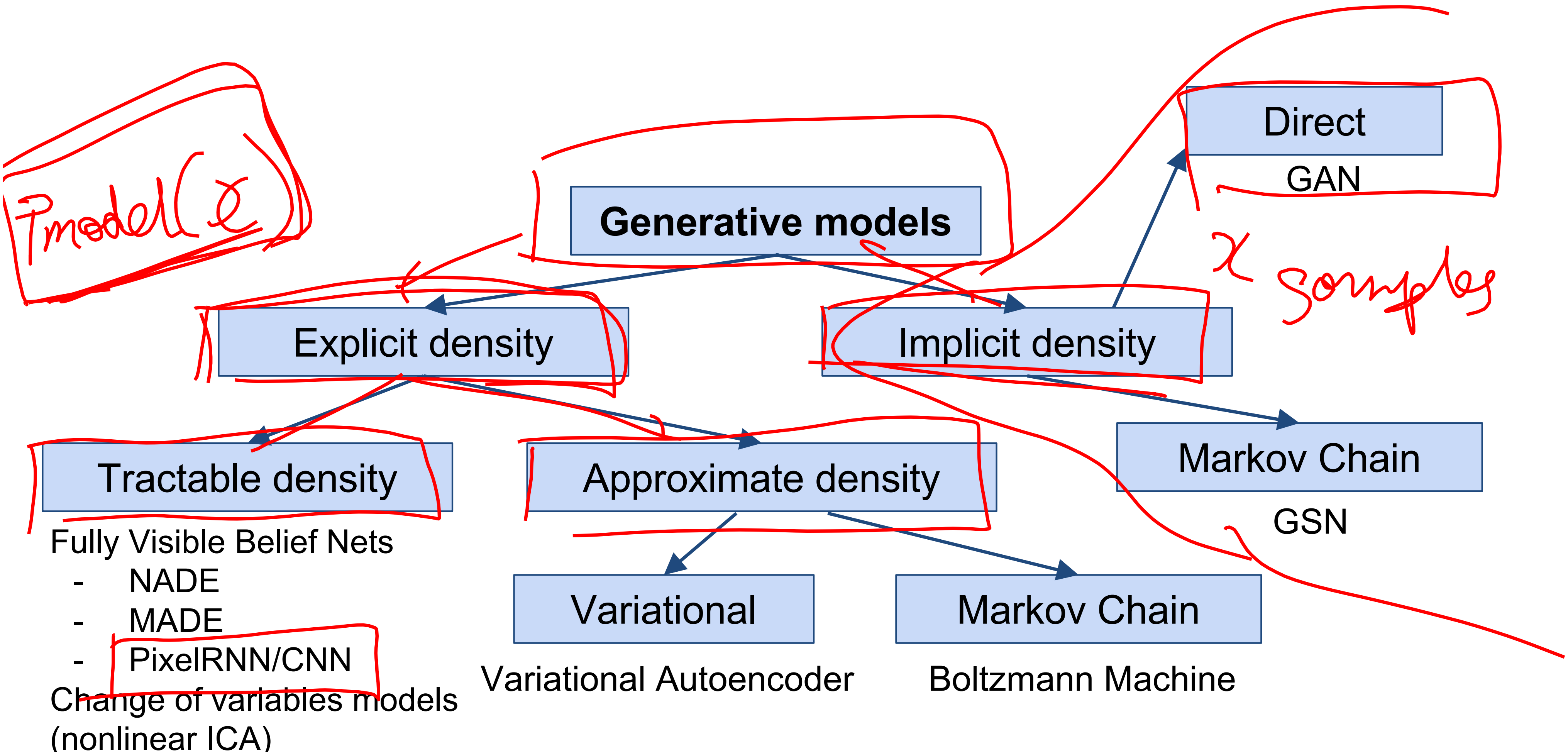


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Taxonomy of Generative Models

Today: discuss 3 most popular types of generative models today

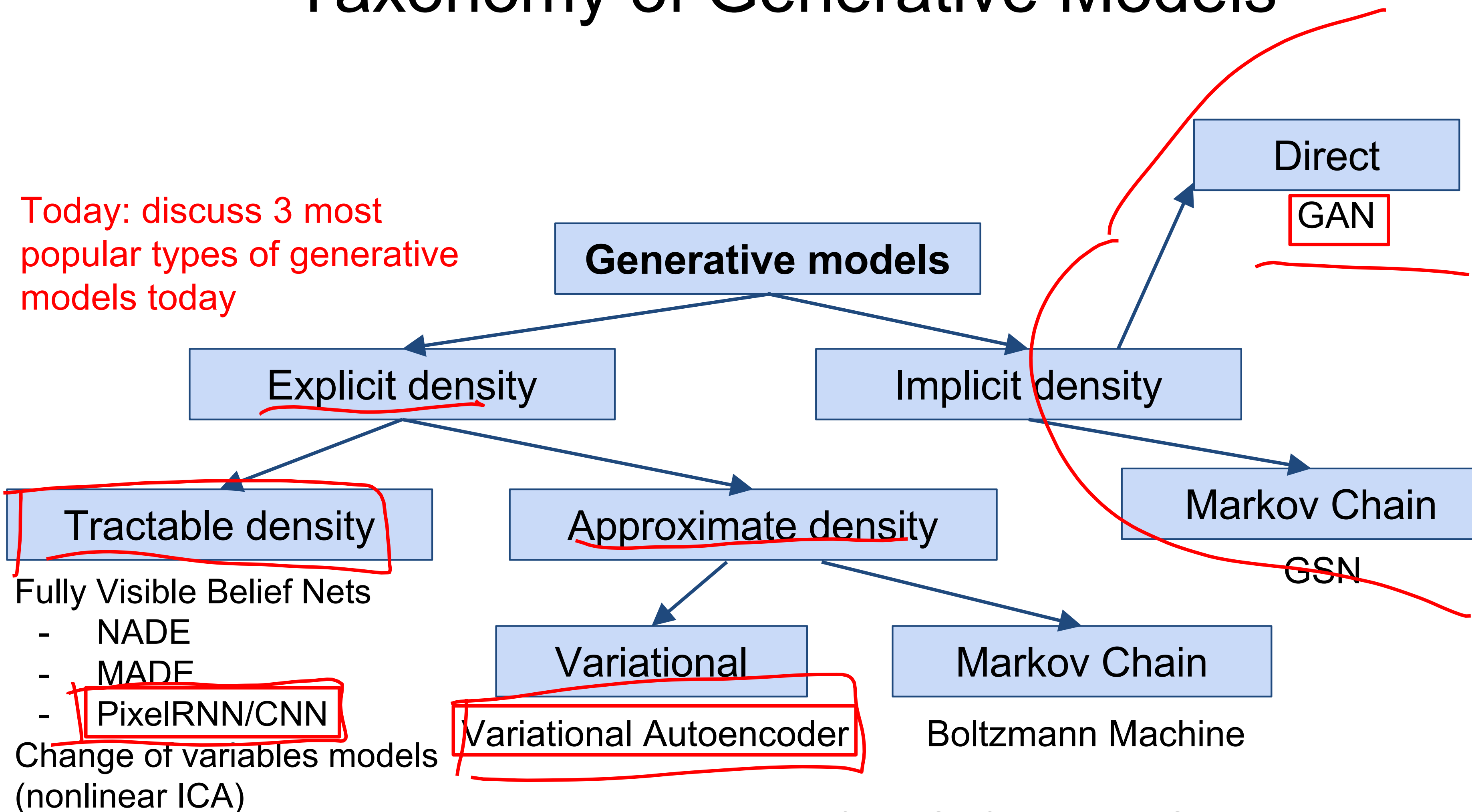
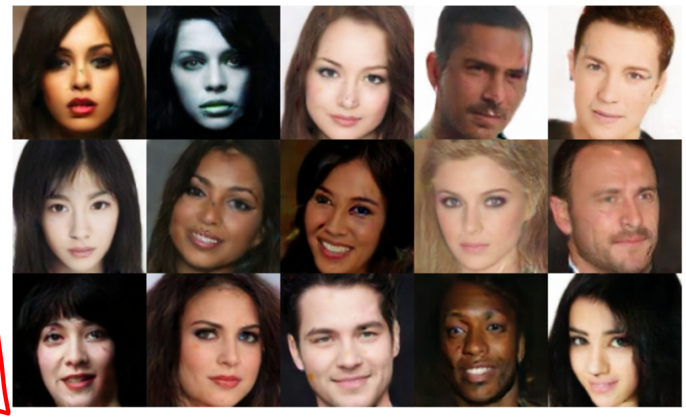


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Why Generative Models?

- Realistic samples for artwork, super-resolution, colorization, etc.


- Generative models of time-series data can be used for simulation and planning (reinforcement learning applications!)
- Training generative models can also enable inference of latent representations that can be useful as general features

PixelRNN and PixelCNN

Fully Observable Model

Explicit density model

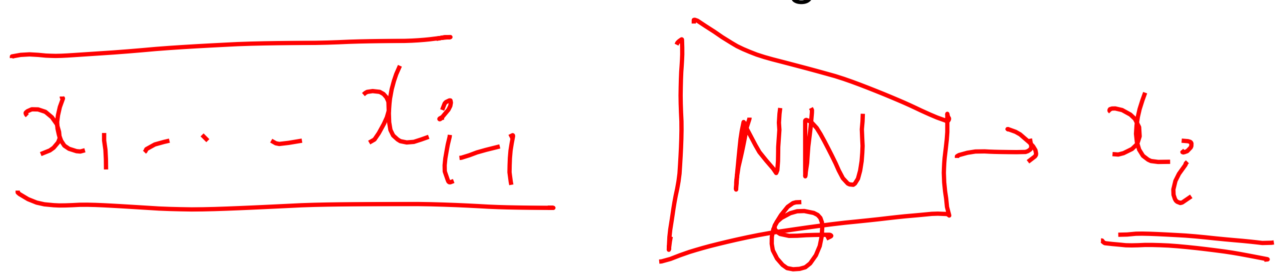
Use chain rule to decompose likelihood of an image x into product of 1-d distributions:

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

Likelihood of image x

Probability of i'th pixel value given all previous pixels

Then maximize likelihood of training data



Fully Observable Model

Explicit density model

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

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

↑ Likelihood of image x

↑ Probability of i'th pixel value given all previous pixels

Then maximize likelihood of training data

Complex distribution over pixel values
=> Express using a neural network!

Fully Observable Model

Explicit density model

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

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

Likelihood of image x

Probability of i'th pixel value given all previous pixels

Will need to define ordering of "previous pixels"

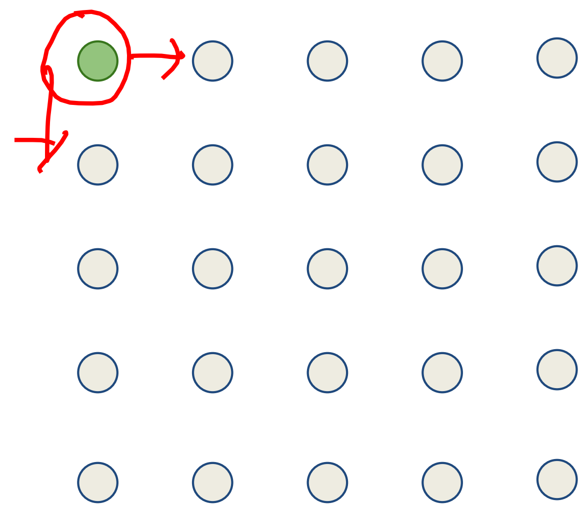
Complex distribution over pixel values => Express using a neural network!

Then maximize likelihood of training data

PixelRNN *[van der Oord et al. 2016]*

Generate image pixels starting from corner

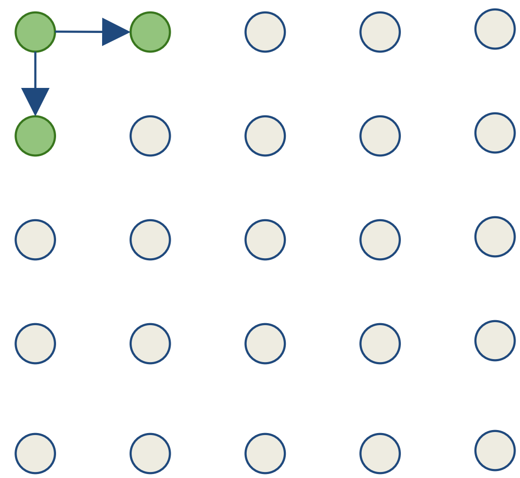
Dependency on previous pixels modeled using an RNN (LSTM)



PixelRNN *[van der Oord et al. 2016]*

Generate image pixels starting from corner

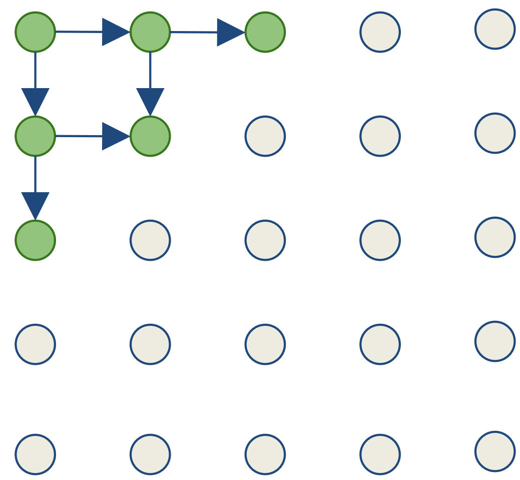
Dependency on previous pixels modeled using an RNN (LSTM)



PixelRNN *[van der Oord et al. 2016]*

Generate image pixels starting from corner

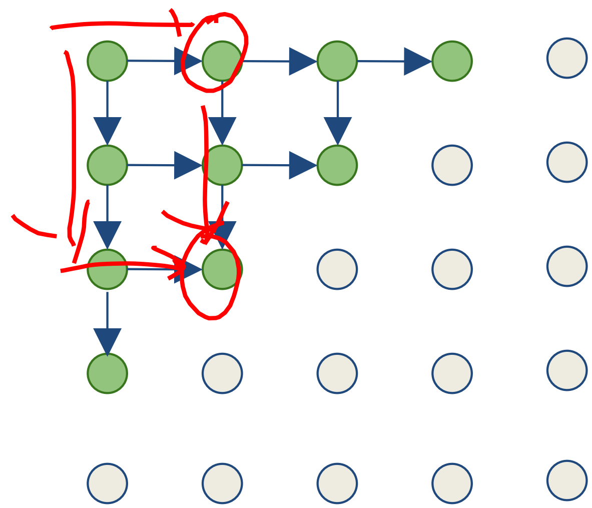
Dependency on previous pixels modeled using an RNN (LSTM)



PixelRNN *[van der Oord et al. 2016]*

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)



Drawback: sequential generation is slow!

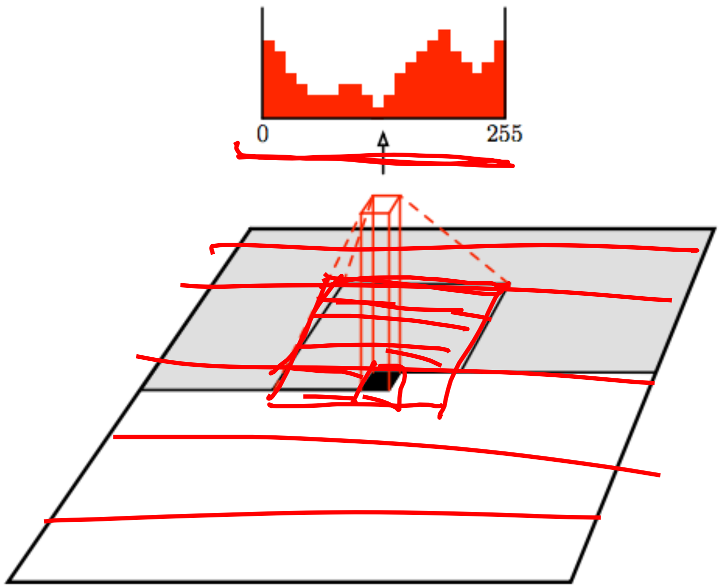
$p(x_i \mid < \quad >) \sim \text{RNN}($

$\text{softmax}(\longrightarrow \mid h_i)$

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



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

Training: maximize likelihood of training images

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

Softmax loss at each pixel

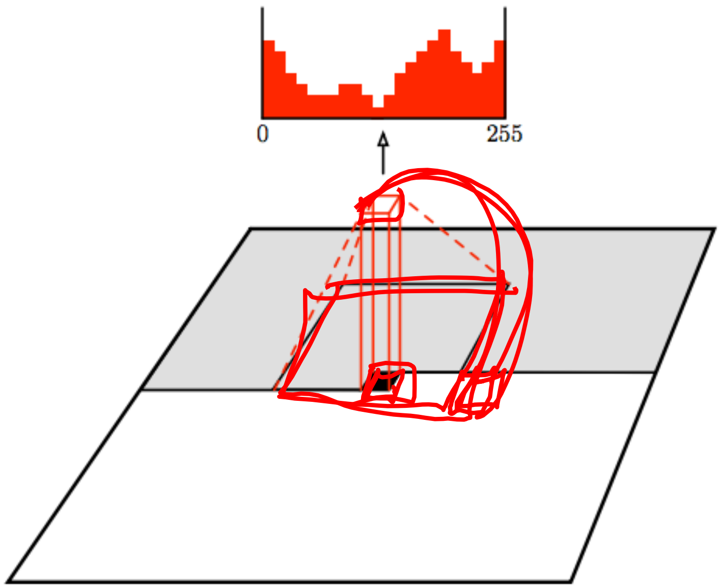


Figure copyright van der Oord et al., 2016. Reproduced with permission.

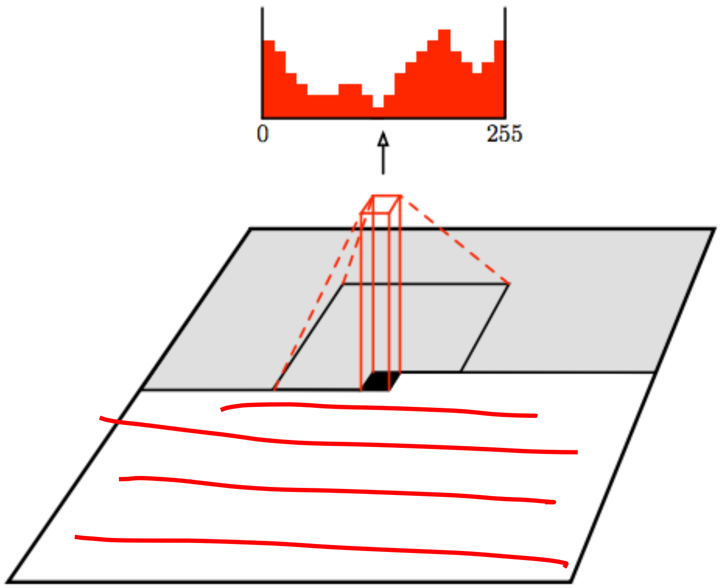
PixelCNN *[van der Oord et al. 2016]*

Still generate image pixels starting from corner

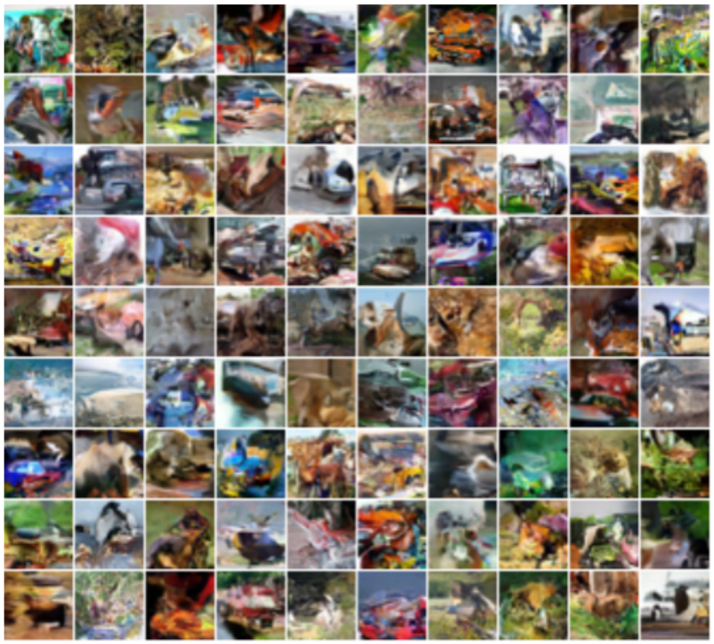
Dependency on previous pixels now modeled using a CNN over context region

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

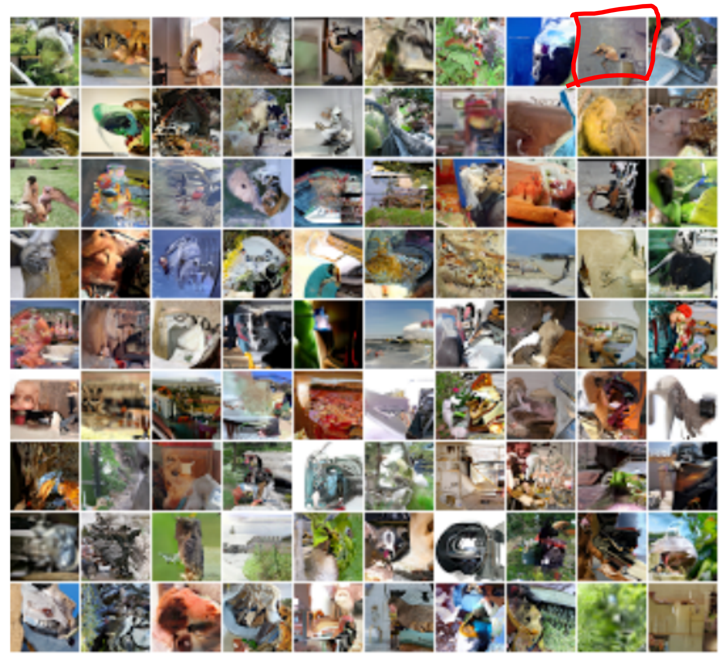
Generation must still proceed sequentially
=> still slow



Generation Samples



32x32 CIFAR-10



32x32 ImageNet

PixelRNN and PixelCNN

Pros:

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

Con:

- Sequential generation => slow

Improving PixelCNN performance

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

See

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

Variational Autoencoders (VAE)

So far...

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

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

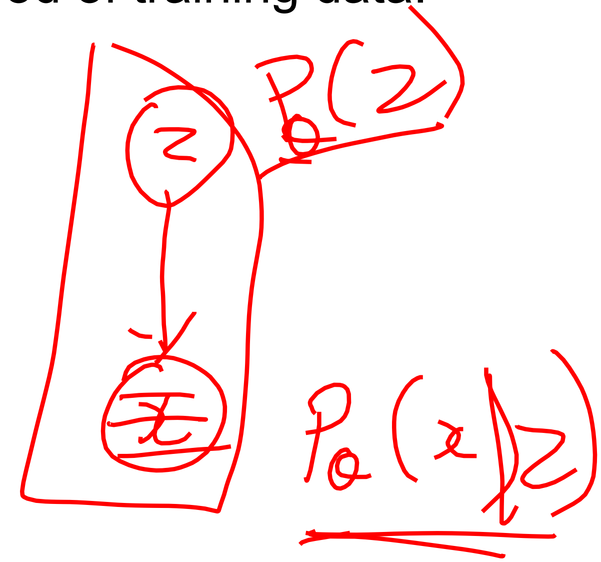
So far...

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

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

VAEs define intractable density function with latent **z**:

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



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

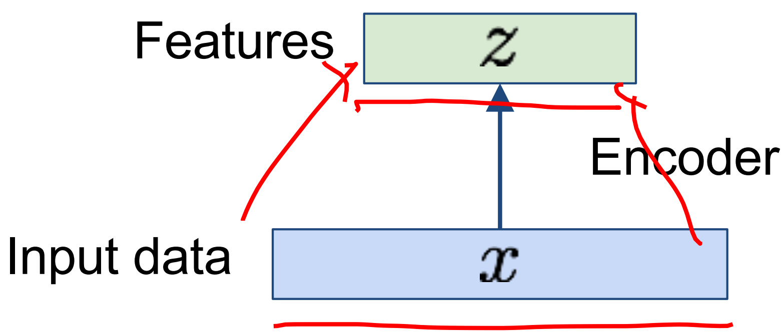
Variational Auto Encoders

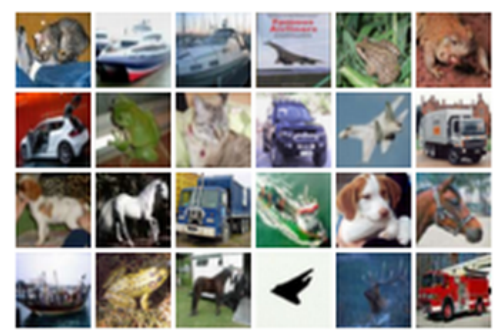
VAEs are a combination of the following ideas:

1. Auto Encoders
2. Variational Approximation
   - Variational Lower Bound / ELBO
3. Amortized Inference Neural Networks
4. "Reparameterization" Trick

Autoencoders

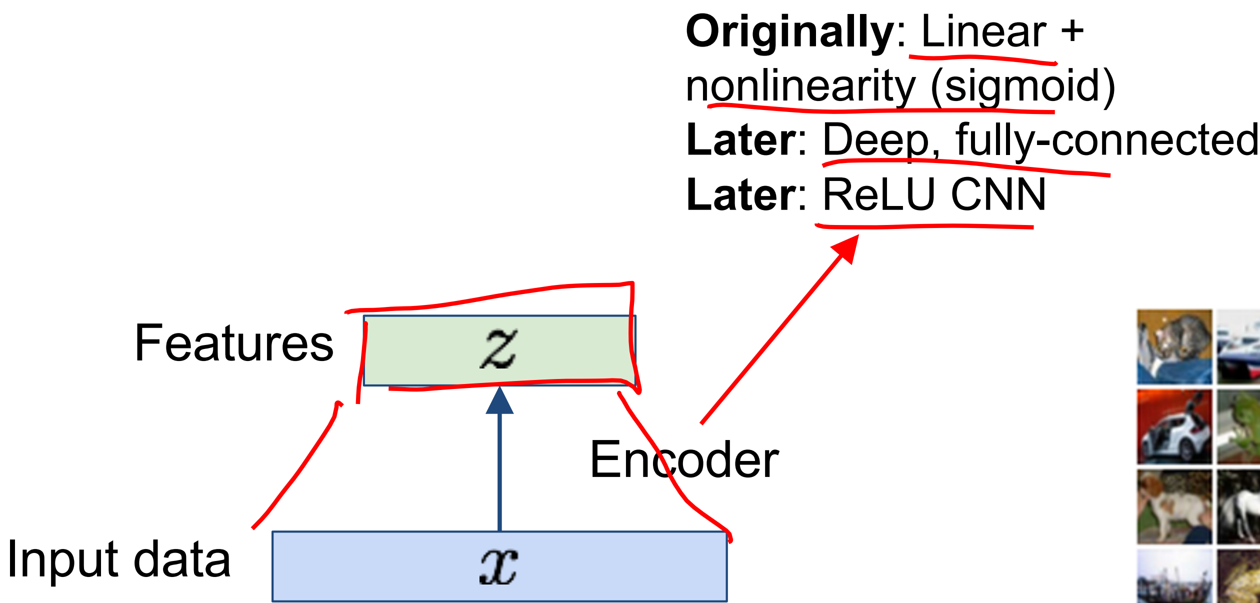
Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

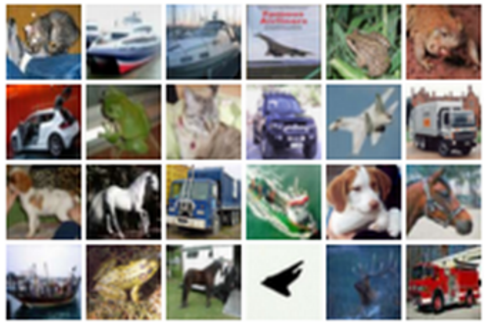




Autoencoders

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





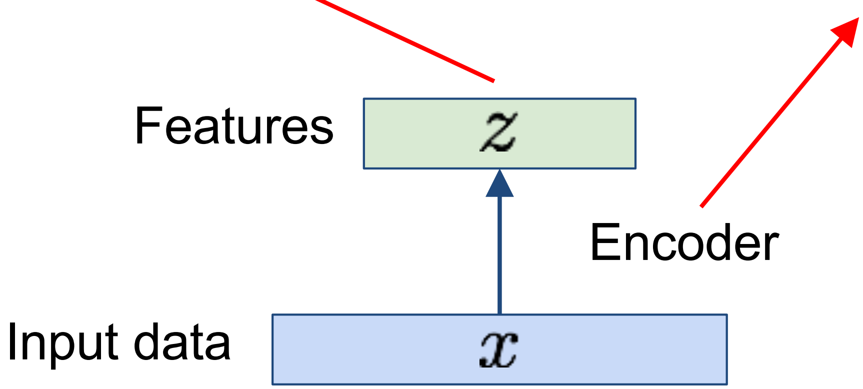
Autoencoders

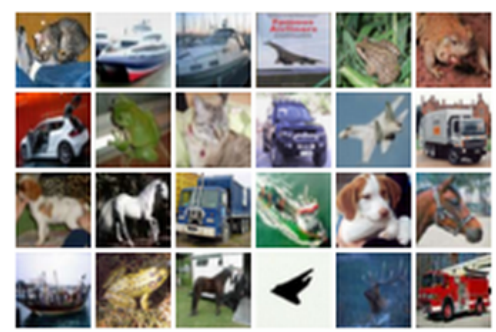
Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

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

Q: Why dimensionality reduction?

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





Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Autoencoders

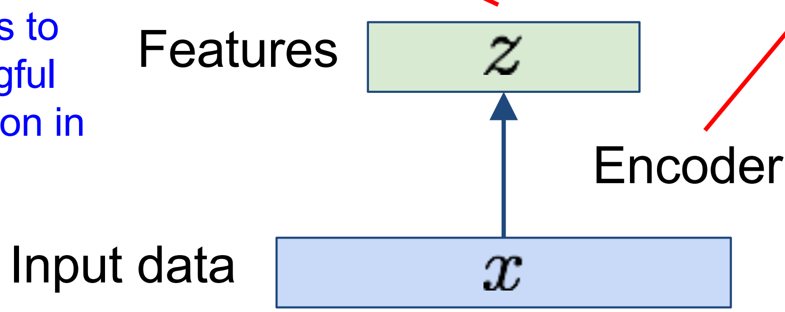
Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

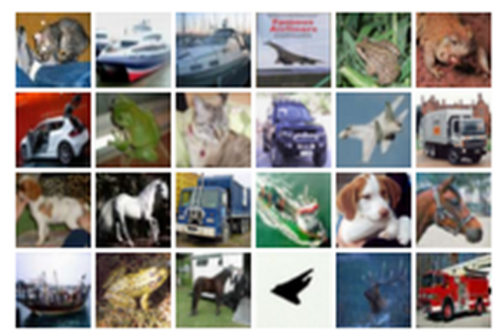
z usually smaller than **x**
(dimensionality reduction)

Q: Why dimensionality reduction?

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

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

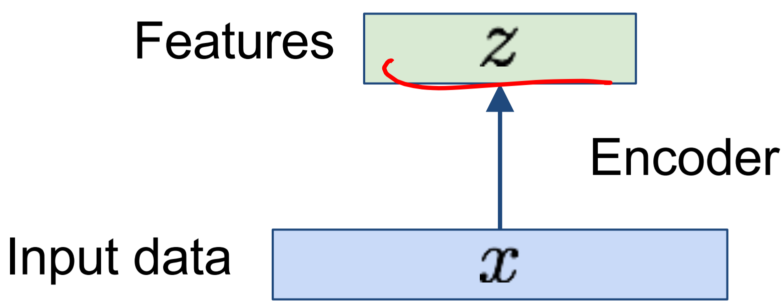


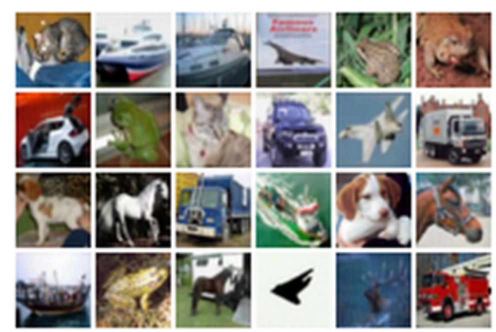


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Autoencoders

How to learn this feature representation?

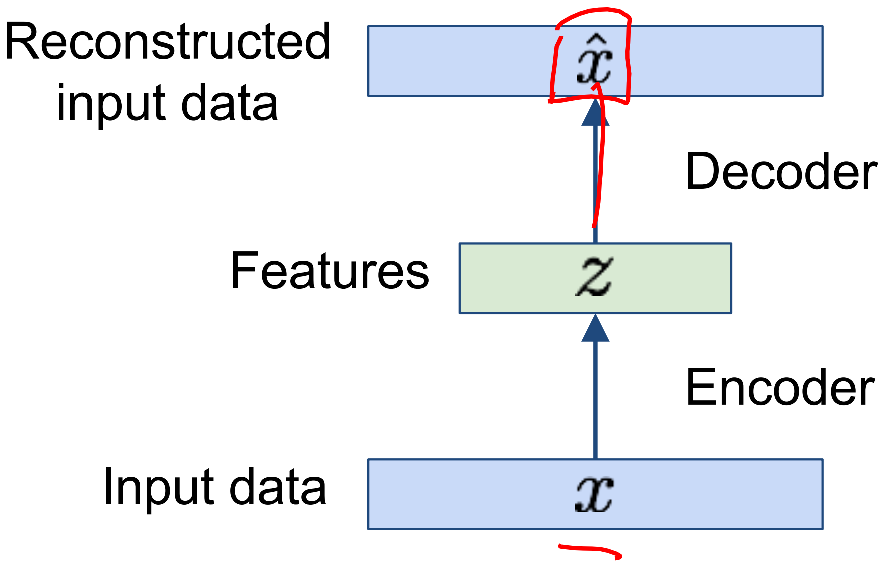


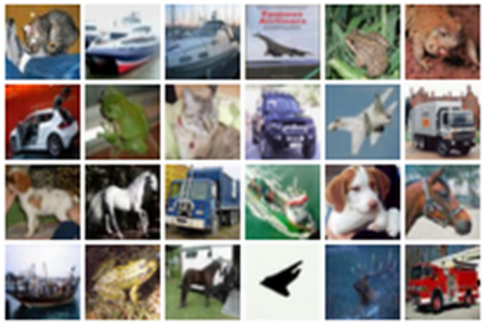


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Autoencoders

How to learn this feature representation?
Train such that features can be used to reconstruct original data
"Autoencoding" - encoding itself





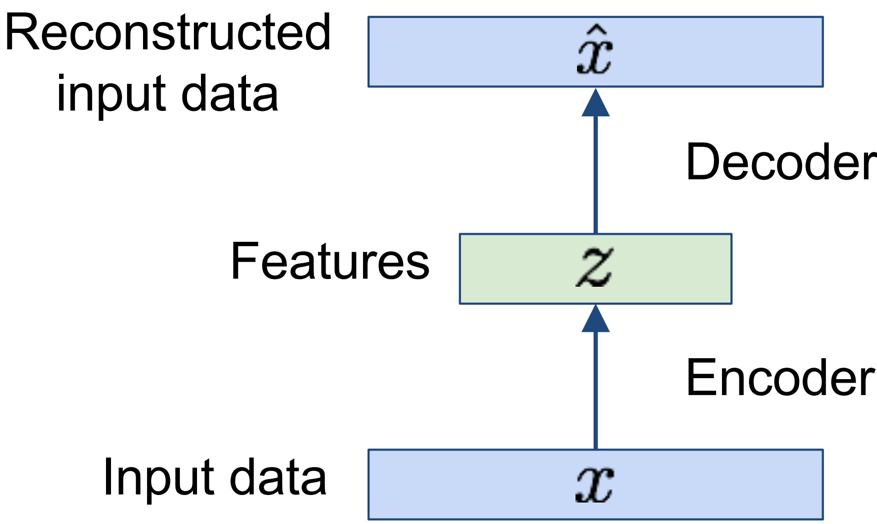
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Autoencoders

How to learn this feature representation?

Train such that features can be used to reconstruct original data

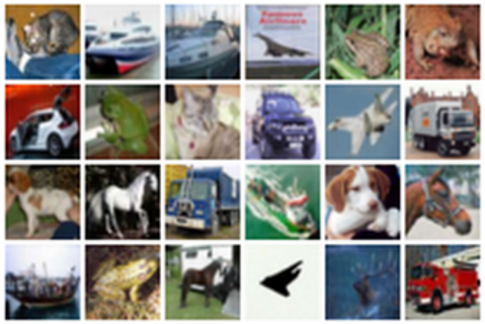
"Autoencoding" - encoding itself



Originally: Linear + nonlinearity (sigmoid)

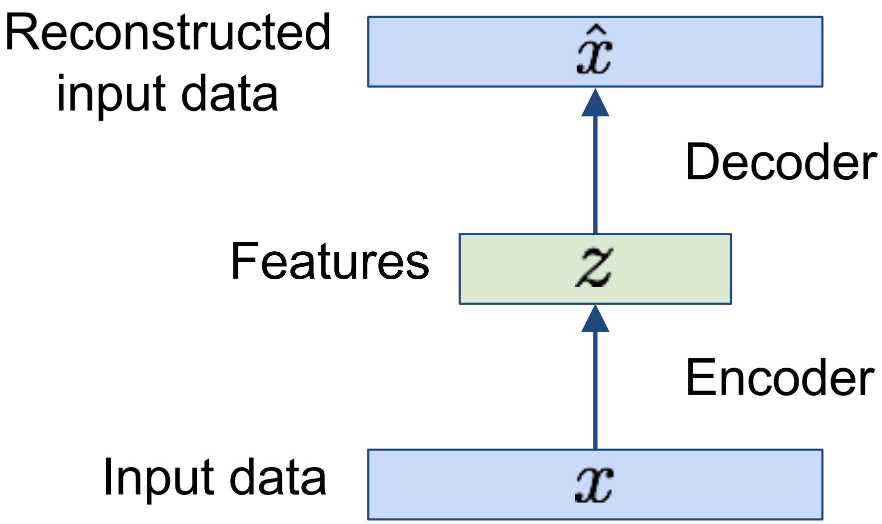
Later: Deep, fully-connected

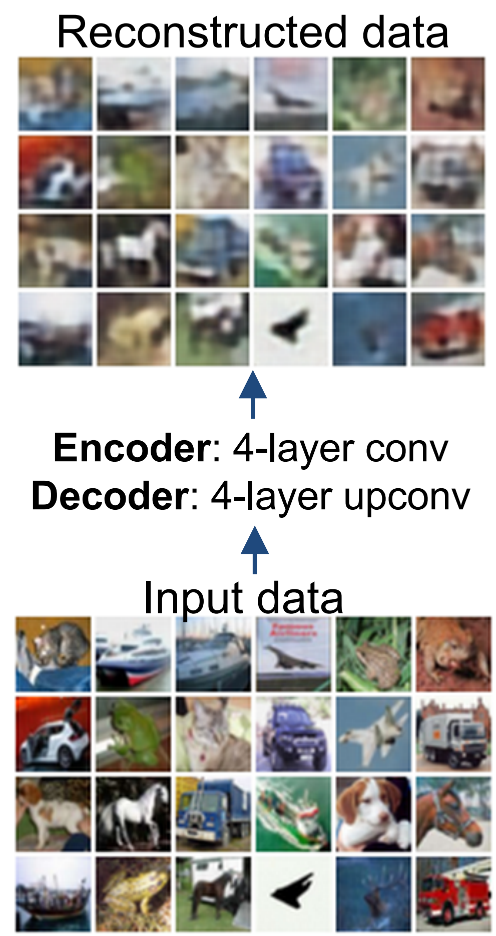
Later: ReLU CNN (upconv)



Autoencoders

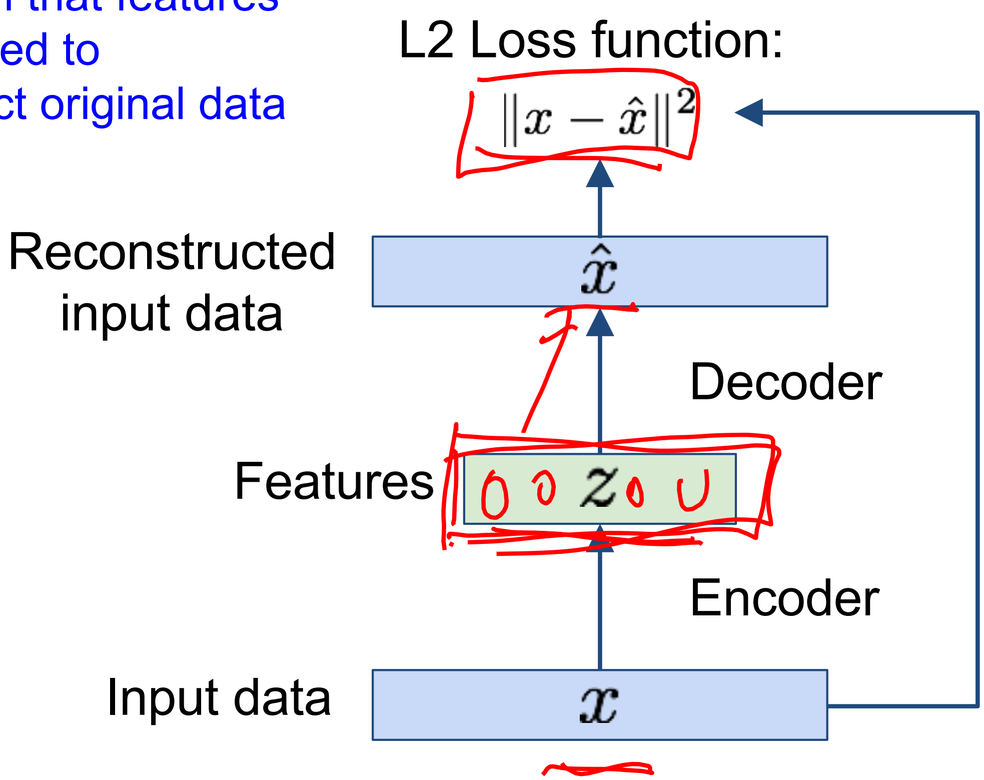
How to learn this feature representation?
Train such that features can be used to reconstruct original data
“Autoencoding” - encoding itself

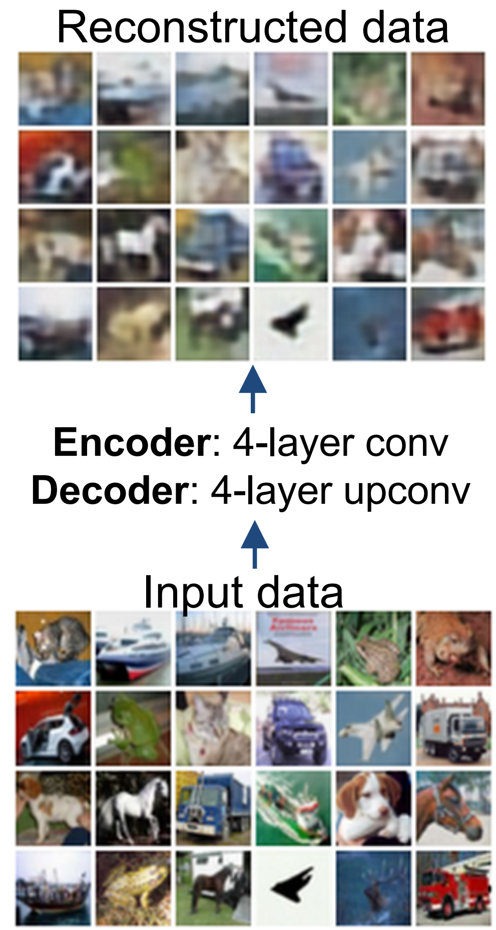




Autoencoders

Train such that features can be used to reconstruct original data

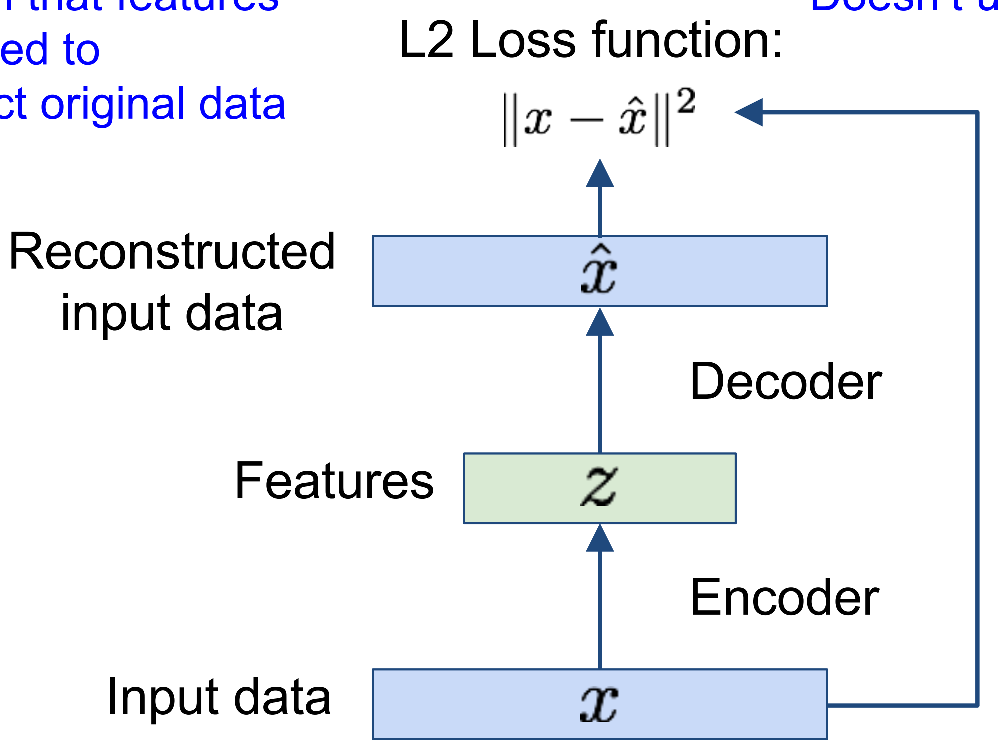


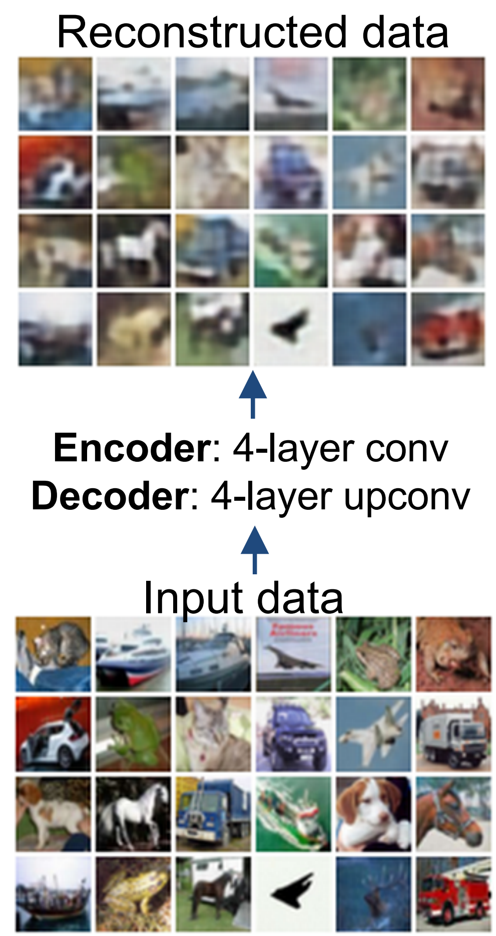


Autoencoders

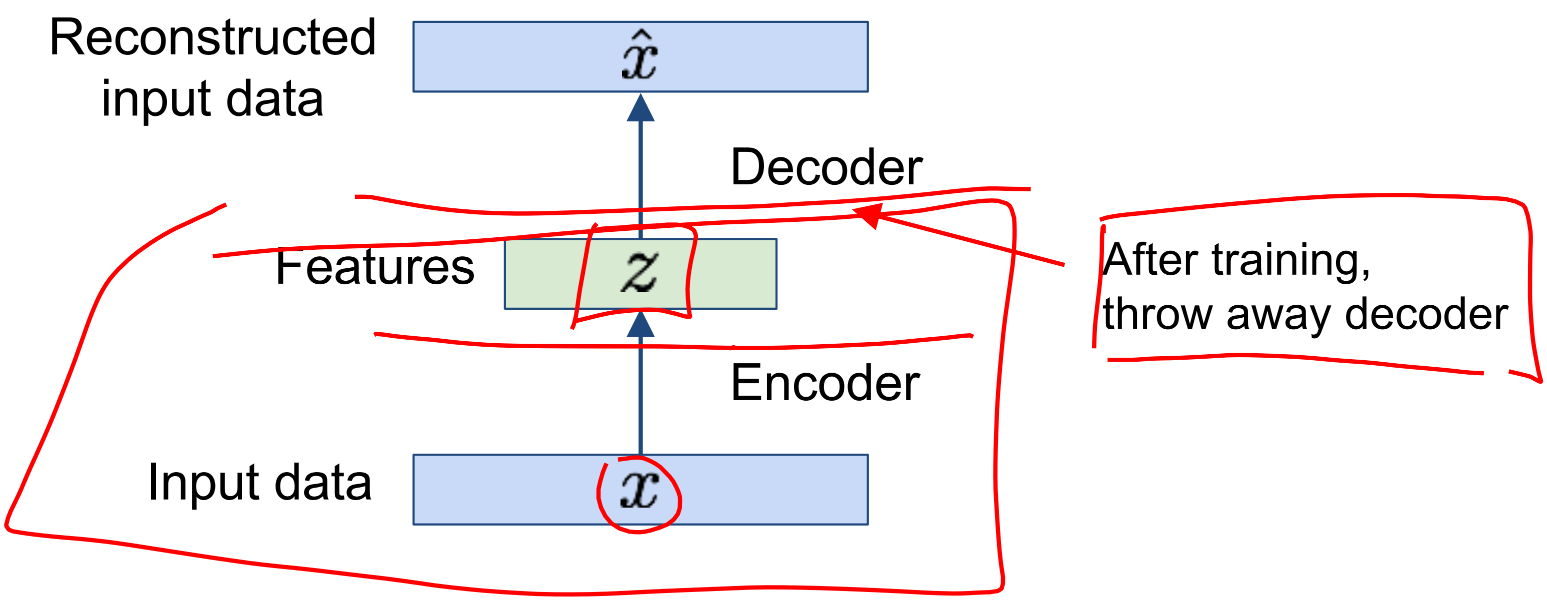
Train such that features can be used to reconstruct original data

Doesn't use labels!



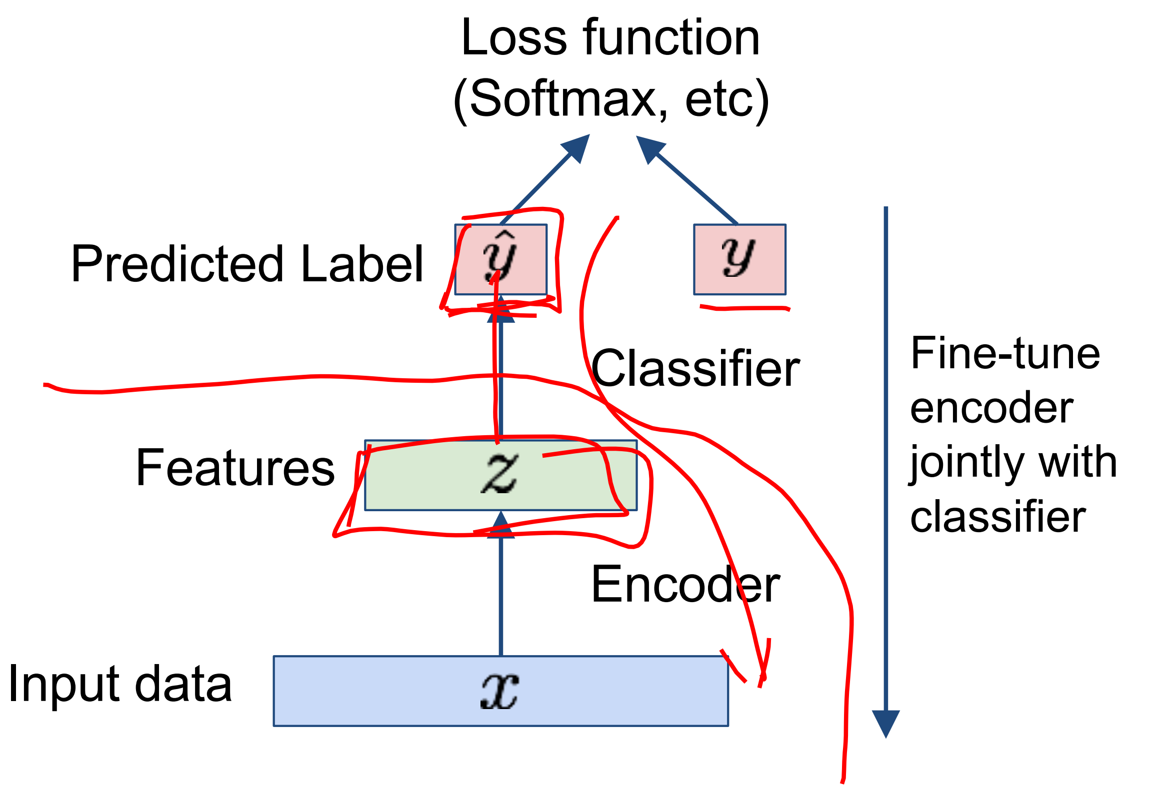


Autoencoders



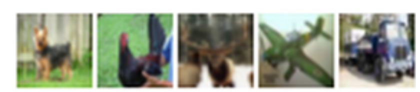
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Autoencoders



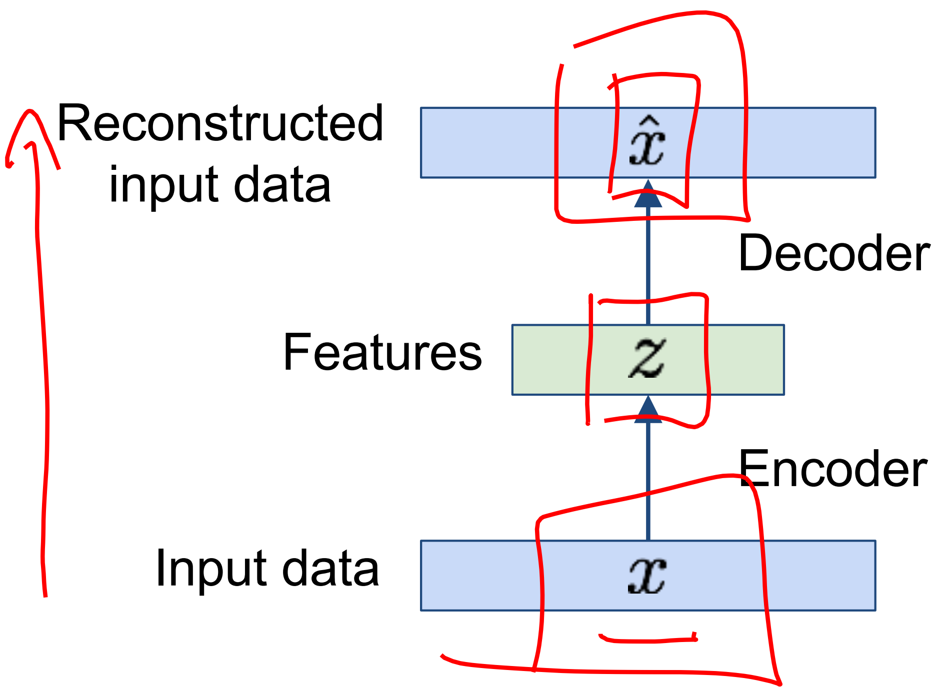
Encoder can be used to initialize a **supervised** model

bird plane
dog deer truck



Train for final task (sometimes with small data)

Autoencoders



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

Features capture factors of variation in training data. Can we generate new images from an autoencoder?

Variational Autoencoders

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

Variational Auto Encoders

VAEs are a combination of the following ideas:

1. Auto Encoders

2. Variational Approximation
   - Variational Lower Bound / ELBO

3. Amortized Inference Neural Networks

4. "Reparameterization" Trick

Basic Problem

- Goal $$\min_{\theta} \mathbb{E}_{z \sim p_\theta(z)}[f(z)]$$

- Need to compute: $$\nabla_\theta \mathbb{E}_{z \sim p_\theta(z)}[f(z)]$$

$$\nabla_\theta E_{x,y \sim p_{data}} [\ell(x, y, \theta)]$$

$$\approx \frac{1}{N} \sum_i \nabla_\theta \ell(x_i, y_i, \theta)$$

Example

$$z \sim p_\theta(z) = N(\theta, 1)$$

$$f(z) = z^2$$

$$\min_\theta E[z^2]$$

$$\min_\theta \int_{-\infty}^{\infty} p(z)\, z^2\, dz$$

$$\int \frac{1}{\sqrt{2\pi}} e^{-\frac{(z-\theta)^2}{2}} z^2\, dz$$

$$Var(z) = E[z^2] - \theta^2$$

Example

Two Options

- Score Function based Gradient Estimator
  aka REINFORCE (and variants)

$$\nabla_\theta \mathbb{E}_z \left[f(z) \right] = \mathbb{E}_z \left[f(z) \nabla_\theta \log p_\theta(z) \right]$$

- Path Derivative Gradient Estimator
  aka "reparameterization trick"

$$\frac{\partial}{\partial \theta} \mathbb{E}_{z \sim p_\theta} \left[f(z) \right] = \frac{\partial}{\partial \theta} \mathbb{E}_\epsilon \left[f(g(\theta, \epsilon)) \right] = \mathbb{E}_{\epsilon \sim p_\epsilon} \left[\frac{\partial f}{\partial g} \frac{\partial g}{\partial \theta} \right]$$

Option 1

- Score Function based Gradient Estimator aka REINFORCE (and variants)

$$\nabla_\theta \mathbb{E}_z\left[f(z)\right] = \mathbb{E}_z\left[f(z) \nabla_\theta \log p_\theta(z)\right]$$

$$\int f(z) \nabla_\theta p_\theta(z)\, dz \cdot \frac{p_\theta(z)}{p_\theta(z)} = \int \left[f(z) \nabla_\theta \log p(z) \right] p(z)\, dz$$

$$\nabla_\theta \log p_\theta(z)$$

$$= E\left[f(z) \nabla_\theta \log p(z) \right]$$

Example

$$p_\theta(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(z-\theta)^2}{2}}$$

$$\frac{\partial}{\partial\theta} \log p(z) = -\frac{(z-\theta)^2}{2} - \frac{1}{2}\log 2\pi$$

$$= -\frac{(z-\theta)(-1)}{2}$$

$$E\left[z^2 (z-\theta)\right] \approx \frac{1}{N}\sum z_i^2 (z_i-\theta)$$

Two Options

- Score Function based Gradient Estimator
  aka REINFORCE (and variants)

$$\nabla_\theta \mathbb{E}_z [f(z)] = \mathbb{E}_z [f(z) \nabla_\theta \log p_\theta(z)]$$

- Path Derivative Gradient Estimator
  aka "reparameterization trick"

$$\frac{\partial}{\partial \theta} \mathbb{E}_{z \sim p_\theta} [f(z))] = \frac{\partial}{\partial \theta} \mathbb{E}_\epsilon [f(g(\theta, \epsilon))] = \mathbb{E}_{\epsilon \sim p_\epsilon} \left[\frac{\partial f}{\partial g} \frac{\partial g}{\partial \theta} \right]$$

Option 2

- Path Derivative Gradient Estimator
  aka “reparameterization trick”

$$\frac{\partial}{\partial\theta}\mathbb{E}_{z\sim p_\theta}[f(z)] = \frac{\partial}{\partial\theta}\mathbb{E}_\epsilon[f(g(\theta,\epsilon))] = \mathbb{E}_{\epsilon\sim p_\epsilon}\left[\frac{\partial f}{\partial g}\frac{\partial g}{\partial\theta}\right] \approx \frac{1}{N}\sum$$

$$z \sim p_\theta(z)$$

$$z = g(\theta, \varepsilon) \qquad \varepsilon \sim N(0,1)$$

$$E_{z\sim p(z)}[f(z)] = E_{\varepsilon\sim p(\varepsilon)}[f(g(\theta,\varepsilon))]$$

Example

$z \sim N(\theta, \sigma^2)$ $\qquad$ $\varepsilon \sim N(0,1)$

$$z = \theta + \sigma \varepsilon$$

$$\frac{\partial z}{\partial \theta} = 1$$

$$f(z) = z^2$$

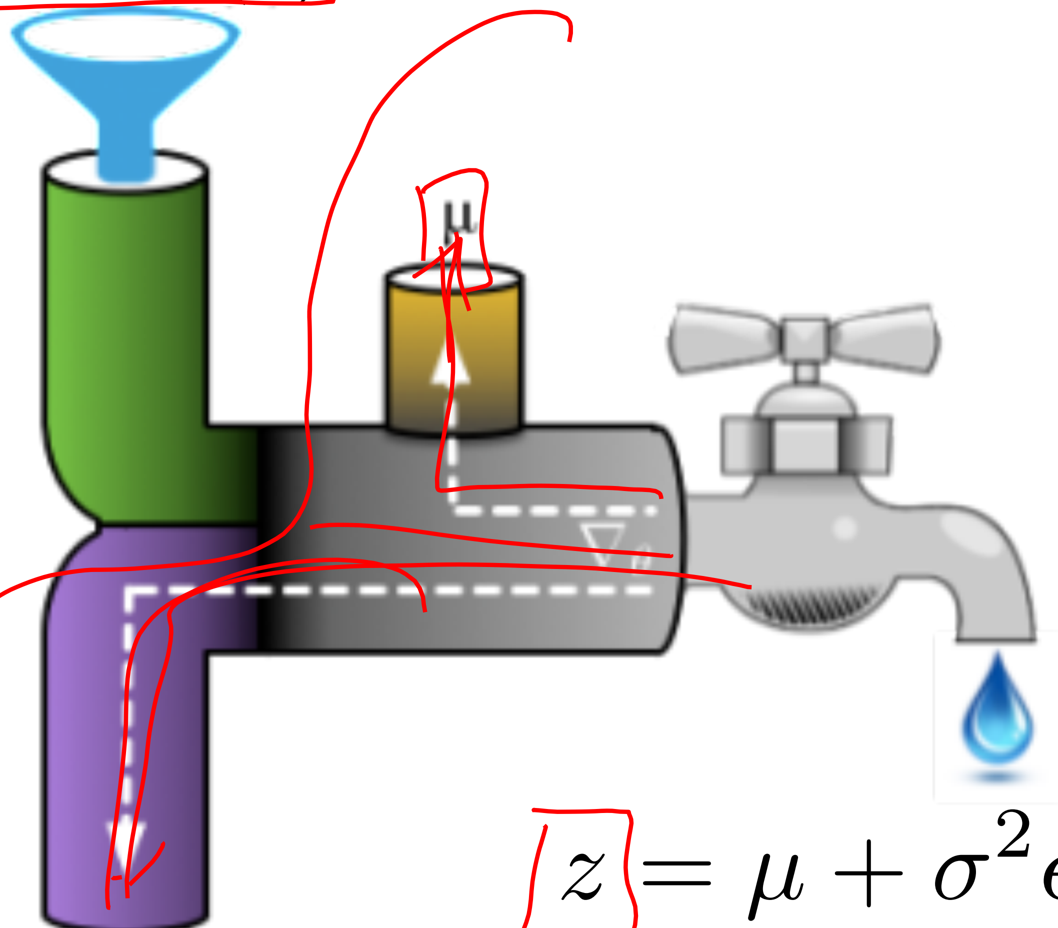
$$\frac{\partial f}{\partial g} = 2z$$

$$E_{\varepsilon \sim p(\varepsilon)}[2z] = E_{\varepsilon}[2(\theta + \varepsilon)]$$

$$= \frac{1}{N} \sum (\_\_)$$

Reparameterization Intuition

$\epsilon_i \sim p(\epsilon)$



σ^2

$z = \mu + \sigma^2 \epsilon_i$

Two Options

- Score Function based Gradient Estimator
  aka REINFORCE (and variants)

$$\nabla_\theta \mathbb{E}_z \left[f(z) \right] = \mathbb{E}_z \left[f(z) \nabla_\theta \log p_\theta(z) \right]$$

- Path Derivative Gradient Estimator
  aka "reparameterization trick"

$$\frac{\partial}{\partial \theta} \mathbb{E}_{z \sim p_\theta} \left[f(z) \right] = \frac{\partial}{\partial \theta} \mathbb{E}_{\epsilon} \left[f(g(\theta, \epsilon)) \right] = \mathbb{E}_{\epsilon \sim p_\epsilon} \left[\frac{\partial f}{\partial g} \frac{\partial g}{\partial \theta} \right]$$

Example

```
import numpy as np
N = 1000
theta = 2.0
x = np.random.randn(N) + theta
eps = np.random.randn(N)

grad1 = lambda x: np.sum(np.square(x)*(x-theta)) / x.size
grad2 = lambda eps: np.sum(2*(theta + eps)) / x.size

print grad1(x)
print grad2(eps)
```

```
4.46239612174
4.1840532024
```


Figure Credit: http://gokererdogan.github.io/2016/07/01/reparameterization-trick/

Example

```
Ns = [10, 100, 1000, 10000, 100000]
reps = 100

means1 = np.zeros(len(Ns))
vars1 = np.zeros(len(Ns))
means2 = np.zeros(len(Ns))
vars2 = np.zeros(len(Ns))

est1 = np.zeros(reps)
est2 = np.zeros(reps)
for i, N in enumerate(Ns):
    for r in range(reps):
        x = np.random.randn(N) + theta
        est1[r] = grad1(x)
        eps = np.random.randn(N)
        est2[r] = grad2(eps)
    means1[i] = np.mean(est1)
    means2[i] = np.mean(est2)
    vars1[i] = np.var(est1)
    vars2[i] = np.var(est2)

print means1
print means2
print
print vars1
print vars2
```

```
[ 3.8409546   3.97298803  4.03007634  3.98531095  3.99579423]
[ 3.97775271  4.00232825  3.99894536  4.00353734  3.99995899]

[  6.45307927e+00   6.80227241e-01   8.69226368e-02   1.00489791e-02
   8.62396526e-04]
[  4.59767676e-01   4.26567475e-02   3.33699503e-03   5.17148975e-04
   4.65338152e-05]
```

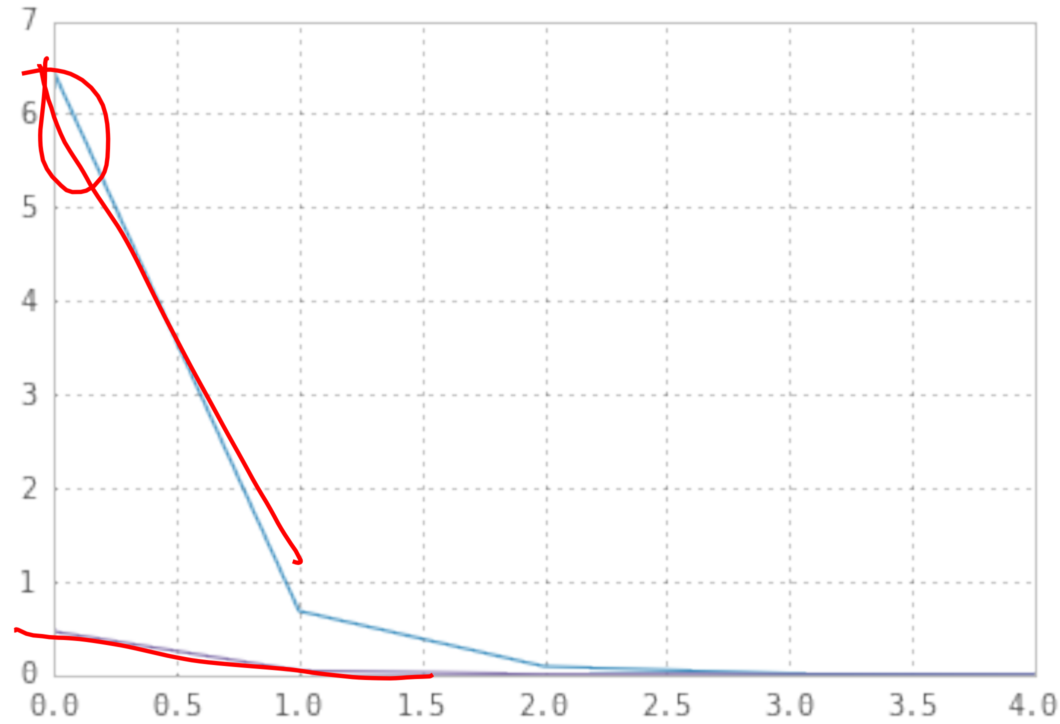
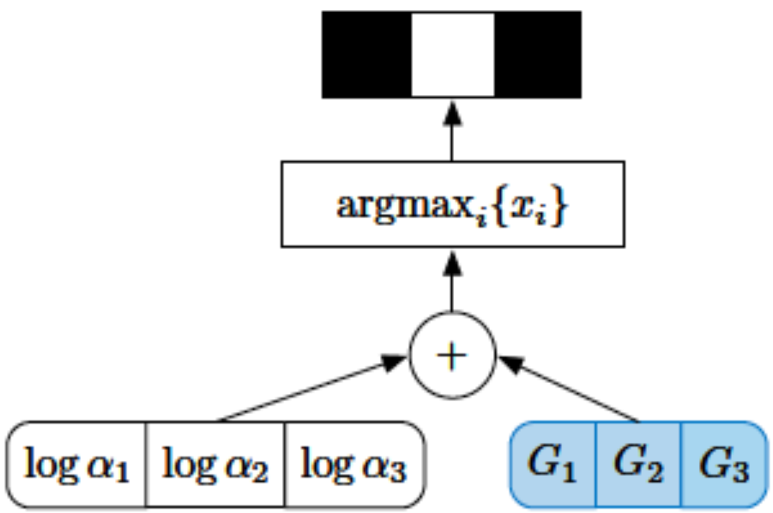
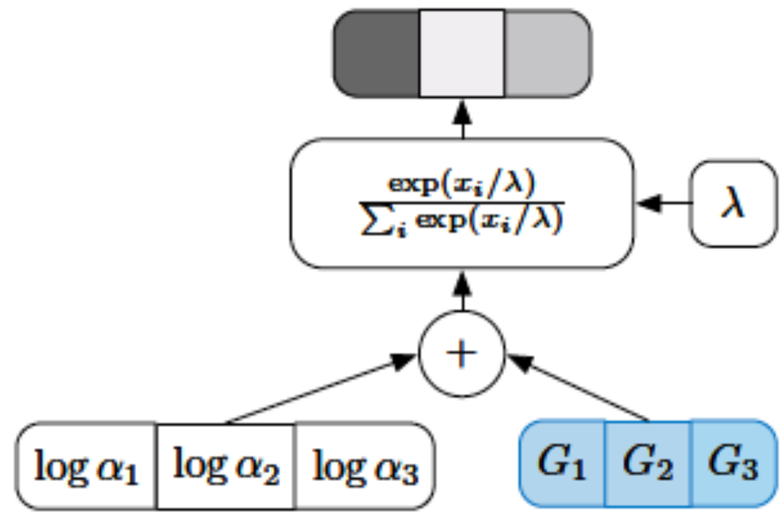


Figure Credit: http://gokererdogan.github.io/2016/07/01/reparameterization-trick/

Aside: Gumbel Softmax

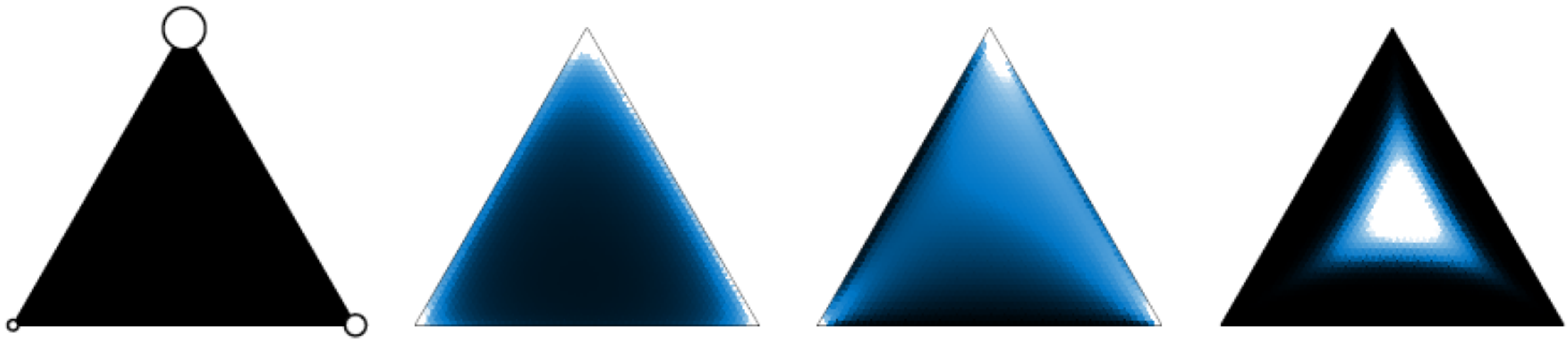
- Meet the Gumbel Softmax "trick"





Aside: Gumbel Softmax

- Sampling on the Simplex



(a) $\lambda = 0$ (b) $\lambda = 1/2$ (c) $\lambda = 1$ (d) $\lambda = 2$

Figure 2: A discrete distribution with unnormalized probabilities $(\alpha_1, \alpha_2, \alpha_3) = (2, 0.5, 1)$ and three corresponding Concrete densities at increasing temperatures λ. Each triangle represents the set of points (y_1, y_2, y_3) in the simplex $\Delta^2 = \{(y_1, y_2, y_3) \mid y_k \in (0, 1), y_1 + y_2 + y_3 = 1\}$. For $\lambda = 0$ the size of white circles represents the mass assigned to each vertex of the simplex under the discrete distribution. For $\lambda \in \{2, 1, 0.5\}$ the intensity of the shading represents the value of $p_{\alpha,\lambda}(y)$.

Variational Auto Encoders

VAEs are a combination of the following ideas:

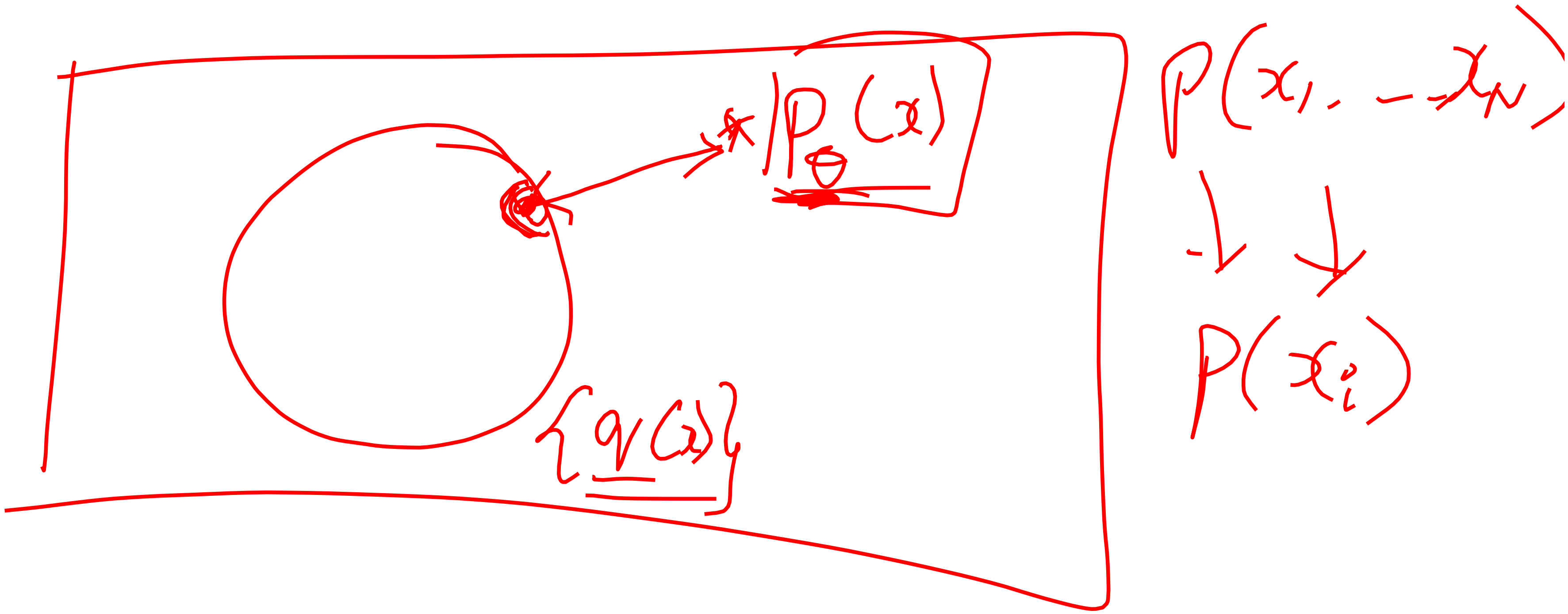
1. Auto Encoders
2. Variational Approximation
   - Variational Lower Bound / ELBO
3. Amortized Inference Neural Networks
4. “Reparameterization” Trick

What is Variational Inference?

- A class of methods for
  - approximate inference, parameter learning
  - And approximating integrals basically..

- Key idea
  - Reality is complex
  - Instead of performing approximate computation in something complex,
  - Can we perform exact computation in something “simple”?
  - Just need to make sure the simple thing is “close” to the complex thing.

Intuition



KL divergence: Distance between distributions

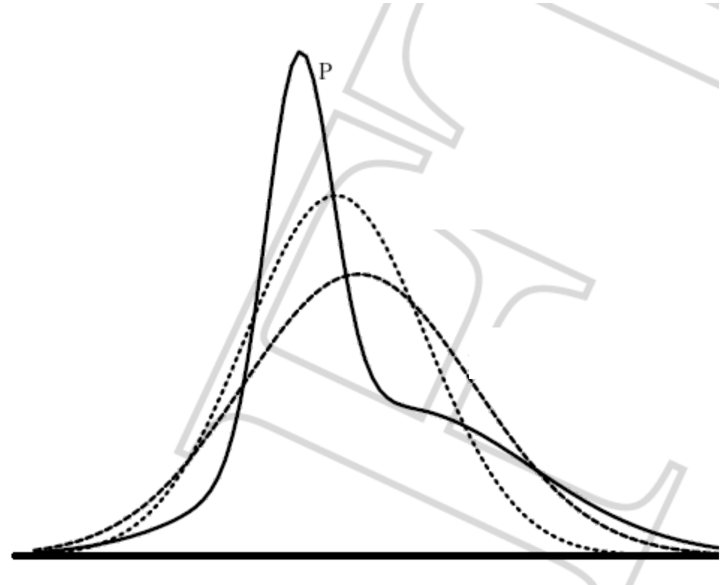
- Given two distributions *p* and *q* KL divergence:

$$KL(p\,||\,q) = \sum_x p(x) \log \frac{p(x)}{q(x)}$$

- D(p||q) = 0 iff p=q
- Not symmetric – p determines where difference is important

Find simple approximate distribution

- Suppose *p* is intractable posterior
- Want to find simple *q* that approximates *p*
- KL divergence not symmetric



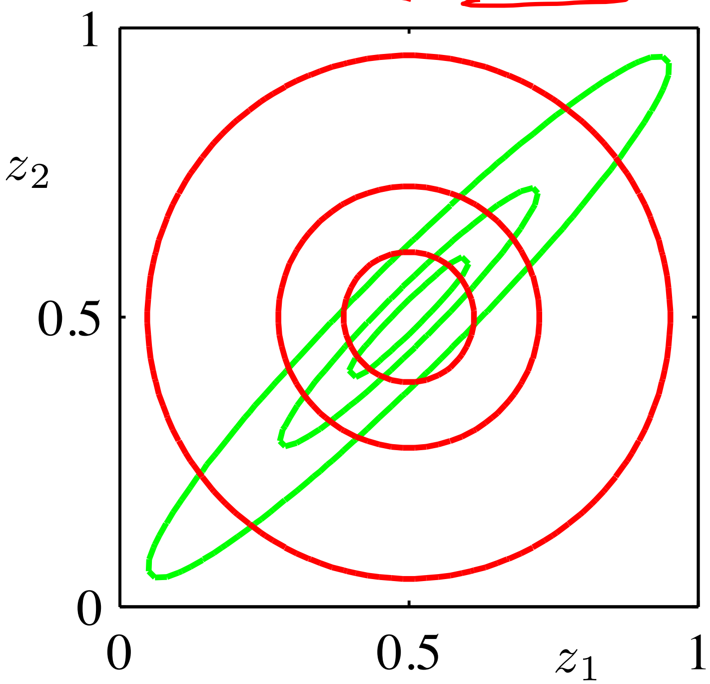
- D(p||q)
  - true distribution p defines support of diff.
  - the "correct" direction
  - will be intractable to compute

- D(q||p)
  - approximate distribution defines support
  - tends to give overconfident results
  - will be tractable

Example 1

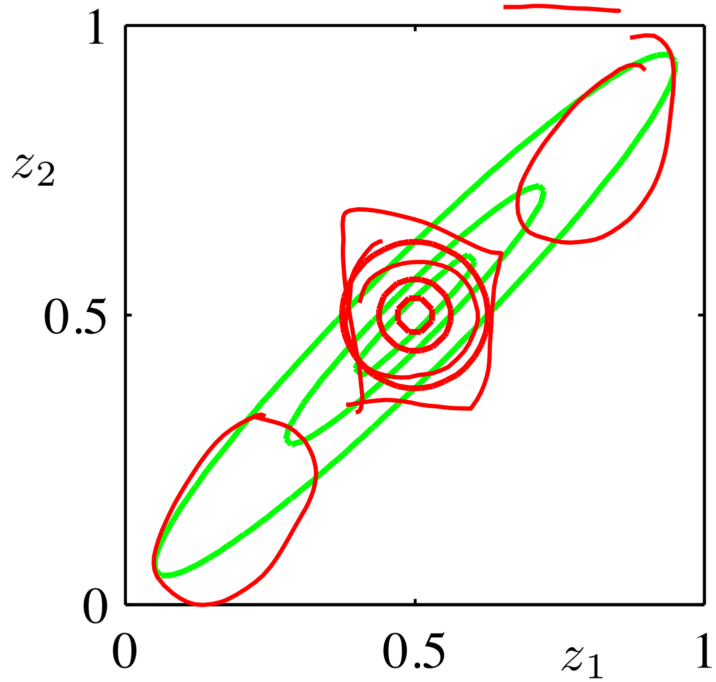
- p = 2D Gaussian with arbitrary co-variance
- q = 2D Gaussian with diagonal co-variance

argmin_q KL (p || q)



(b)

argmin_q KL (q || p)



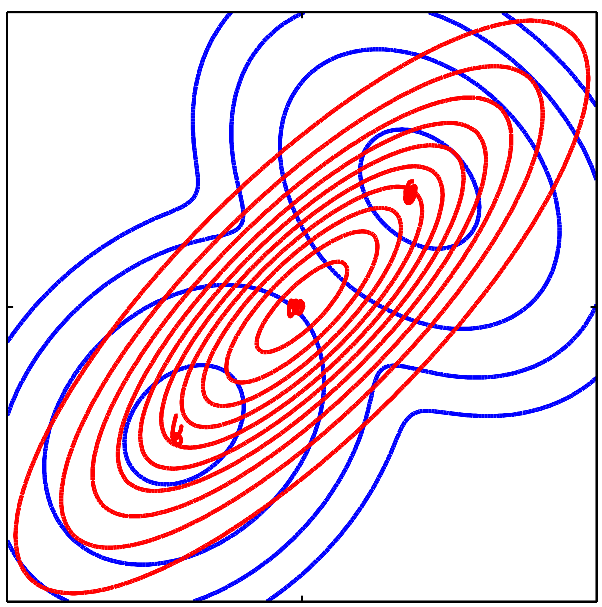
(a)

p = Green; q = Red

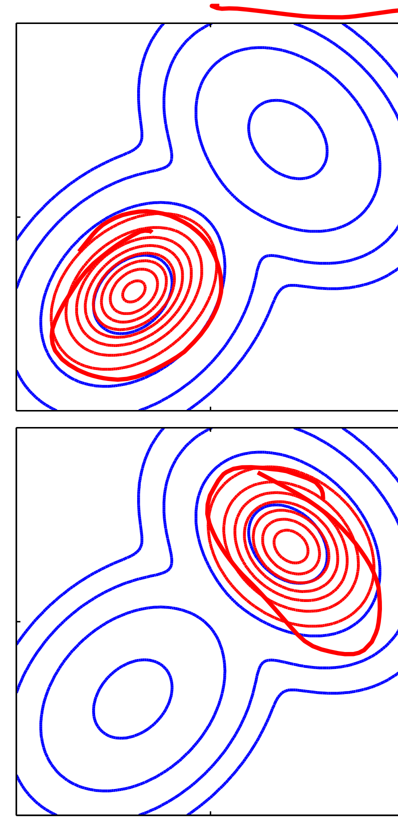
Example 2

- p = Mixture of Two Gaussians
- q = Single Gaussian

argmin_q KL (p || q)



argmin_q KL (q || p)



p = Blue; q = Red

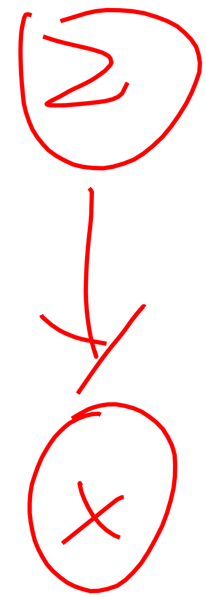
The general learning problem with missing data

- Marginal likelihood – **x** is observed, **z** is missing:

$$ll(\theta : \mathcal{D}) = \log \prod_{i=1}^{N} P(\mathbf{x}_i \mid \theta)$$

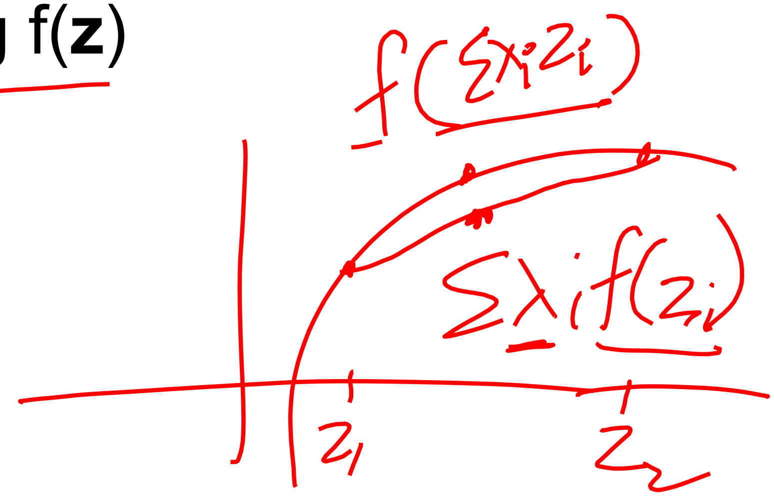
$$= \sum_{i=1}^{N} \log P(\mathbf{x}_i \mid \theta)$$

$$= \sum_{i=1}^{N} \log \sum_{\mathbf{z}} P(\mathbf{x}_i, \mathbf{z} \mid \theta)$$



Applying Jensen's inequality

- Use: $\log \sum_{\mathbf{z}} P(\mathbf{z})\, f(\mathbf{z}) \geq \sum_{\mathbf{z}} P(\mathbf{z}) \log f(\mathbf{z})$



Applying Jensen's inequality

- Use: $\log \sum_{\mathbf{z}} P(\mathbf{z})\, f(\mathbf{z}) \geq \sum_{\mathbf{z}} P(\mathbf{z}) \log f(\mathbf{z})$

$$ll(\theta : \mathcal{D}) = \sum_{i=1}^{N} \log \sum_{\mathbf{z}} Q_i(\mathbf{z}) \frac{P(\mathbf{x}_i, \mathbf{z} \mid \theta)}{Q_i(\mathbf{z})}$$

Evidence Based Lower Bound

- Define potential function F(θ,Q):

$$ll(\theta : \mathcal{D}) \geq F(\theta, Q_i) = \sum_{i=1}^{N} \sum_{\mathbf{z}} Q_i(\mathbf{z}) \log \frac{P(\mathbf{x}_i, \mathbf{z} \mid \theta)}{Q_i(\mathbf{z})}$$

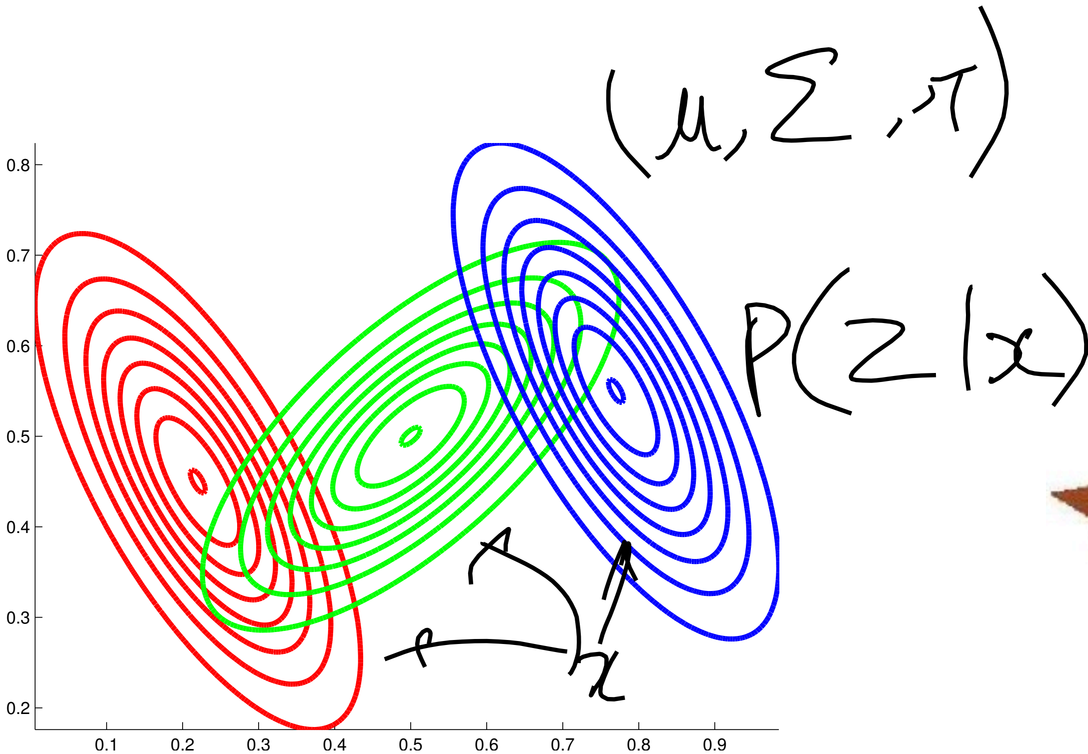
Evidence Based Lower Bound

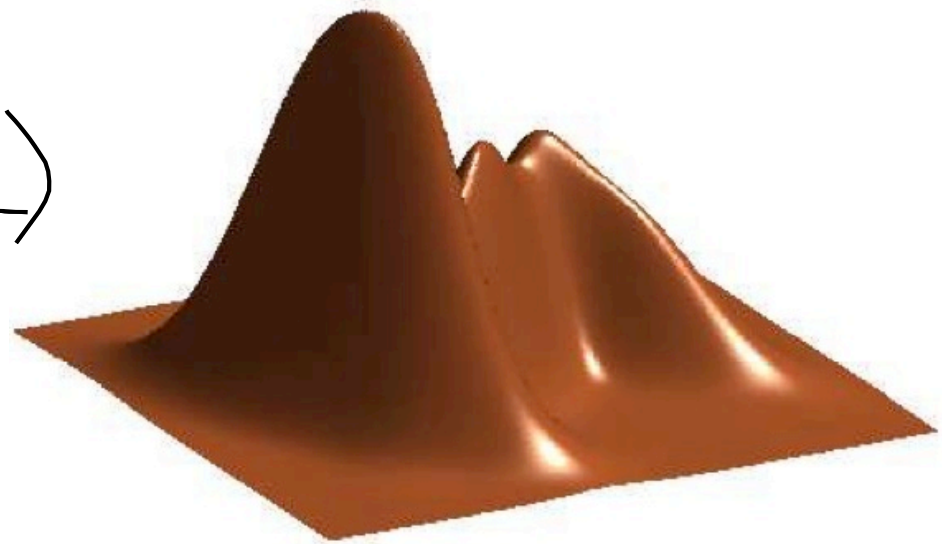
- Define potential function F(θ,Q):

$$ll(\theta : \mathcal{D}) \;\geq\; F(\theta, Q_i) \;=\; \sum_{i=1}^{N} \sum_{\mathbf{z}} Q_i(\mathbf{z}) \log \frac{P(\mathbf{x}_i, \mathbf{z} \mid \theta)}{Q_i(\mathbf{z})}$$

- EM corresponds to coordinate ascent on F
  - Thus, maximizes lower bound on marginal log likelihood

GMM

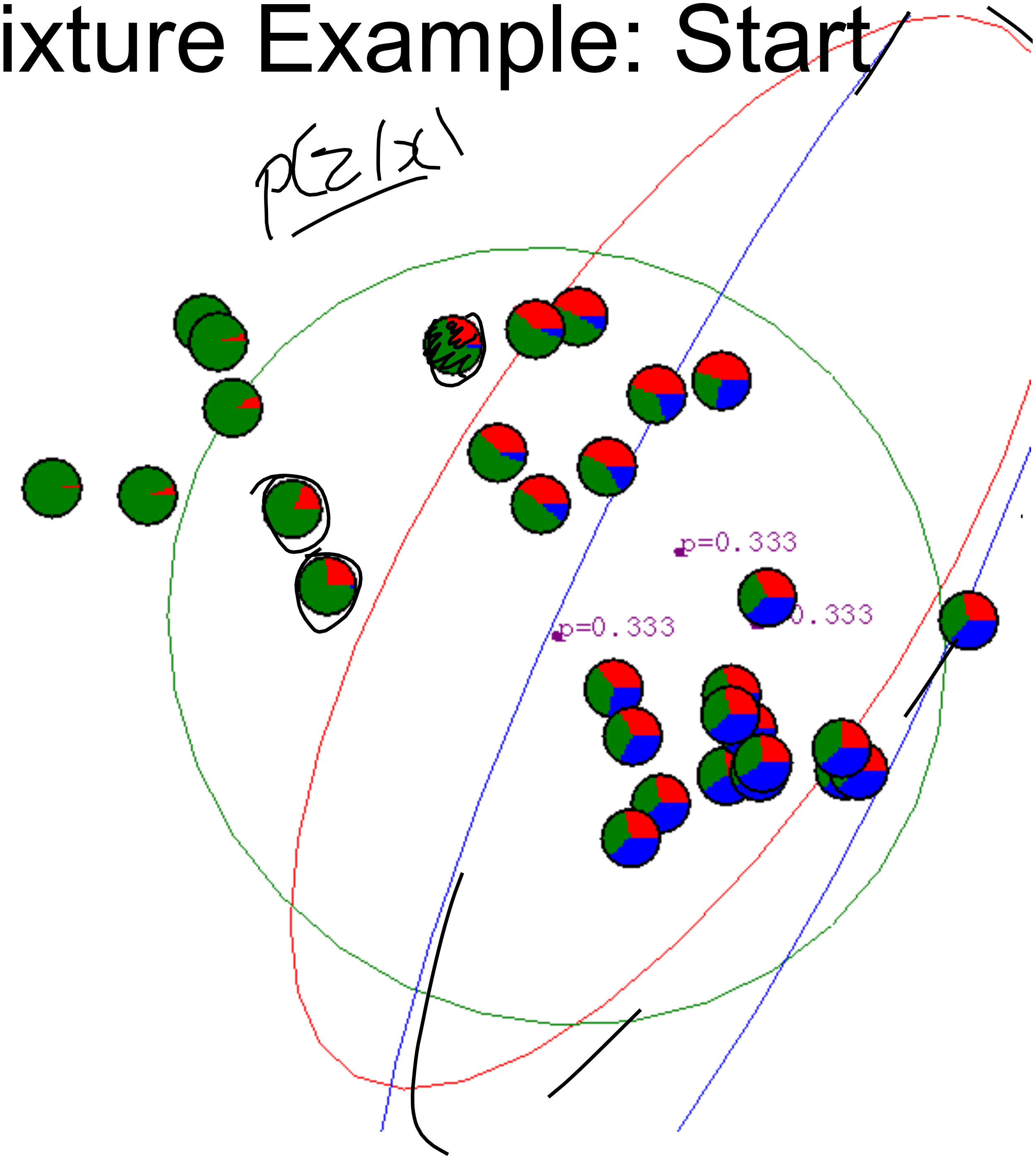




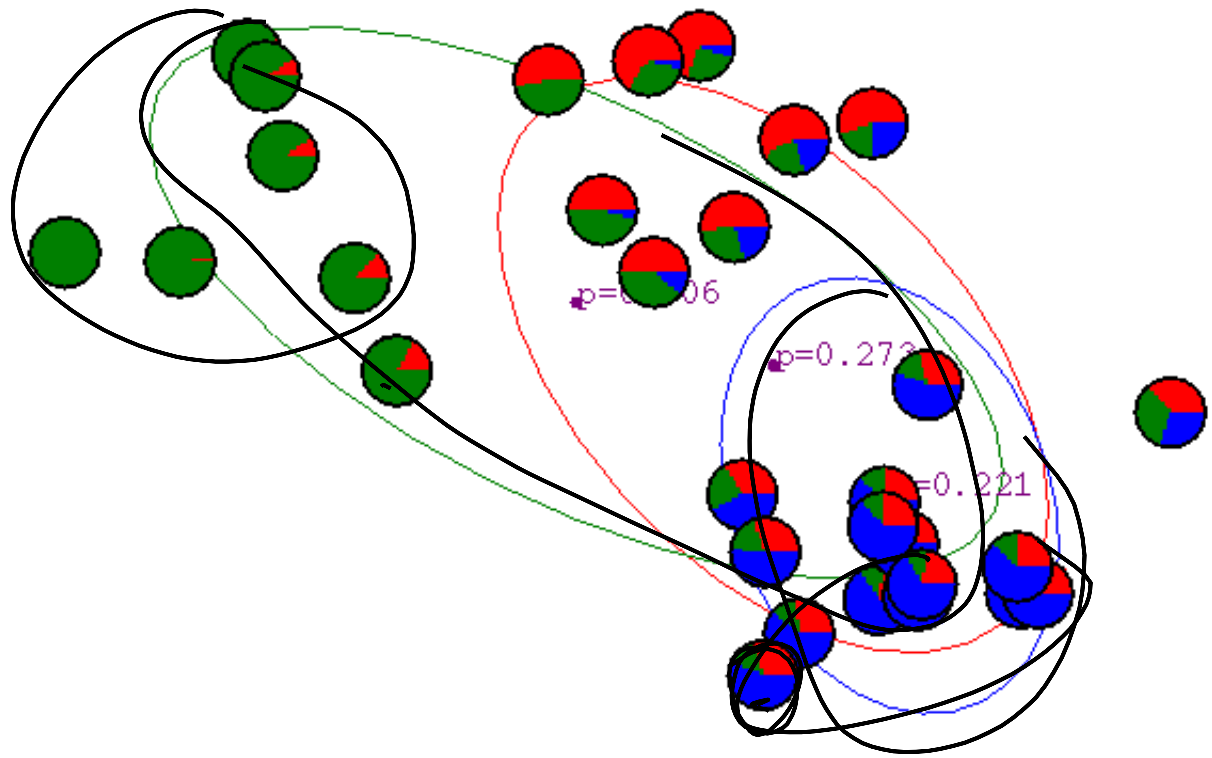
EM for Learning GMMs

- Simple Update Rules
  - E-Step: estimate $Q_i(z) = \Pr(z = j \mid x_i)$
  - M-Step: maximize full likelihood weighted by posterior

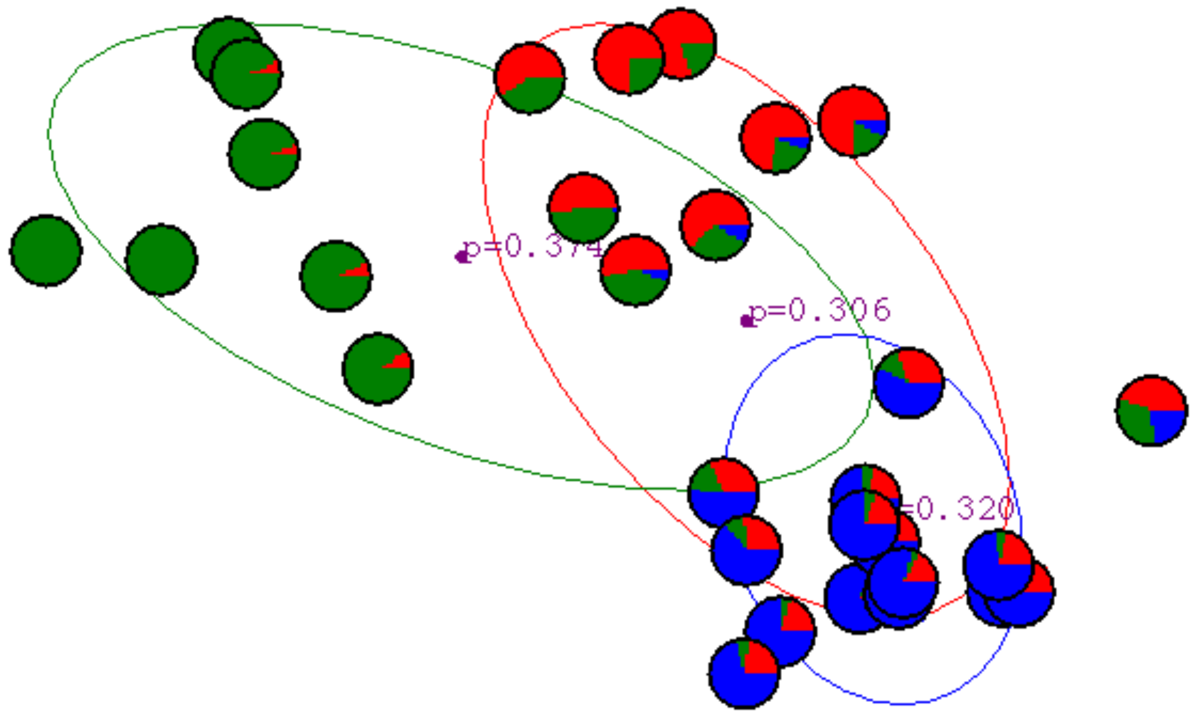
Gaussian Mixture Example: Start



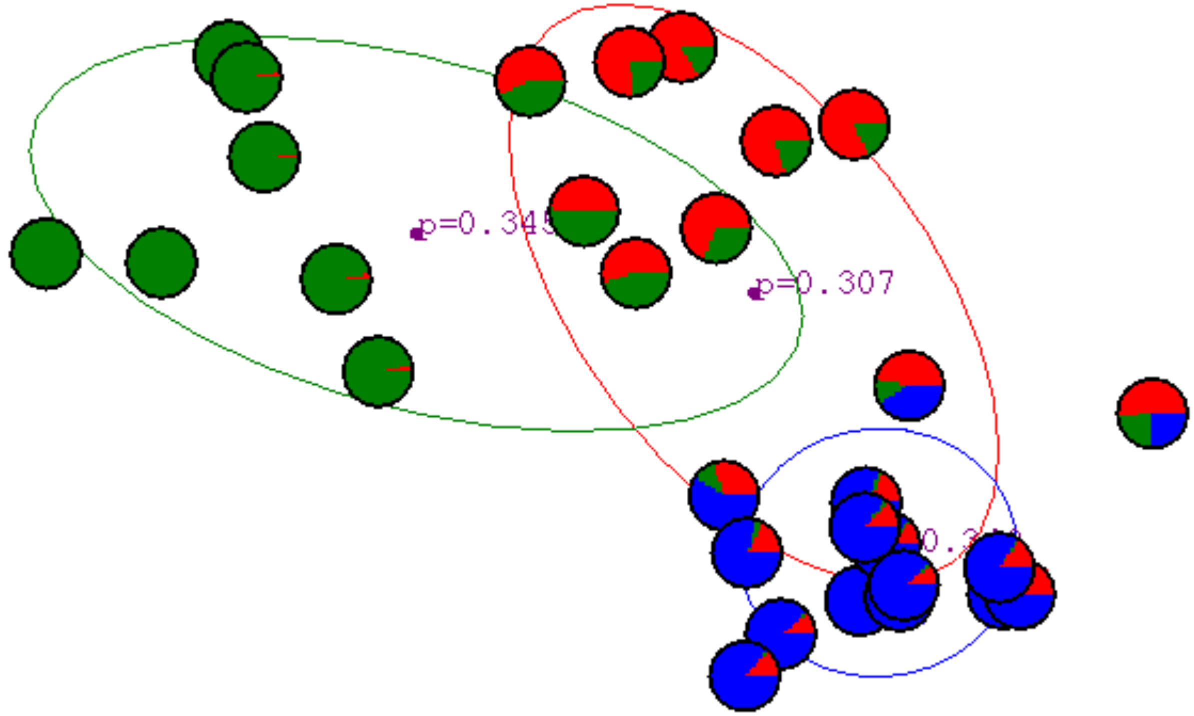
After 1st iteration



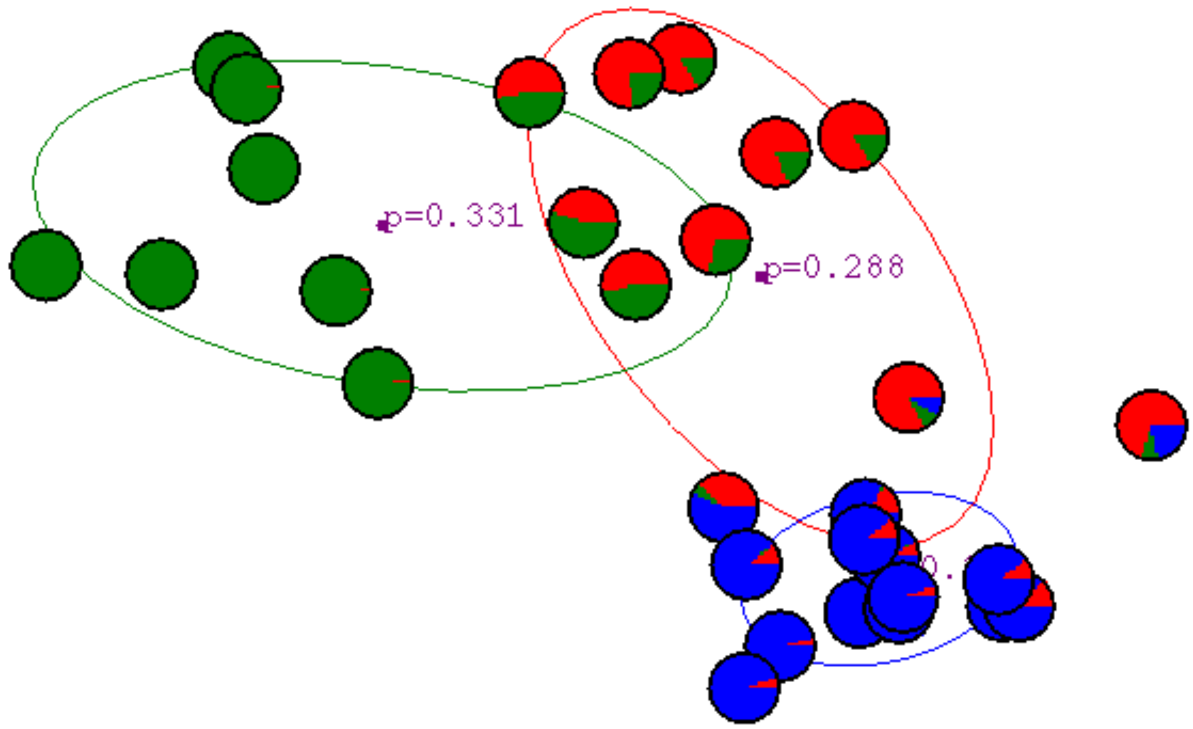
After 2nd iteration



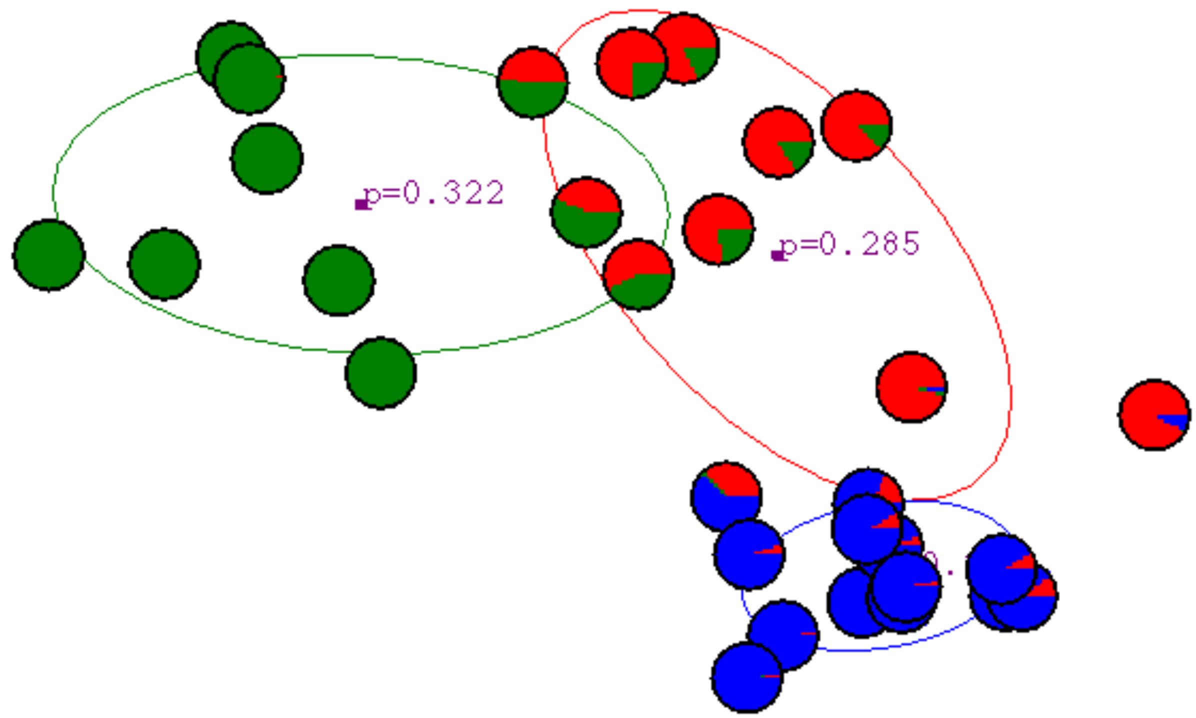
After 3rd iteration



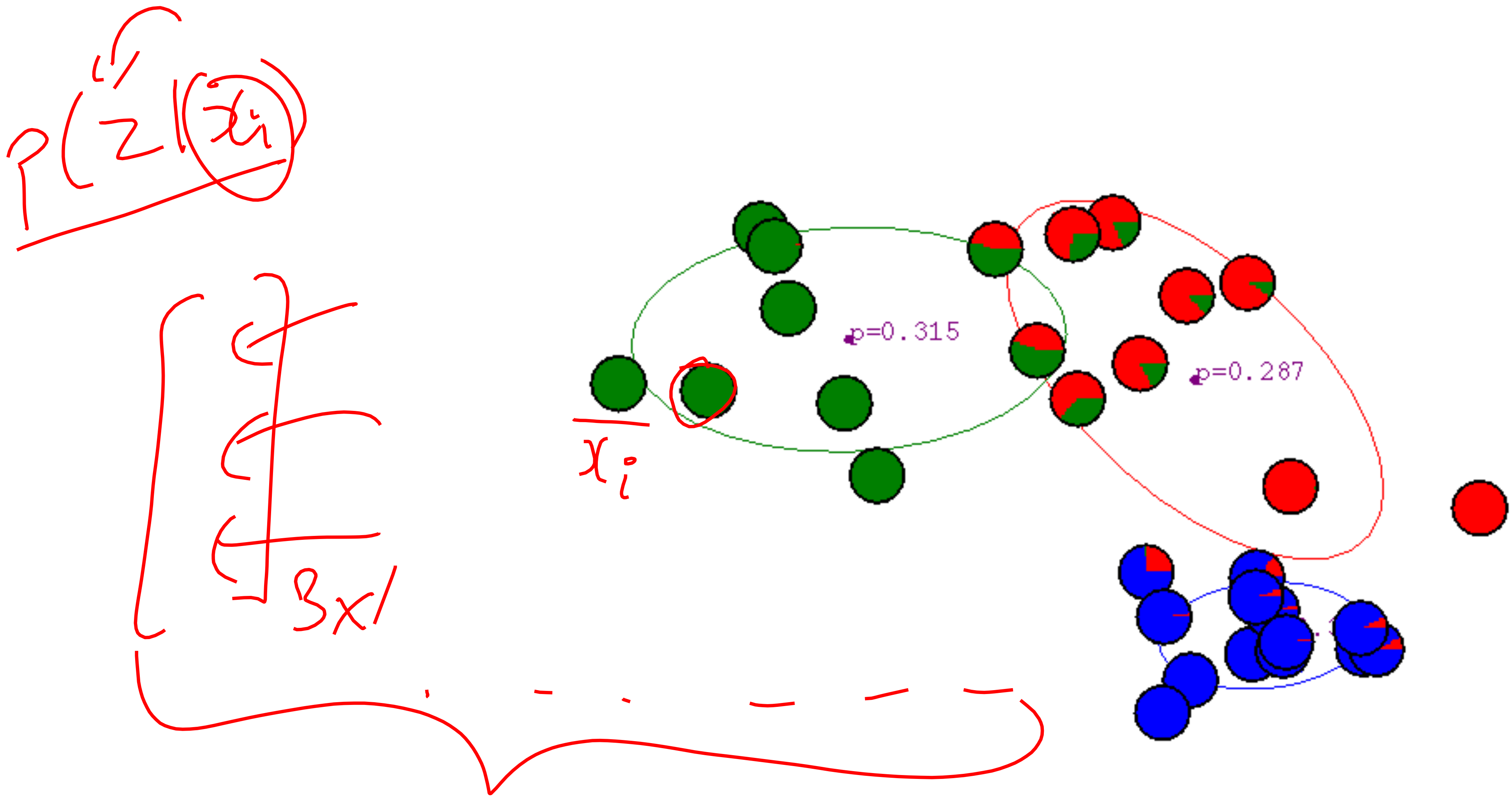
After 4th iteration



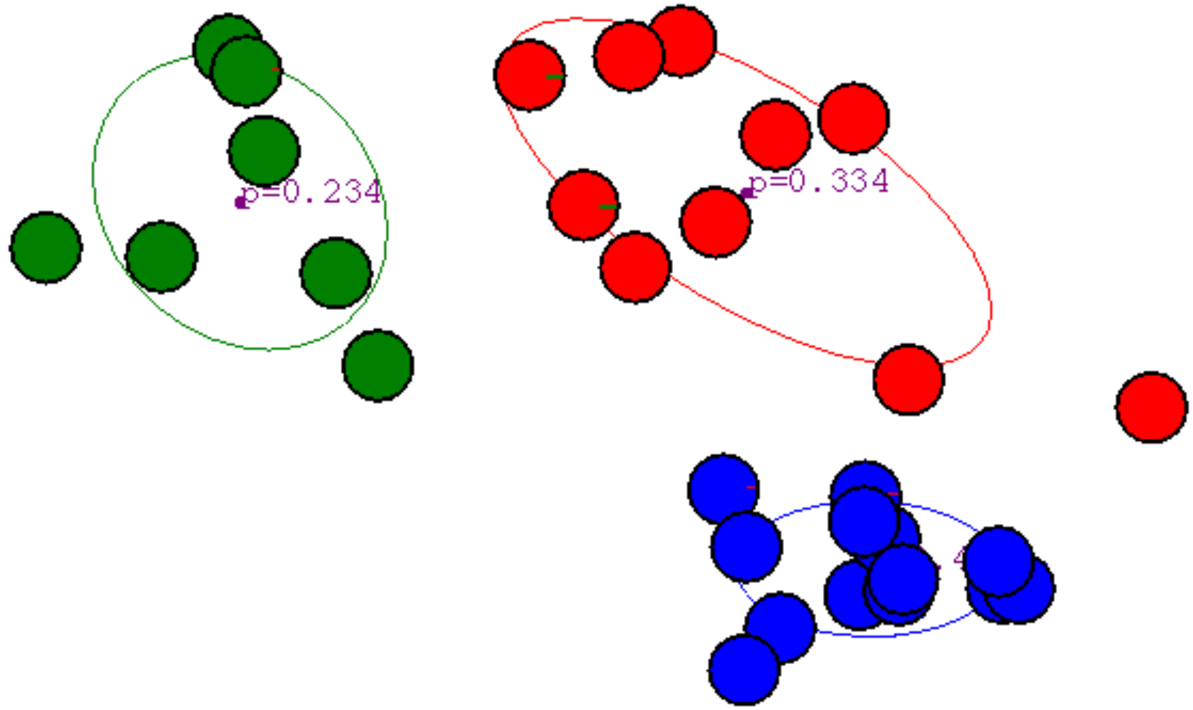
After 5th iteration



After 6th iteration



After 20th iteration



Variational Auto Encoders

VAEs are a combination of the following ideas:

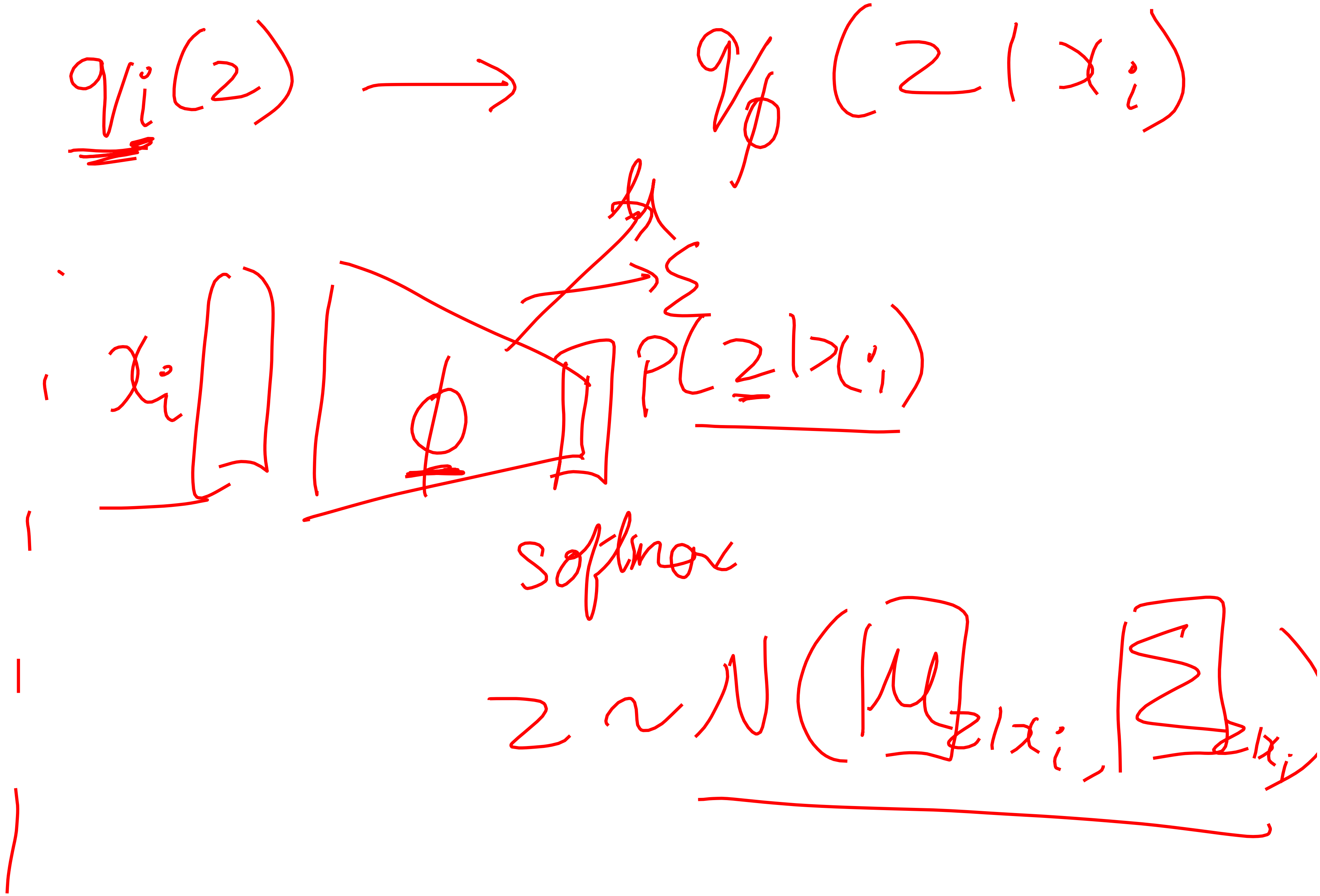
1. Auto Encoders

2. Variational Approximation
   - Variational Lower Bound / ELBO

3. Amortized Inference Neural Networks

4. "Reparameterization" Trick

Amortized Inference Neural Networks



Variational Auto Encoders

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 Auto Encoders

Putting it all together: maximizing the likelihood lower bound

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

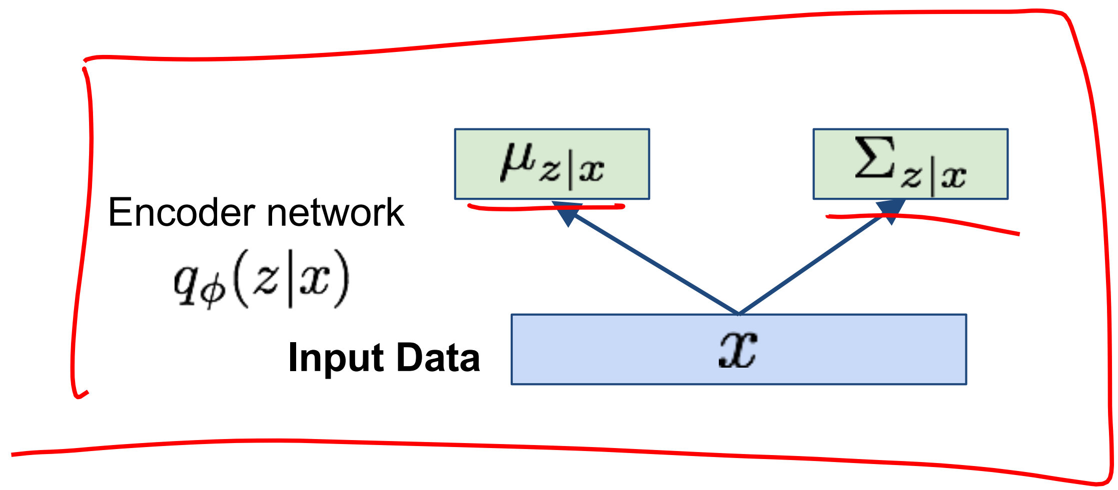
Let's look at computing the bound (forward pass) for a given minibatch of input data

Input Data x

Variational Auto Encoders

Putting it all together: maximizing the likelihood lower bound

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

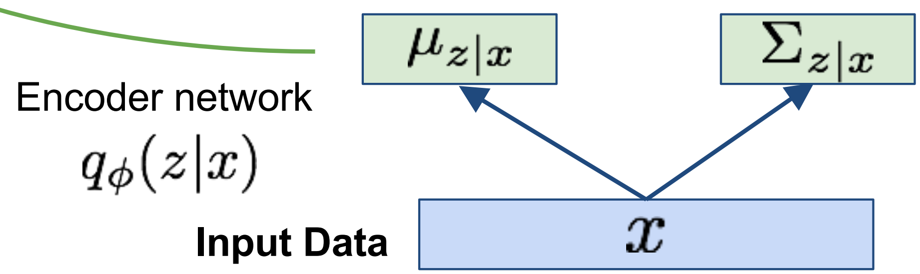


Variational Auto Encoders

Putting it all together: maximizing the likelihood lower bound

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

Make approximate posterior distribution close to prior

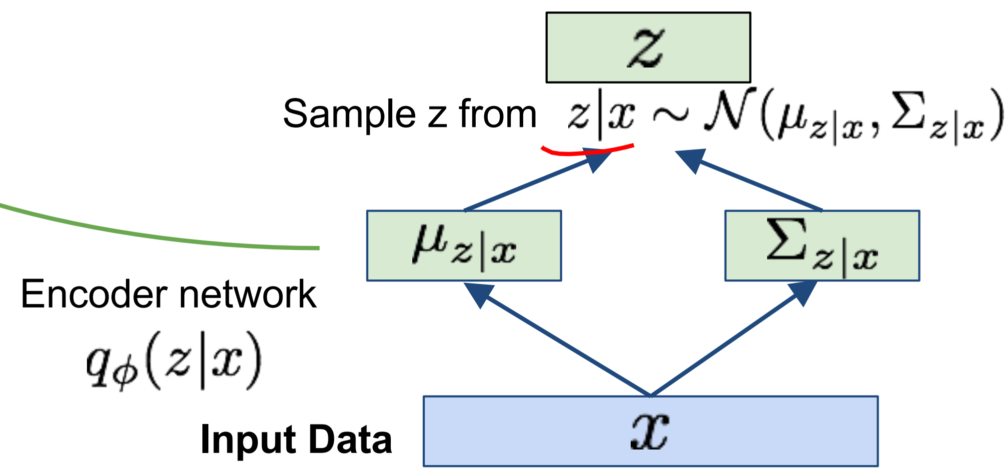


Variational Auto Encoders

Putting it all together: maximizing the likelihood lower bound

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

Make approximate posterior distribution close to prior

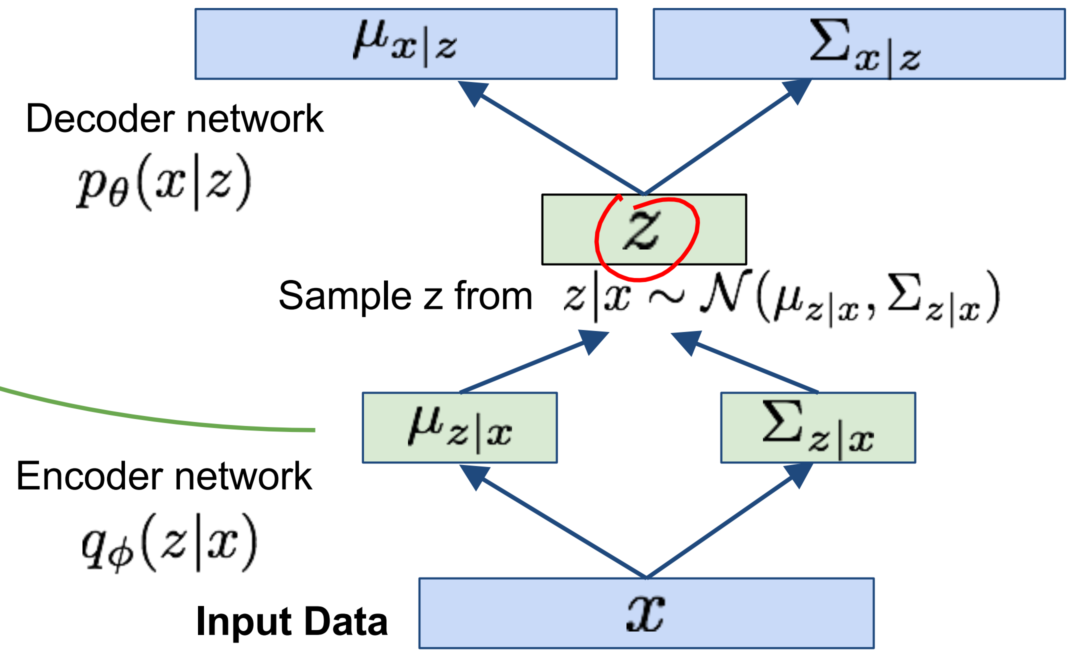


Variational Auto Encoders

Putting it all together: maximizing the likelihood lower bound

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

Make approximate posterior distribution close to prior



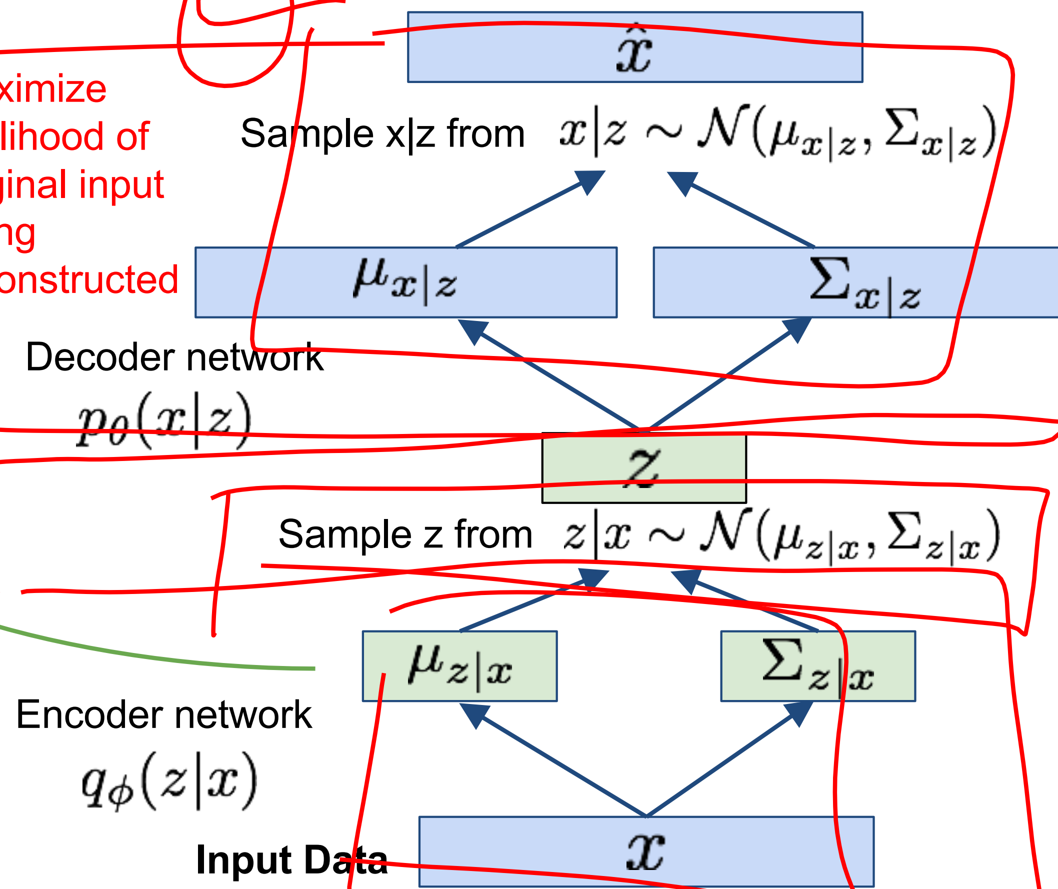
Variational Auto Encoders

Putting it all together: maximizing the likelihood lower bound

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

Maximize likelihood of original input being reconstructed

Make approximate posterior distribution close to prior



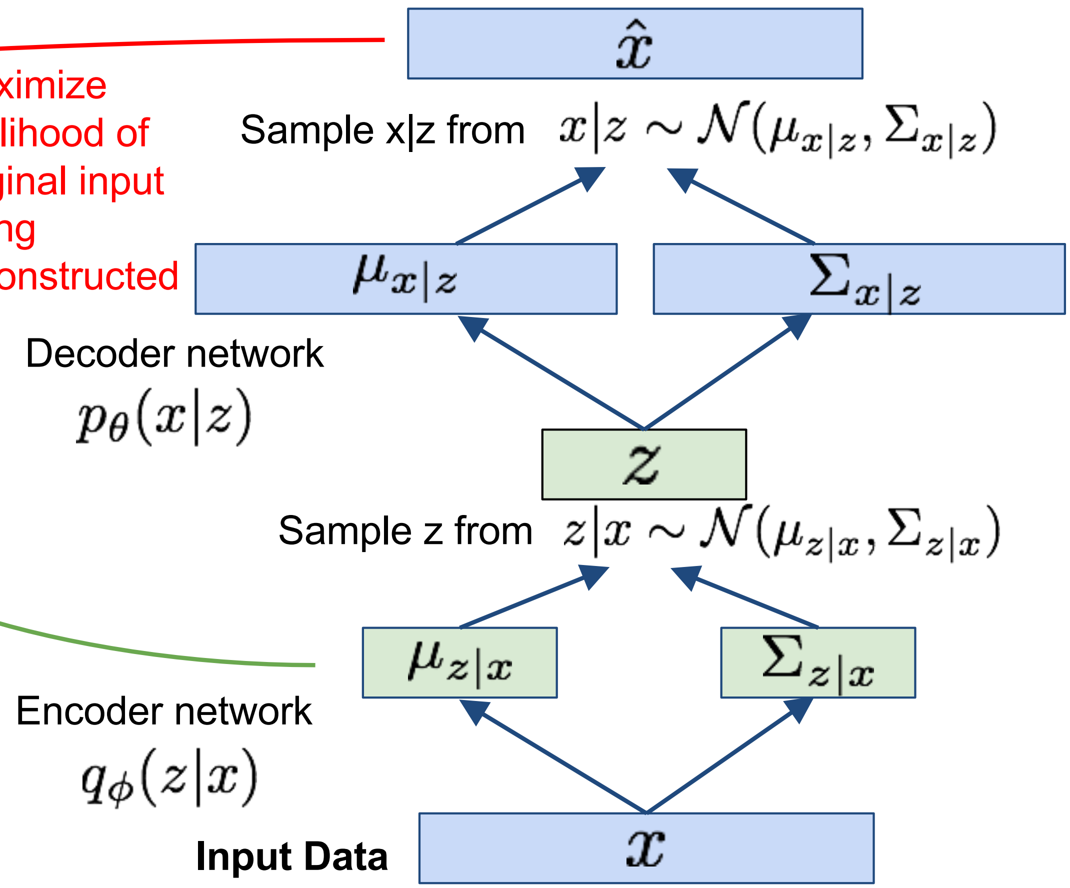
Variational Auto Encoders

Putting it all together: maximizing the likelihood lower bound

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

Maximize likelihood of original input being reconstructed

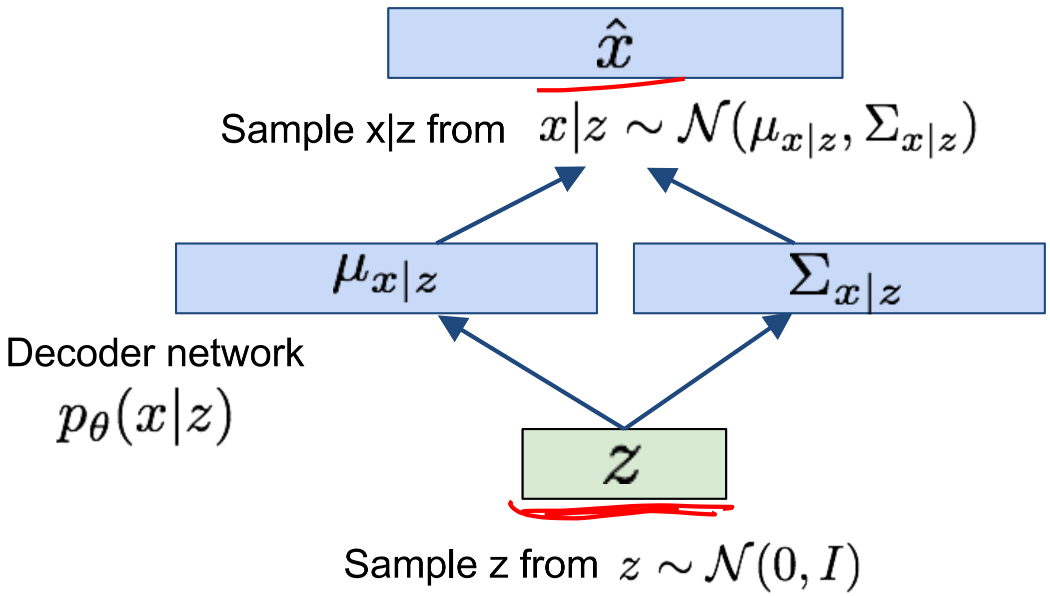
Make approximate posterior distribution close to prior



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

Variational Auto Encoders: Generating Data

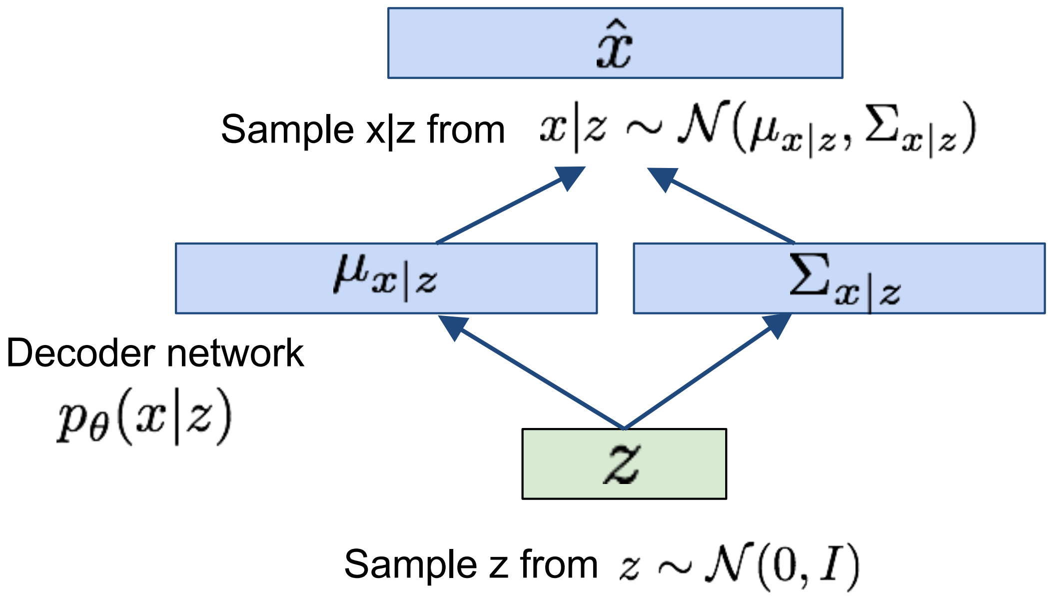
Use decoder network. Now sample z from prior!

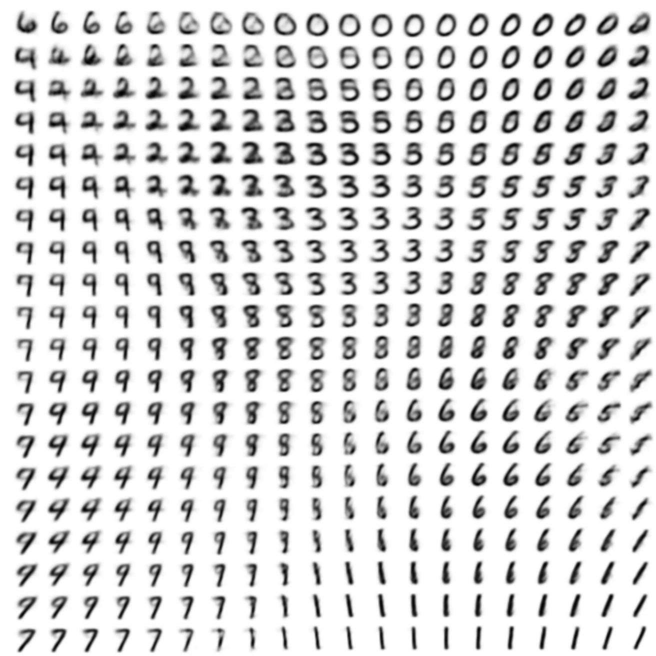


Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Auto Encoders: Generating Data

Use decoder network. Now sample z from prior!

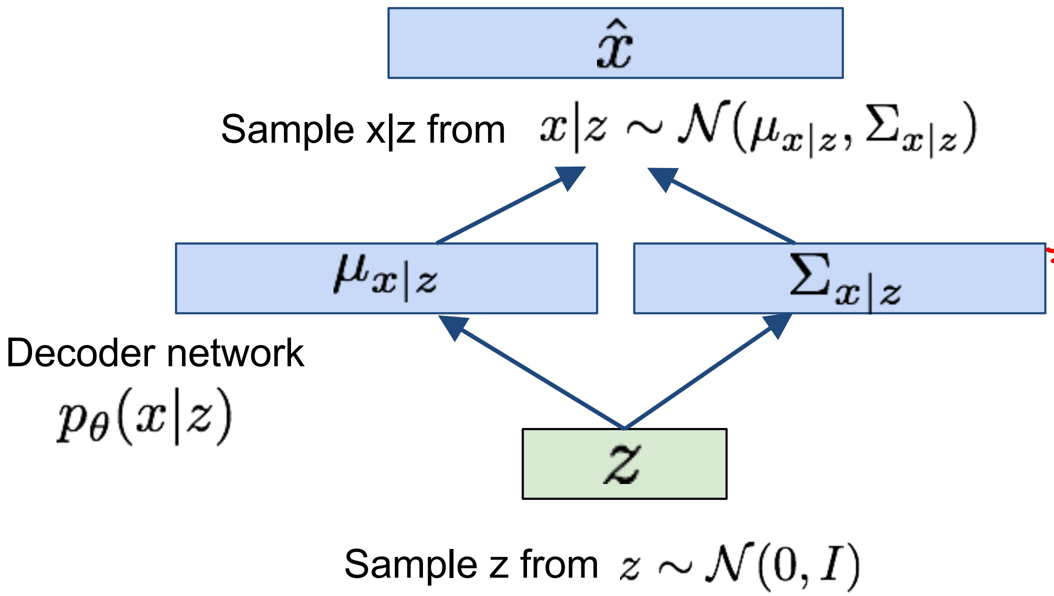




Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

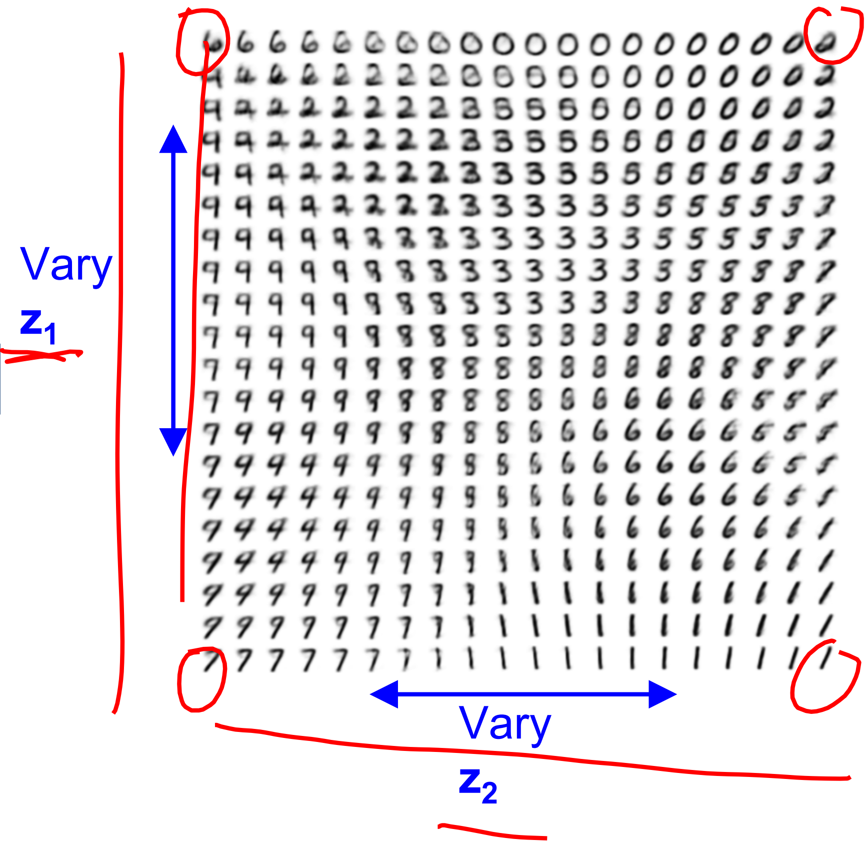
Variational Auto Encoders: Generating Data

Use decoder network. Now sample z from prior!



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

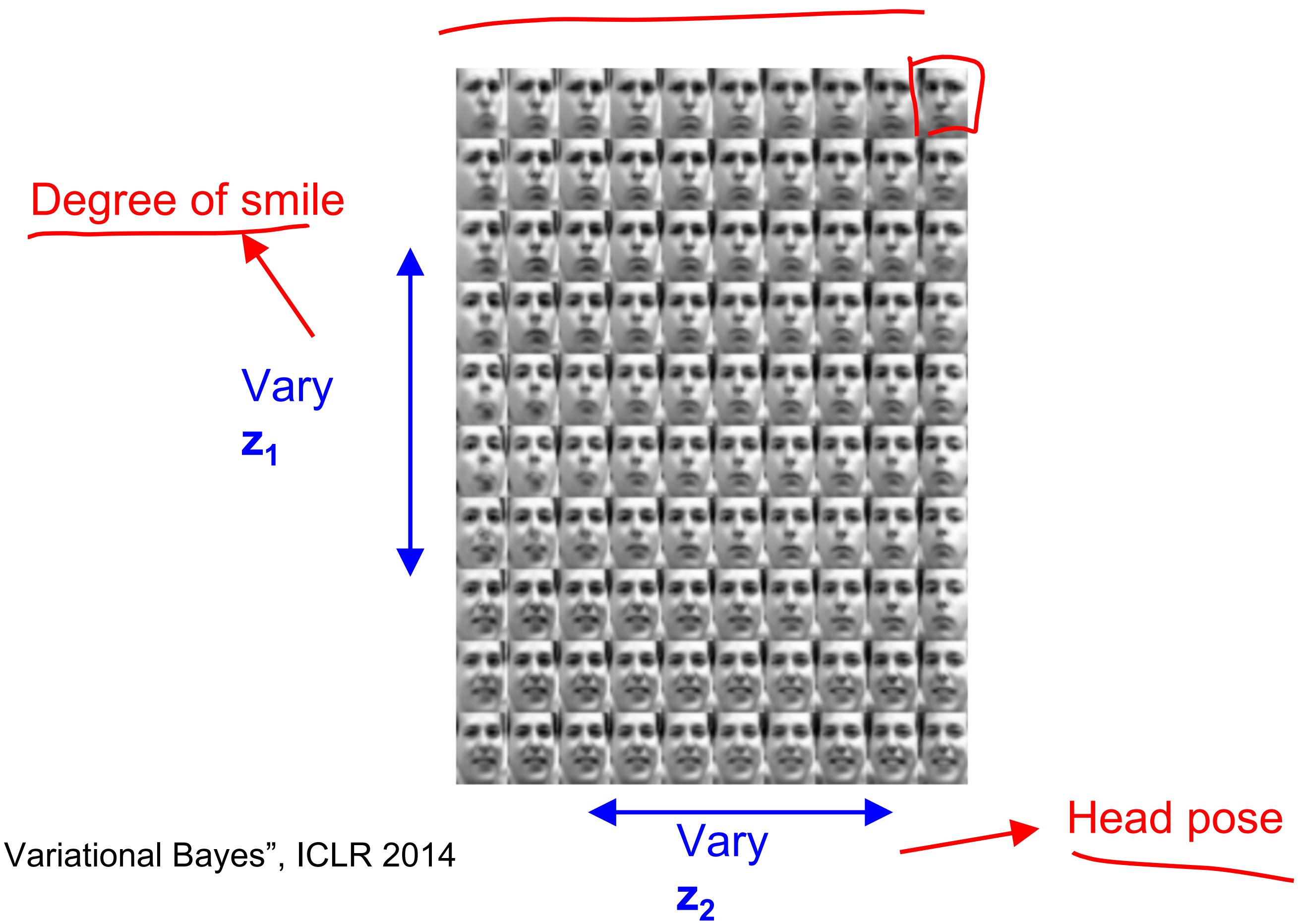
Data manifold for 2-d **z**



Variational Auto Encoders: 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 Auto Encoders: Generating Data

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

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

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

Degree of smile

Vary
$\mathbf{z_1}$

Vary
$\mathbf{z_2}$

Head pose

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

Variational Auto Encoders: Generating Data

32x32 CIFAR-10

Labeled Faces in the Wild

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
- 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
- Incorporating structure in latent variables

$P(x_{high} \mid x_{low})$

Generative Adversarial Networks (GAN)

So far...

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

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

VAEs define intractable density function with latent **z**:

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

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

So far...

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

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

VAEs define intractable density function with latent **z**:

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

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

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

So far...

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

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

VAEs define intractable density function with latent **z**:

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

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

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

GANs: don’t work with any explicit density function!

Generative Adversarial Networks

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

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

Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.

Q: What can we use to represent this complex transformation?

Generative Adversarial Networks

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

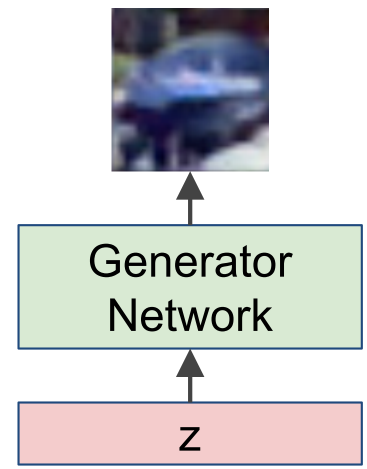
Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution, e.g. random noise. Learn transformation to training distribution.

Q: What can we use to represent this complex transformation?

A: A neural network!

Output: Sample from training distribution



Input: Random noise

Training GANs: Two-player game

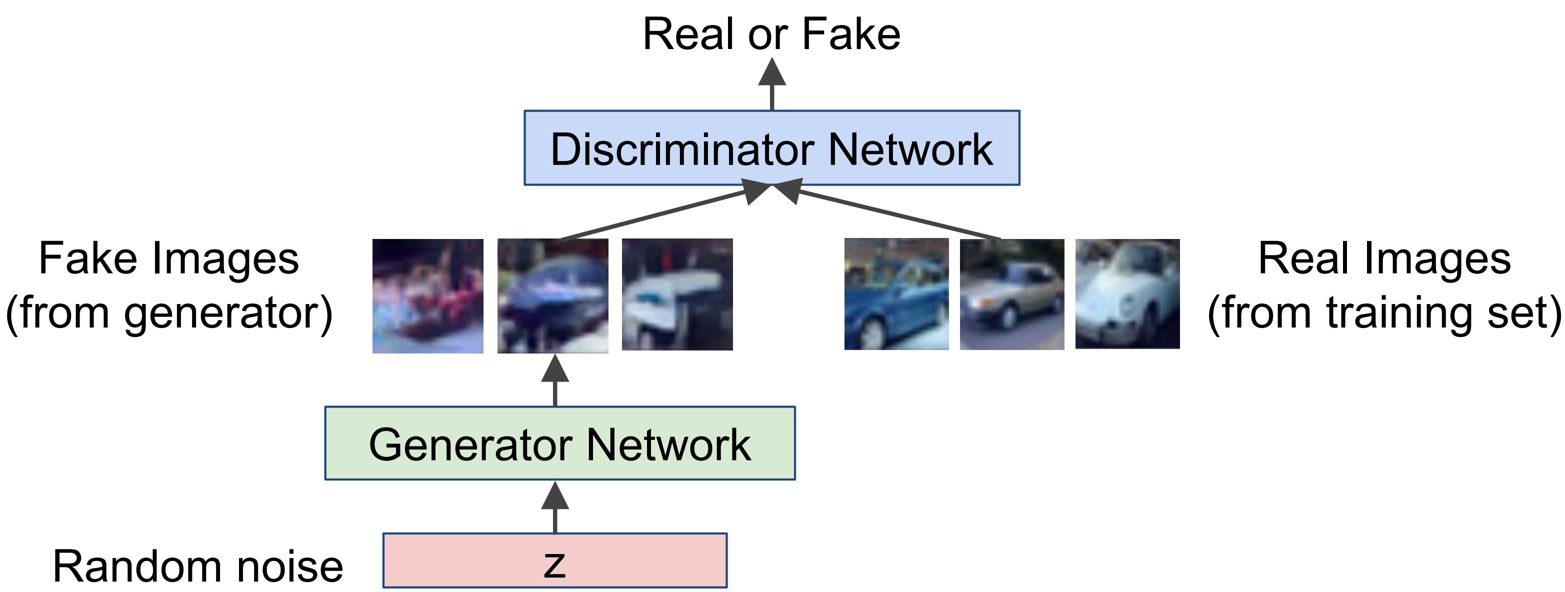
Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images

Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images



Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images

Train jointly in **minimax game**

Minimax objective function:

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

Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images

Train jointly in **minimax game**

Minimax objective function:

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

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

Discriminator output for real data x

Discriminator output for generated fake data G(z)

Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images

Train jointly in **minimax game**

Minimax objective function:

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

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

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

Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Minimax objective function:

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

Alternate between:

1. **Gradient ascent** on discriminator

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

2. **Gradient descent** on generator

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

Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Minimax objective function:

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

Alternate between:

1. **Gradient ascent** on discriminator

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

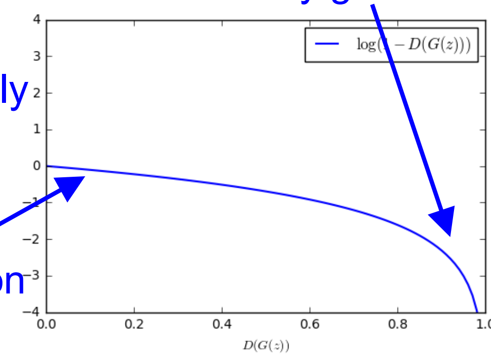
2. **Gradient descent** on generator

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

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

When sample is likely fake, want to learn from it to improve generator. But gradient in this region is relatively flat!

Gradient signal dominated by region where sample is already good



Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Minimax objective function:

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

Alternate between:

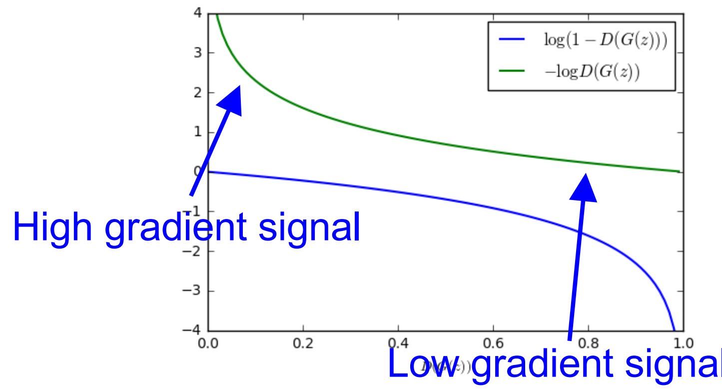
1. **Gradient ascent** on discriminator

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

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

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

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.
Same objective of fooling discriminator, but now higher gradient signal for bad samples => works much better! Standard in practice.



Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Minimax objective function:

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

Aside: Jointly training two networks is challenging, can be unstable. Choosing objectives with better loss landscapes helps training, is an active area of research.

Alternate between:

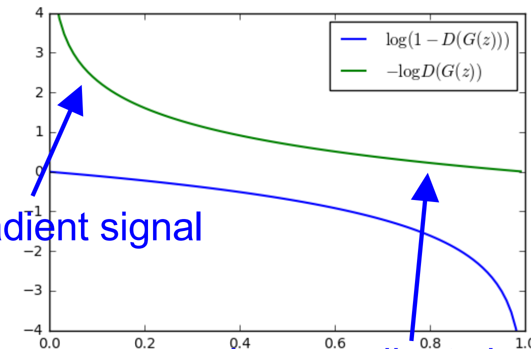
1. **Gradient ascent** on discriminator

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

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

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

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.
Same objective of fooling discriminator, but now higher gradient signal for bad samples => works much better! Standard in practice.



High gradient signal

Low gradient signal