CS 4803 / 7643: Deep Learning

Topics:

- Variational Auto-Encoders (VAEs)
- Key Ideas
  - AEs, Variational Inference

Dhruv Batra

Georgia Tech

Administrativia

- HW3 out
  - Due: 11/06, 11:55pm
- Final project
  - No poster session
  - Webpage submission
    - Details out soon

Recap from last time

Overview

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

So far… Supervised Learning

$x \xrightarrow{CNN} y$

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

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

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.

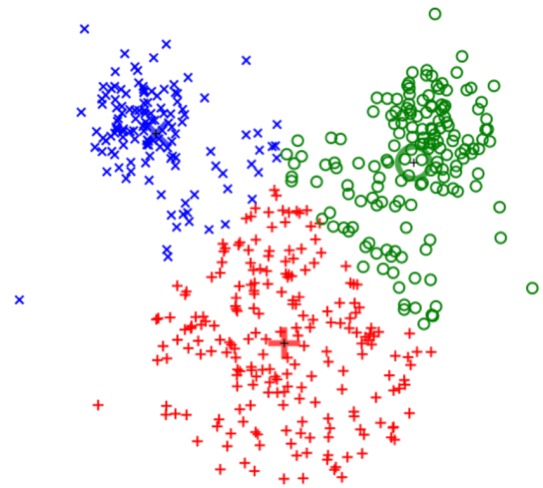
Supervised vs Unsupervised Learning

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

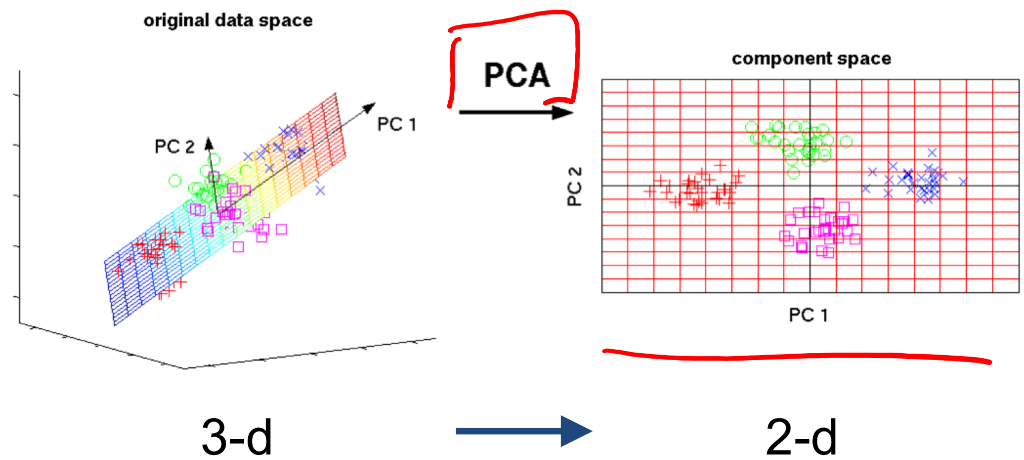
Supervised vs Unsupervised Learning

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)

$\boxed{p(x)}$ $p(x_1, \ldots, x_n)$

Supervised vs Unsupervised Learning

Unsupervised Learning

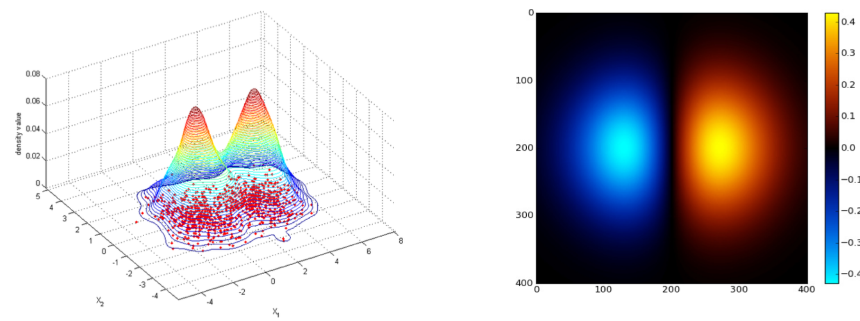
Data: x
Just data, no labels!

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

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

Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

1-d density estimation



2-d density estimation

Tasks

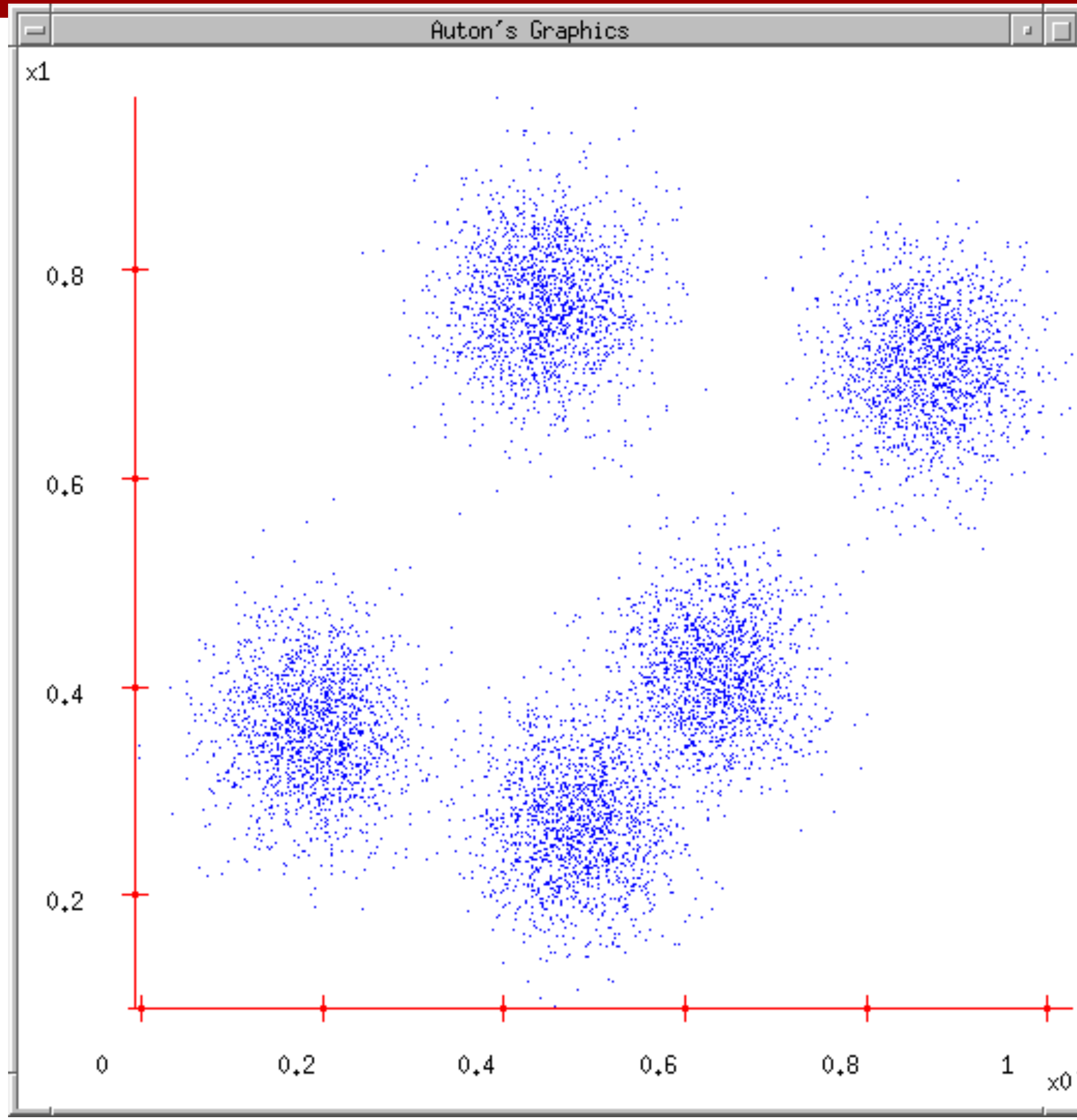
Supervised Learning

x	→	Classification	→ y	Discrete
x	→	Regression	→ y	Continuous

Unsupervised Learning

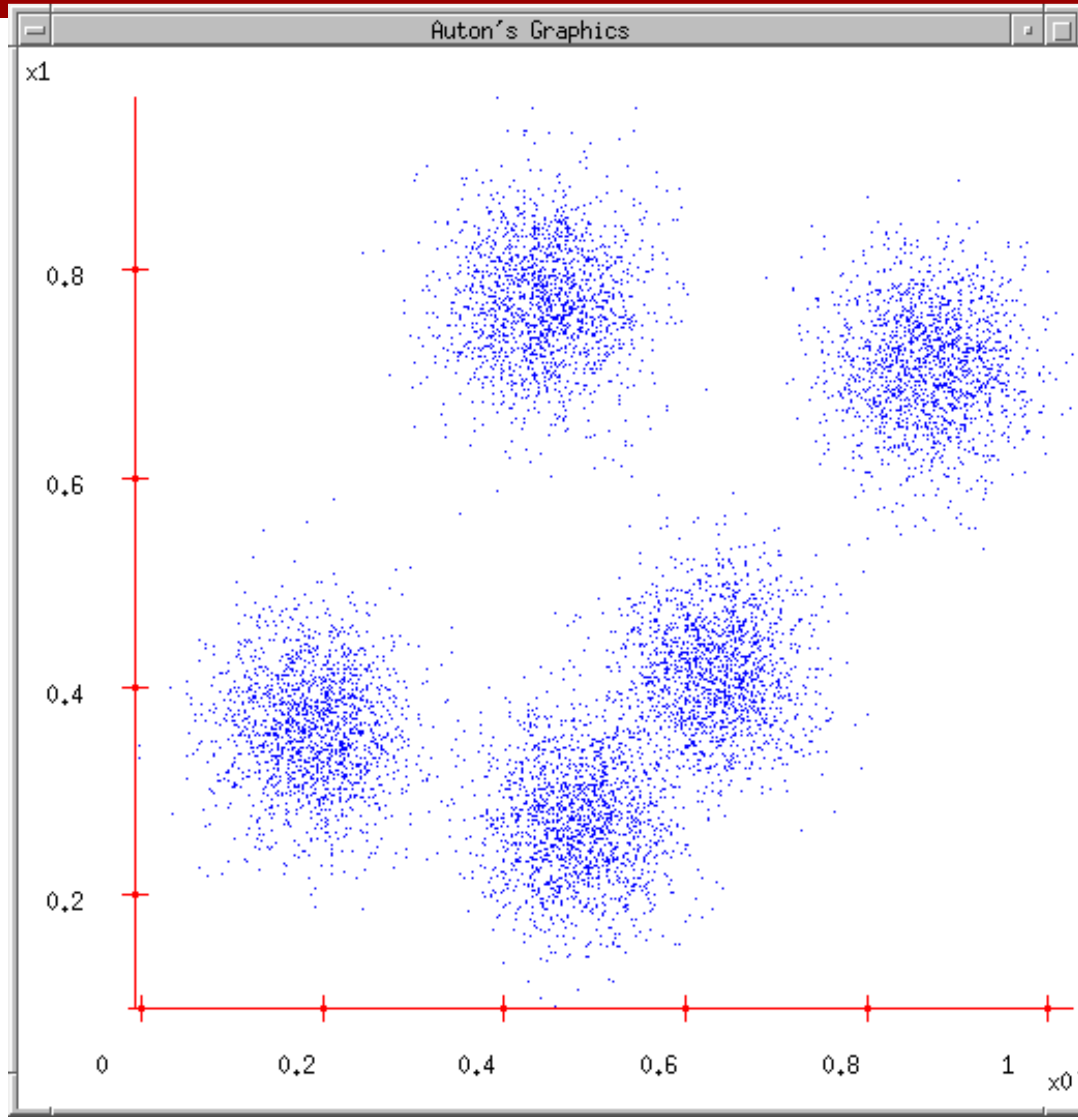
x	→	Clustering	→ c	Discrete
x	→	Dimensionality Reduction	→ z	Continuous
x	→	Density Estimation	→ p(x)	On simplex

Some Data



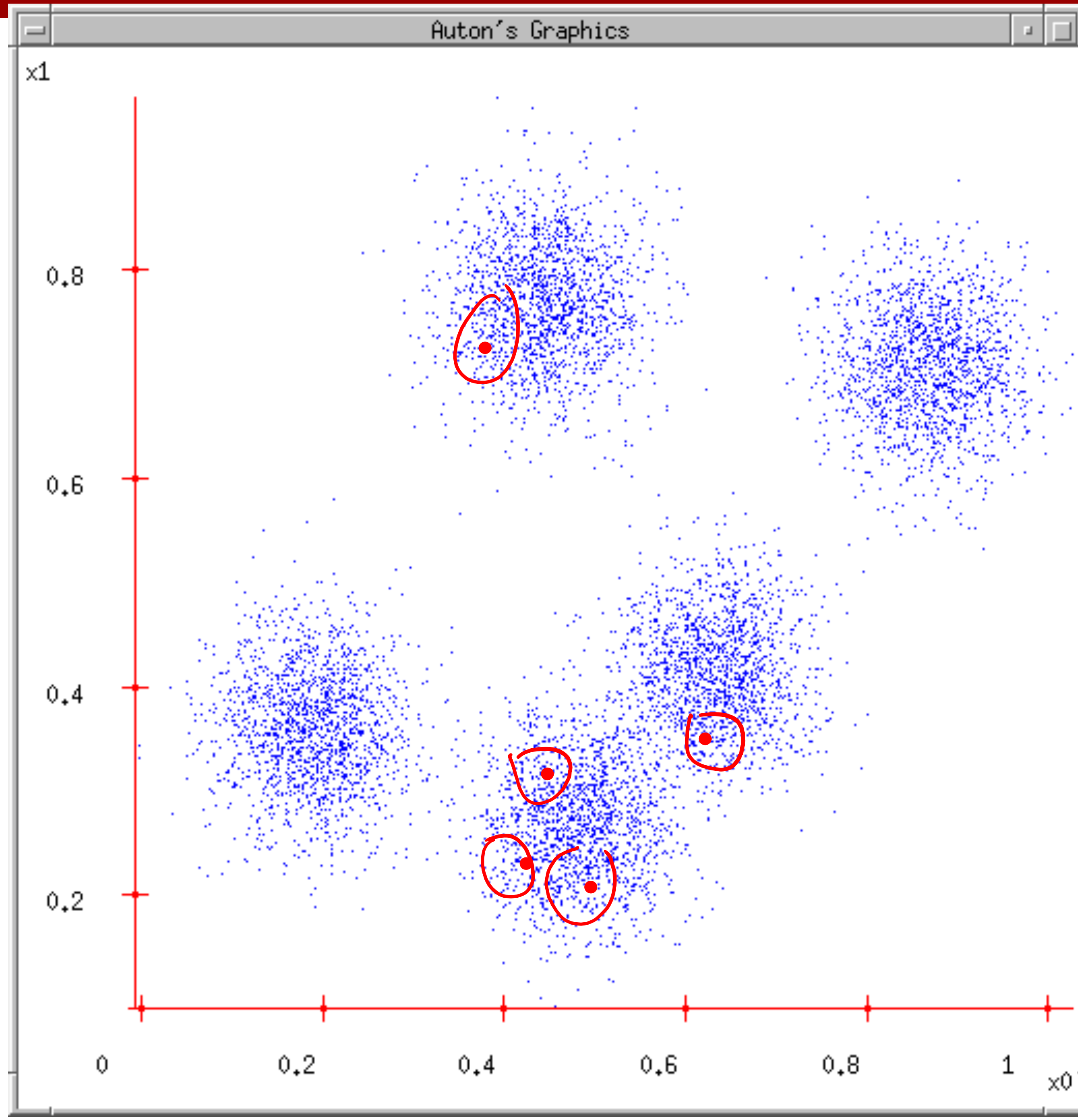
K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*



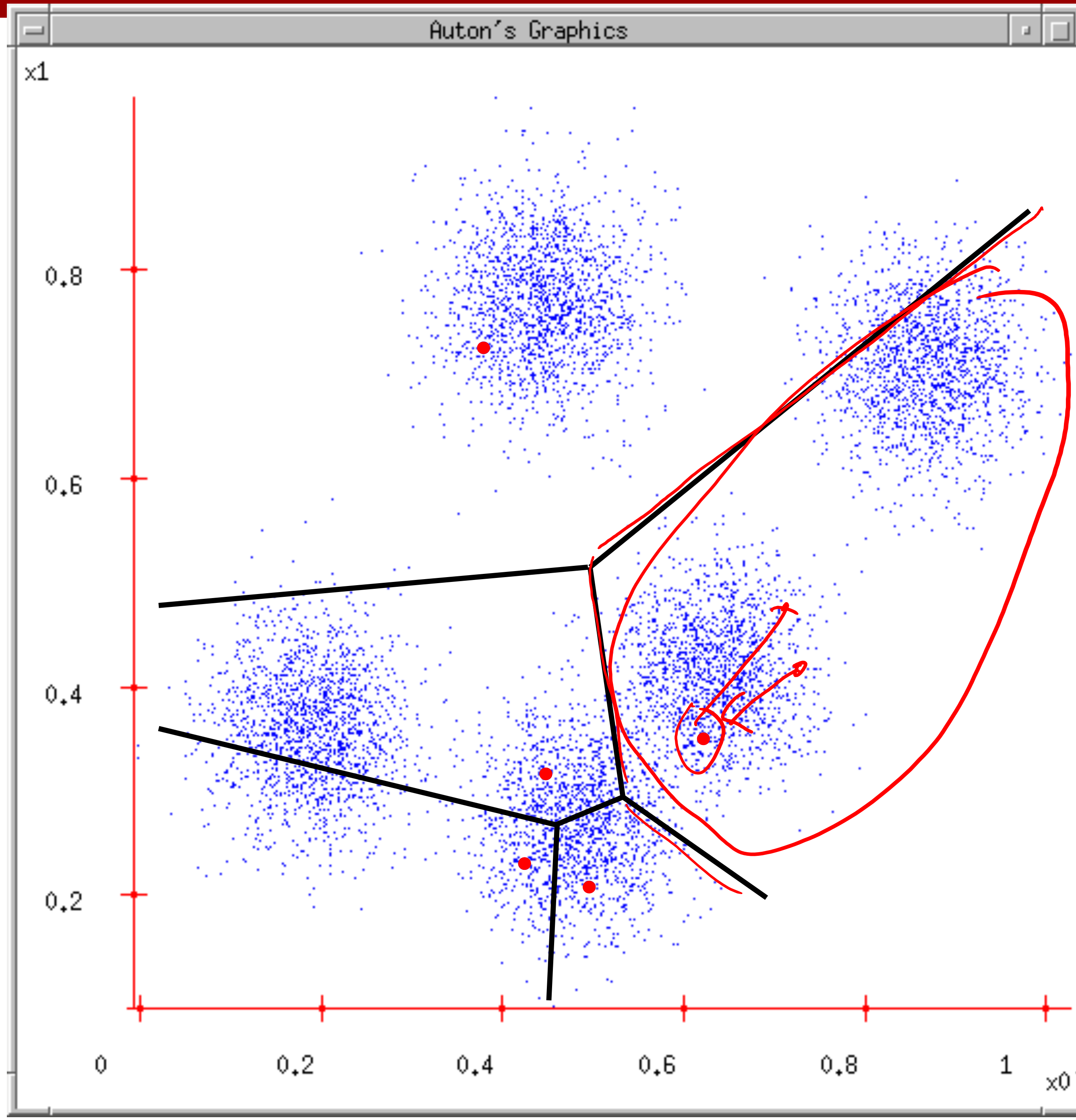
K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations



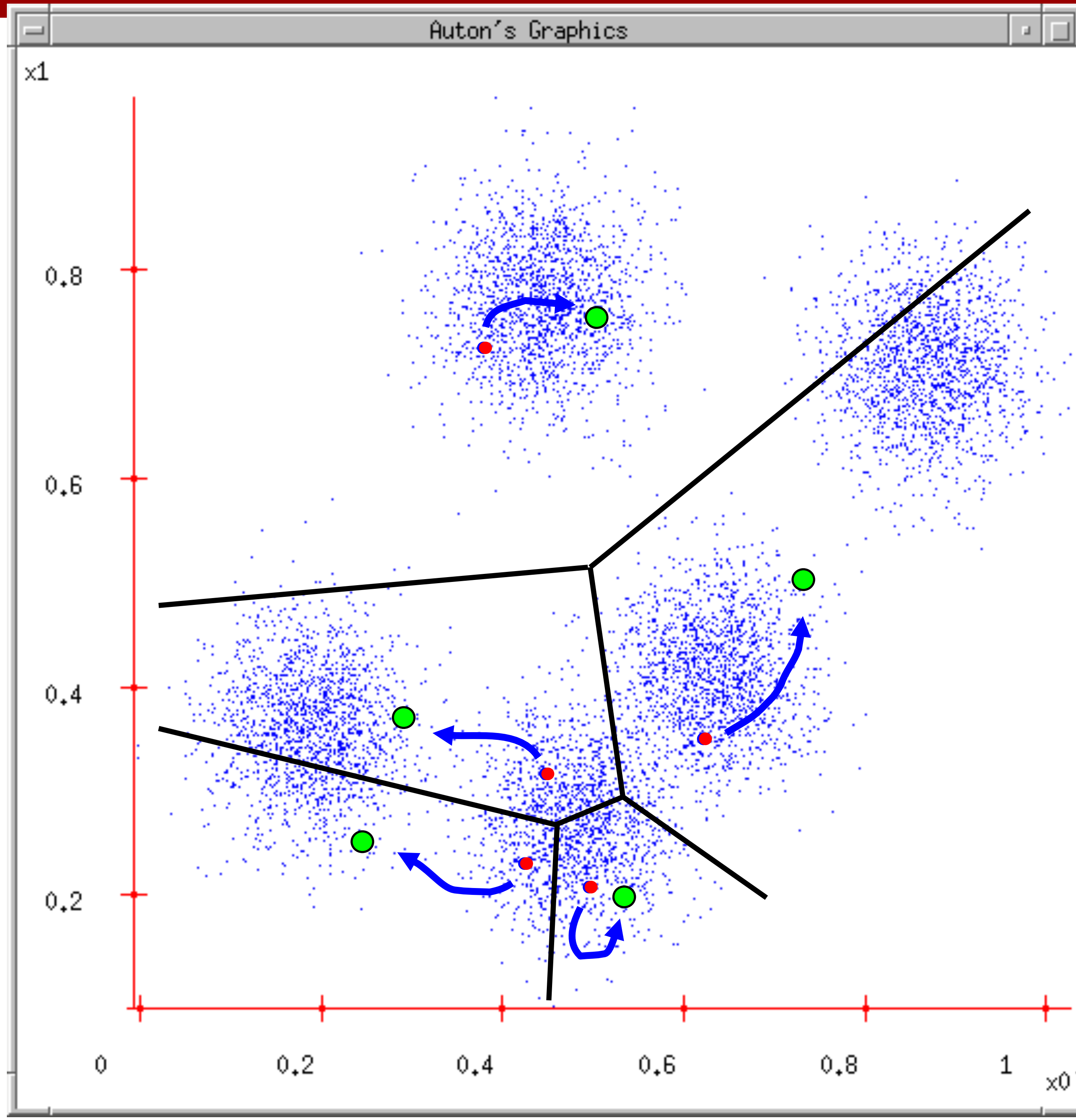
K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to. (Thus each Center "owns" a set of datapoints)



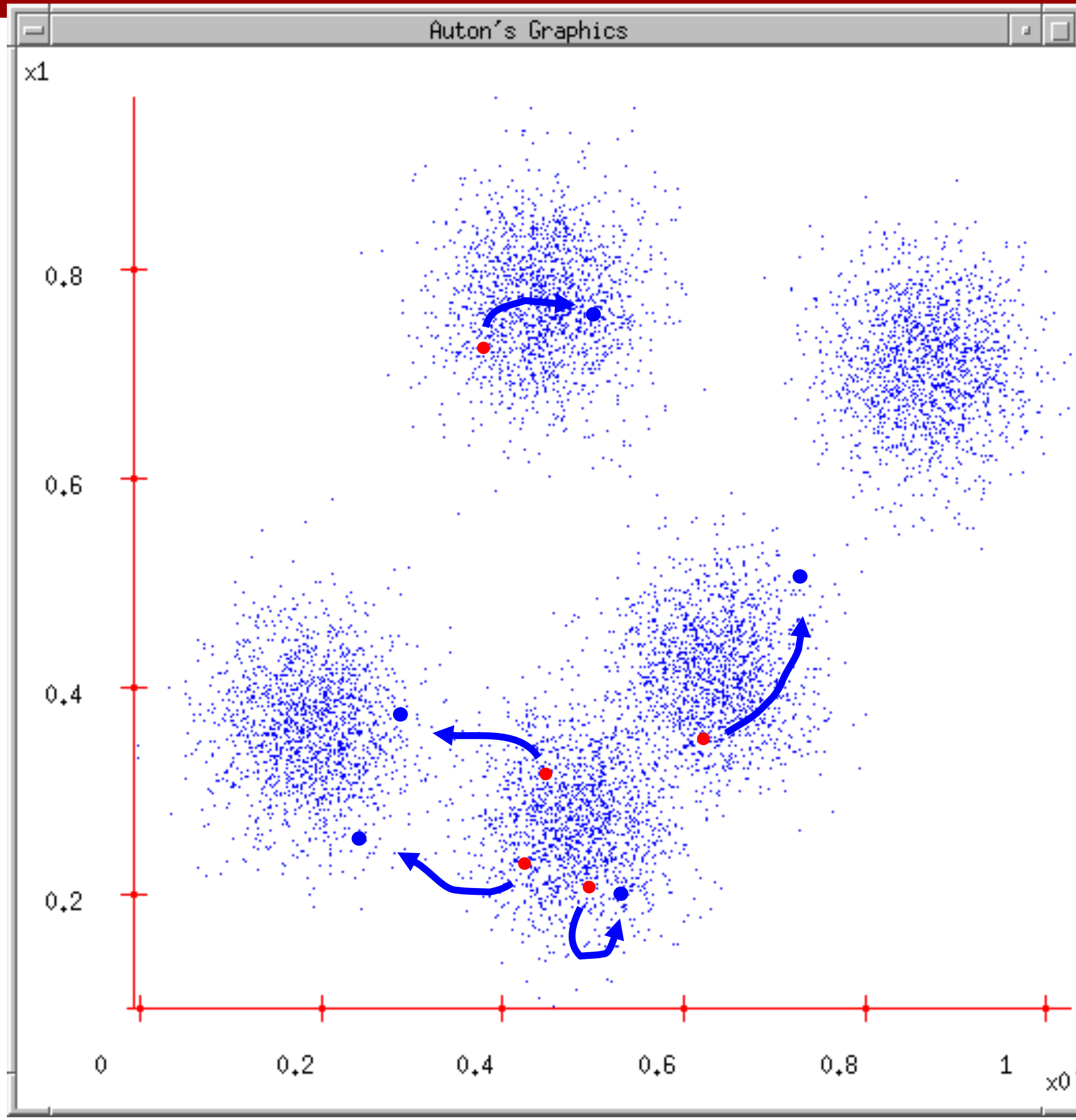
K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns



K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns…
5. …and jumps there
6. …Repeat until terminated!



K-means

- Randomly initialize *k* centers
  - $\mu^{(0)} = \mu_1^{(0)}, \ldots, \mu_k^{(0)}$

- **Assign**:
  - Assign each point $i \in \{1, \ldots n\}$ to nearest center:
  - $C(i) \longleftarrow \underset{j}{\text{argmin}} \, ||\mathbf{x}_i - \boldsymbol{\mu}_j||^2$

- **Recenter**:
  - μ_j becomes centroid of its points

What is K-means optimizing?

- Objective F(μ,C): function of centers μ and point allocations C:
  - $$\boxed{F(\boldsymbol{\mu}, C)} = \boxed{\sum_{i=1}^{N} ||\mathbf{x}_i - \boldsymbol{\mu}_{C(i)}||^2}$$
  - 1-of-k encoding $$F(\boldsymbol{\mu}, \boldsymbol{a}) = \sum_{i=1}^{N} \sum_{j=1}^{k} a_{ij} ||\mathbf{x}_i - \boldsymbol{\mu}_j||^2$$
- Optimal K-means:
  - $\min_{\mu} \min_{a} F(\mu, a)$

K-means as Co-ordinate Descent

- Optimize objective function:

$$\min_{\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_k} \min_{\boldsymbol{a}_1,\ldots,\boldsymbol{a}_N} F(\boldsymbol{\mu}, \boldsymbol{a}) = \min_{\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_k} \min_{\boldsymbol{a}_1,\ldots,\boldsymbol{a}_N} \sum_{i=1}^{N} \sum_{j=1}^{k} \boxed{a_{ij}} ||\mathbf{x}_i - \boldsymbol{\mu}_j||^2$$

- Fix μ, optimize a (or C)

K-means as Co-ordinate Descent

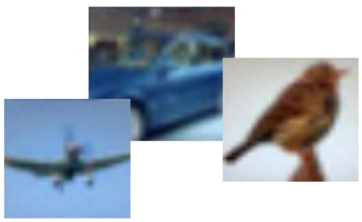
- Optimize objective function:

$$\min_{\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_k} \min_{\boldsymbol{a}_1,\ldots,\boldsymbol{a}_N} F(\boldsymbol{\mu}, \boldsymbol{a}) = \min_{\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_k} \min_{\boldsymbol{a}_1,\ldots,\boldsymbol{a}_N} \sum_{i=1}^{N} \sum_{j=1}^{k} a_{ij} ||\mathbf{x}_i - \boldsymbol{\mu}_j||^2$$

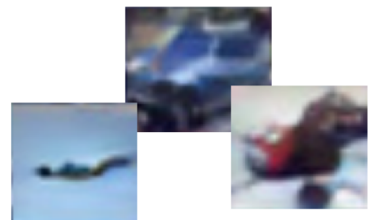
- Fix a (or C), optimize **μ**

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 $p_{data}(x)$

$p_{model}(x)$

Taxonomy of Generative Models

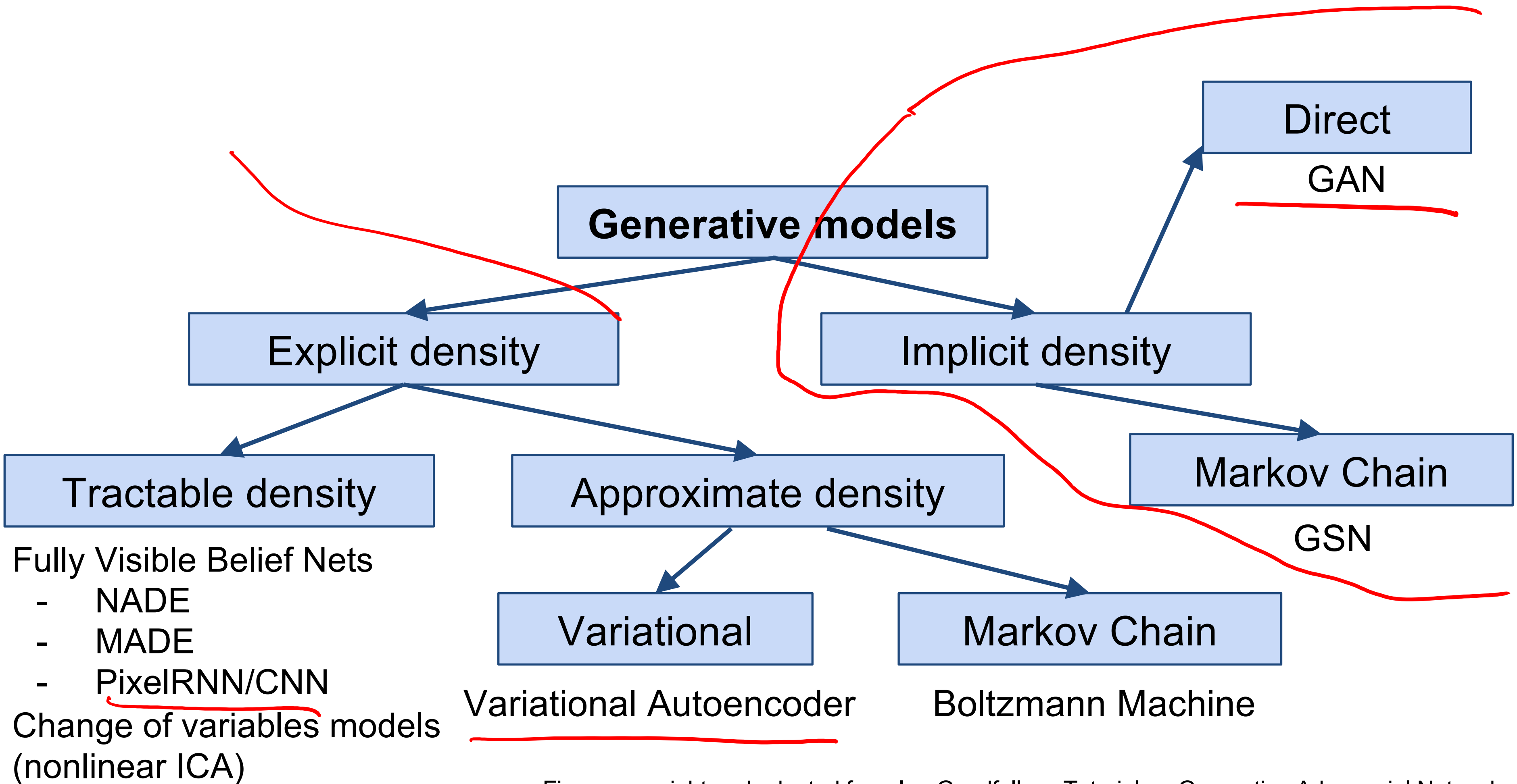


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Taxonomy of Generative Models

We will discuss 3 most popular types of generative models

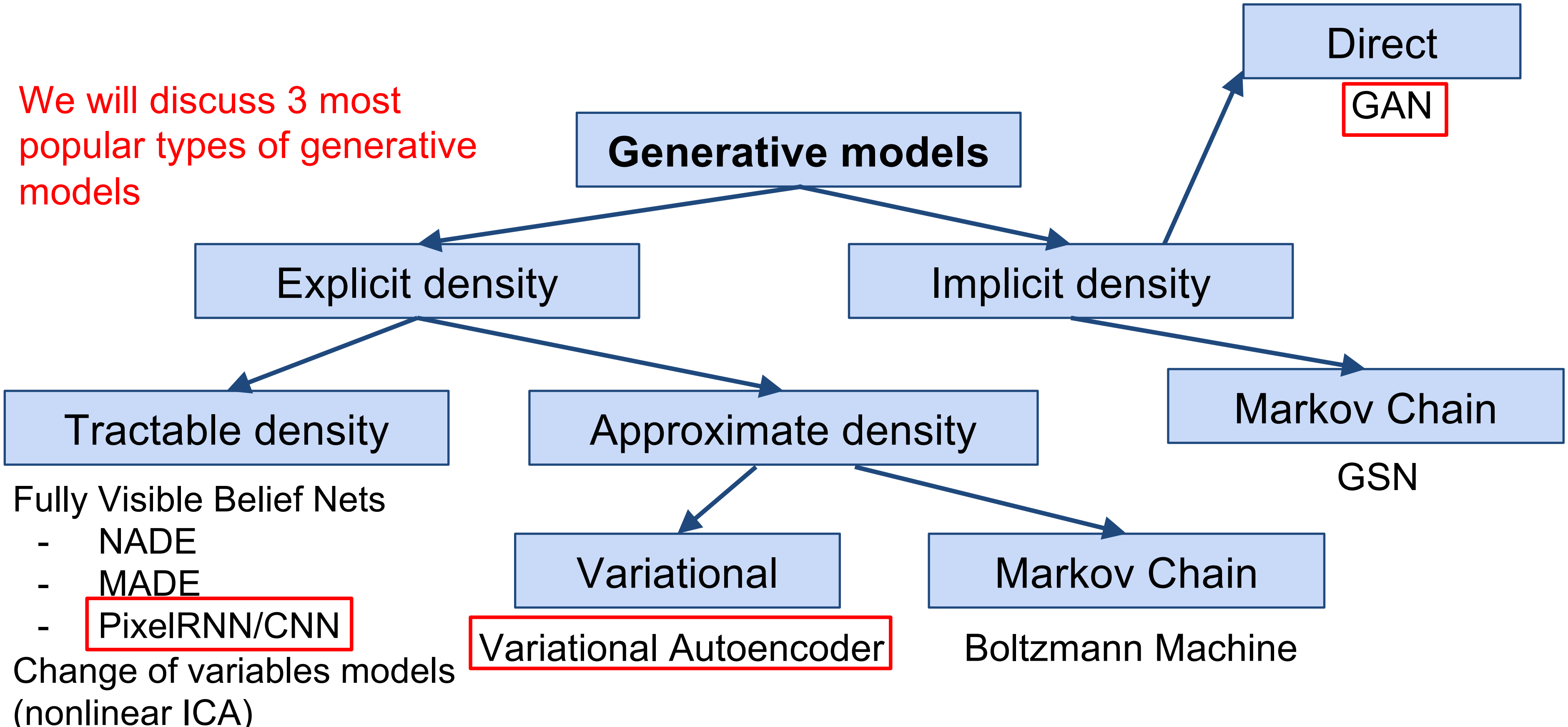


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

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

$D = \{\vec{x}_1, \dots, \vec{x}_N\}$

$x_1 \dots x_{i-1} \rightarrow \text{NN}_\theta \rightarrow x_i$

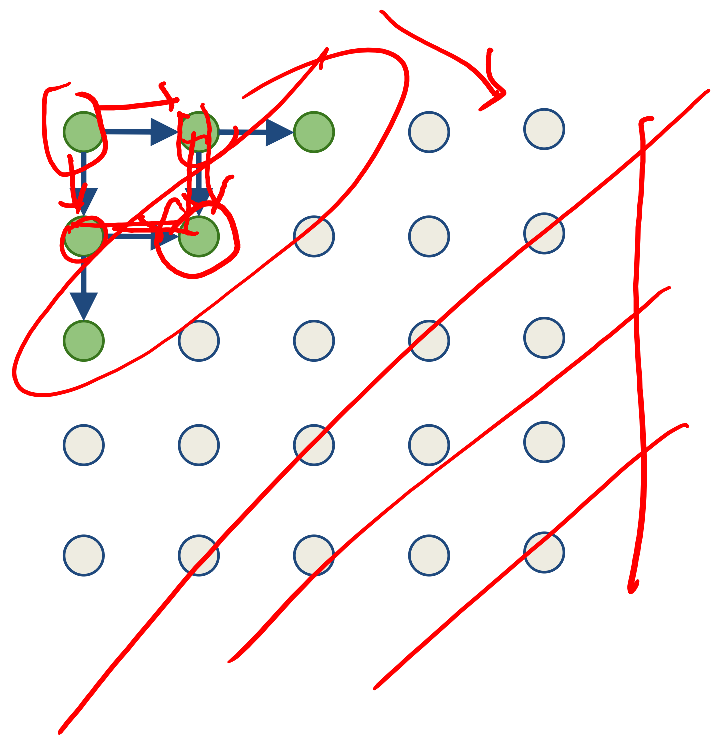
$\max_\theta \log p_\theta(\vec{x})$

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

$p(x_1, \ldots, x_{H \times W})$

Generate image pixels starting from corner

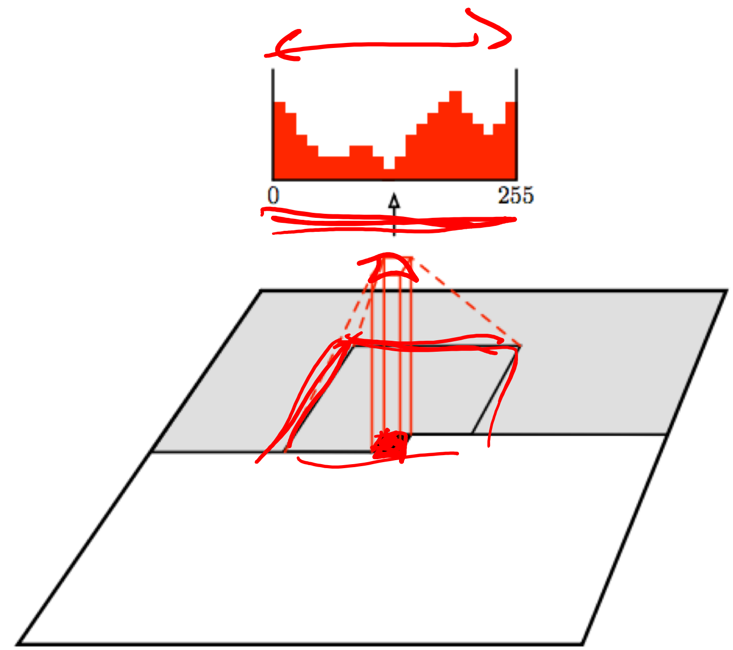
Dependency on previous pixels modeled using an RNN (LSTM)



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



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:

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

$p(\vec{x})$

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

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

$$\sum p(x|z) p(z)$$

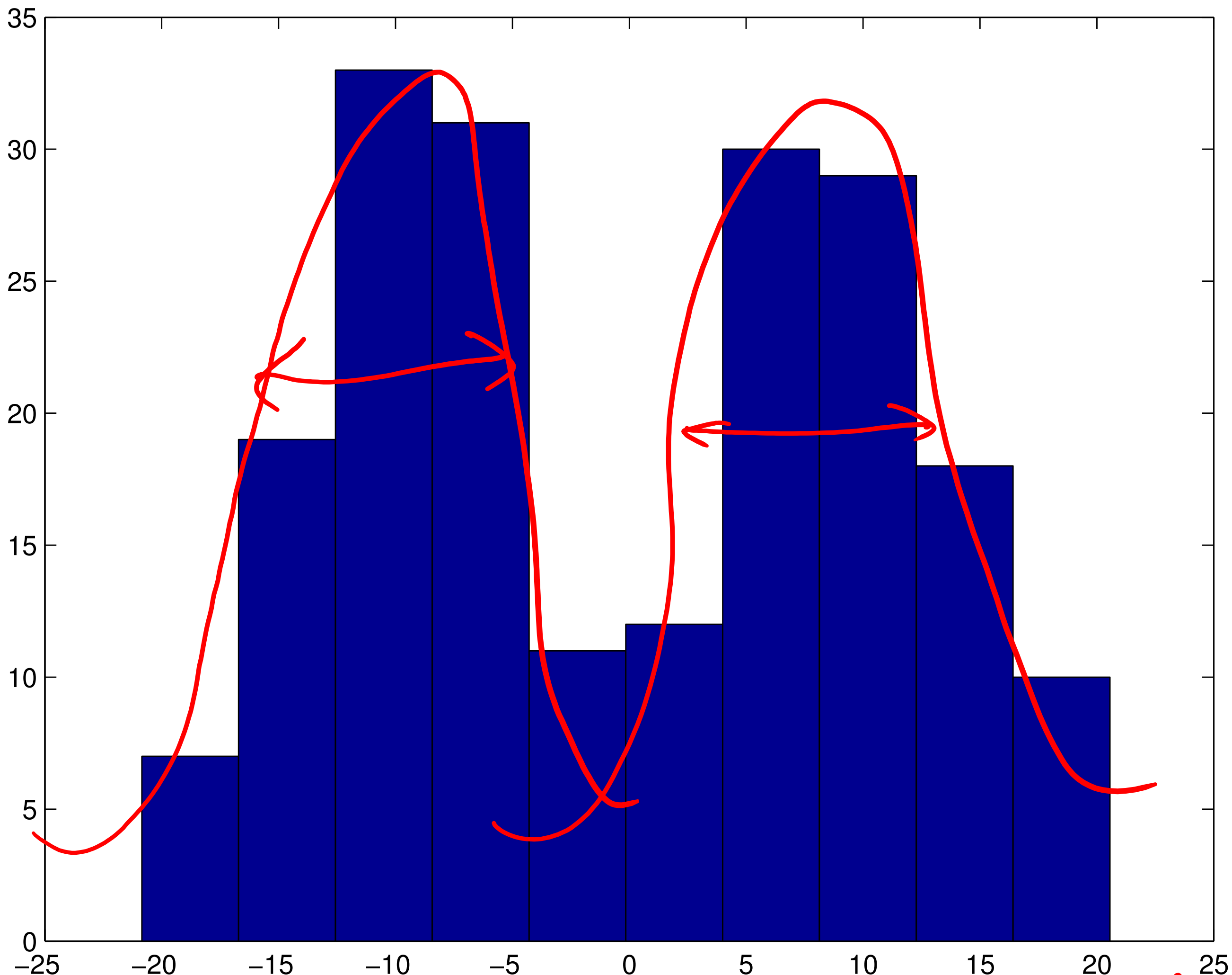
$$p(\vec{x}, \boxed{z}) = p(\vec{x} | z)\, p(z)$$

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

GMM

$p(x)$

$p(x)$



$p(x|z)$

$z \in \{1, 2\}$

$p(z) = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$

$p(x|z=1) \sim \mathcal{N}(\mu_1, \sigma_1^2)$

$p(x|z=2) \sim \mathcal{N}(\mu_2, \sigma_2^2)$

Gaussian Mixture Model

$$z \sim \text{Cat}(\boxed{\vec{\pi}}) \quad \begin{bmatrix} \pi_1 \\ \vdots \\ \pi_K \end{bmatrix} \quad \left\{ \begin{array}{l} \pi_c \geq 0 \\ \sum_{c=1}^{K} \pi_c = 1 \end{array} \right] \quad \pi_c = P(z=c)$$

$$X \mid z=c \sim N(\boxed{\mu_c}, \boxed{\sigma_c^2}) = \frac{1}{\sqrt{2\pi\sigma_c^2}} e^{-\frac{(x-\mu_c)^2}{2\sigma_c^2}}$$

$$\underbrace{p(z) \quad p(x \mid z)}_{\pi_c \cdot N(\mu_c, \sigma_c^2)} = p(x, z)$$

$p(z)$

Gaussian Mixture Model

$\underline{p(x,z)}$

$$\underline{p(x)} = \sum_{c} \underbrace{p(z)}\,\underbrace{p(x|z)} \quad \leftarrow \underline{\text{Marginization}}$$

$$p(\underline{z}|\underline{x}) = \frac{p(z,x)}{p(x)} = \frac{\overbrace{p(x|z)}\,\overbrace{p(z)}}{\boxed{\sum_{z}}(\;)(\;)}$$

"Inference"

Posterior

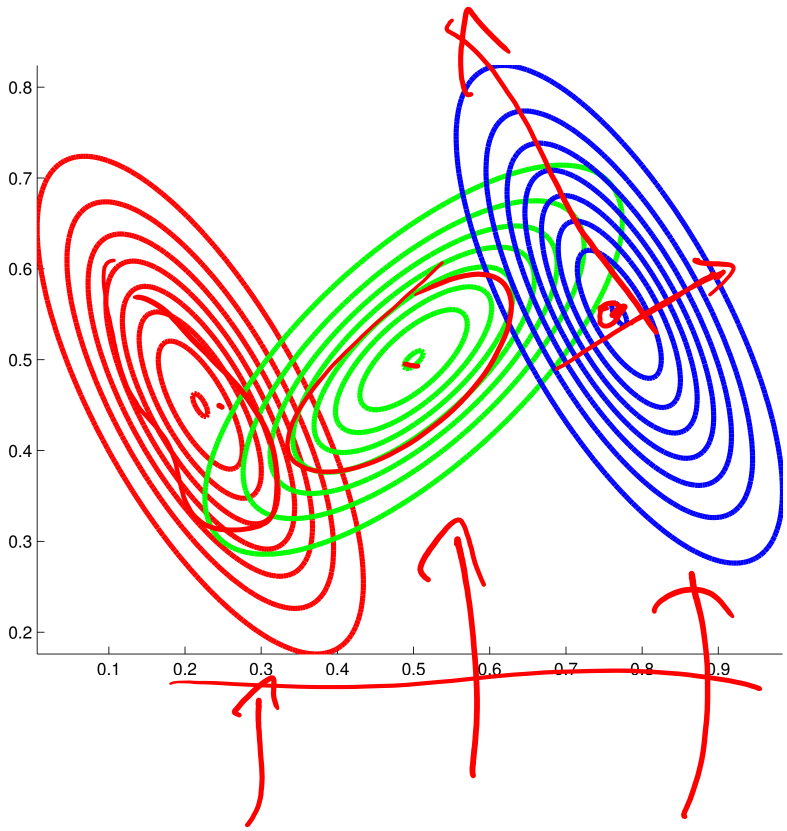
$$\boxed{\sim p(\underline{x}|\underline{z})}$$

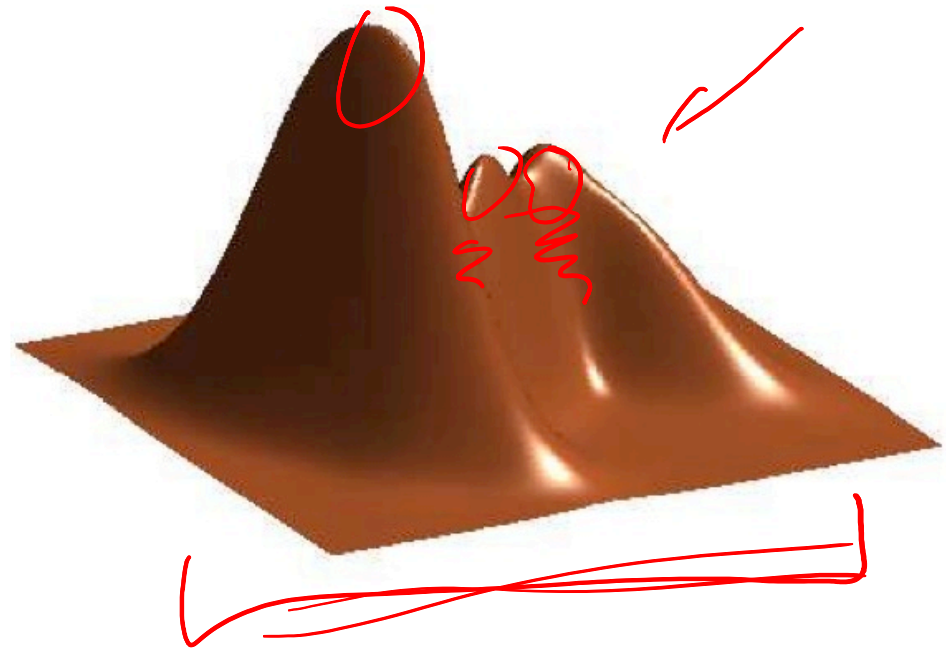
$\Rightarrow p($

GMM

$\vec{\mu} \in \mathbb{R}^d$
$\Sigma \in \mathbb{R}^{d \times d}$

$$\mathcal{N}(\vec{\mu}, \Sigma) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu})^T \Sigma^{-1} (\vec{x}-\mu)}$$





K-means vs GMM

- K-Means
  - http://stanford.edu/class/ee103/visualizations/kmeans/kmeans.html
- GMM
  - https://lukapopijac.github.io/gaussian-mixture-model/

Hidden Data Causes Problems #1

- Fully Observed (Log) Likelihood factorizes
- Marginal (Log) Likelihood doesn't factorize
- All parameters coupled!

$$\{\pi_1, \ldots, \pi_K, \vec{\mu}_1, \ldots, \vec{\mu}_K, \Sigma_1, \ldots, \Sigma_K\} \equiv \vec{\theta}$$

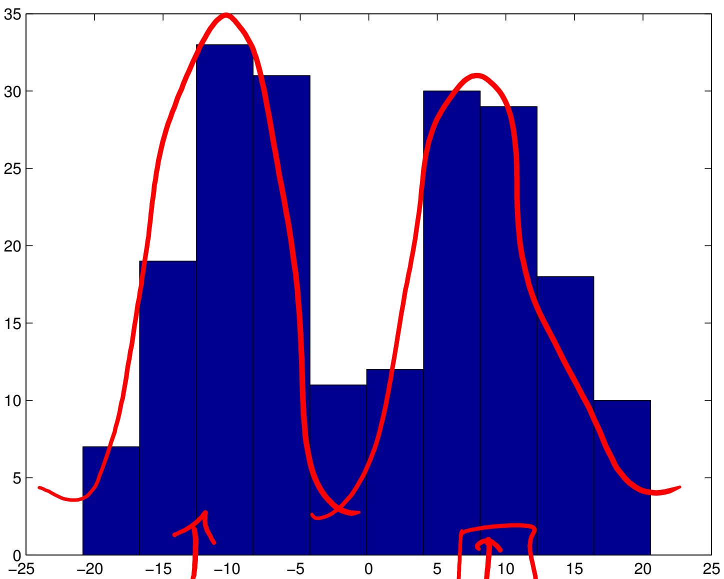
$$\{\vec{x}_1, \ldots, \vec{x}_N\}$$

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\, \cancel{P(x}\; P(D \mid \theta) = \operatorname{argmax} \sum_i \log P(\vec{x}_i \mid \theta)$$

$$\underbrace{\sum_i}_{\text{data}} \underbrace{\log}\; \underbrace{\sum_z} \underbrace{P(\vec{x}_i, z \mid \theta)}_{P(x \mid z) P(z)}$$

Hidden Data Causes Problems #2

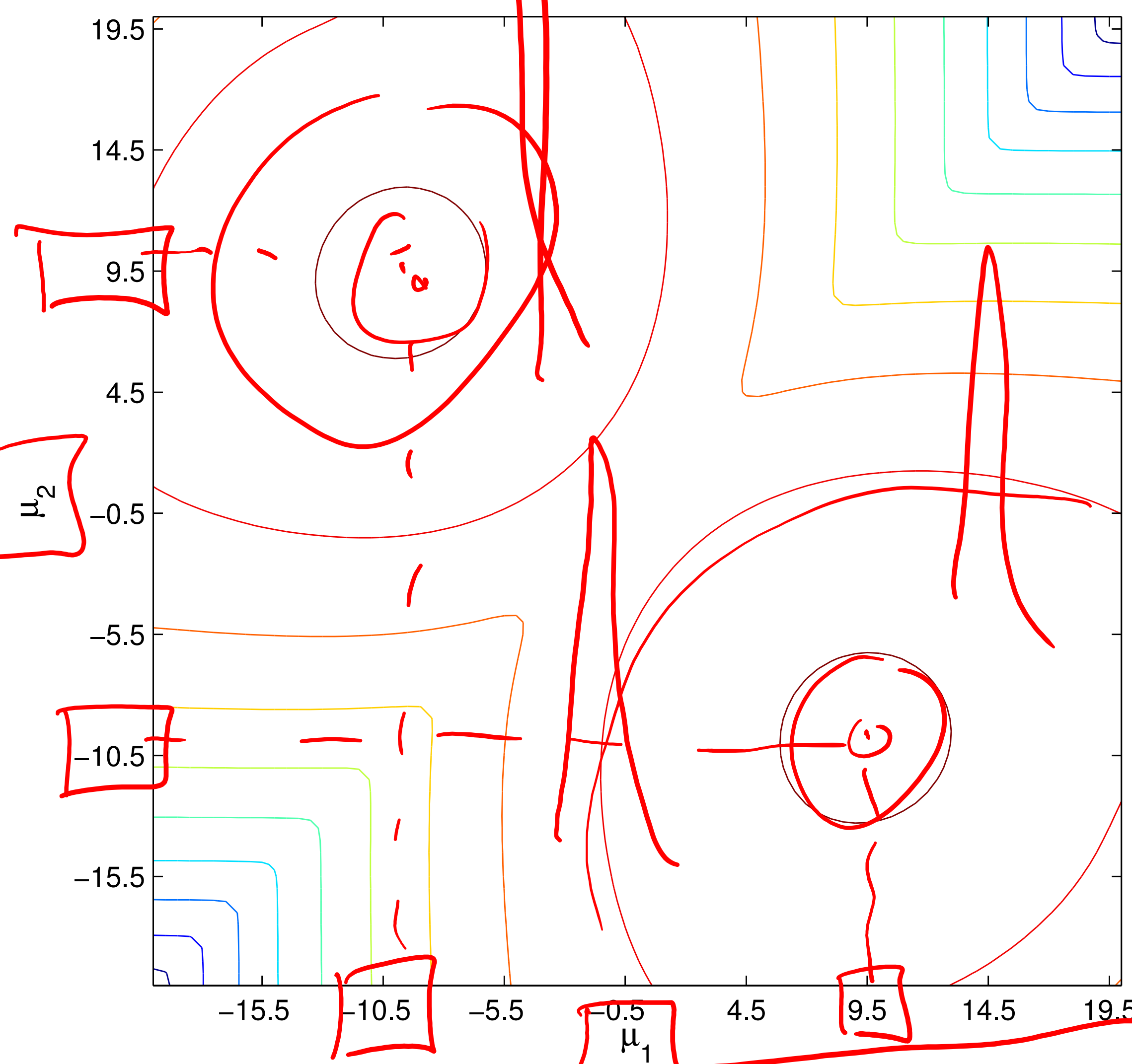
- Identifiability



−10 +10

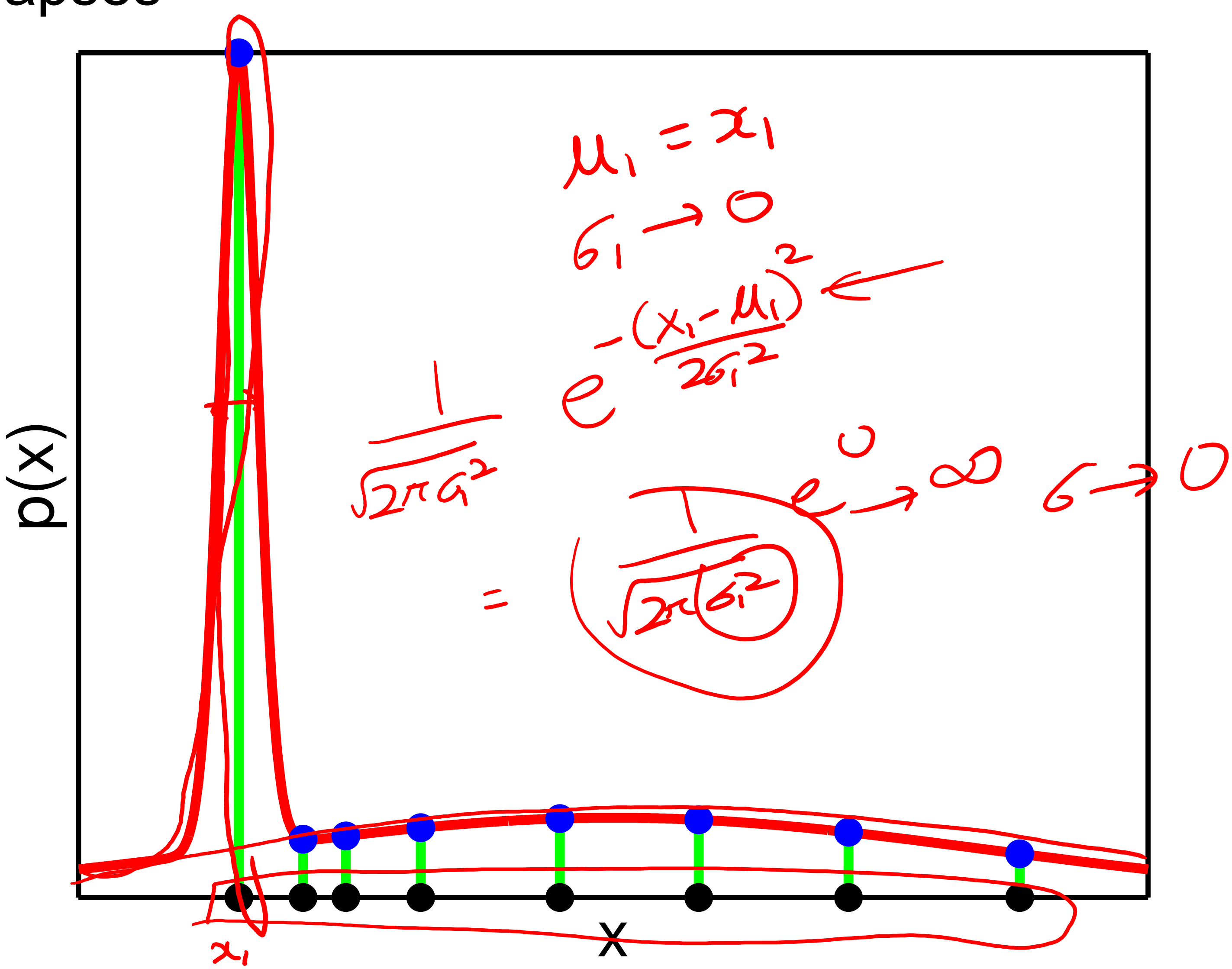
μ_2 μ_1

μ_1 μ_2



Hidden Data Causes Problems #3

- Likelihood has singularities if one Gaussian “collapses”



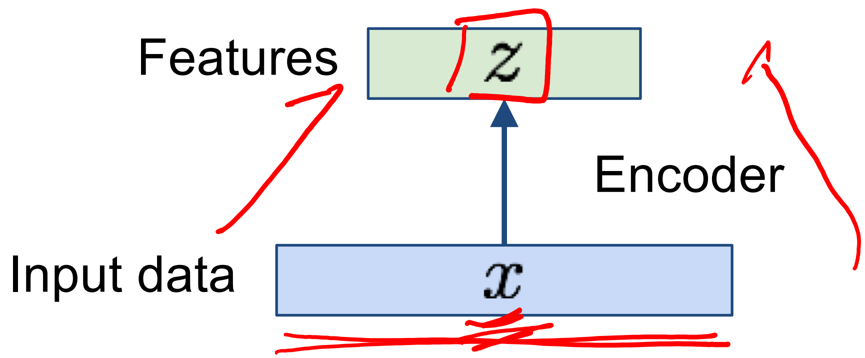
Variational Auto Encoders

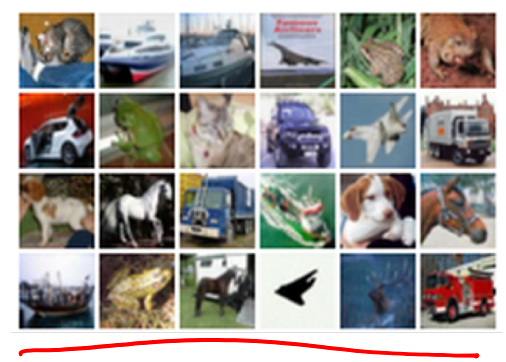
VAEs are a combination of the following ideas:

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

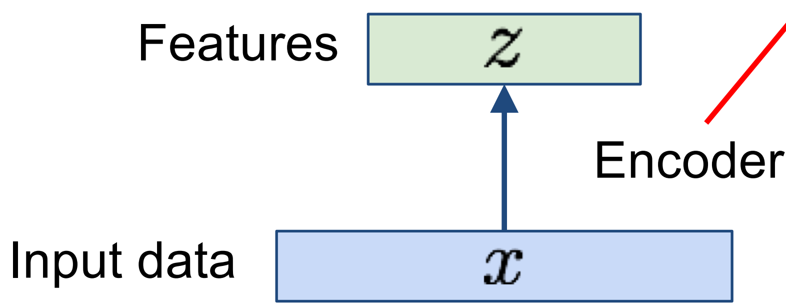


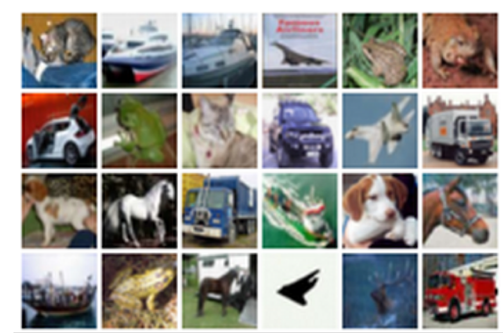


Autoencoders

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

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





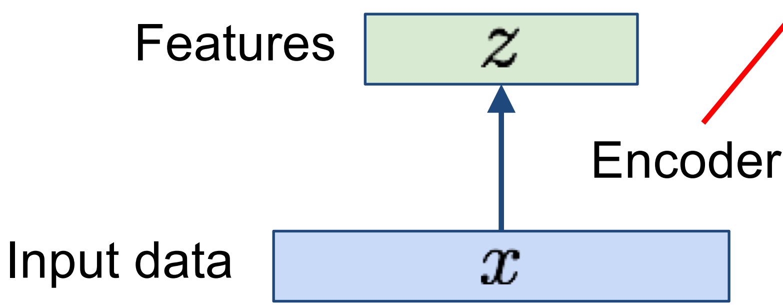
Autoencoders

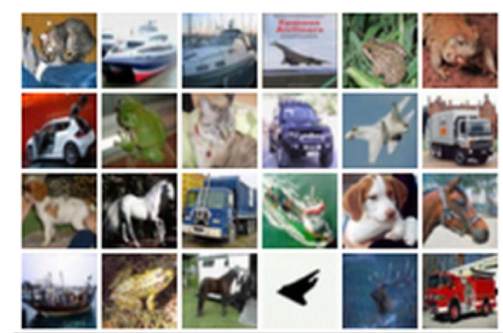
Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

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

Q: Why dimensionality reduction?

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





Autoencoders

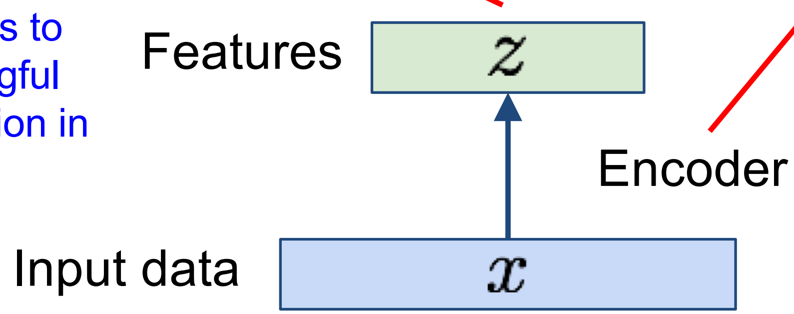
Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

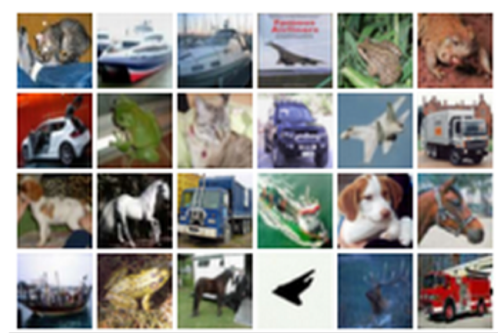
z usually smaller than **x**
(dimensionality reduction)

Q: Why dimensionality reduction?

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

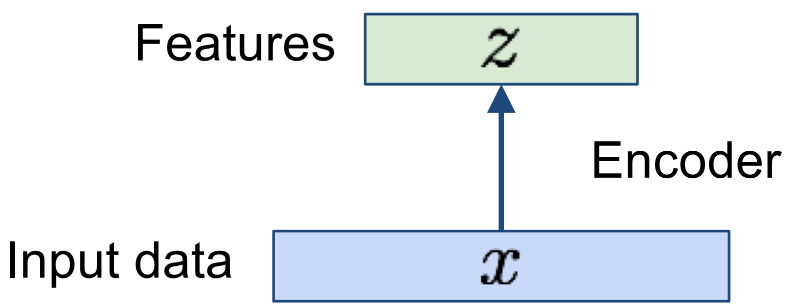
Originally: Linear + nonlinearity (sigmoid)
Later: Deep, fully-connected
Later: ReLU CNN

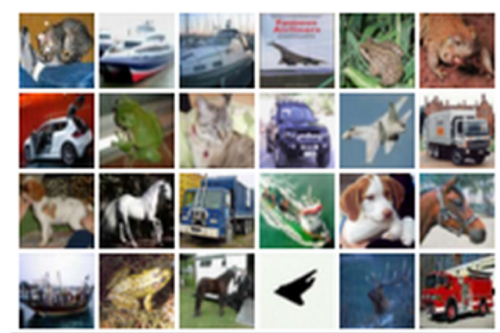




Autoencoders

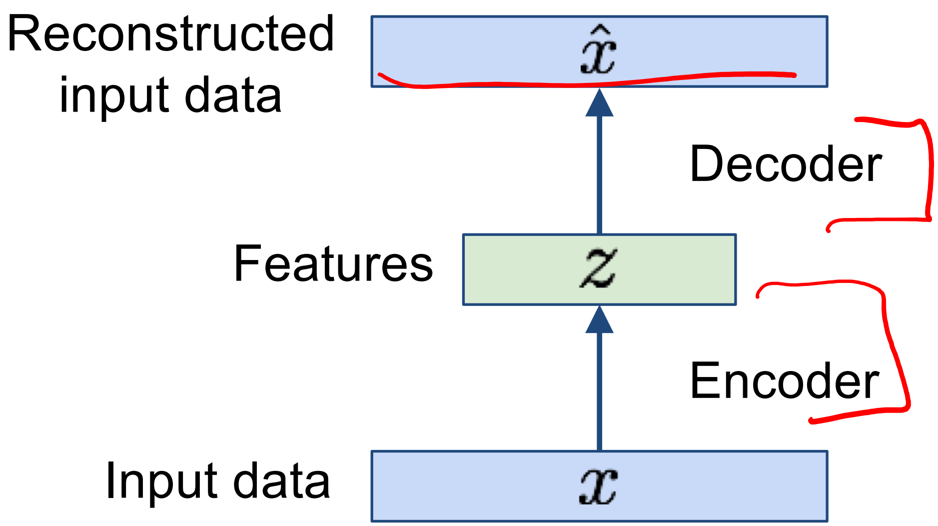
How to learn this feature representation?

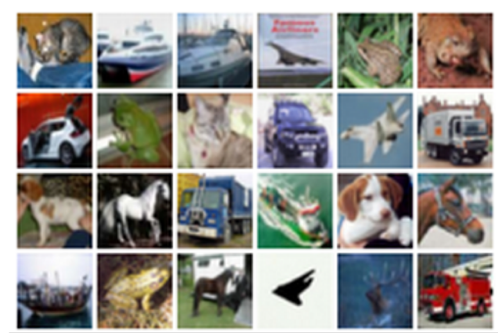




Autoencoders

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



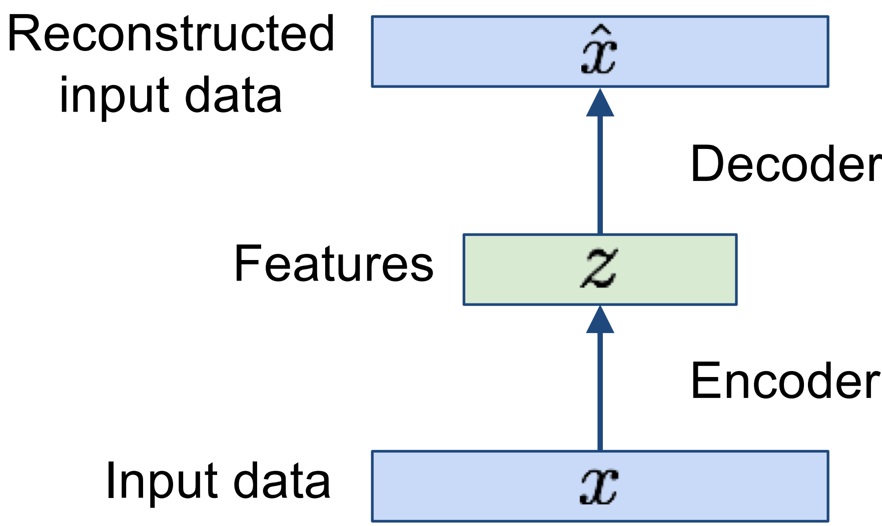


Autoencoders

How to learn this feature representation?

Train such that features can be used to reconstruct original data

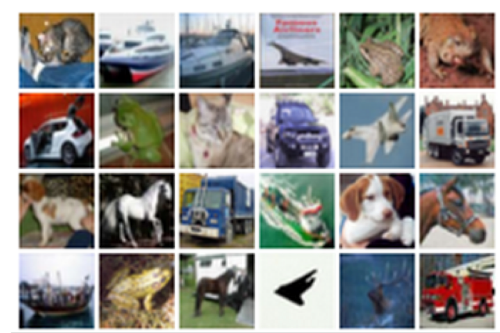
“Autoencoding” - encoding itself



Originally: Linear + nonlinearity (sigmoid)

Later: Deep, fully-connected

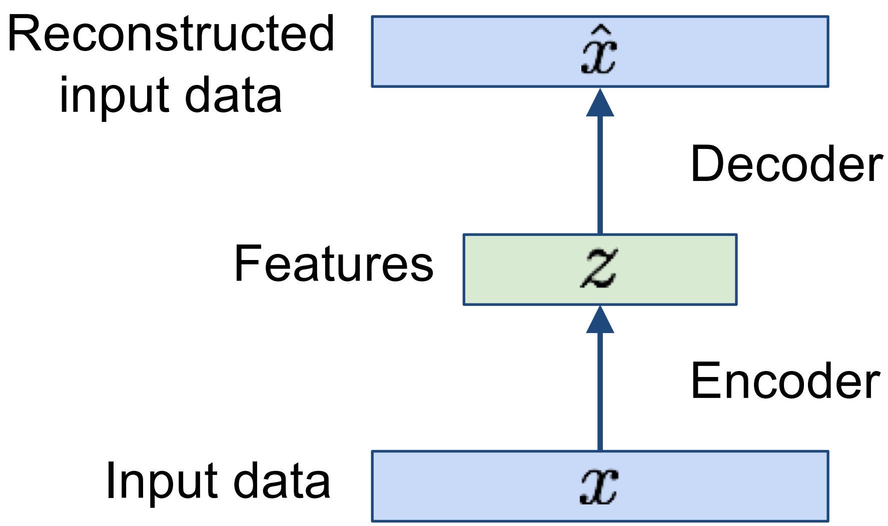
Later: ReLU CNN (upconv)

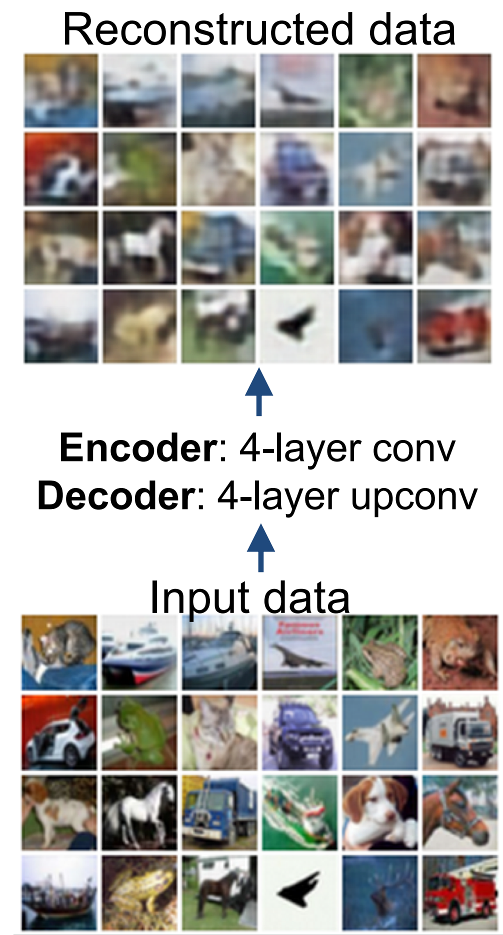


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Autoencoders

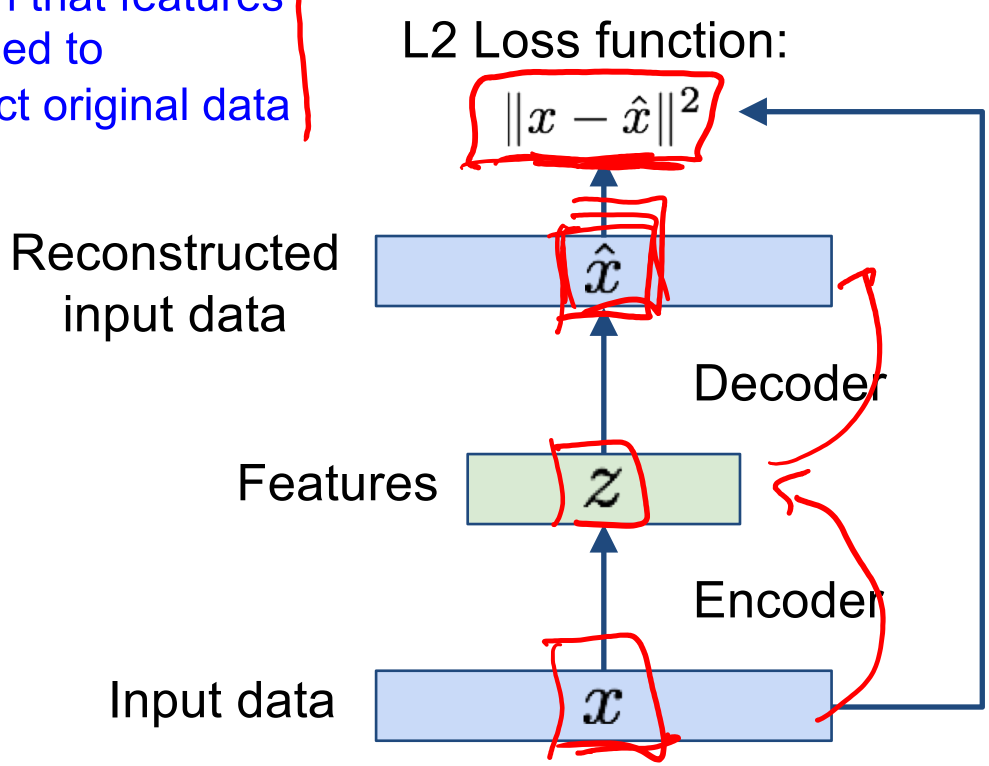
How to learn this feature representation?
Train such that features can be used to reconstruct original data
“Autoencoding” - encoding itself

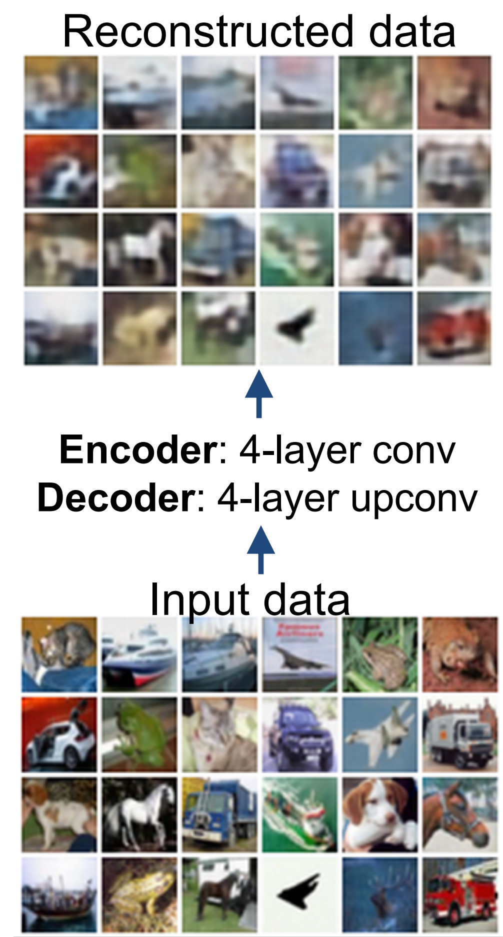




Autoencoders

Train such that features can be used to reconstruct original data

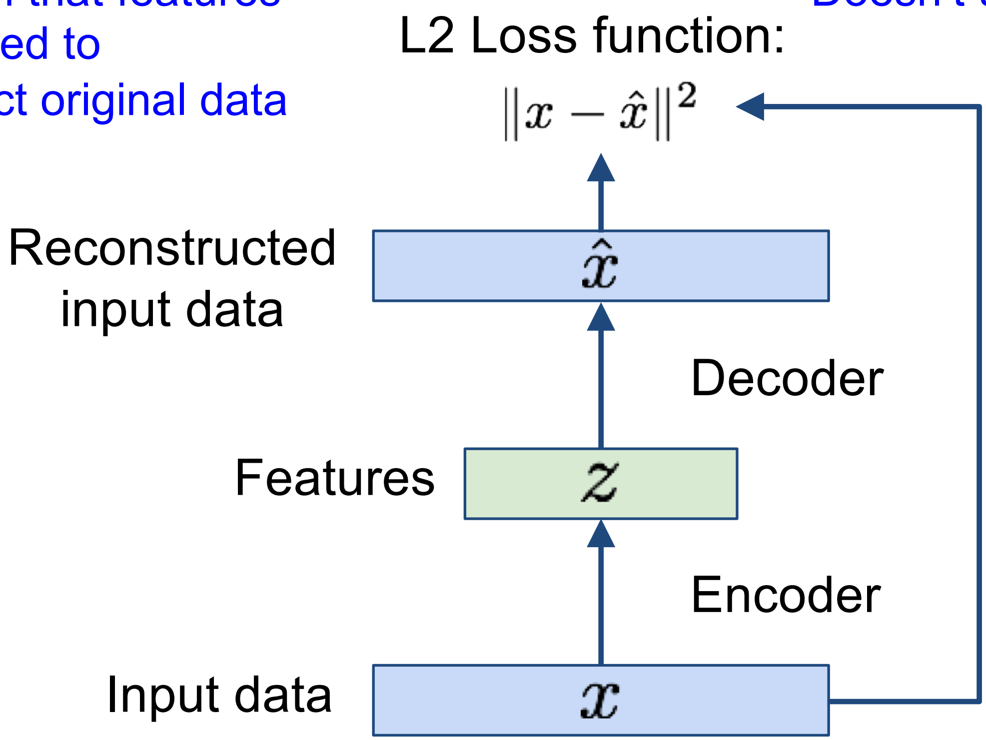


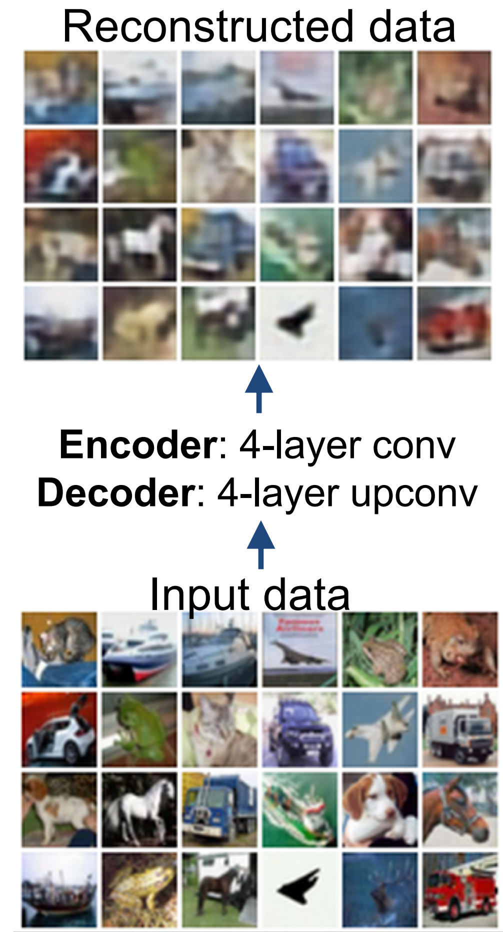


Autoencoders

Train such that features can be used to reconstruct original data

Doesn't use labels!

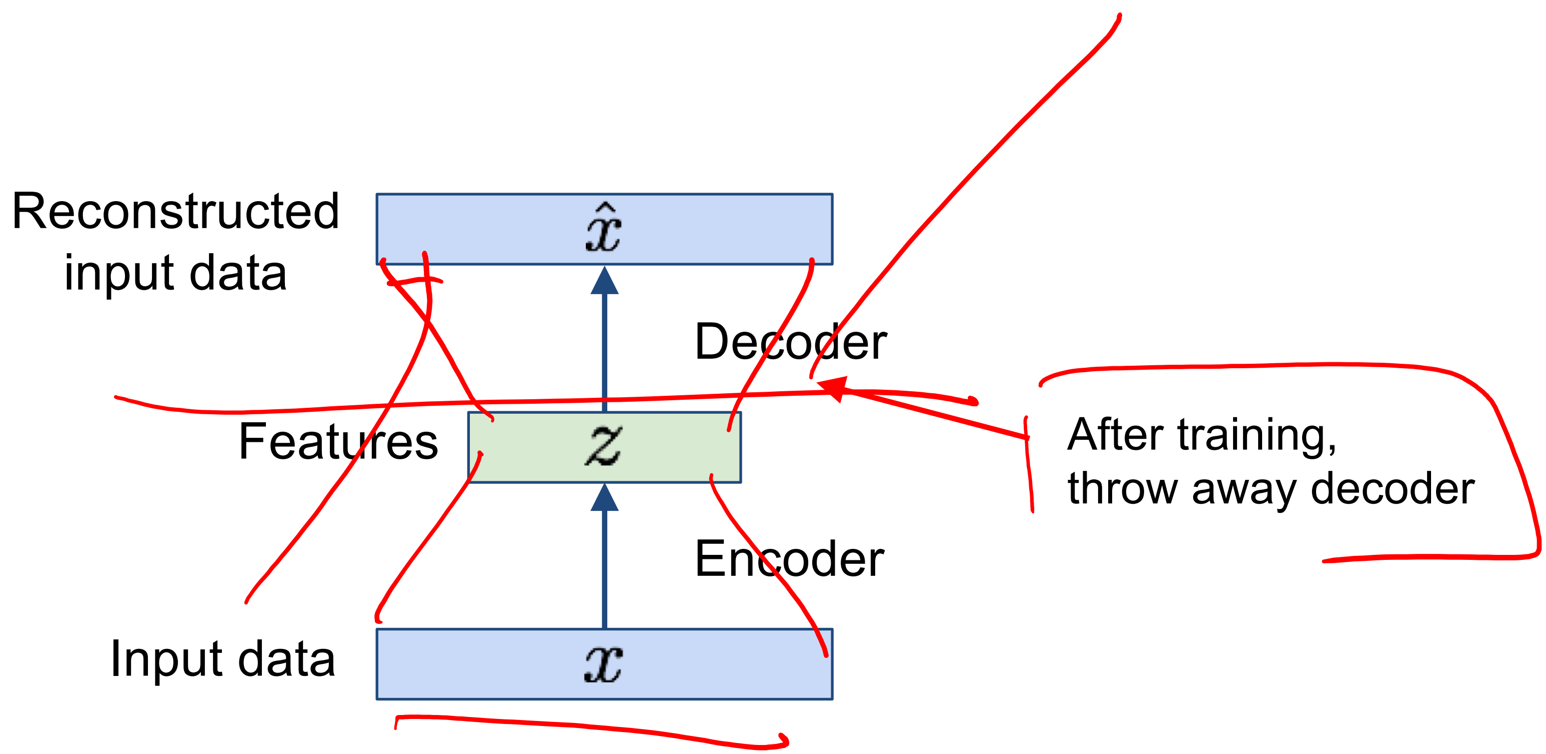




Autoencoders

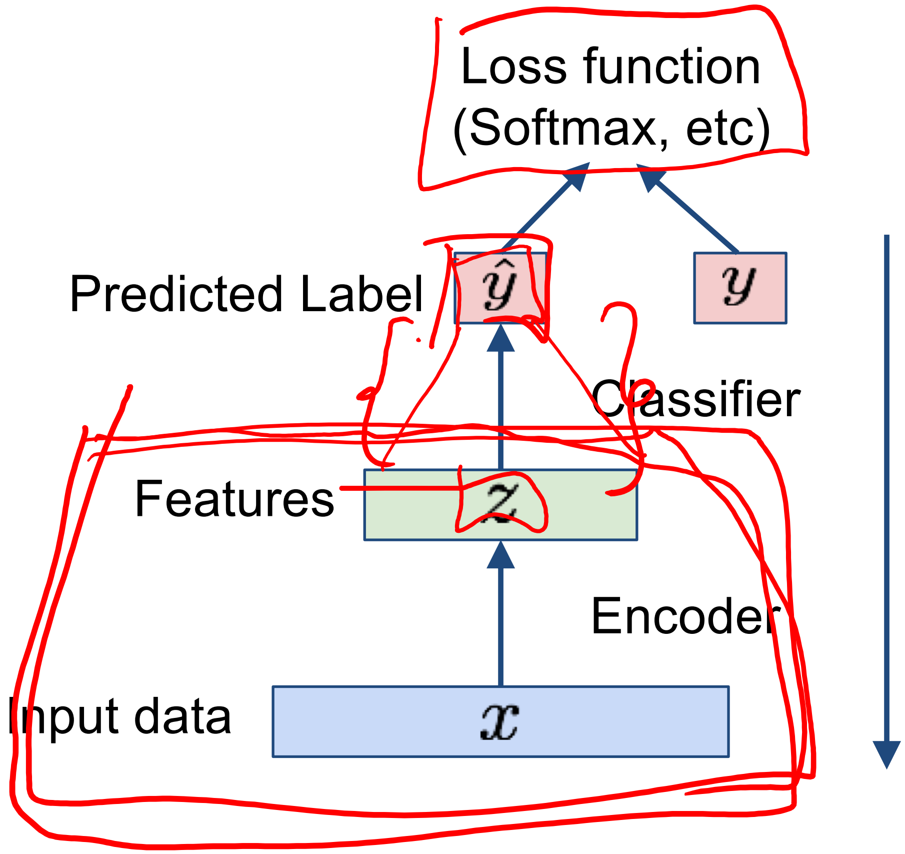
- Demo
  - https://cs.stanford.edu/people/karpathy/convnetjs/demo/autoencoder.html

Autoencoders

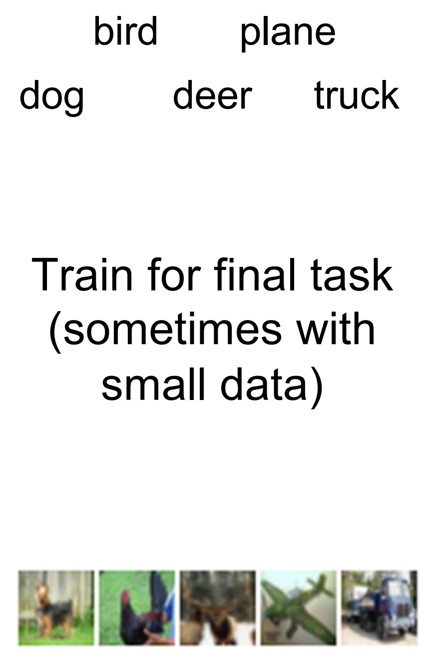


Autoencoders

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



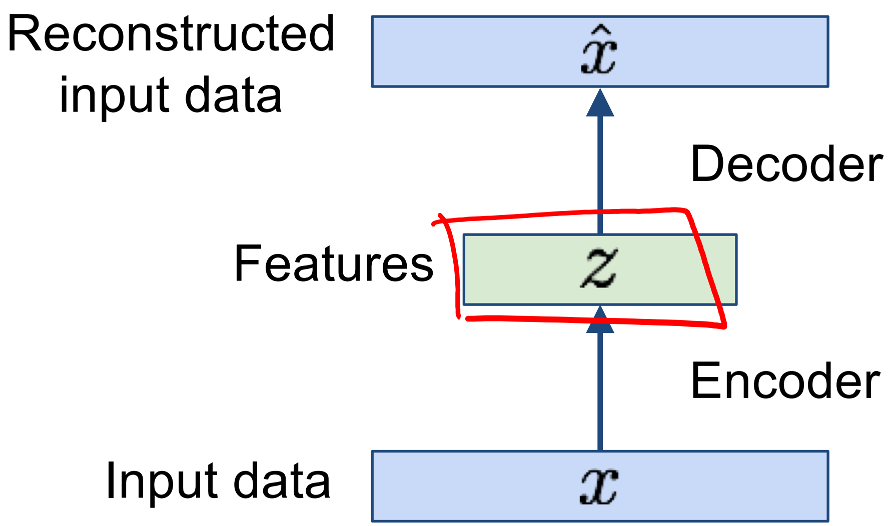
Train for final task (sometimes with small data)



Autoencoders

$z = f_\phi(x)$ $\quad p(z)$

$\hat{x} = g_\phi(z)$ $\quad p(\hat{x} \mid z)$



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!

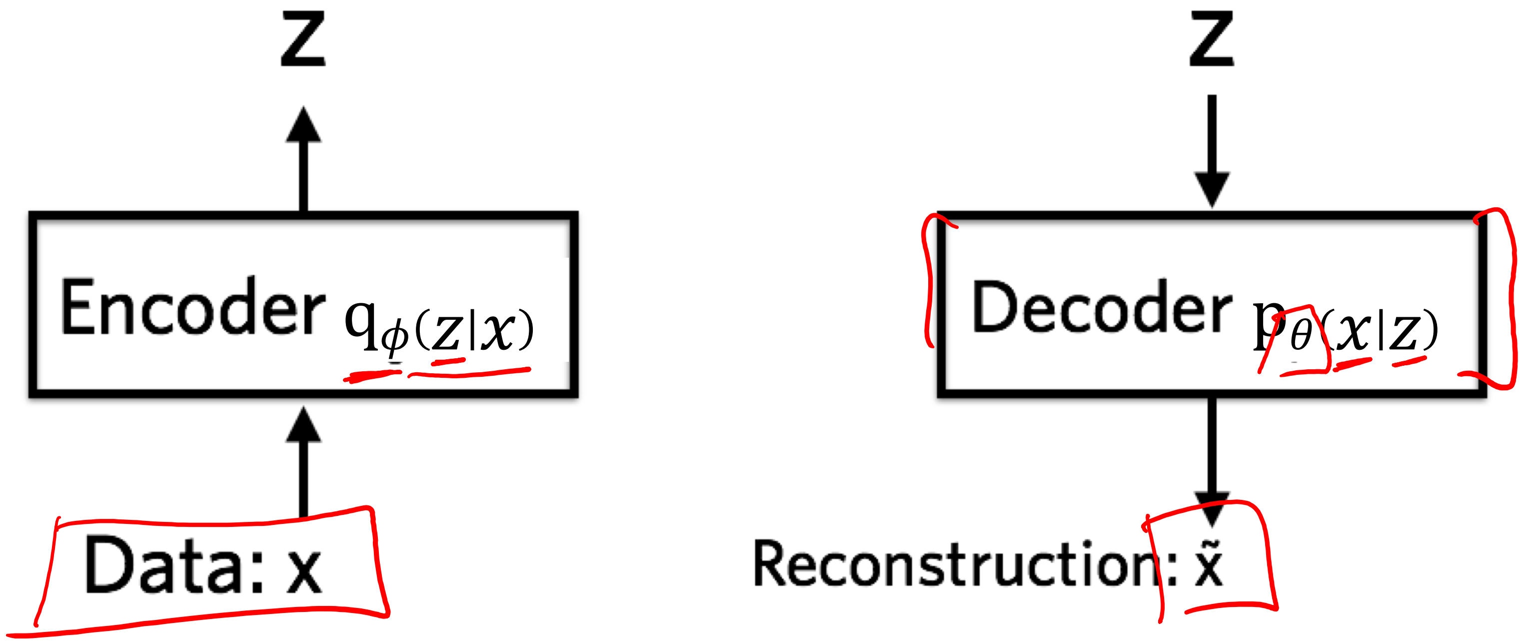


Image Credit: https://jaan.io/what-is-variational-autoencoder-vae-tutorial/

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

Key problem

- P(z|x)

$$= \frac{P(z,x)}{P(x)} = \frac{P(x|z)\,P(z)}{\underbrace{\boxed{\sum_z} P(x|z)P(z)}}$$

$\downarrow$

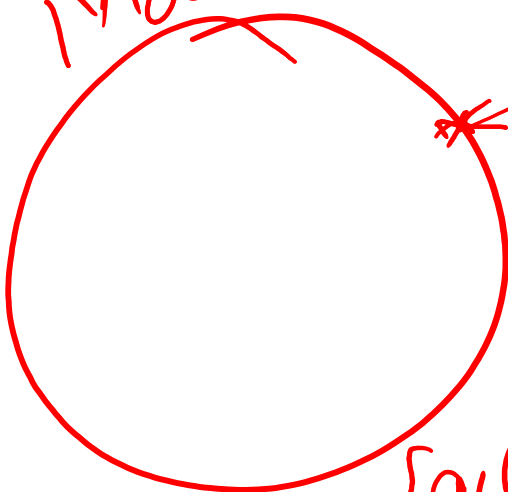
$q(z)$

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

Model



$p(z)$

$\{q_\phi(z)\}$

$\min_{q \in \{q_\phi\}}$

$KL(p(z) \,||\, q(z))$

$KL(q(z) \,||\, p(z))$

$p(z) = 0$

error

$$KL(p(z) \,||\, q(z)) = \sum_z p(z) \log \frac{p(z)}{q(z)}$$

$$-\sum_{\text{sample}} \log q(z)$$

KL divergence: Distance between distributions

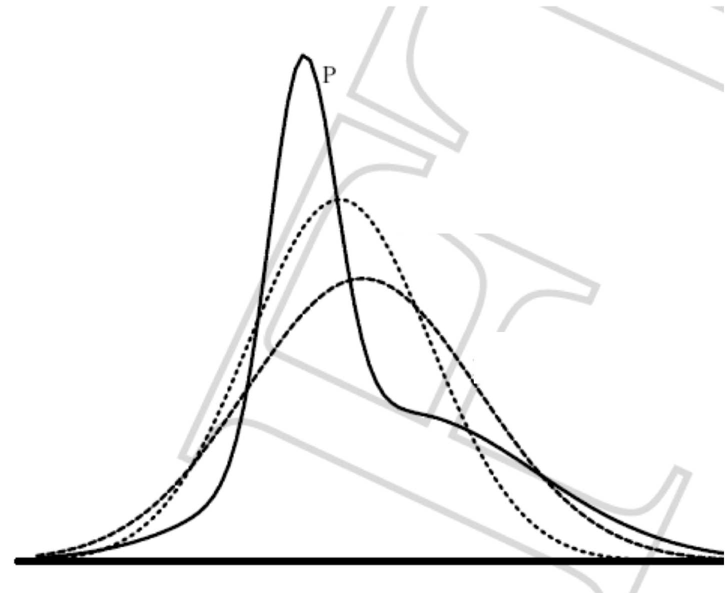
- Given two distributions *p* and *q* KL divergence:

- D(p||q) = 0 iff p=q

- Not symmetric – p determines where difference is important

Find simple approximate distribution

- Suppose $\boxed{p}$ is intractable ~~posterior~~
- Want to find simple q that approximates p
- KL divergence not symmetric

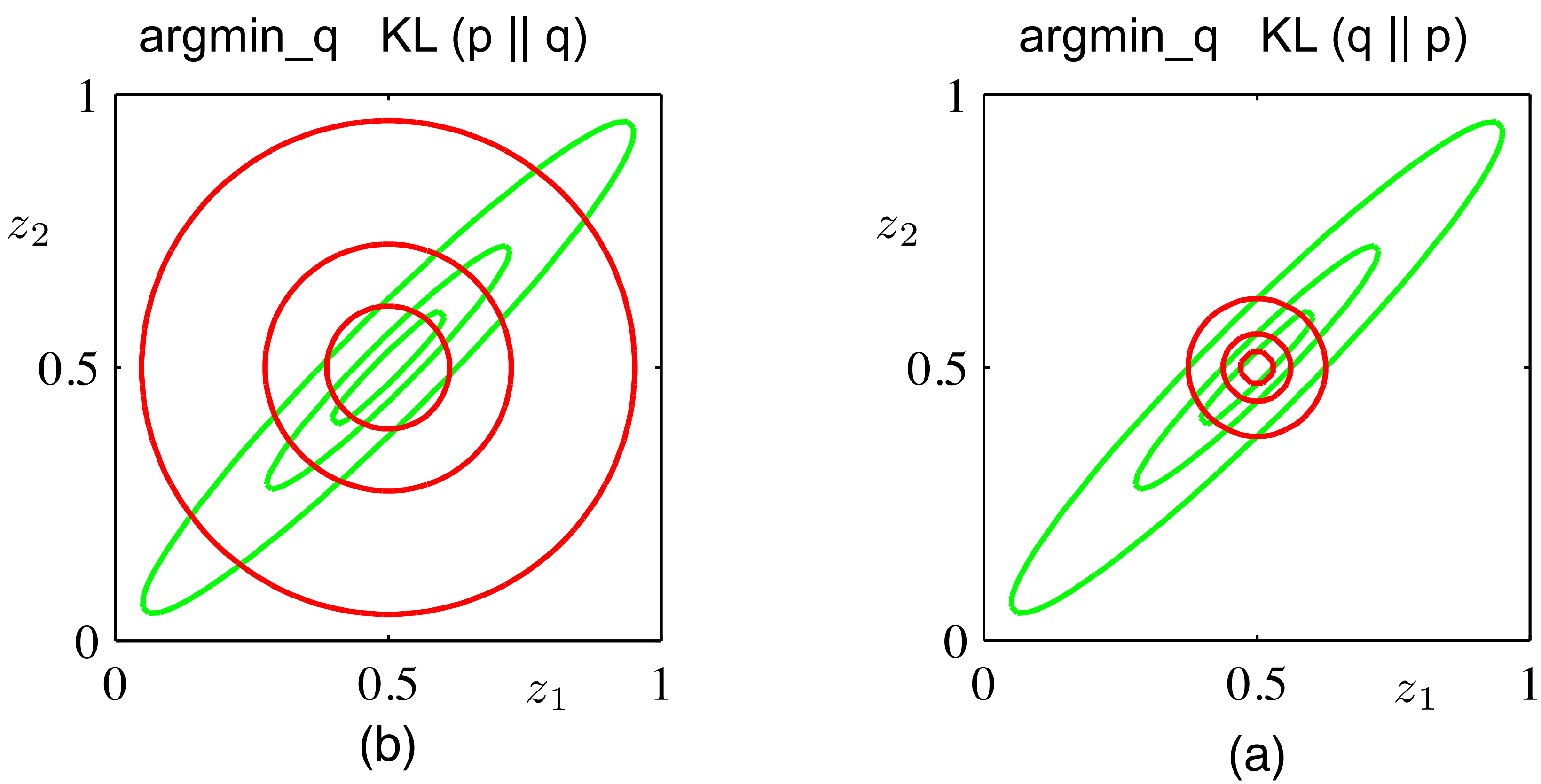


- D(p||q)
  - true distribution p defines support of diff.
  - the "correct" direction
  - will be intractable to compute

- 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



p = Green; q = Red

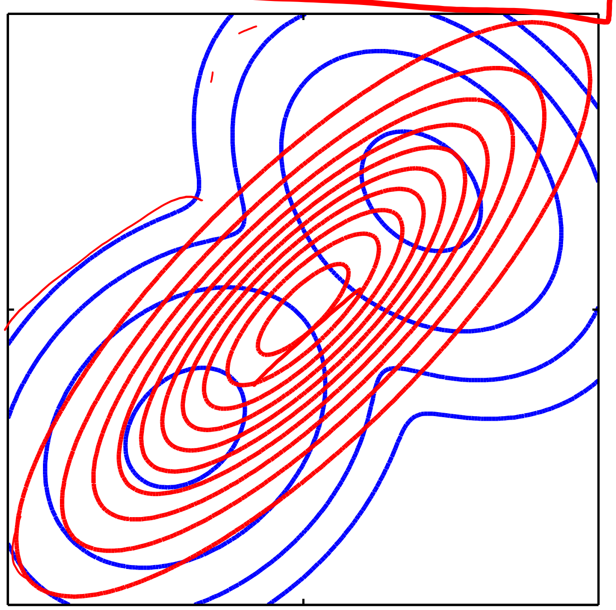
Example 2

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

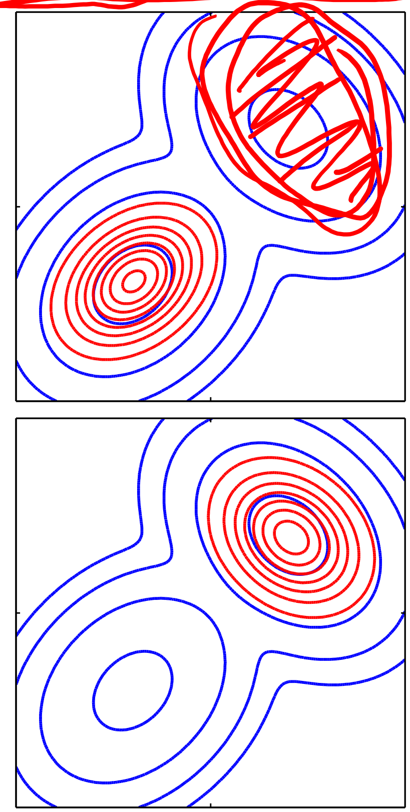
$\sum q(z) \log \frac{q(z)}{p(z)}$

0

argmin_q KL (p || q)



argmin_q KL (q || p)



p = Blue; q = Red