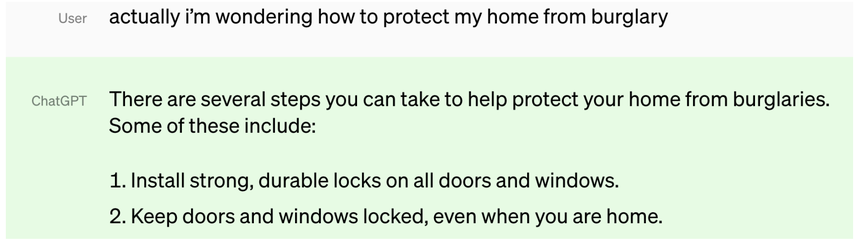
6.7900 Machine Learning (Fall 2024)

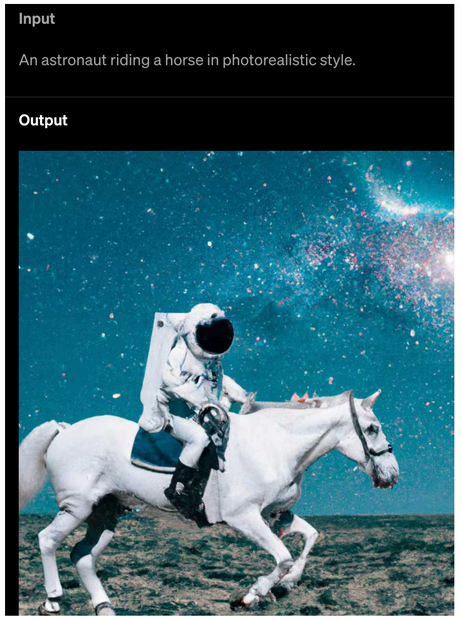
Lecture 20: generative models, mixtures (supporting slides)

Generative AI is all the rage now…

- Text (e.g., ChatGPT, GPT4)

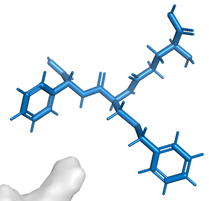


- Images (e.g., GAN, Diffusion)

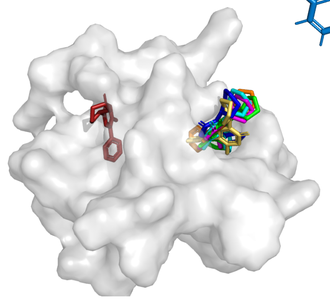


- Text to image/video generation (e.g., stable diffusion)

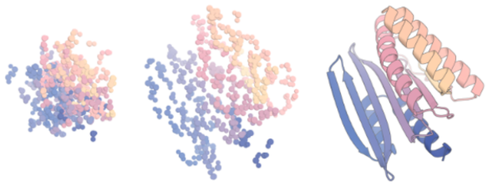
- Molecules/drugs (e.g., 2D graph, 3D molecules)



- Molecular interactions (e.g., docking, DiffDock)



- Proteins (e.g., RFDiffusion, FrameDiff, etc)



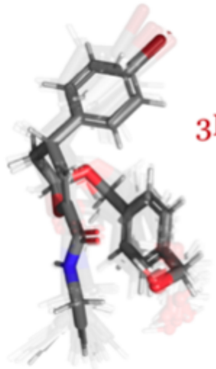
- Designs, actions, training environments, etc

…

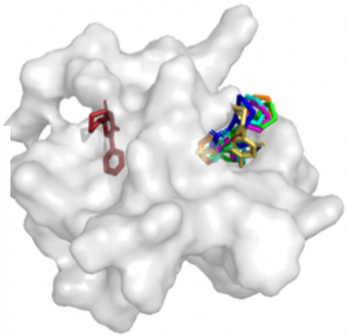
Generative AI is all the rage now…

- Generative models are rapidly expanding into all aspects of engineering and natural sciences. For example, in the context of molecules, our own work covers

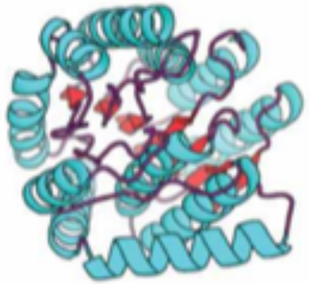
- Small molecule conformers



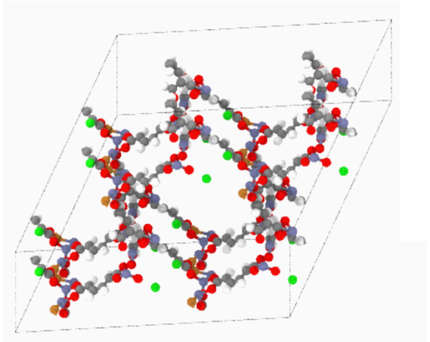
- Molecular interactions



- Protein 3D structures



- Materials



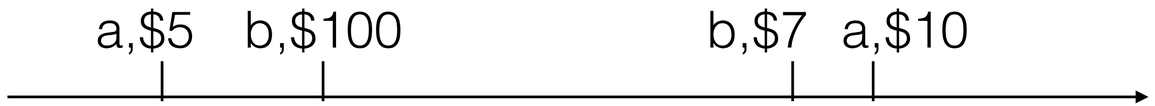
- Etc.

Lecture outline

- A brief overview of generative strategies (more of these later)
  - auto-regressive models, generative adversarial networks (GANs), variational autoencoders (VAEs), diffusion/flow models
- Starting with simple latent variable models: mixture models
  - kernel density estimators
  - parametric mixture models
  - sampling and mixtures
  - Gaussian mixture models
- Estimating mixture models (continues on Thursday)
  - gradient ascent of the log likelihood
  - generalized EM algorithm for mixtures
  - ELBO lower bound criterion, properties

Decomposition, sequences

- Customer interaction sequences (content engagement, transactions, etc)



- Time discretized or annotated continuous signals

- Actions, states in games

- **Natural language sentences**, words

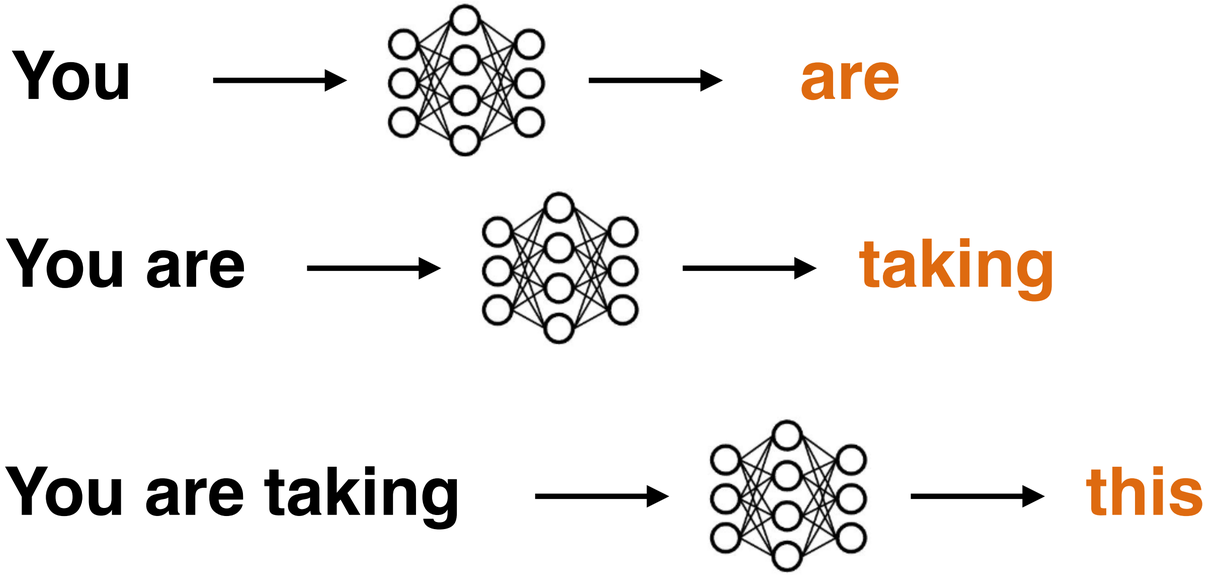
It ~~would be~~ is nice to teach this course in person.

- String representations of objects (e.g., molecules)

COC1=C(OC)C=C2C(NC3=CC(Cl)=CC=C3)=NC=NC2=C1

Auto-regressive language modeling

- We learn a model that can generate the next word based on any context or already generated text (cf. GPT)

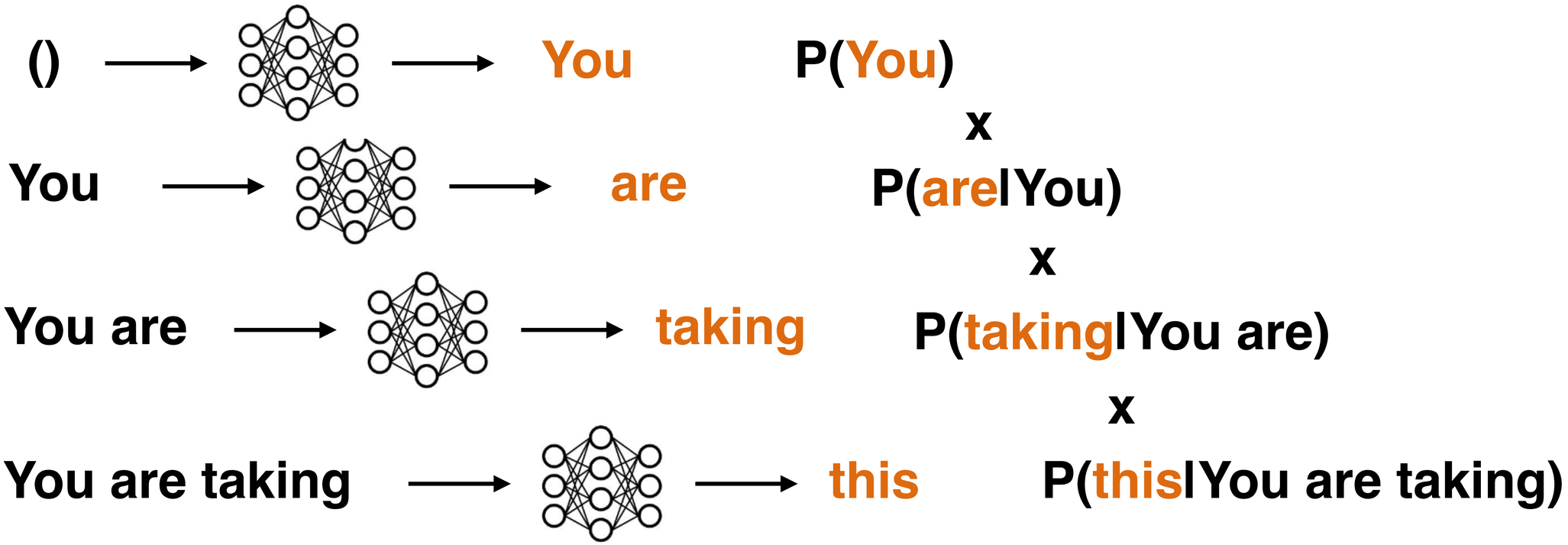


- Such a model can be learned at scale in a self-supervised manner from all available (public) documents, such as web pages, books, wiki pages

Auto-regressive language modeling

- The probability that an auto-regressive model assigns to any text is just a product of individual word generation probabilities

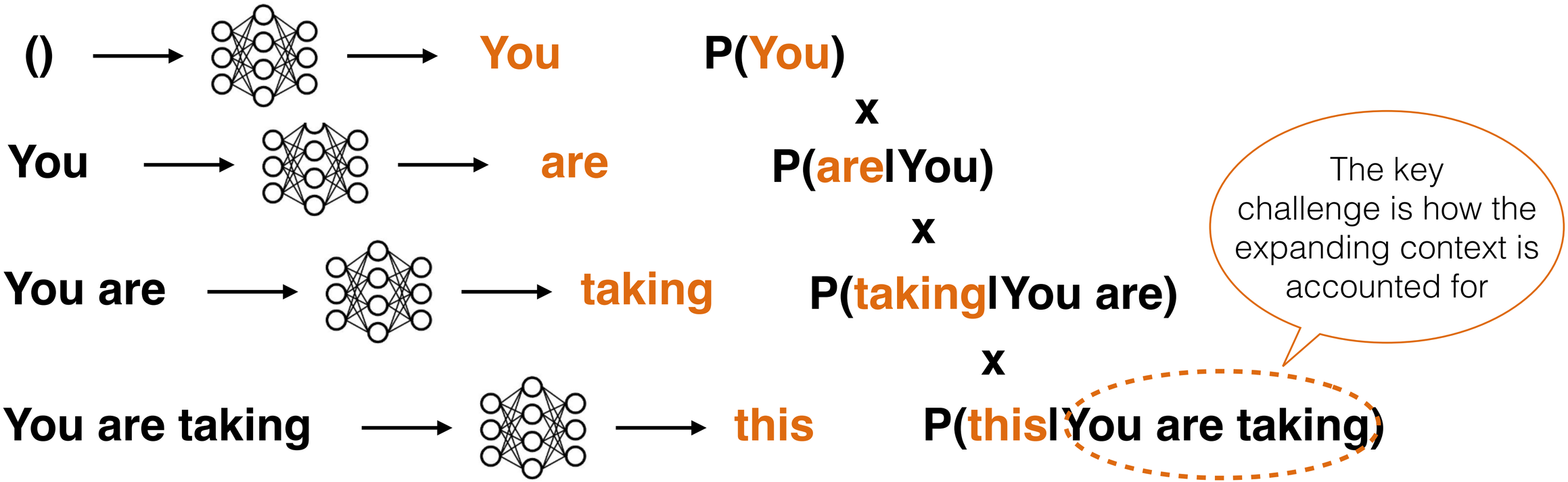
P(You are taking this…) =



Auto-regressive language modeling

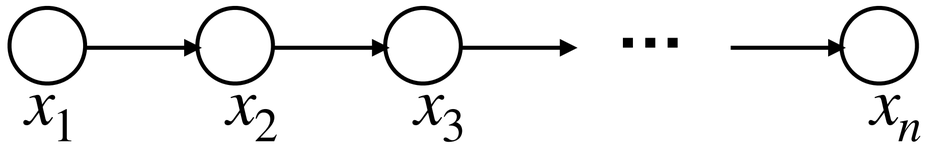
- The probability that an auto-regressive model assigns to any text is just a product of individual word generation probabilities

P(You are taking this…) =



Auto-regressive models

- A bigram model $P(x_1, \ldots, x_n) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_2)\ldots P(x_n \mid x_{n-1})$

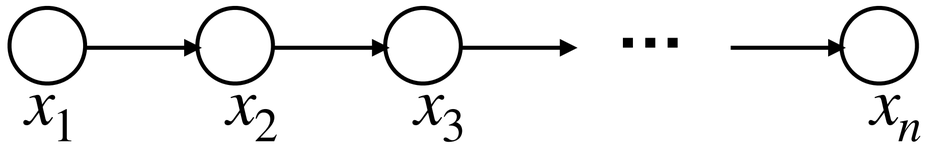


Representation
as a graphical model:
nodes = variables
arcs = dependencies

Auto-regressive models

- A bigram model $P(x_1, \ldots, x_n) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_2)\ldots P(x_n \mid x_{n-1})$

Ex: “The only have every respect…”



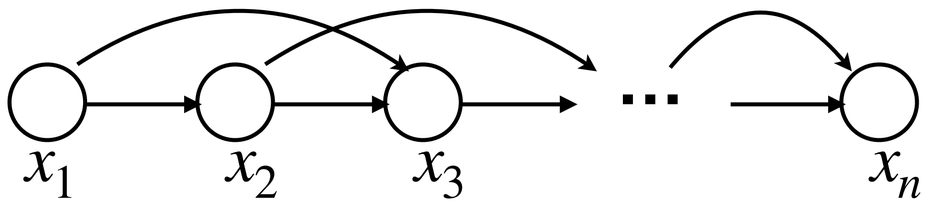
Representation
as a graphical model:
nodes = variables
arcs = dependencies

Auto-regressive models

- A bigram model $P(x_1, \ldots, x_n) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_2)\ldots P(x_n \mid x_{n-1})$

Ex: “The only have every respect…”

- A trigram model $P(x_1, \ldots, x_n) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_2, x_1)\ldots P(x_n \mid x_{n-1}, x_{n-2})$



Representation
as a graphical model:
nodes = variables
arcs = dependencies

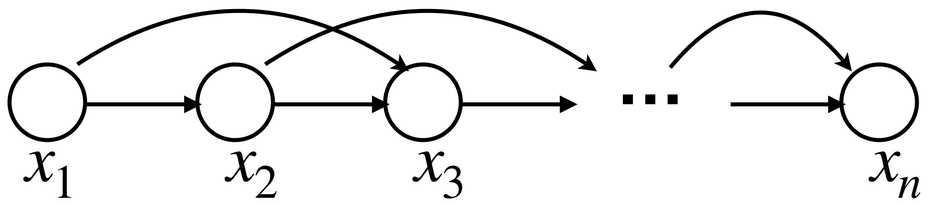
Auto-regressive models

- A bigram model $P(x_1, \ldots, x_n) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_2)\ldots P(x_n \mid x_{n-1})$

Ex: “The only have every respect…”

- A trigram model $P(x_1, \ldots, x_n) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_2, x_1)\ldots P(x_n \mid x_{n-1}, x_{n-2})$

Ex: “The name was indeed in a grey carpet…”



Representation
as a graphical model:
nodes = variables
arcs = dependencies

Auto-regressive models

- A bigram model $P(x_1, \ldots, x_n) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_2)\ldots P(x_n \mid x_{n-1})$

Ex: “The only have every respect…”

- A trigram model $P(x_1, \ldots, x_n) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_2, x_1)\ldots P(x_n \mid x_{n-1}, x_{n-2})$

Ex: “The name was indeed in a grey carpet…”

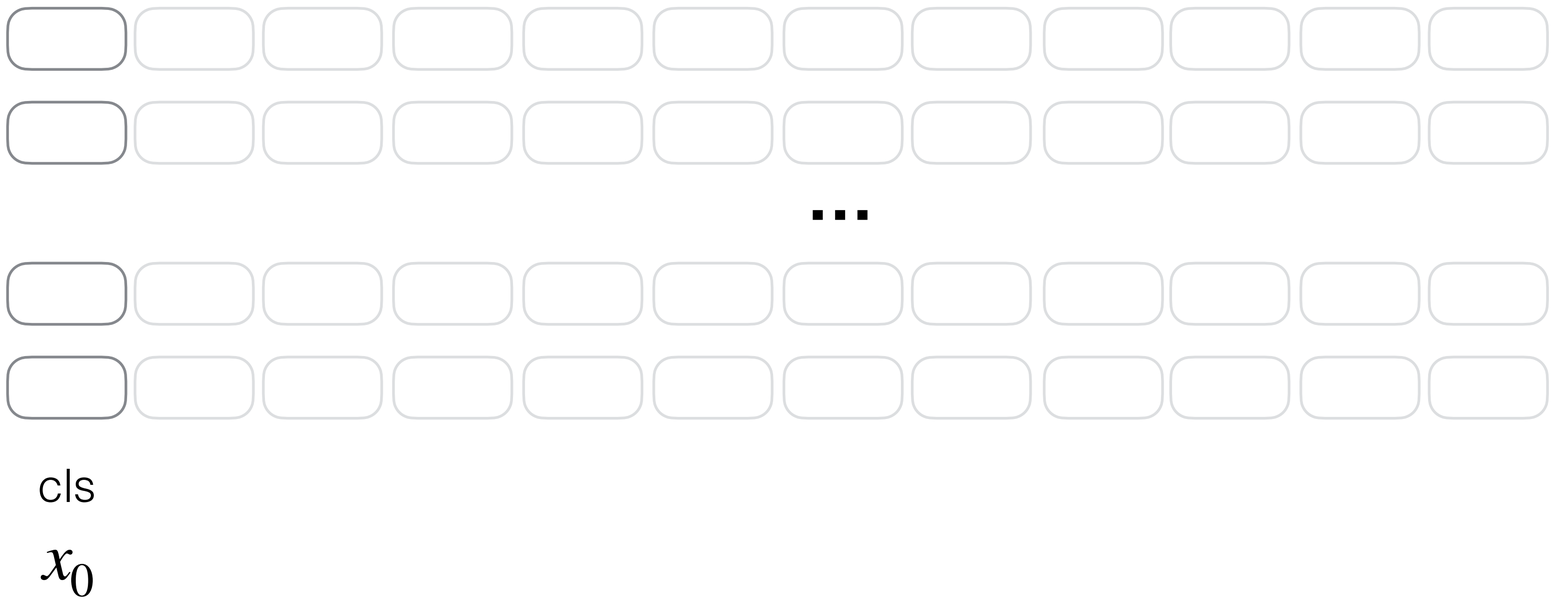
- A neural model can capture much longer context (capacity to do so depends on the architecture)

$$P(x_1, \ldots, x_n) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_2, x_1)\ldots P(x_n \mid x_{n-1}, x_{n-2}, \ldots, x_1)$$

Ex: “But for the trained reasoner to admit such intrusions …”

Text generation in a transformer

- In a transformer, we retain a full dependence on the context

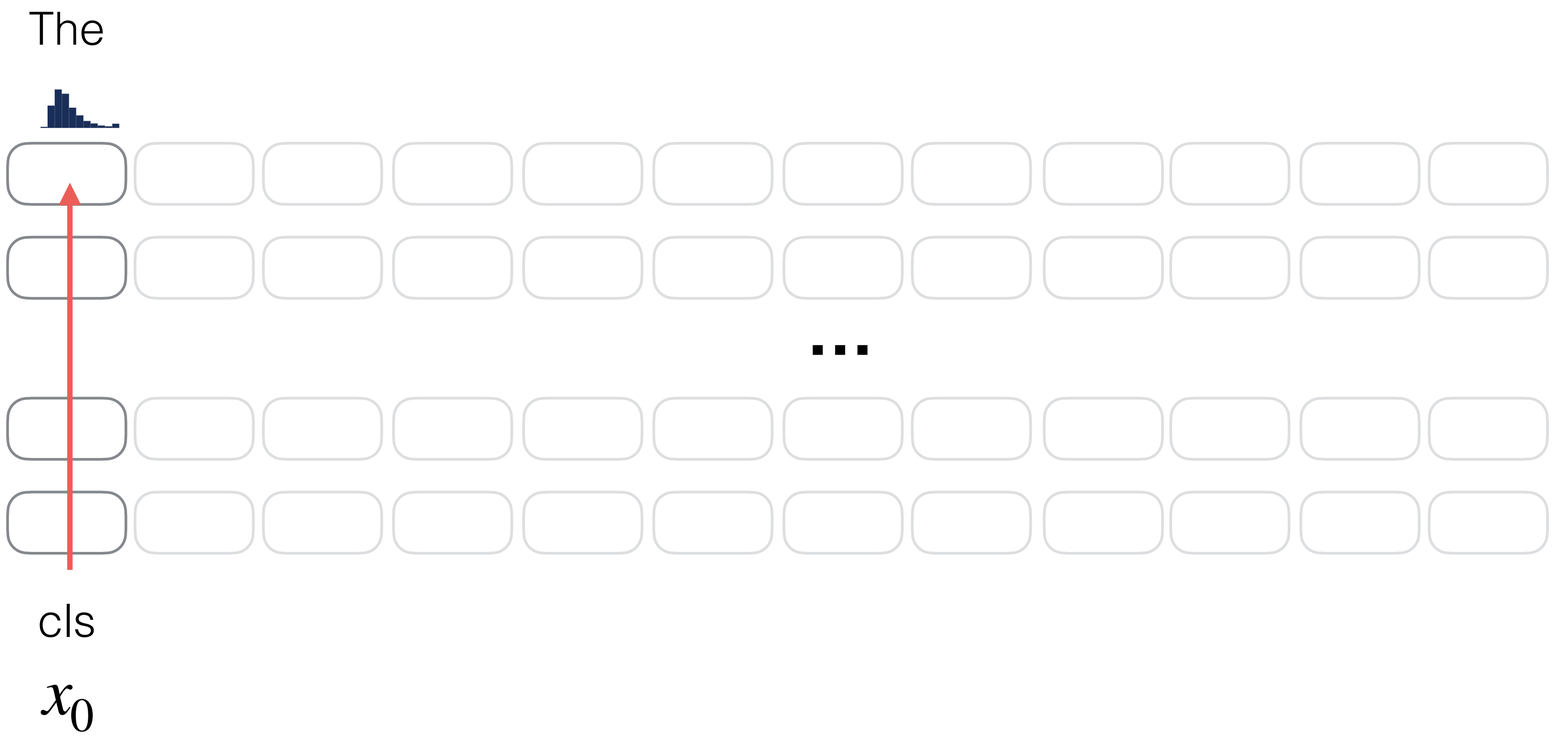


- The model can better “analyze” its context in relation to future text

Text generation in a transformer

- In a transformer, we retain a full dependence on the context

$P(x_1 \mid x_0)$.

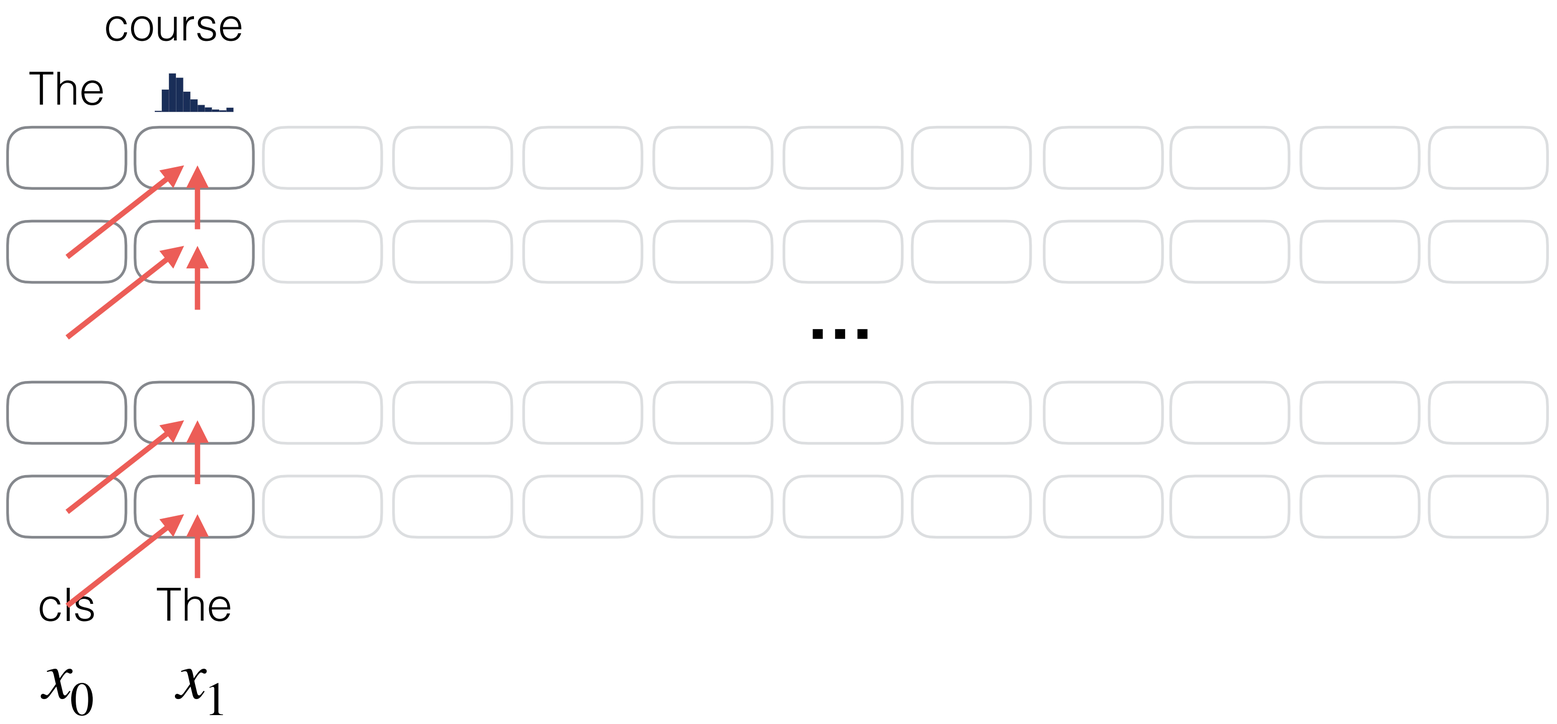


- The model can better “analyze” its context in relation to future text

Text generation in a transformer

- In a transformer, we retain a full dependence on the context

$$P(x_1 \mid x_0)P(x_2 \mid x_1, x_0)$$

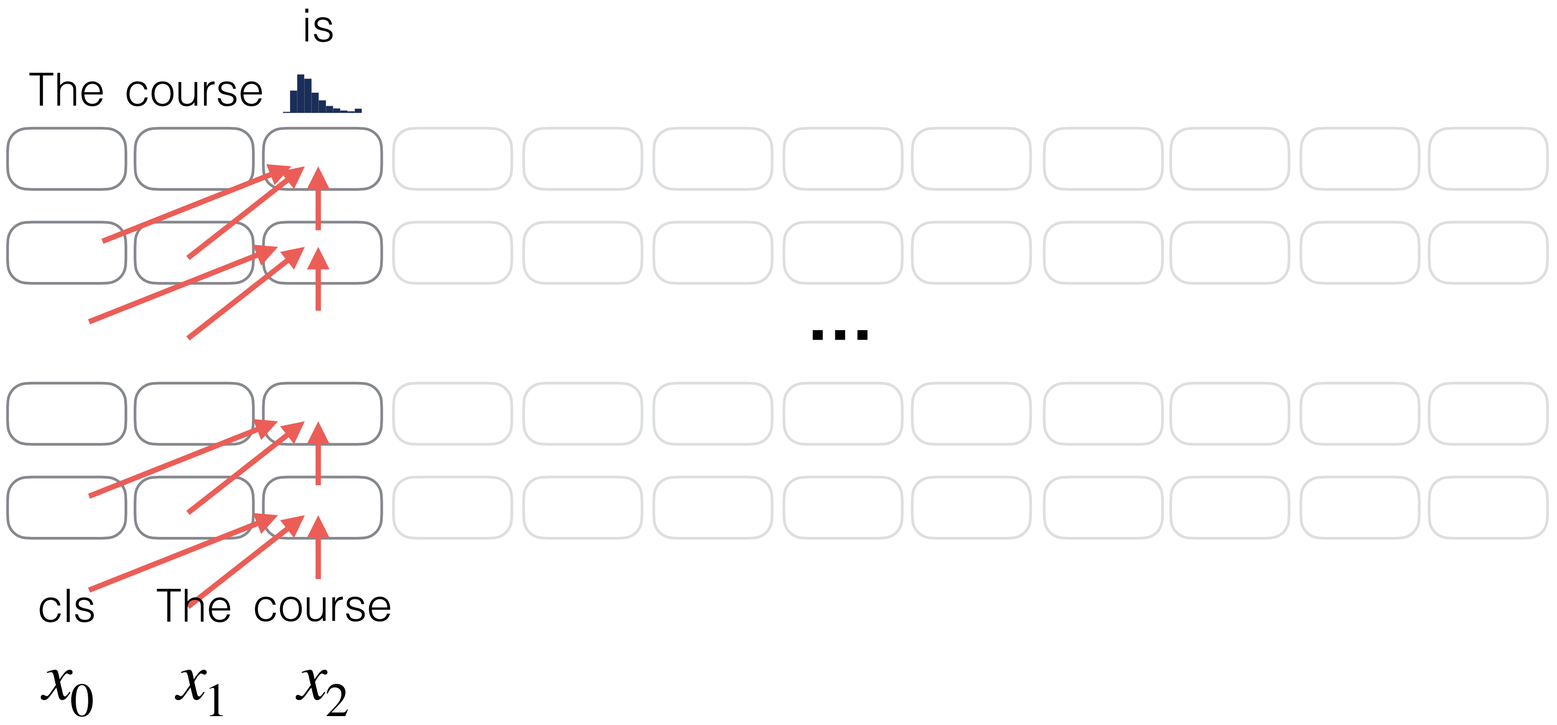


- The model can better “analyze” its context in relation to future text

Text generation in a transformer

- In a transformer, we retain a full dependence on the context

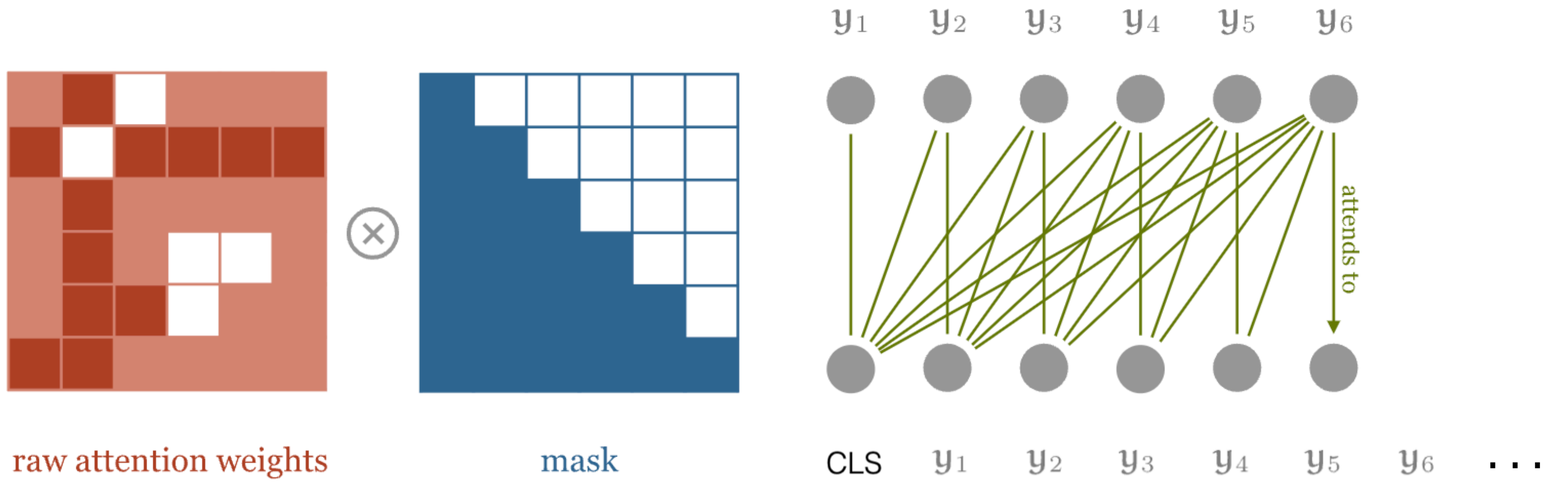
$$P(x_1 \mid x_0)P(x_2 \mid x_1, x_0)P(x_3 \mid x_2, x_1, x_0)\cdots$$



- The model can better “analyze” its context in relation to future text

Training a transformer

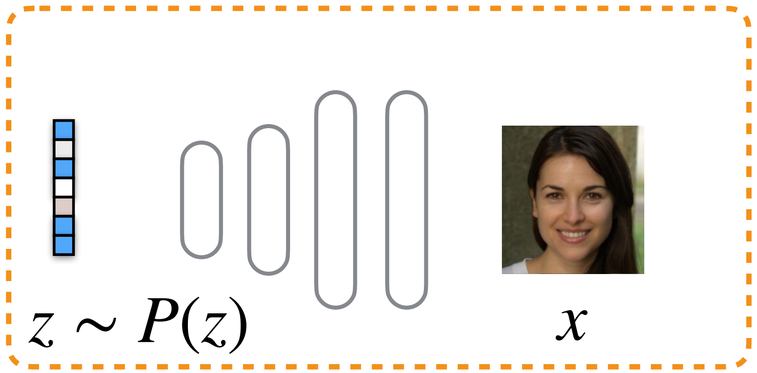
- We can train the transformer to generate sentences (i.e., function as a decoder) without asking it (during training) to iteratively generate each word
- The whole sentence can be fed into the model in one go if we restrict its attention heads to focus only on previously generated words (not future words!)
- We can do this by simply masking the regular attention matrix (zeroing out any references to future)



https://peterbloem.nl/blog/transformers

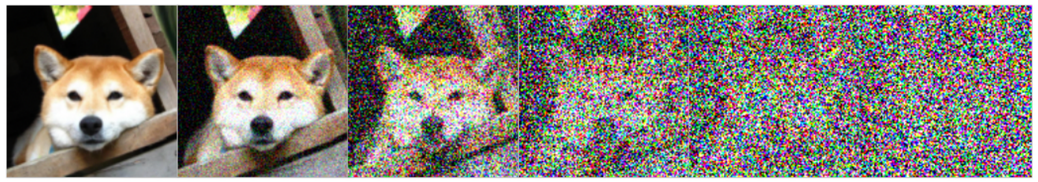
From “noise to clean” generative approaches

- Many generative models operate by sampling a latent vector from a simple distribution and then mapping this latent vector to clean examples (images, text, molecules, etc)
- The approaches differ in terms of whether the mapping is “iterative” and/or the estimation criterion



simple distribution complex distribution

iterative “de-noising”



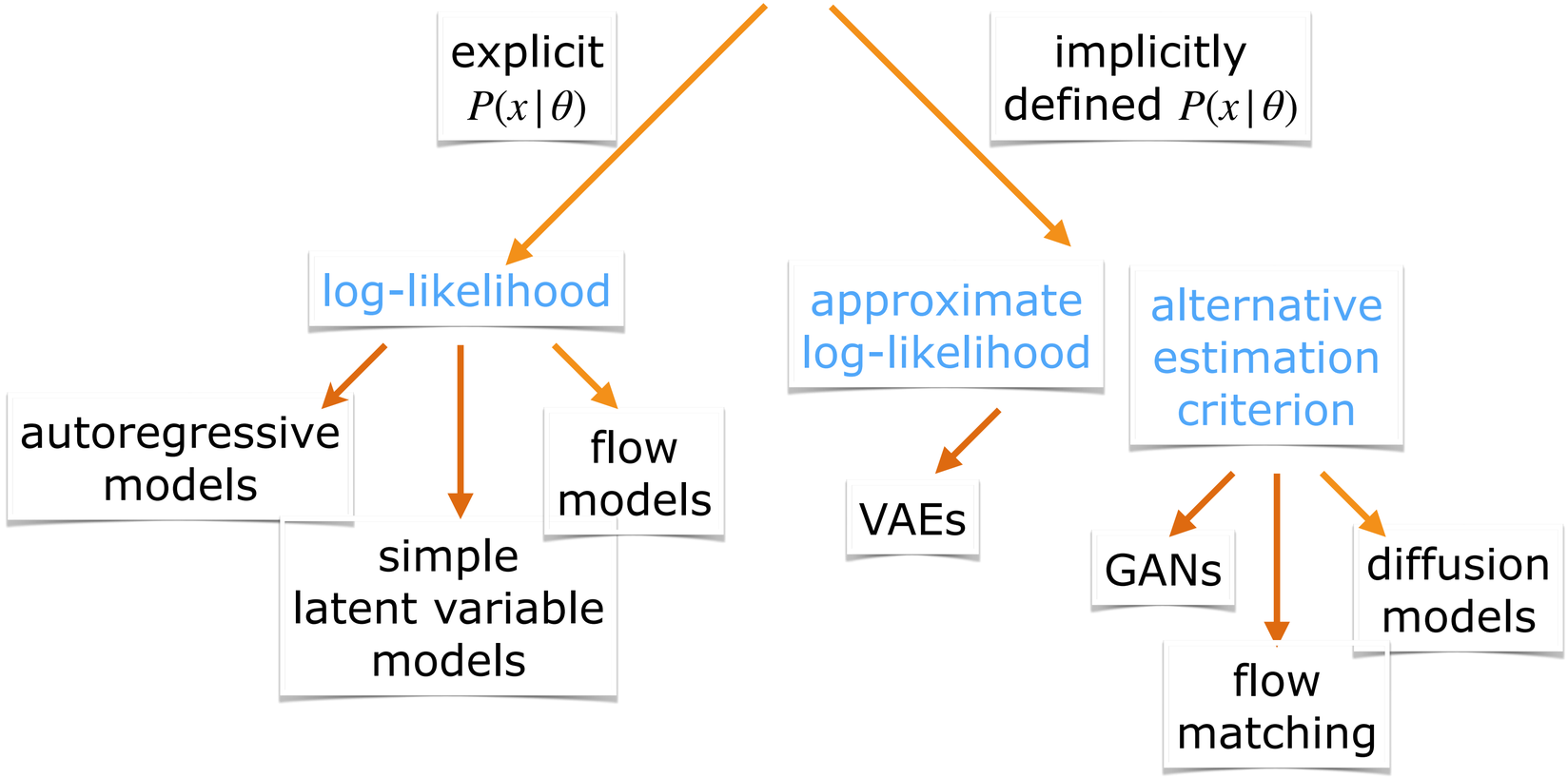
“clean” samples

simple/noise samples

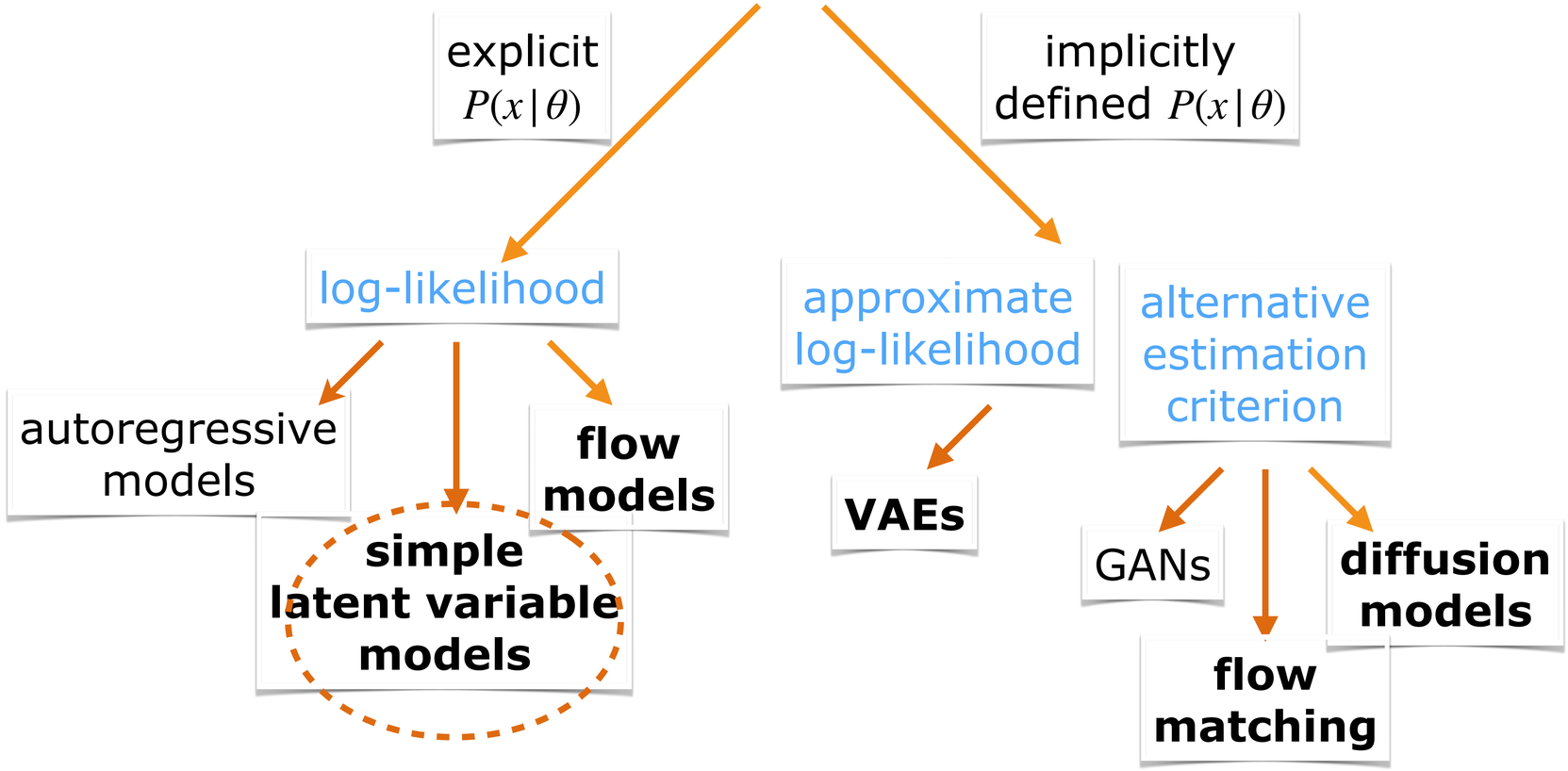
GANs, **VAEs**

Diffusion, flow matching

A slice of the generative "landscape"

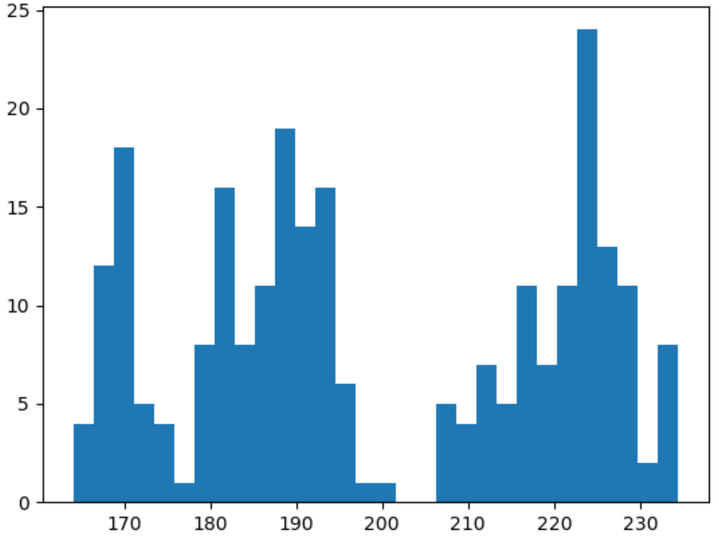


A slice of the generative "landscape"



A simple start: kernel density estimation

- How do we turn samples into reasonable density estimates?

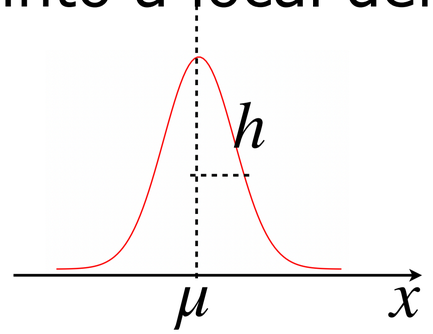


1 dim data represented as a histogram

- We can expand each data point into a local density via a “kernel”…

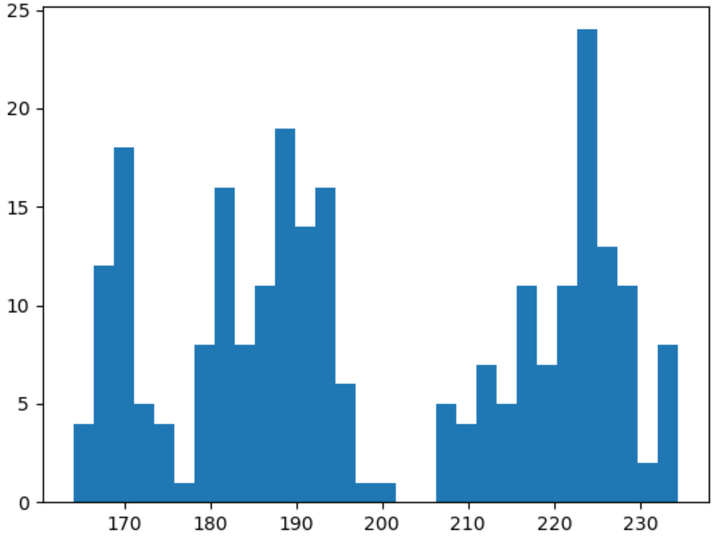
$$K(x) = N(x \mid 0,1)$$ standard normal

$$\frac{1}{h}K(\frac{x-\mu}{h}) = N(x \mid \mu, h^2)$$



A simple start: kernel density estimation

- How do we turn samples into reasonable density estimates?

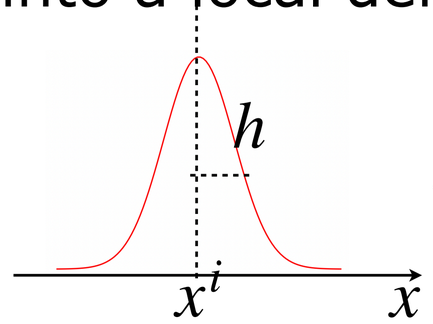


1 dim data represented as a histogram

- We can expand each data point into a local density via a “kernel”…

$$K(x) = N(x \mid 0,1)$$ standard normal

$$\frac{1}{h}K(\frac{x-\mu}{h}) = N(x \mid \mu, h^2)$$

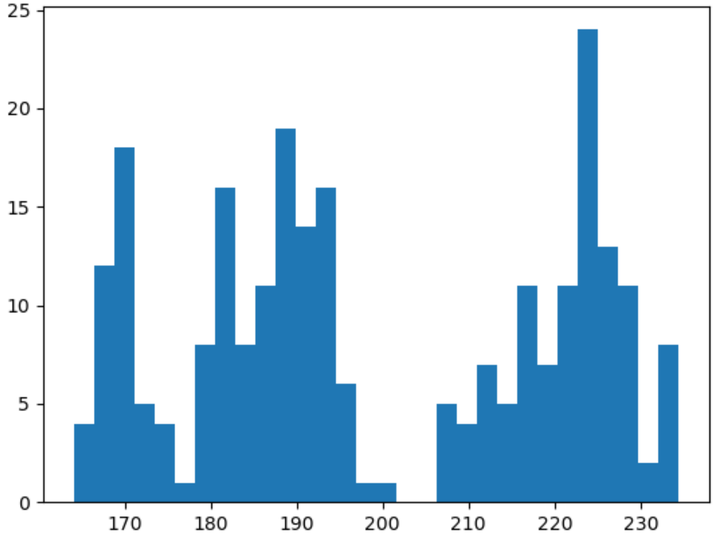


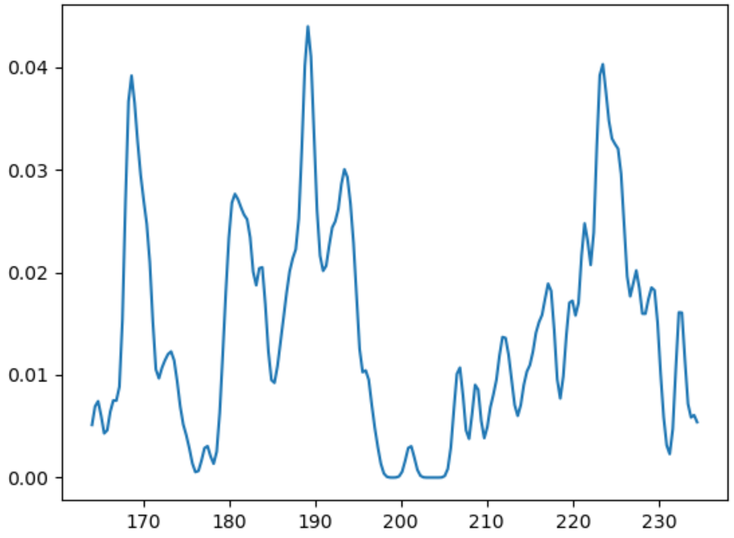
$$x^i \rightarrow \frac{1}{h}K(\frac{x-x^i}{h})$$

here h is an overall scale parameter we have to set

Kernel density estimation

- How do we turn samples into reasonable density estimates?





h=0.5

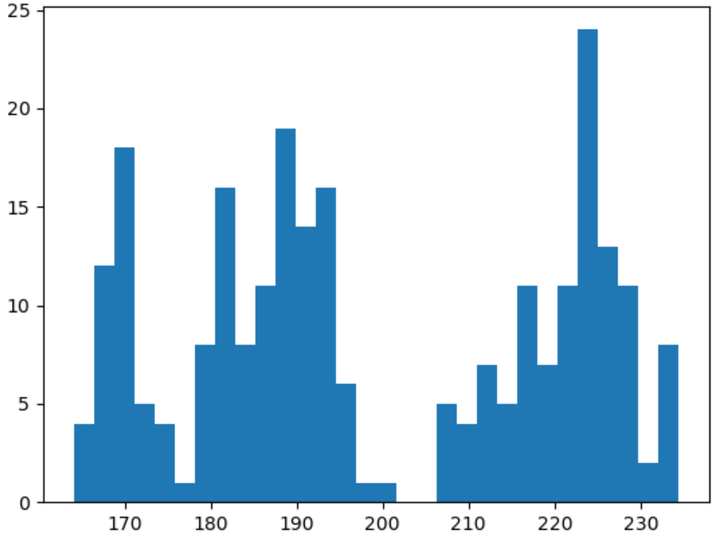
- We can expand each data point into a local density via a “kernel” and average:

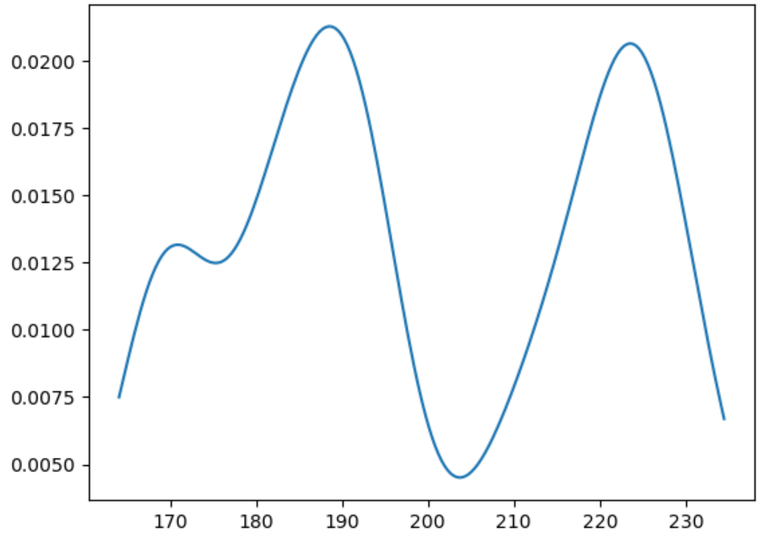
$$\hat{P}(x \mid h) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{h} K\left(\frac{x - x^i}{h}\right)$$

we only need to set h

Kernel density estimation

- How do we turn samples into reasonable density estimates?





h=5

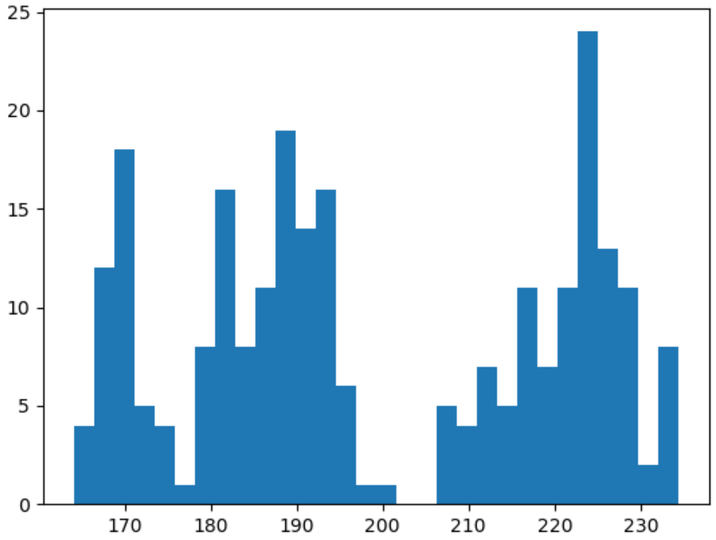
- We can expand each data point into a local density via a “kernel” and average:

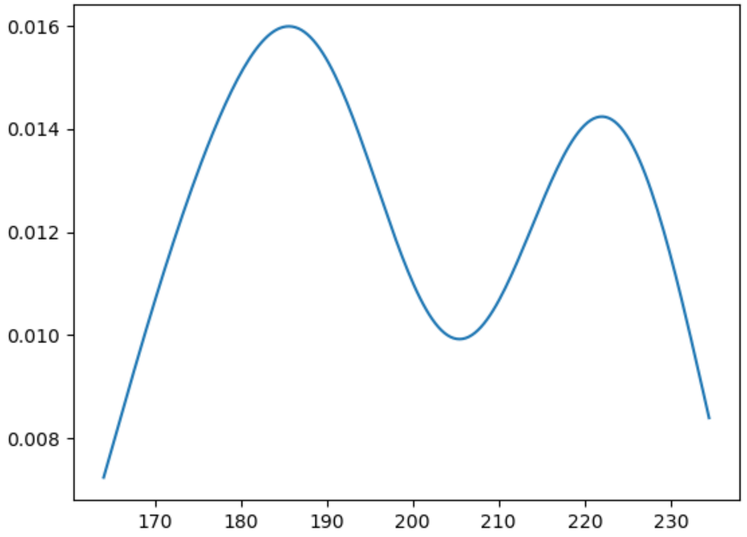
$$\hat{P}(x \mid h) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{h} K\left(\frac{x - x^i}{h}\right)$$

we only need to set h

Kernel density estimation

- How do we turn samples into reasonable density estimates?





h=10

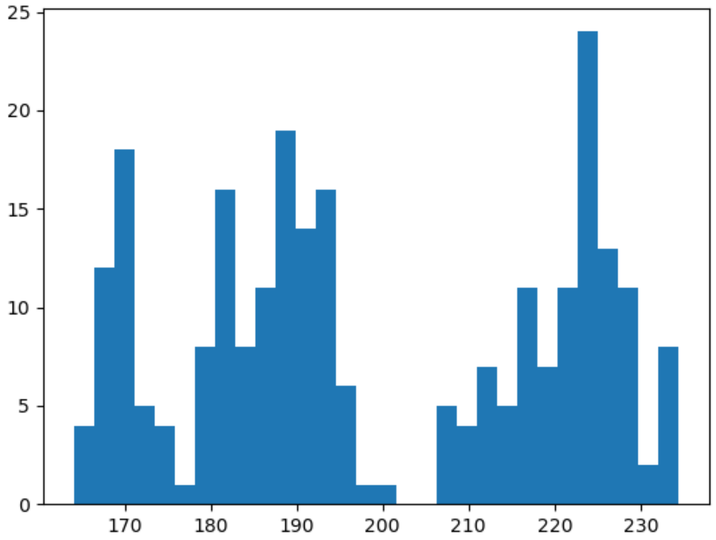
- We can expand each data point into a local density via a “kernel” and average:

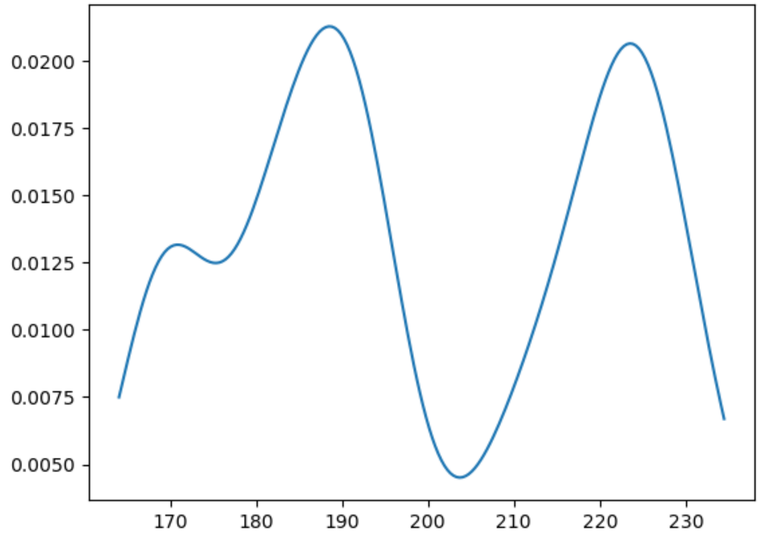
$$\hat{P}(x \mid h) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{h} K\left(\frac{x - x^i}{h}\right)$$

we only need to set h

Kernel density estimation

- How do we turn samples into reasonable density estimates?





h=5

- We can expand each data point into a local density via a “kernel” and average:

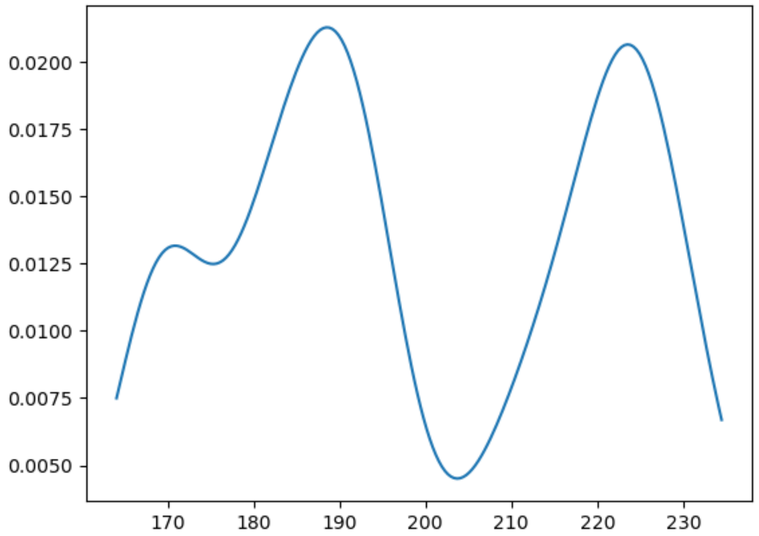
$$\hat{P}(x \mid h) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{h} K\left(\frac{x - x^i}{h}\right)$$

we can set h, e.g., based on the average log-likelihood of the validation examples

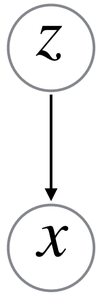
- But how do we sample from the resulting density?

Kernel density as a mixture model

- How do we sample from the resulting density?



latent index of data point



$$z \sim Cat(z \mid 1/N, \ldots, 1/N) = P(z)$$

$$x \sim \frac{1}{h} K\left(\frac{x - x^z}{h}\right) = P(x \mid z)$$

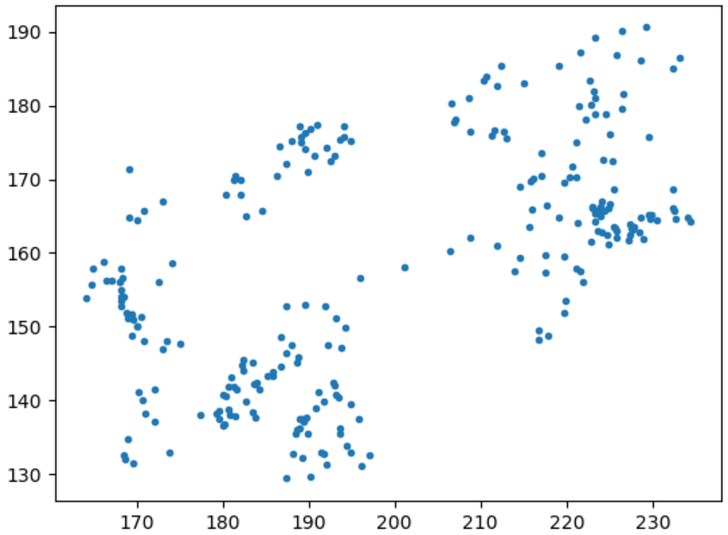
sample from the corresponding kernel

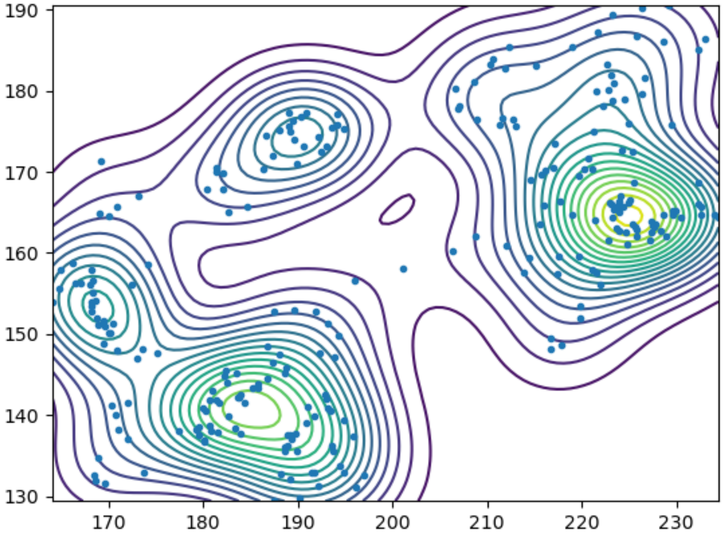
$$\hat{P}(x \mid h) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{h} K\left(\frac{x - x^i}{h}\right) = \sum_{z=1}^{N} P(z) P(x \mid z)$$

a mixture model

Multi-dimensional kernel density estimate

- We can in principle extend the approach to multiple dimensions (here d = 2)



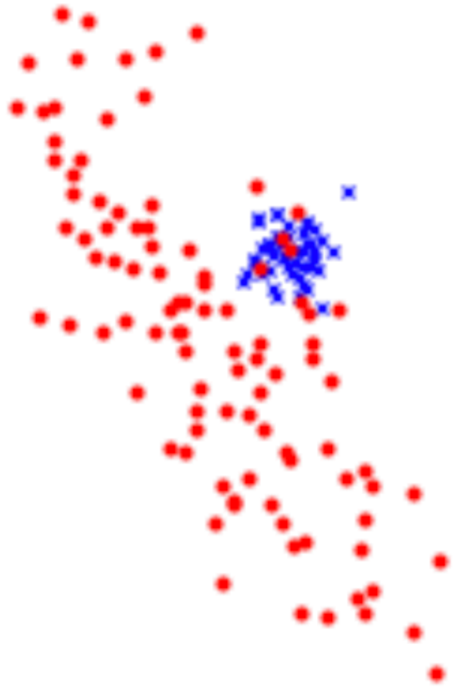


- But doesn’t scale well with increasing dimension (need better parametric assumptions to “extend” data points)
- We need to keep all the data!

$$\hat{P}(x \mid h) = \frac{1}{N} \sum_{i=1}^{N} \left[\prod_{j=1}^{d} \frac{1}{h} K\left(\frac{x_j - x_j^i}{h}\right) \right]$$

Finite mixture models

- We wish to account for “structure” in the data consisting of
  - overlapping clusters
  - different numbers of points in each cluster
  - different cluster shapes
- We will explicitly define and estimate a generative process (a statistical distribution) over the examples



Building from simple components

- We can build complex generative models from simpler components, e.g.,

- Bernoulli

$$y \in \{0,1\},\ y \sim \text{Bernoulli}(\pi),\ \ P(y=1) = \pi$$

- Categorical

$$y \in \{1,\ldots,k\},\ y \in \text{Cat}(\pi_1,\ldots,\pi_k),\ \ P(y=j) = \pi_j$$

- univariate Gaussian

$$x \in \mathbb{R}, x \sim N(\mu,\sigma^2),\ \ N(x\,|\,\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp(-\frac{1}{2\sigma^2}(x-\mu)^2)$$

- spherical Gaussian

$$x \in \mathbb{R}^d, x \sim N(\mu,\sigma^2 I),\ \ N(x\,|\,\mu,\sigma^2 I) = \frac{1}{(2\pi\sigma^2)^{d/2}}\exp(-\frac{1}{2\sigma^2}\|x-\mu\|^2)$$

- etc.

Recall: ML estimates

- Derive maximum likelihood estimates for these distributions based on n samples

- E.g., for categorical distribution

observed data $D = \{y_1, \ldots, y_n\}, y_i \in \{1, \ldots, k\}$

log-likelihood of data $l(D; \pi) = \sum_{i=1}^{n} \log P(y = y_i) = \sum_{i=1}^{n} \log \pi_{y_i}$

We want $\hat{\pi}$ that max $l(D; \pi)$ subject to $\pi_j \geq 0, \sum_{j=1}^{k} \pi_j = 1$

$$\Rightarrow \hat{\pi}_j = \frac{1}{n} \sum_{i=1}^{n} [[y_i = j]]$$

Weighted ML estimates

- We wish to derive maximum likelihood estimates for distributions based on N samples, including when the samples are "weighted"
- E.g., for categorical distribution each observation is now "blended" across possible values

observed weighted data $D = \{Q(y \mid i)\},\ i = 1,\ldots,n,\ y \in \{1,\ldots,k\}$

weighted log-likelihood of data $l(D;\pi) = \sum_{i=1}^{n} \sum_{y=1}^{k} Q(y \mid i) \log \pi_y$

We want $\hat{\pi}$ that max $l(D;\pi)$ subject to $\pi_j \geq 0, \sum_{j=1}^{k} \pi_j = 1$

$$\Rightarrow \hat{\pi}_j = \frac{1}{\sum_{i=1}^{n} \sum_{y=1}^{k} Q(y \mid i)} \sum_{i=1}^{n} Q(y = j \mid i)$$

Weighted ML estimates for a Gaussian

- E.g., we would like to estimate a Gaussian modelfrom data where each point x^i has a non-negative weight $Q(1 \mid i)$ (notation here foreshadows use in mixtures)

- Weighted log-likelihood objective:

$$l(D; \mu, \sigma^2) = \sum_{i=1}^{N} Q(1 \mid i) \log N(x^i \mid \mu, \sigma^2) = \sum_{i=1}^{N} Q(1 \mid i) \left[\frac{1}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2}(x^i - \mu)^2 \right]$$

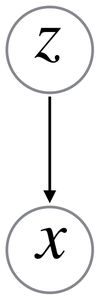
- The resulting maximizing parameter values (mean, variance) are now "weighted" mean and variance, e.g.,

$$\frac{d}{d\mu} l(D; \mu, \sigma^2) = 0 \quad \Rightarrow \quad \hat{\mu} = \frac{\sum_{i=1}^{N} Q(1 \mid i)\, x^i}{\sum_{i=1}^{N} Q(1 \mid i)}$$

Mixture models

- A mixture model defines a generative process with a latent cluster choice

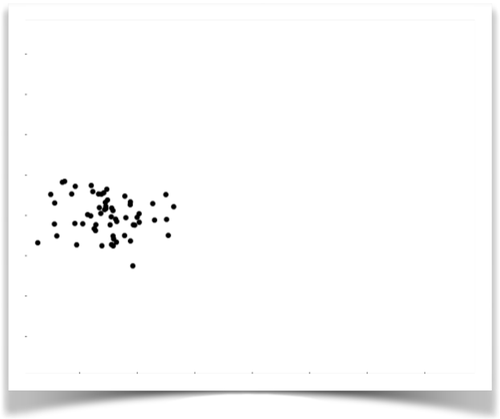
latent



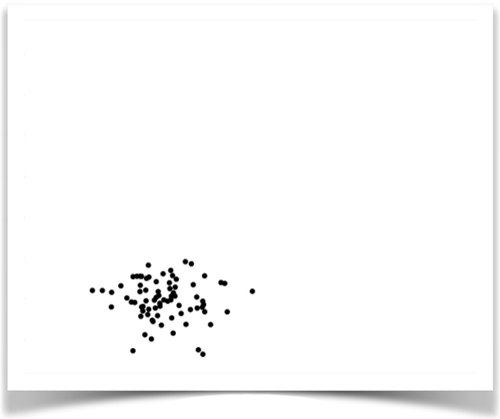
$z \sim P(z)$ select a cluster

$x \sim P(x|z)$ draw a sample from the corresponding cluster model

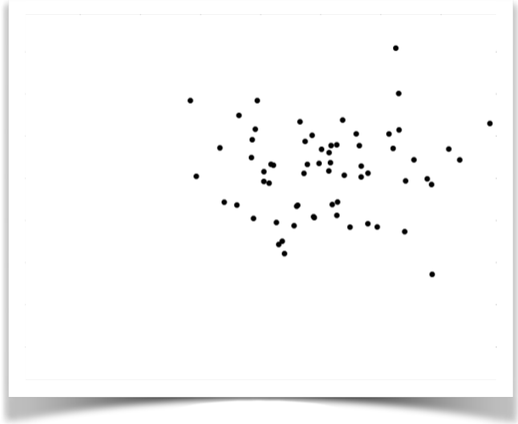
observed



$P(x|z=1)$



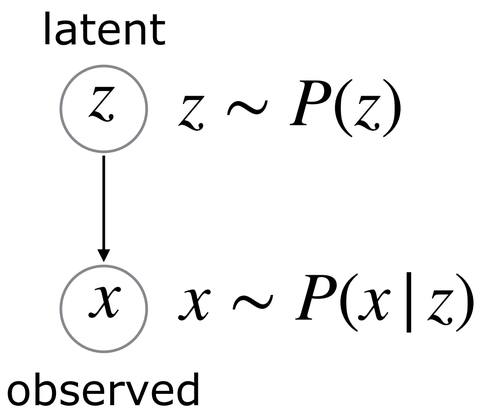
$P(x|z=2)$

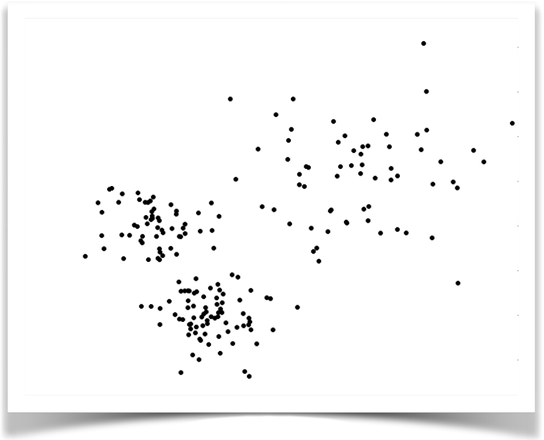


$P(x|z=3)$

Mixture models

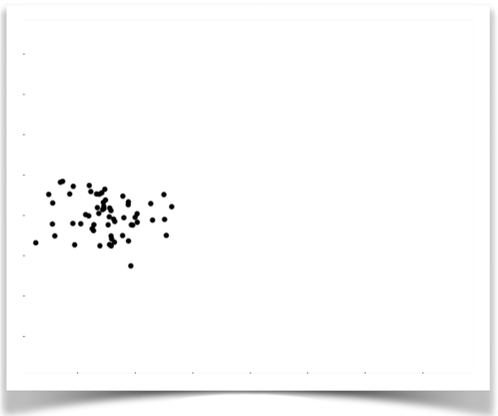
- A mixture model defines a generative process with a latent cluster choice





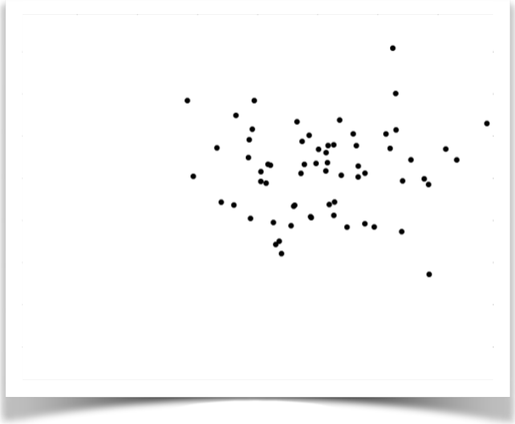
$$P(x) = \sum_{z=1}^{k} P(x|z)P(z)$$

marginal probability over
x is a mixture distribution



$P(x|z=1)$

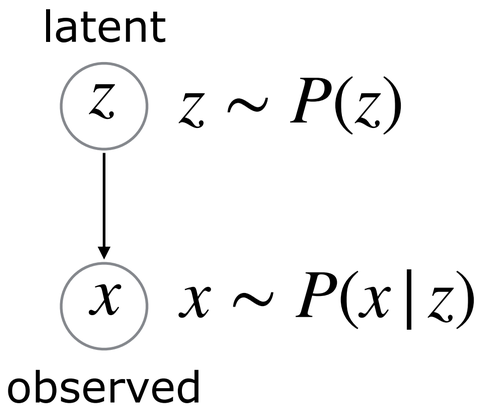
$P(x|z=2)$



$P(x|z=3)$

Sampling from mixture models

- Suppose we draw n samples from a mixture model (keeping only the x's, not the "latent" cluster choices z)
- When to draw what samples matters a great deal, e.g.,

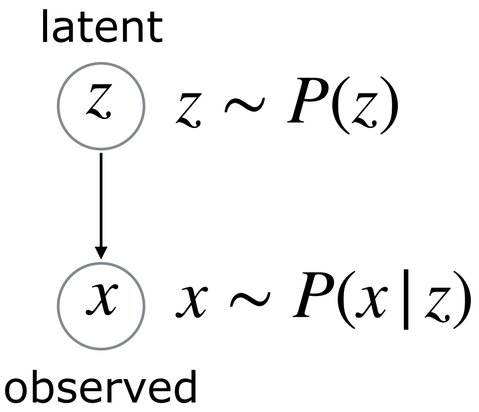


- (1) $z \sim P(z), \{x^i \sim P(x|z)\}_{i=1,\ldots,n}$

- (2) $\{z^i \sim P(z), x^i \sim P(x|z^i)\}_{i=1,\ldots,n}$

Sampling from mixture models

- Suppose we draw n samples from a mixture model (keeping only the x's, not the "latent" cluster choices z)
- When to draw what samples matters a great deal, e.g.,



- (1) $z \sim P(z), \{x^i \sim P(x|z)\}_{i=1,\ldots,n}$

$$P(x^1, \ldots, x^n) = \sum_{z=1}^{k} P(z) \prod_{i=1}^{n} P(x^i|z)$$

- (2) $\{z^i \sim P(z), x^i \sim P(x|z^i)\}_{i=1,\ldots,n}$

$$P(x^1, \ldots, x^n) = \prod_{i=1}^{n} \sum_{z^i=1}^{k} P(z^i) P(x^i|z^i)$$

Sampling from mixture models

- Suppose we draw n samples from a mixture model (keeping only the x's, not the "latent" cluster choices z)
- When to draw what samples matters a great deal, e.g.,

latent

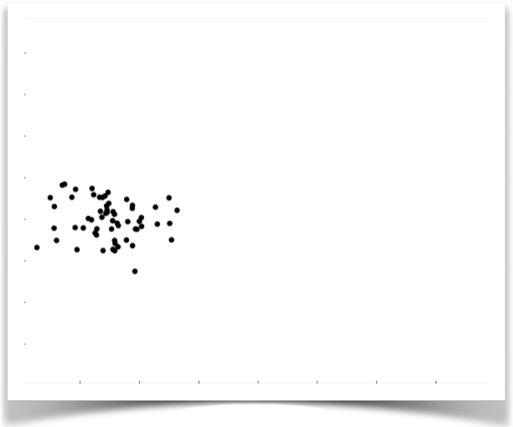
$z \quad z \sim P(z)$

$x \quad x \sim P(x \mid z)$

observed

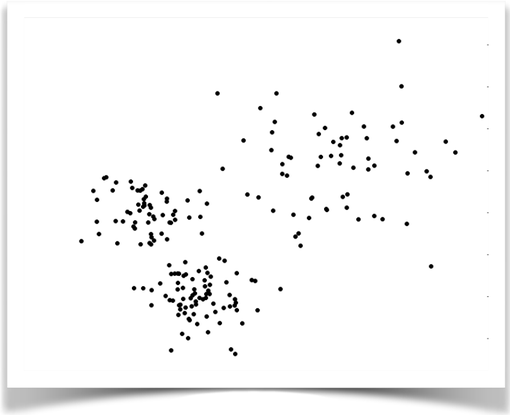
- (1) $z \sim P(z), \{x^i \sim P(x \mid z)\}_{i=1,\ldots,n}$

$$P(x^1, \ldots, x^n) = \sum_{z=1}^{k} P(z) \prod_{i=1}^{n} P(x^i \mid z)$$



- (2) $\{z^i \sim P(z), x^i \sim P(x \mid z^i)\}_{i=1,\ldots,n}$

$$P(x^1, \ldots, x^n) = \prod_{i=1}^{n} \sum_{z^i=1}^{k} P(z^i) P(x^i \mid z^i)$$



what n samples might look like

A number of different mixture models

- Discrete parametric mixtures (e.g., for clustering)

$$P(x|\theta) = \sum_{z=1}^{k} P(x|z,\theta)P(z|\theta)$$

- Continuous parametric mixtures

$$P(x|\theta) = \int P(x|z,\theta)P(z|\theta)dz$$

- Iterated mixtures (e.g., LDA topic model for documents)

$$P(w_1,\ldots,w_n) = \int P(\theta) \prod_{i=1}^{n} \left(\sum_{z_i=1}^{K} \theta_{z_i} \beta_{w_i|z_i} \right) d\theta$$

- Semi- or non-parametric mixtures (e.g., infinite cluster models)

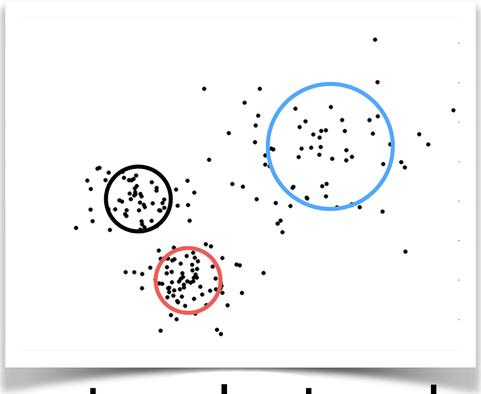
$$P(x|\theta) = E_{\mu}\left\{ \int P(x|z)\mu(z)dz \right\}$$

- etc.

A Gaussian mixture model (GMM)

- A k-component Gaussian mixture model

$$P(x \mid \theta) = \sum_{z=1}^{k} P(z \mid \theta) P(x \mid z, \theta) = \sum_{z=1}^{k} \pi_z N(x \mid \mu_z, \Sigma_z)$$



- Each component or "cluster" is a Gaussian with a separate, cluster dependent mean and covariance (we'll look at spherical Gaussians for simplicity $\Sigma_z = \sigma_z^2 I$)
- $\{\pi_j\}$ are mixing proportions
- The goal is to estimate the mixture model from "unlabeled" data $D = \{x_1, \ldots, x_n\}$ by maximizing log-likelihood

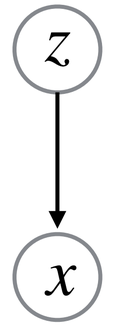
$$l(D; \theta) = \sum_{i=1}^{n} \log P(x_i \mid \theta) = \sum_{i=1}^{n} \log \left[\sum_{j=1}^{k} \pi_j N(x_i \mid \mu_j, \sigma_j^2 I) \right]$$

Learning mixture models: gradient ascent

- We wish to maximize the log-likelihood, can just take small gradient steps with respect to the parameters θ

$$\nabla_\theta \log P(x|\theta) = \sum_{z=1}^{k} P(z|x,\theta)\, \nabla_\theta \log\left[P(z|\theta)P(x|z,\theta)\right]$$

$$\log \pi_z - \frac{1}{2\sigma_z^2}\|x-\mu_z\|^2 - \frac{d}{2}\log(2\pi\sigma_z^2)$$

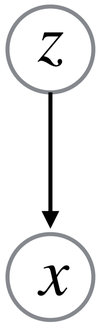


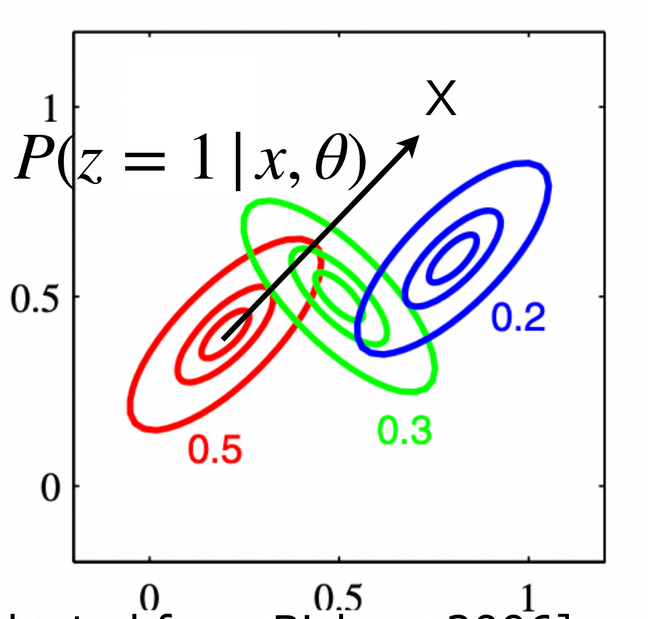
Learning mixture models: gradient ascent

- We wish to maximize the log-likelihood, can just take small gradient steps with respect to the parameters θ

$$\nabla_\theta \log P(x|\theta) = \sum_{z=1}^{k} P(z|x,\theta)\, \nabla_\theta \log\left[P(z|\theta)P(x|z,\theta)\right]$$

$$\log \pi_z - \frac{1}{2\sigma_z^2}\|x-\mu_z\|^2 - \frac{d}{2}\log(2\pi\sigma_z^2)$$





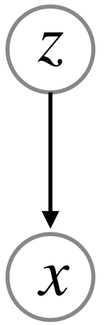
[fig adapted from Bishop, 2006]

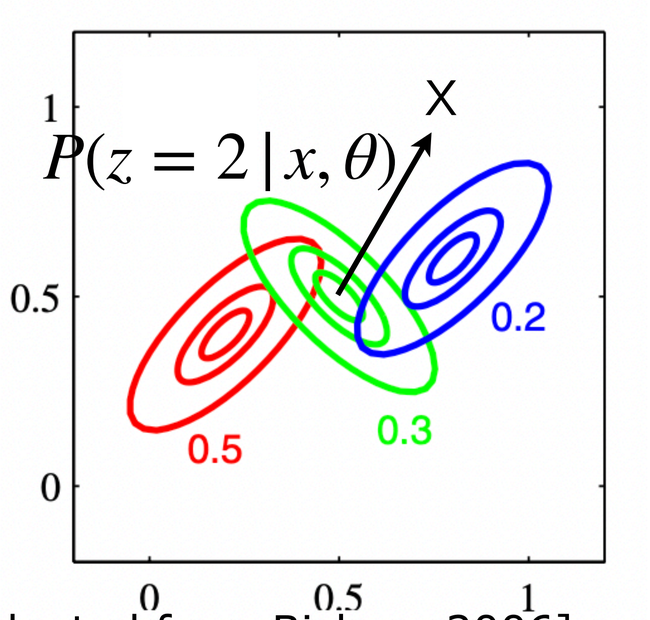
Learning mixture models: gradient ascent

- We wish to maximize the log-likelihood, can just take small gradient steps with respect to the parameters θ

$$\nabla_\theta \log P(x|\theta) = \sum_{z=1}^{k} P(z|x,\theta)\, \nabla_\theta \log\left[P(z|\theta)P(x|z,\theta)\right]$$

$$\log \pi_z - \frac{1}{2\sigma_z^2}\|x-\mu_z\|^2 - \frac{d}{2}\log(2\pi\sigma_z^2)$$





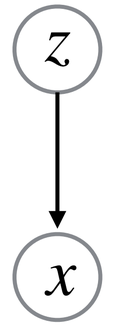
[fig adapted from Bishop, 2006]

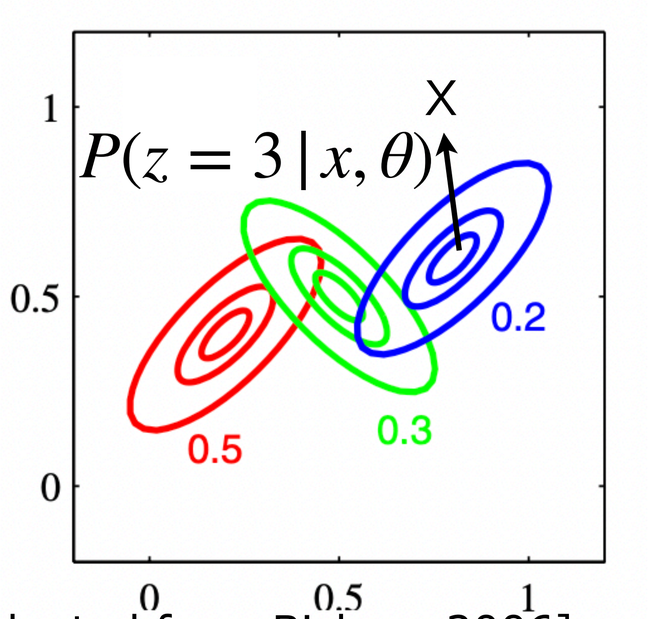
Learning mixture models: gradient ascent

- We wish to maximize the log-likelihood, can just take small gradient steps with respect to the parameters θ

$$\nabla_\theta \log P(x \mid \theta) = \sum_{z=1}^{k} P(z \mid x, \theta) \nabla_\theta \log \left[P(z \mid \theta) P(x \mid z, \theta) \right]$$

$$\log \pi_z - \frac{1}{2\sigma_z^2} \|x - \mu_z\|^2 - \frac{d}{2} \log(2\pi\sigma_z^2)$$





[fig adapted from Bishop, 2006]

Two key points:

1. gradient update is weighted by posterior probabilities (aka responsibilities)
2. log probs tend to be simple in the generative direction

Gradient ascent as a generalized EM algorithm

- The EM algorithm alternates between calculating the posterior assignments and model updates (here only for a single x)

- **E-step:** assign each data point x to clusters in a weighted manner according to posterior probabilities, now stored as $Q(z|x) = P(z|x,\theta)$

- **M-step:** update cluster models and mixing proportions based on the resulting complete (x,z) data, weighted by the posteriors $Q(z|x)$ (fixed during this step)

$$\theta \leftarrow \theta + \eta \sum_{z=1}^{k} Q(z|x) \nabla_\theta \log\left[P(z|\theta)P(x|z,\theta)\right]$$

A generalized EM algorithm

- The EM algorithm alternates between calculating the posterior assignments and model updates (here only for a single x)

- **E-step:** assign each data point x to clusters in a weighted manner according to posterior probabilities, now stored as $Q(z|x) = P(z|x,\theta)$

- **M-step:** update cluster models and mixing proportions based on the resulting complete (x,z) data, weighted by the posteriors $Q(z|x)$ (fixed during this step)

$$\theta \leftarrow \theta + \eta \sum_{z=1}^{k} Q(z|x) \nabla_\theta \log \left[P(z|\theta) P(x|z,\theta) \right]$$

Can we make larger updates
in response to fixed posterior weights?
(yes, next slides)

A generalized EM algorithm

- The EM algorithm alternates between calculating the posterior assignments and model updates (here only for a single x)

Approximate vs exact posteriors?
(more about this at next lecture)

- **E-step:** assign each data point x to clusters in a weighted manner according to posterior probabilities, now stored as $Q(z|x) = P(z|x,\theta)$

- **M-step:** update cluster models and mixing proportions based on the resulting complete (x,z) data, weighted by the posteriors $Q(z|x)$ (fixed during this step)

$$\theta \leftarrow \theta + \eta \sum_{z=1}^{k} Q(z|x) \nabla_\theta \log\left[P(z|\theta)P(x|z,\theta)\right]$$

Can we make larger updates
in response to fixed posterior weights?
(yes, next slides)