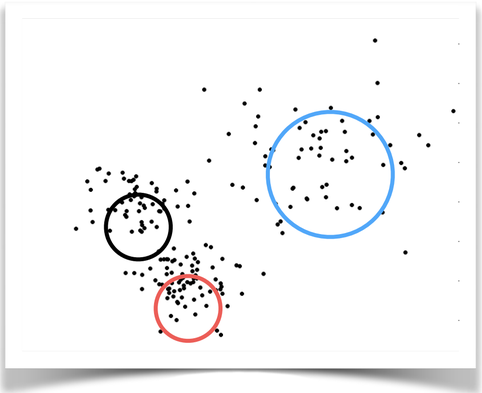
6.7900 Machine Learning (Fall 2024)

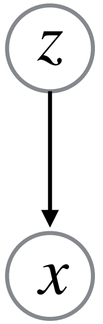
Lecture 21: mixtures cont'd, VAEs (supporting slides)

Recall: Gaussian mixture model (GMM)

- A k-component Gaussian mixture model

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





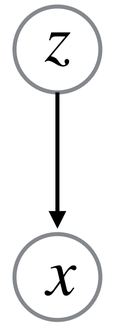
- Each component or "cluster model" is a Gaussian with its separate, cluster dependent mean and covariance (we'll limit ourselves to spherical Gaussians for simplicity so that $\Sigma_z = \sigma_z^2 I$). The parameters $\{\pi_j\}$ are called mixing proportions.
- Our first goal is to estimate a particular mixture model, i.e., find parameters $\theta = \{\pi_1, \ldots, \pi_k, \mu_1, \ldots, \mu_k, \sigma_1, \ldots, \sigma_k\}$ on the basis of "unlabeled" data $D = \{x^1, \ldots, x^n\}$. Note that here $x \in \mathbb{R}^d$ and each $\mu_z \in \mathbb{R}^d$ as well.
- We find the parameters that maximize the log-likelihood of data

$$l(D; \theta) = \sum_{i=1}^{N} \log P(x^i \mid \theta) = \sum_{i=1}^{N} \log \left[\sum_{z=1}^{k} \pi_z N(x^i \mid \mu_z, \sigma_z^2 I) \right]$$

Learning mixture models: gradient ascent

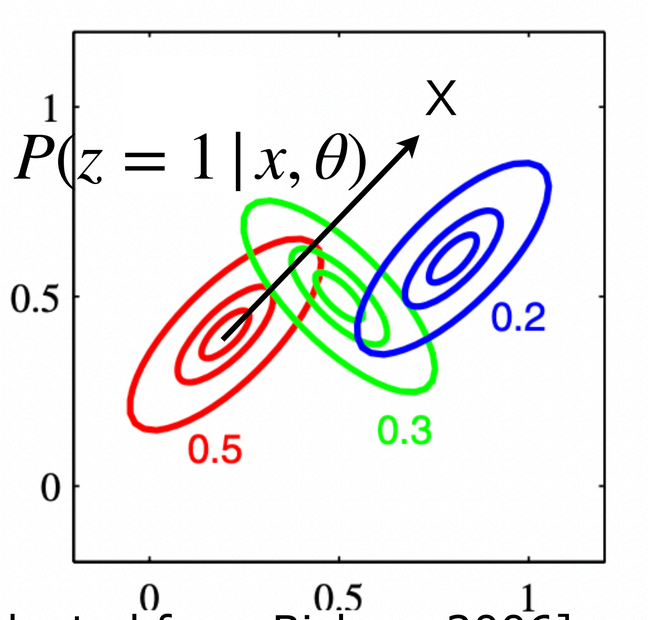
- We already derived that the gradient with respect to the parameters θ of the mixture model has a nice, posterior weighted form (here just for a single x)

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



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

complete log-likelihood of (x,z)
has a nice, simple form



[fig adapted from Bishop, 2006]

Gradient ascent as a generalized EM algorithm

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

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

Gradient ascent as a generalized EM algorithm

- The EM algorithm alternates between calculating the posterior assignments and model updates (here only for a single data point 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 many updates
in response to fixed posterior weights?

EM algorithm

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

$$\theta \leftarrow argmax_\theta \left[\sum_{z=1}^{k} Q(z|x) \log \left[P(z|\theta) P(x|z,\theta) \right] \right] \quad \textbf{?}$$

Can we just solve the weighted problem in the M-step in response to fixed posterior weights?

The EM algorithm

- The EM algorithm alternates between calculating the posterior assignments and model updates (here many data points)

- **E-step:** assign each data point x^i to clusters in a weighted manner according to posterior probabilities, now stored as $Q(z|x^i) = P(z|x^i, \theta),\ z = 1,\ldots,k$

- **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)

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

The M-step is a weighted maximum likelihood problem, decomposes into separate problems for mixing proportions and each cluster model (Gaussian). We can often get exact (weighted) solutions to such subproblems.

EM justification: ELBO lower bound

- The EM algorithm operates by monotonically increasing a lower bound on the log-likelihood, a bound that holds for any choice of Q(z|x)

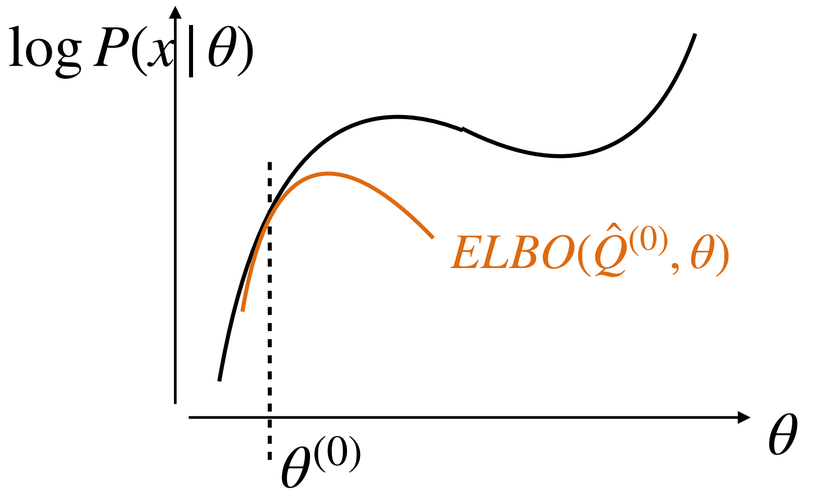
entropy of Q(z|x) for fixed x

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

(true for all Q, θ derived on the board)

$$\text{ELBO}(Q;\theta)$$

- The bound is tight when $Q(z|x) = P(z|x,\theta)$ (maximum wrt Q)



EM justification: ELBO lower bound

- The EM algorithm operates by monotonically increasing a lower bound on the log-likelihood, a bound that holds for any choice of Q(z|x)

entropy of Q(z|x) for fixed x

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

(true for all Q, θ derived on the board)

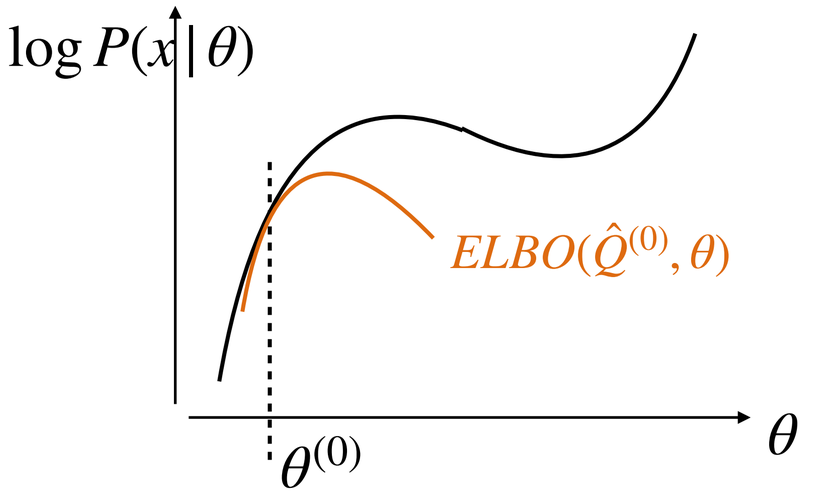
$$\text{ELBO}(Q;\theta)$$

- The bound is tight when $Q(z|x) = P(z|x,\theta)$ (maximum wrt Q)

- We can further show that the gap in the bound depends on how well Q approximates the posterior

$$ELBO(Q;\theta) = \log P(x|\theta) - KL(Q_{z|x} \| P_{z|x,\theta})$$

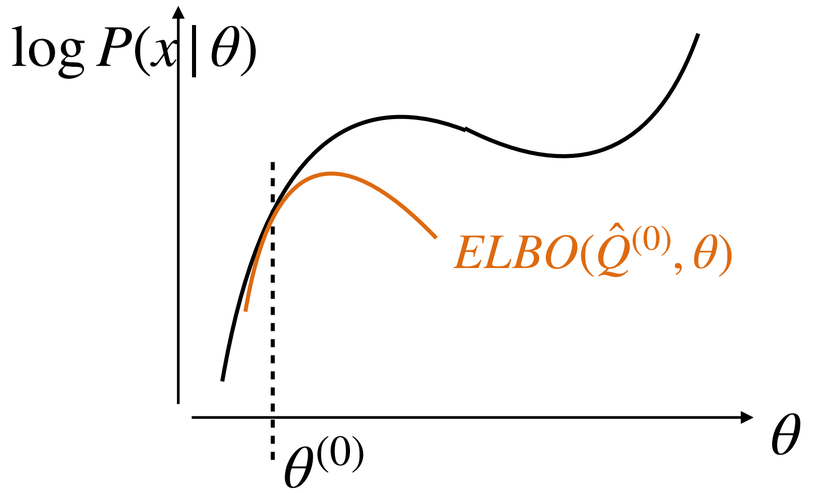
KL divergence ≥ 0
zero iff distributions agree



The EM algorithm (from a new perspective)

- Abstractly, the EM algorithm is an alternating maximization algorithm

- initialize θ
- **E-step (max wrt Q):** fix θ, store $\hat{Q}(z|x) = P(z|x,\theta)$ for all data points. This is equivalent to maximizing $ELBO(Q,\theta)$ with respect to Q
- **M-step (max wrt θ):** fix posterior weights $\hat{Q}$, update θ by maximizing $ELBO(\hat{Q},\theta)$. This is a weighted maximum likelihood estimation problem, weighted by $\hat{Q}$. We can get closed form updates (e.g., in the case of a mixture of Gaussians)



The EM algorithm (from a new perspective)

- Abstractly, the EM algorithm is an alternating maximization algorithm

- initialize θ
- **E-step (max wrt Q):** fix θ, store $\hat{Q}(z|x) = P(z|x,\theta)$ for all data points. This is equivalent to maximizing $ELBO(Q,\theta)$ with respect to Q
- **M-step (max wrt θ):** fix posterior weights $\hat{Q}$, update θ by maximizing $ELBO(\hat{Q},\theta)$. This is a weighted maximum likelihood estimation problem, weighted by $\hat{Q}$. We can get closed form updates (e.g., in the case of a mixture of Gaussians)

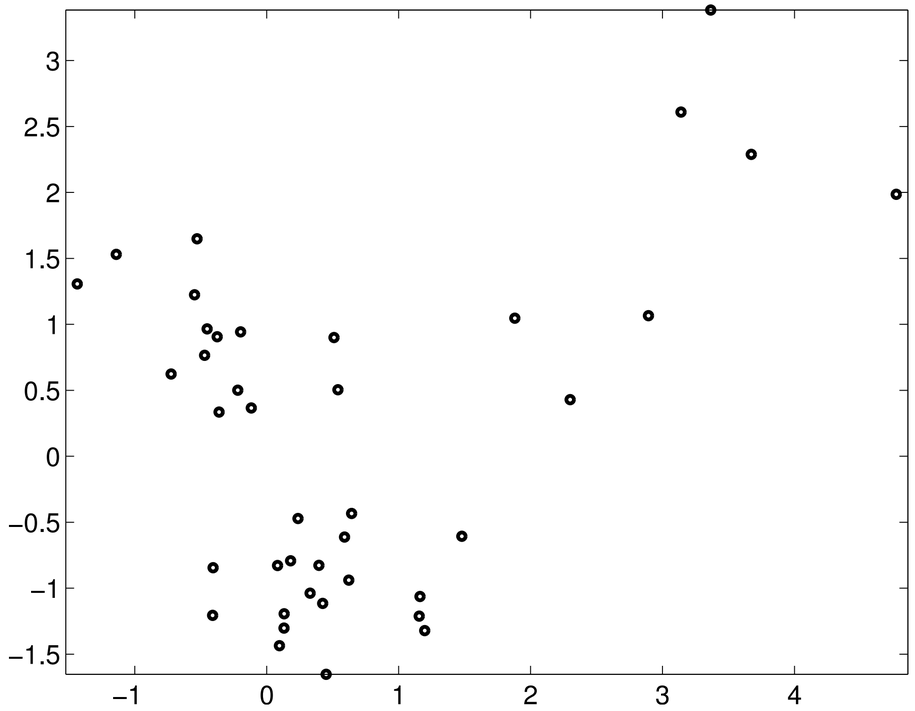
- A simple proof of monotone progress on the log-likelihood (for one x here):

$$\log P(x|\theta^{(0)}) = ELBO(Q^{(0)},\theta^{(0)}) \leq ELBO(Q^{(0)},\theta^{(1)}) \leq ELBO(Q^{(1)};\theta^{(1)}) = \log P(x|\theta^{(1)})$$

E-step M-step E-step

GMM example

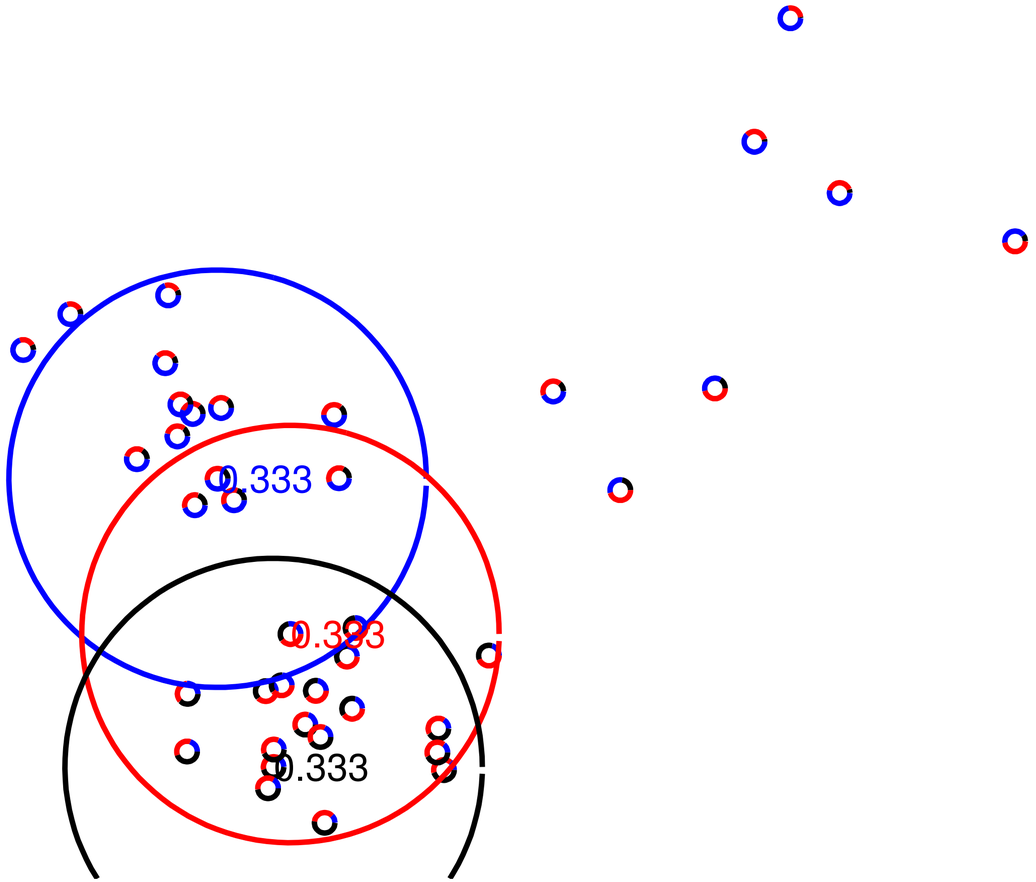
- A simple 3-component spherical GMM example



- Initialization matters:
  - uniform mixing proportions $\pi_j = 1/k, \;\; j = 1,\ldots,k$
  - means $\{\mu_j\}$ set to randomly selected points
  - variances $\{\sigma_j\}$ set to single Gaussian estimates (from all data)

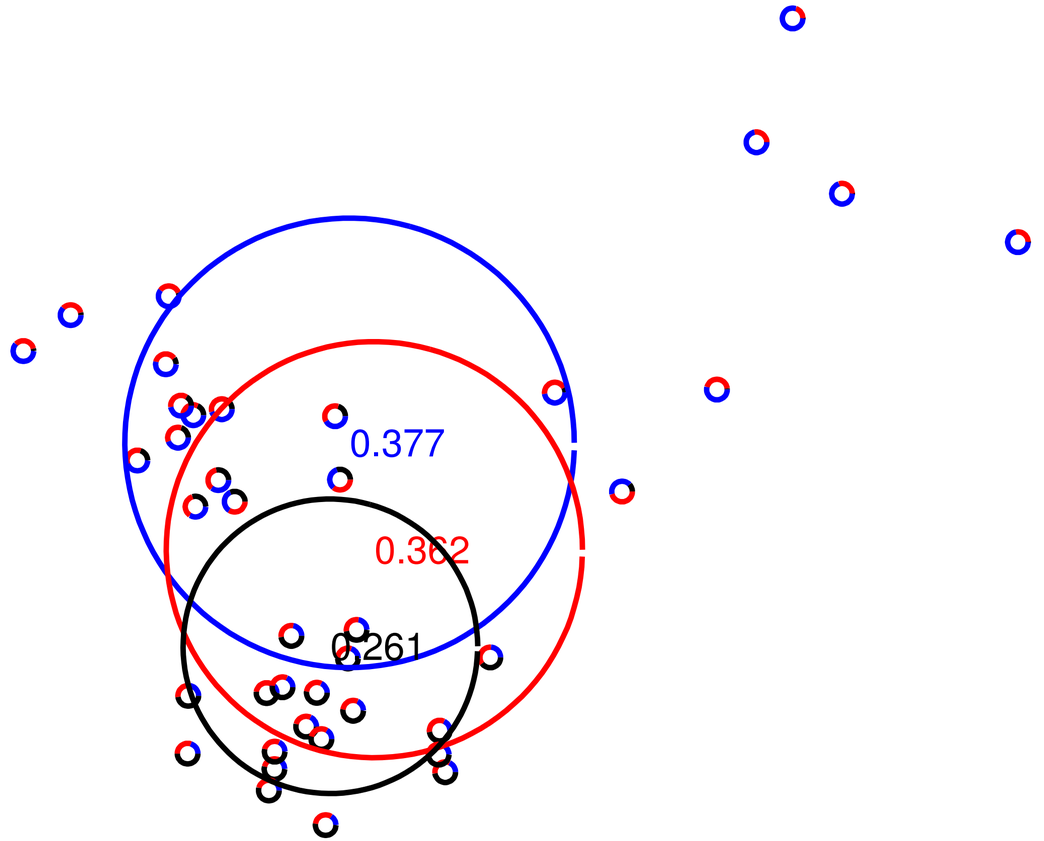
GMM example

- a 3-component GMM



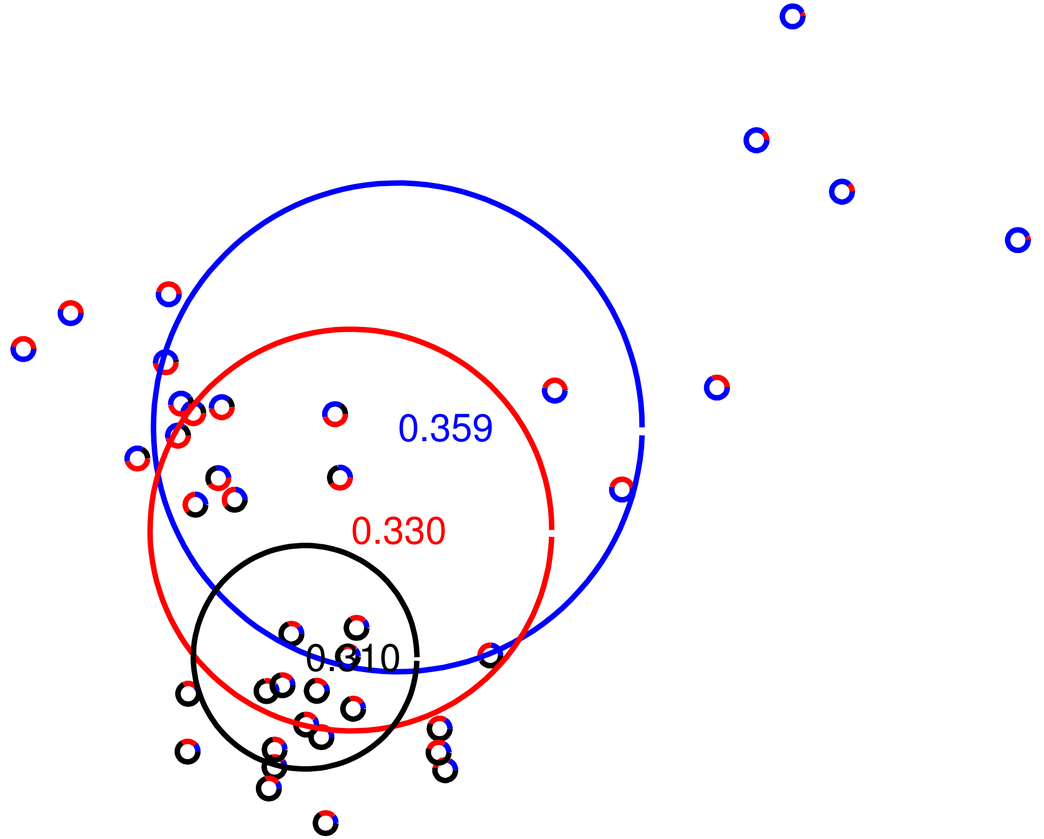
GMM example

- a 3-component GMM



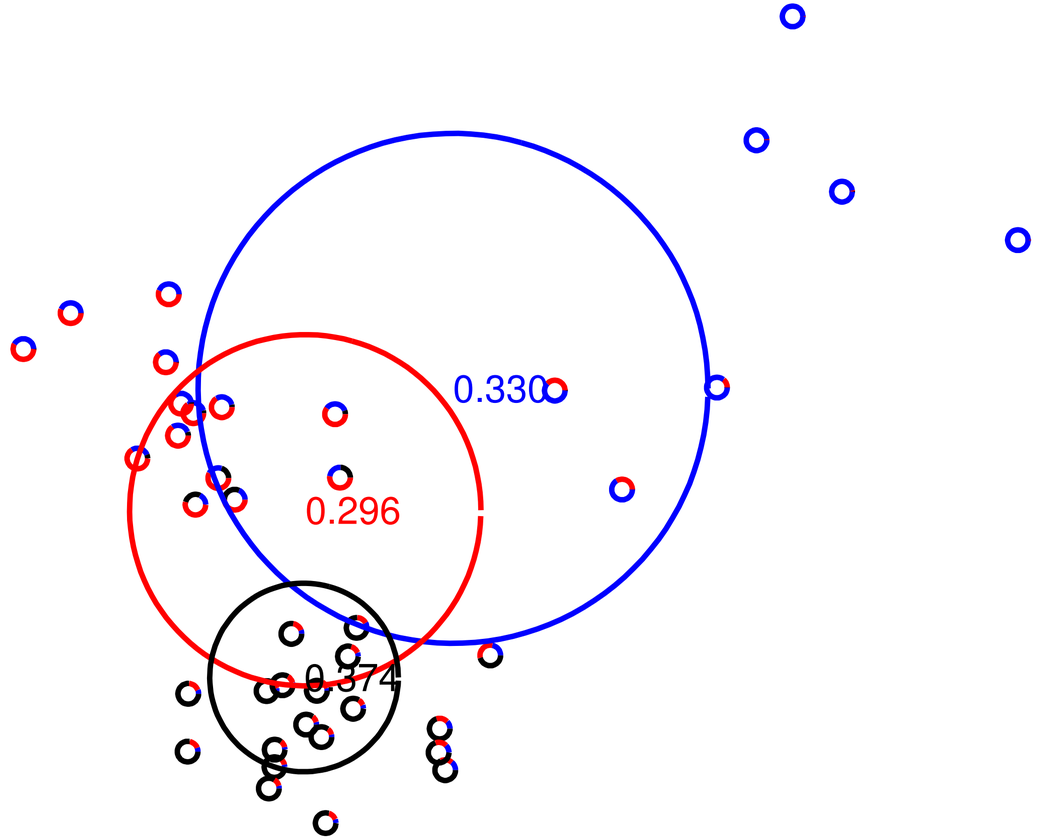
GMM example

- a 3-component GMM



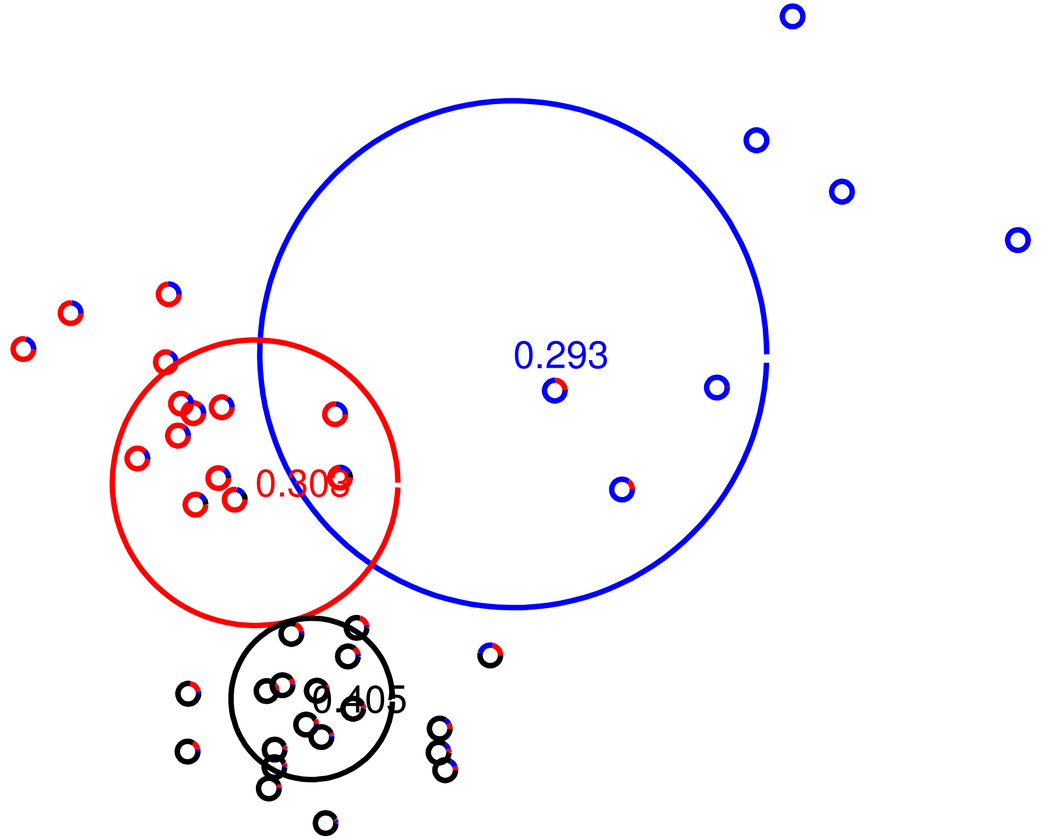
GMM example

- a 3-component GMM



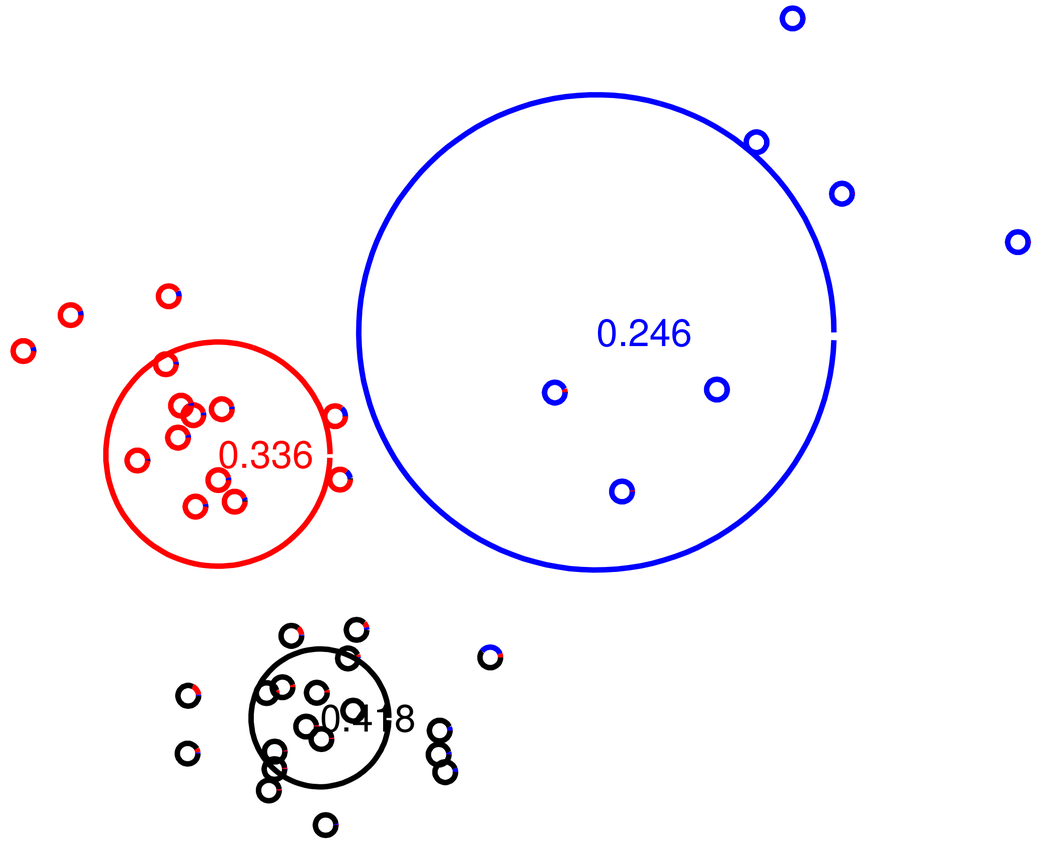
GMM example

- a 3-component GMM



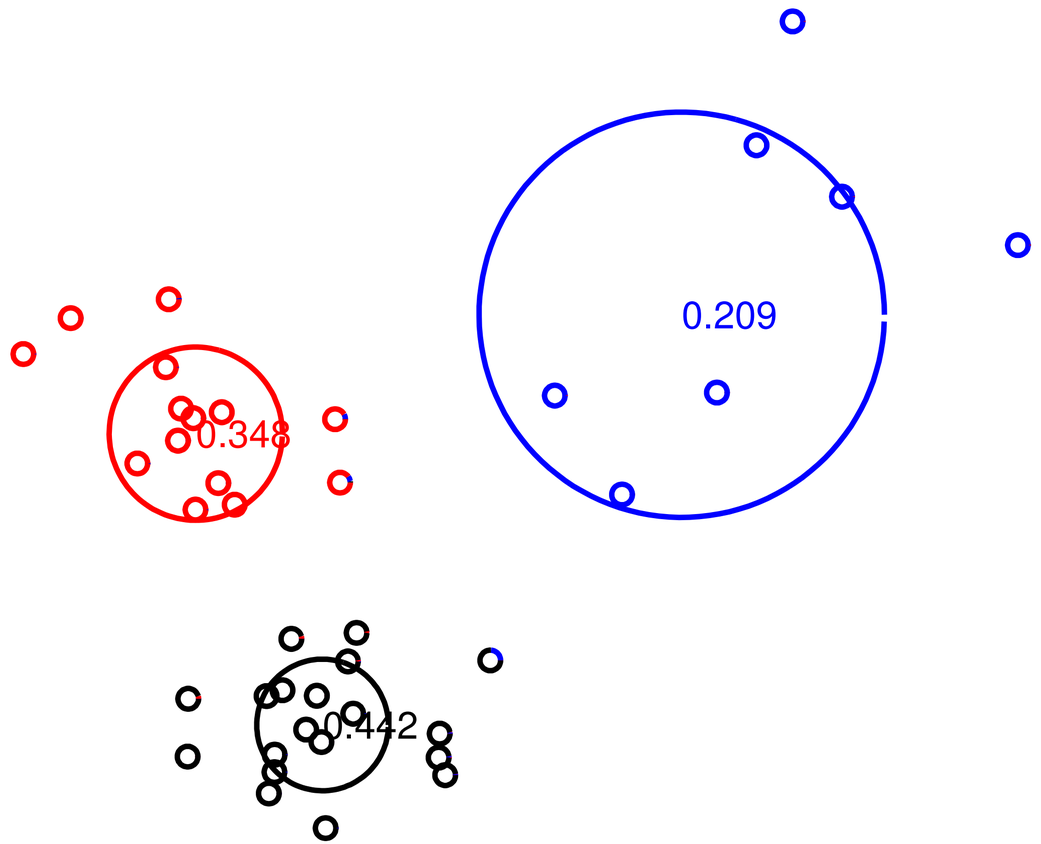
GMM example

- a 3-component GMM



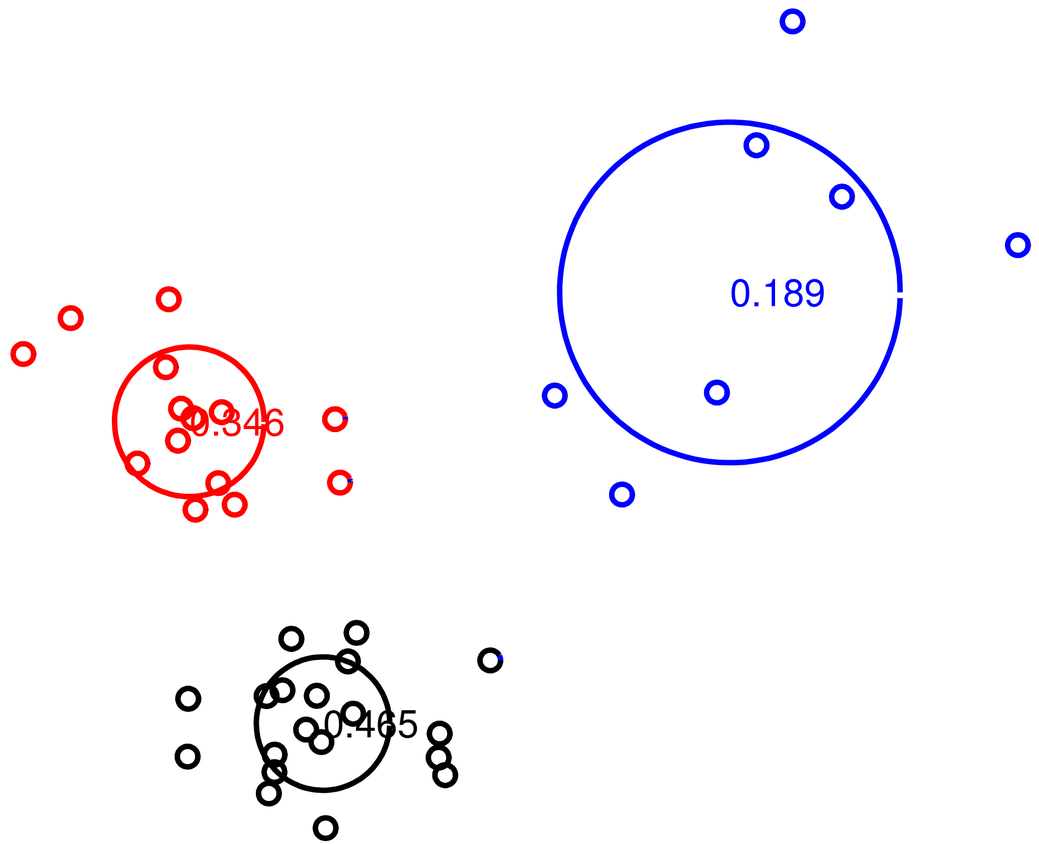
GMM example

- a 3-component GMM



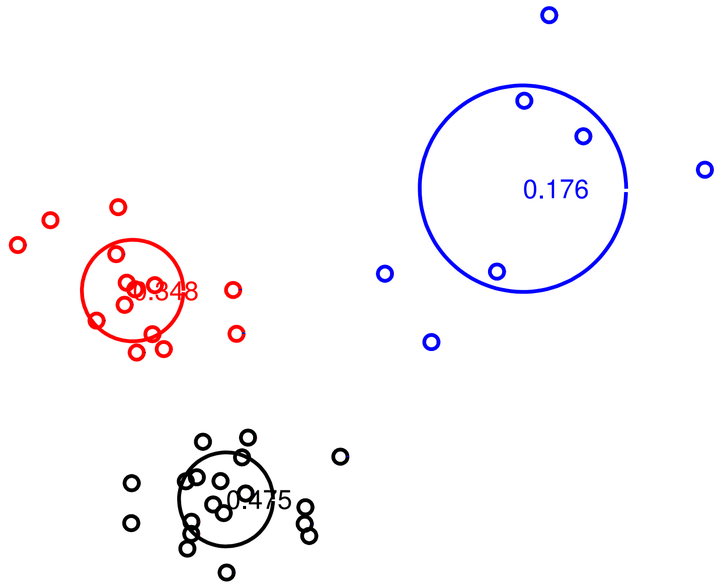
GMM example

- a 3-component GMM

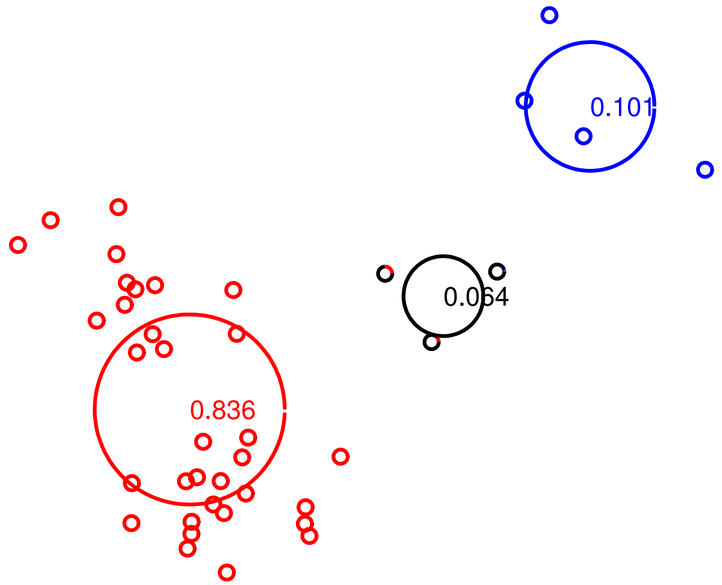


GMM example

- Different (random) initializations can lead to different solutions; EM only finds a locally optimal solution



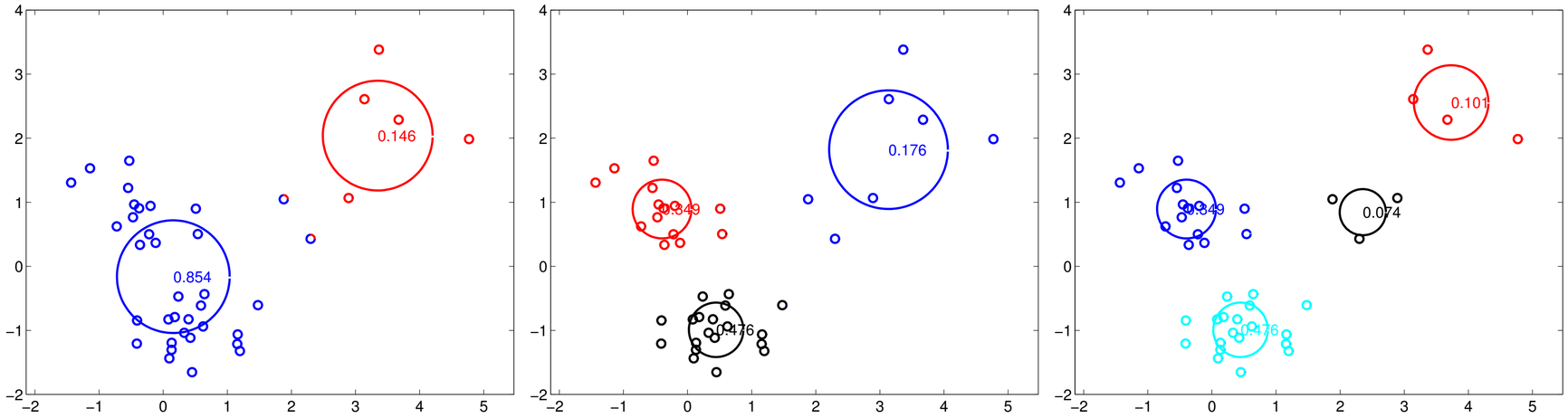
$$l(D;\hat{\theta}) = -98.64$$



$$l(D;\hat{\theta}) = -114.82$$

GMM solutions: varying k

- We can run the GMM with different numbers of components… which one should we choose?

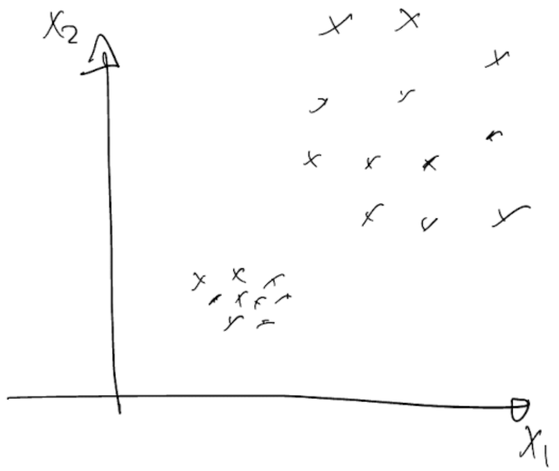


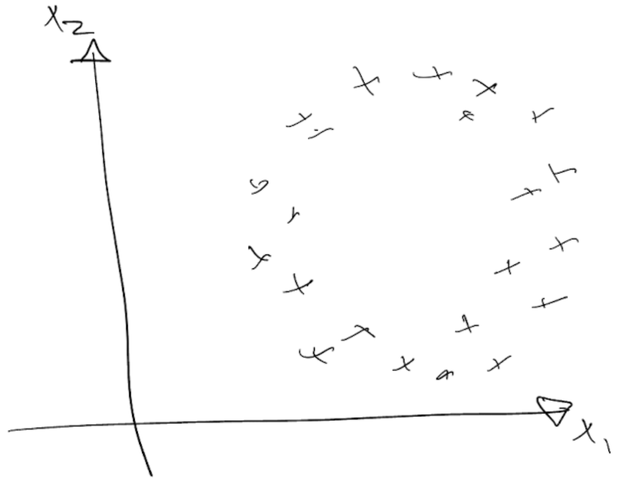
$$l(D;\hat{\theta}) = -118.25 \qquad l(D;\hat{\theta}) = -98.64 \qquad l(D;\hat{\theta}) = -94.11$$

Towards deep generative models, VAEs

Why latent variables?

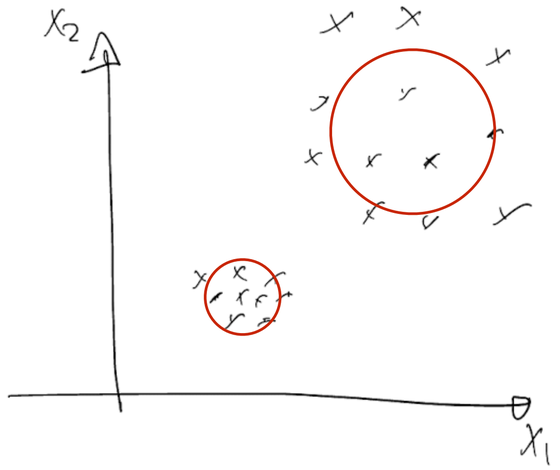
- Generative models specify a mechanism for creating objects such as text, images, molecules, designs, etc.
- Latent variable models try to identify (specify) some of the underlying choices involved in realizing such objects
- Latent variables may be discrete, continuous, as well as high dimensional (as in deep generative models)
- E.g., as simple examples consider data that looks like

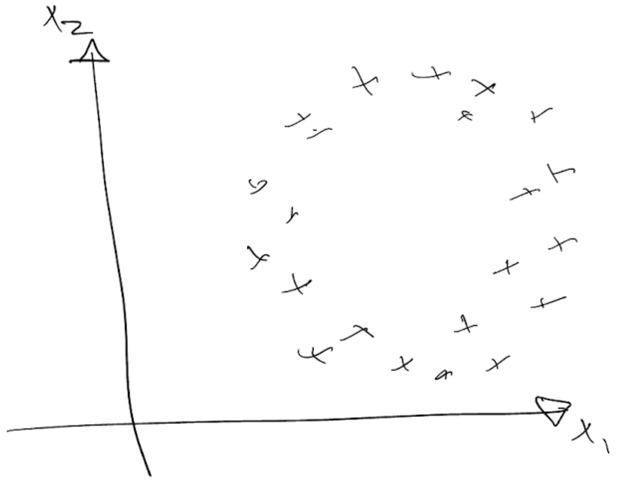




Latent mechanisms

- Data





- Model, parameters

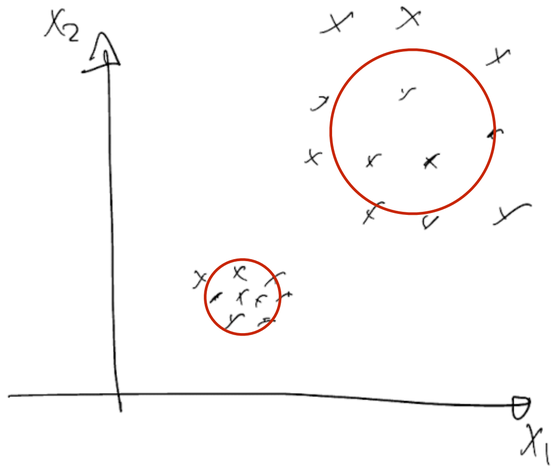
$$k = 2$$

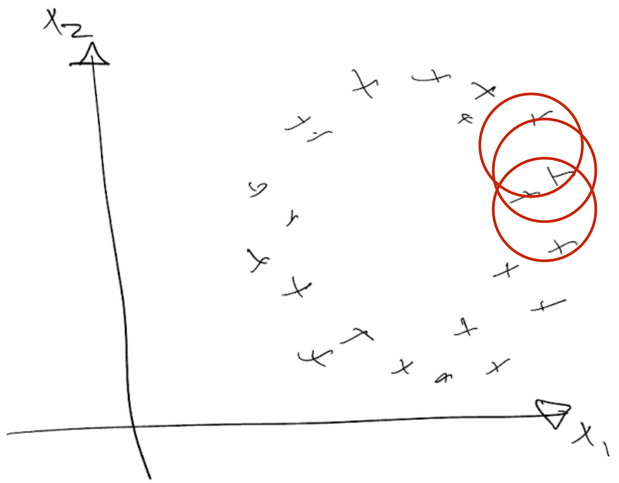
$$z \sim \text{Cat}(z; \pi_1, \ldots, \pi_k)$$

$$x \sim N(x \mid \mu_z, \sigma_z^2 I)$$

Latent mechanisms

- Data





- Model, parameters

$k = 2$

$z \sim \text{Cat}(z; \pi_1, \ldots, \pi_k)$

$x \sim N(x \mid \mu_z, \sigma_z^2 I)$

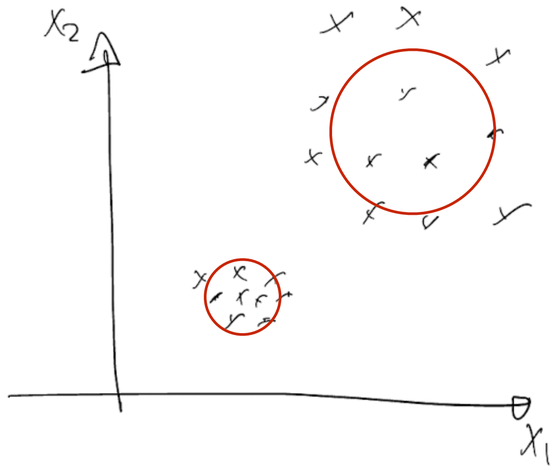
$k = \text{large}$

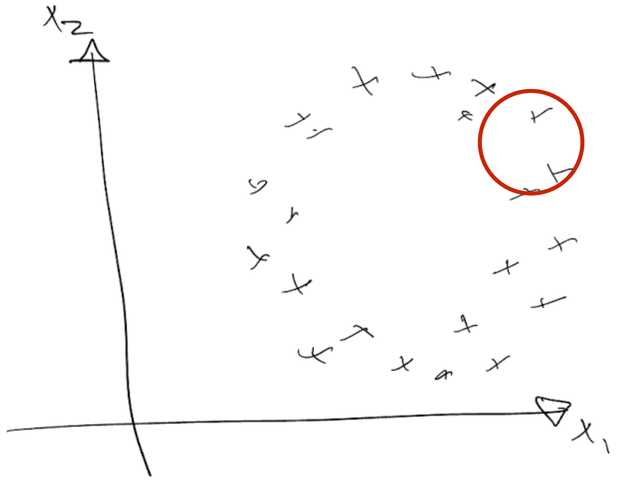
$z \sim \text{Cat}(z; \pi_1, \ldots, \pi_k)$

$x \sim N(x \mid \mu_z, \sigma_z^2 I)$

Latent mechanisms

- Data





- Model, parameters

$k = 2$

$z \sim \text{Cat}(z; \pi_1, \ldots, \pi_k)$

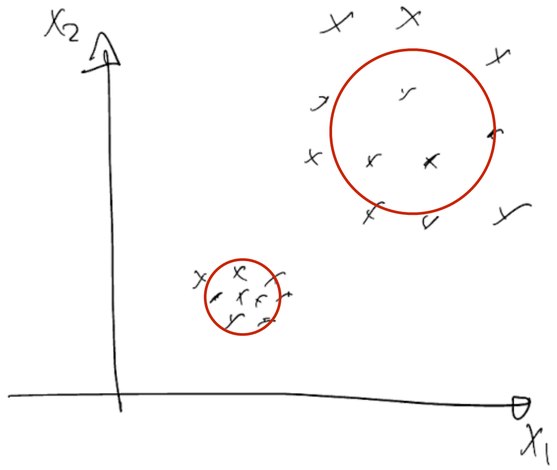
$x \sim N(x \mid \mu_z, \sigma_z^2 I)$

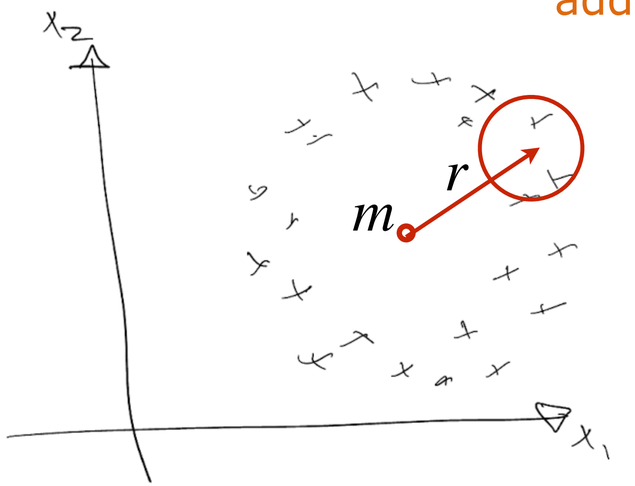
$z \sim N(0, I)$ (dim = ?)

$x \sim N(x \mid \mu_z, \sigma_z^2 I)$

Latent mechanisms

- Data



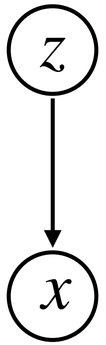


additional parameters

$$\mu_z = r\frac{z}{\|z\|} + m$$

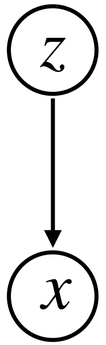
- Model, parameters

$k = 2$



$z \sim \text{Cat}(z; \pi_1, \ldots, \pi_k)$

$x \sim N(x \mid \mu_z, \sigma_z^2 I)$

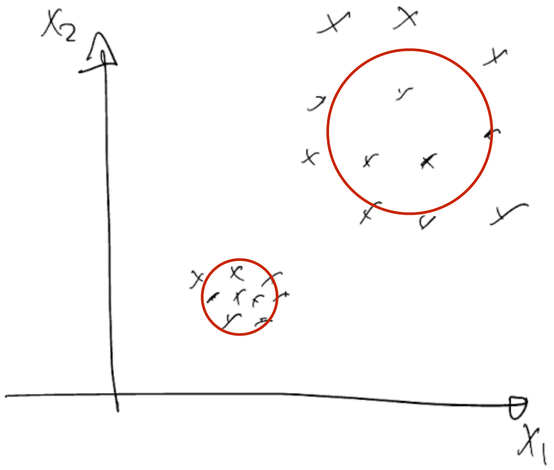


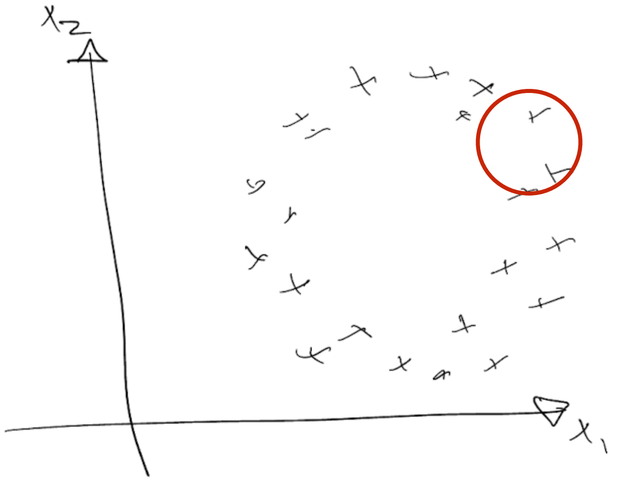
$z \sim N(0, I)$ (dim = 2)

$x \sim N(x \mid \mu_z, \sigma_z^2 I)$

Latent mechanisms

- Data





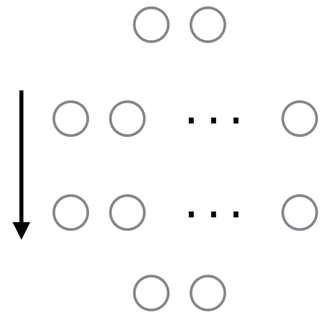
- Model, parameters

$k = 2$

$z \sim \text{Cat}(z; \pi_1, \ldots, \pi_k)$

$x \sim N(x \mid \mu_z, \sigma_z^2 I)$

$z \sim N(0, I)$ (dim = 2)



deep neural model

$x \sim N(x \mid g(z; \theta), \sigma^2 I)$

ELBO for latent variable models

- ELBO lower bound can be used to optimize any mixture (discrete or continuous)

$$\log P(x|\theta) \geq \int Q(z|x) \log \underbrace{\left[P(x|z,\theta)P(z)\right]}_{\text{network}} dz + H(Q_{z|x}) = ELBO(Q,\theta)$$

- **E-step:** infer likely z that generated x (i.e., complete the data)

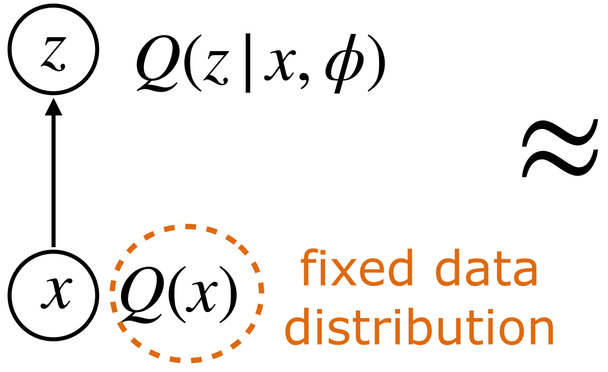
- **M-step:** update θ based on the complete data

ELBO for complex latent variable models

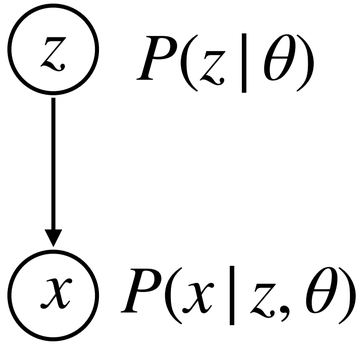
- ELBO lower bound can be used to optimize any mixture (discrete or continuous)

$$\log P(x\,|\,\theta) \geq \int \underbrace{Q(z\,|\,x,\phi)}_{\text{also a network}} \log \Big[\underbrace{P(x\,|\,z,\theta)}_{\text{network}} P(z)\Big]\, dz + H(Q_{z|x,\phi}) = ELBO(Q_\phi, \theta)$$

inferential model or "encoder"



$\approx$



generative model or "decoder"

ELBO for complex latent variable models

- ELBO lower bound can be used to optimize any mixture (discrete or continuous)

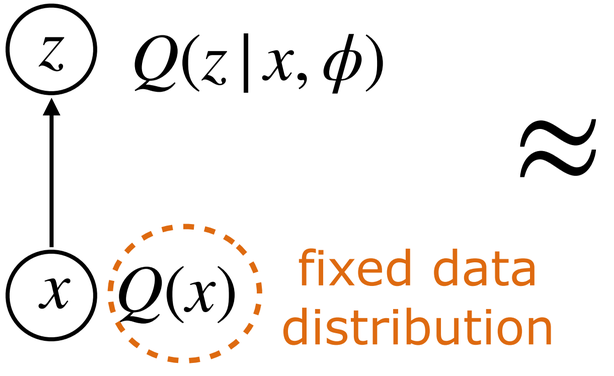
$$\log P(x|\theta) \geq \int Q(z|x,\phi)\log\left[P(x|z,\theta)P(z)\right]dz + H(Q_{z|x,\phi}) = ELBO(Q_\phi,\theta)$$

also a network — network

- **E-step:** update $Q(z|x,\phi)$ to approximate $P(z|x,\theta)$ by maximizing ELBO(Q,θ)

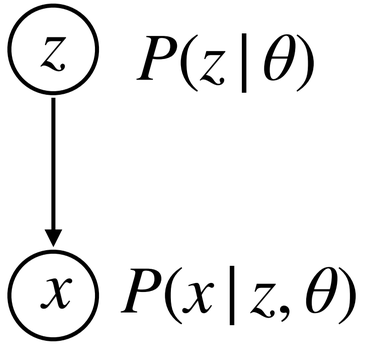
$$\phi \leftarrow \phi + \eta\nabla_\phi ELBO(Q_\phi,\theta)$$

inferential model or "encoder"



fixed data distribution

≈



generative model or "decoder"

ELBO for complex latent variable models

- ELBO lower bound can be used to optimize any mixture (discrete or continuous)

$$\log P(x|\theta) \geq \int \underbrace{Q(z|x,\phi)}_{\text{also a network}} \log \left[\underbrace{P(x|z,\theta)}_{\text{network}} P(z)\right] dz + H(Q_{z|x,\phi}) = ELBO(Q_\phi, \theta)$$

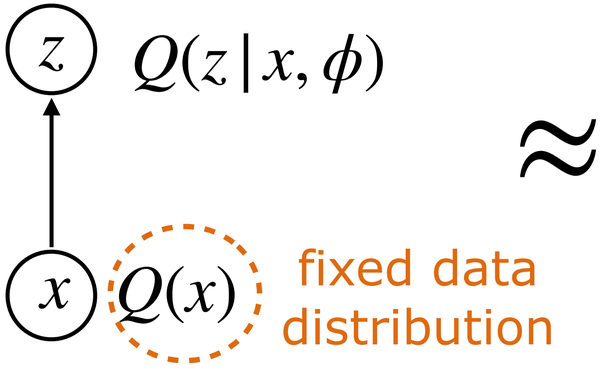
- **E-step:** update $Q(z|x,\phi)$ to approximate $P(z|x,\theta)$ by maximizing ELBO(Q,θ)

$$\phi \leftarrow \phi + \eta \nabla_\phi ELBO(Q_\phi, \theta)$$

- **M-step:** update $P(x|z,\theta)$ to maximize (increase) ELBO(Q,θ) for fixed $Q(z|x,\phi)$

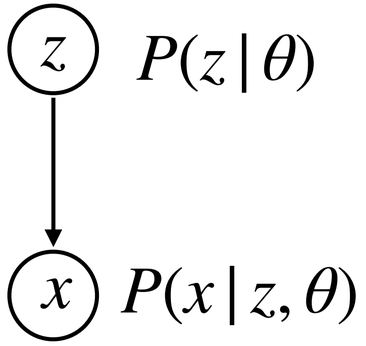
$$\theta \leftarrow \theta + \eta \nabla_\theta ELBO(Q_\phi, \theta)$$

inferential model or "encoder"



fixed data distribution

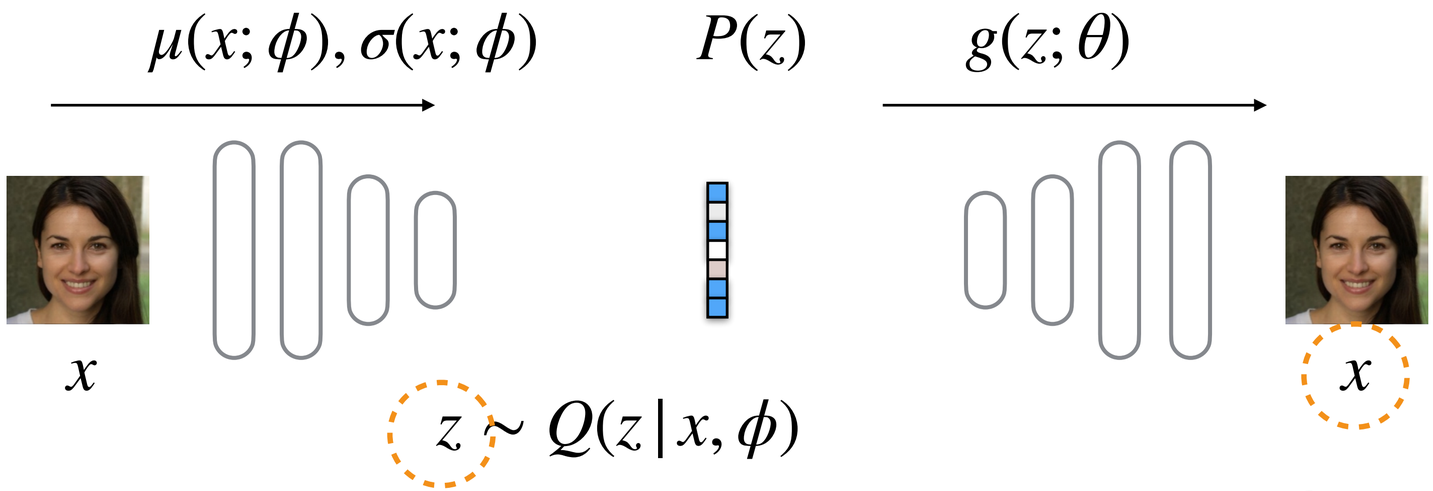
$\approx$



generative model or "decoder"

Variational autoencoders (VAEs)

- Generative model/decoder:
  - sample $z \sim P(z)$ from a simple, fixed distribution, e.g., $N(0,I)$
  - map the resulting z through a deep model $x = g(z;\theta)$
  - observed images assumed to be noisy versions, i.e., $P(x|z;\theta) = N(x; g(z;\theta), \sigma^2 I)$

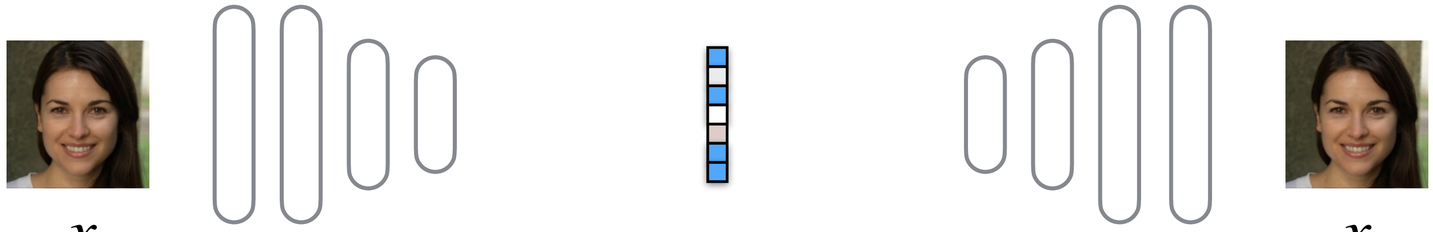


$$z \sim Q(z|x,\phi)$$
$$= N(z; \mu(x;\phi), \text{diag}(\sigma(x;\phi)^2))$$

- Inference model/encoder:
  - we use another network (encoder) to infer what z was for any given x.
  - this encoder network predicts, e.g., the mean and stdv of each coordinate of z, conditioned on x (this is an approximation to the posterior)
- The two networks are learned together.

Learning VAEs

$\mu(x;\phi), \sigma(x;\phi)$ $P(z)$ $g(z;\theta)$ $P(x \mid z;\theta) = N(x; g(z;\theta), \sigma^2 I)$



x x

$$z \sim Q(z \mid x, \phi)$$

$$= N(z; \mu(x;\phi), \text{diag}(\sigma(x;\phi)^2))$$

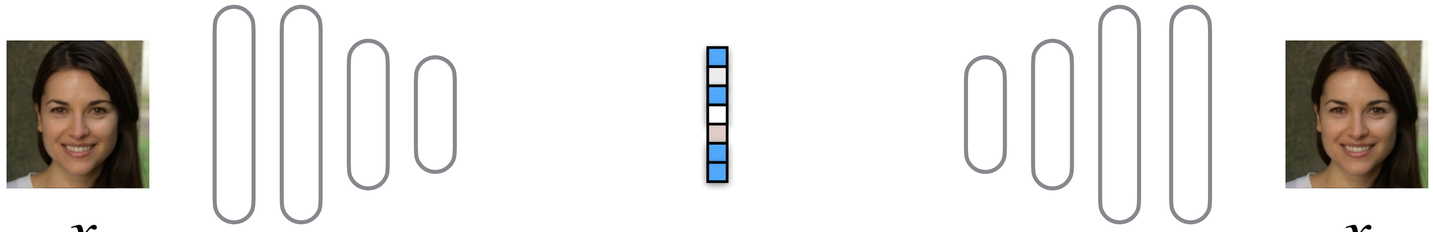
- The encoder and generator are both learned via stochastic gradient ascent steps on the lower bound objective (ELBO)

$$\log \left[\int P(x \mid z, \theta) P(z) dz \right] \geq \int Q(z \mid x, \phi) \log \left[P(x \mid z, \theta) P(z) \right] dz + H(Q_{z \mid x, \phi})$$

$$= ELBO(Q_\phi; \theta)$$

Learning VAEs

$\mu(x;\phi), \sigma(x;\phi)$ $P(z)$ $g(z;\theta)$ $P(x \mid z;\theta) = N(z; g(z;\theta), \sigma^2 I)$



x x

$$z \sim Q(z \mid x, \phi)$$

$$= N(z; \mu(x;\phi), \text{diag}(\sigma(x;\phi)^2))$$

- The encoder and generator are both learned via stochastic gradient ascent steps on the lower bound objective (ELBO)

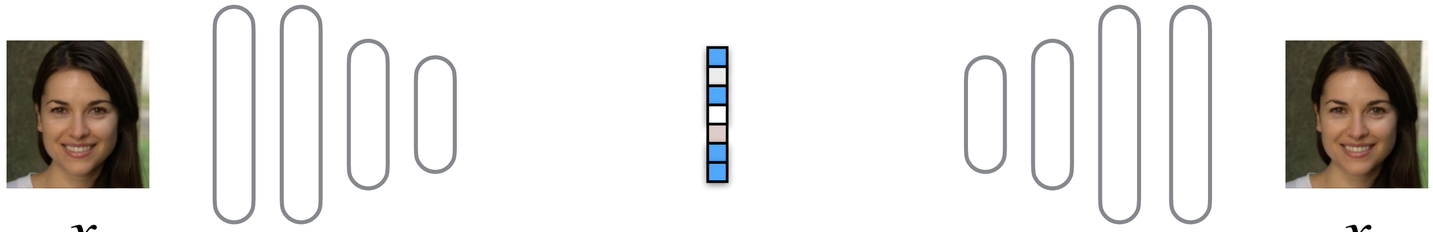
$$\log \left[\int P(x \mid z, \theta) P(z) dz \right] \geq \int Q(z \mid x, \phi) \log \left[P(x \mid z, \theta) P(z) \right] dz + H(Q_{z \mid x, \phi})$$

$$= ELBO(Q_\phi; \theta)$$

$$-KL(Q_{z \mid x;\phi} \mid P_z)$$

Learning VAEs

$\mu(x;\phi), \sigma(x;\phi)$ $P(z)$ $g(z;\theta)$ $P(x \mid z;\theta) = N(z; g(z;\theta), \sigma^2 I)$



x x

$$z \sim Q(z \mid x, \phi)$$

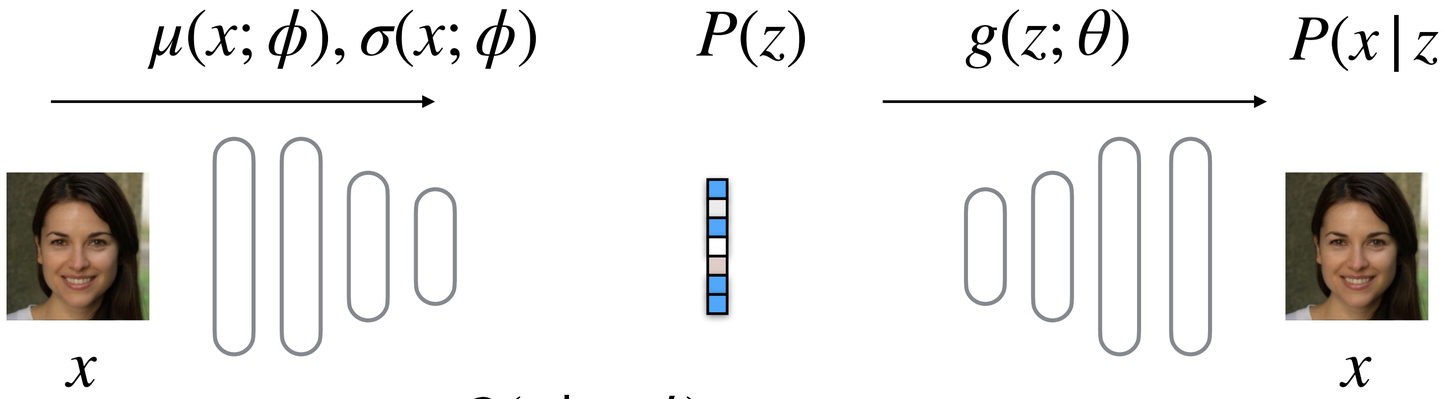
$$= N(z; \mu(x;\phi), \text{diag}(\sigma(x;\phi)^2))$$

- The encoder and generator are both learned via stochastic gradient ascent steps on the lower bound objective (ELBO)

$$\log \left[\int P(x \mid z, \theta) P(z) dz \right] \geq \int Q(z \mid x, \phi) \log \left[P(x \mid z, \theta) \right] dz - KL(Q_{z \mid x;\phi} \| P_z)$$

$$= ELBO(Q_\phi; \theta)$$

Learning VAEs



$$z \sim Q(z \mid x, \phi)$$

$$= N(z; \mu(x;\phi), \text{diag}(\sigma(x;\phi)^2))$$

- The encoder and generator are both learned via stochastic gradient ascent steps on a lower bound objective (ELBO)

need to sample — can calc exactly

$$\log \left[\int P(x \mid z, \theta) P(z) dz \right] \geq \int Q(z \mid x, \phi) \log \left[P(x \mid z, \theta) \right] dz - KL(Q_{z|x;\phi} \| P_z)$$

$$= ELBO(Q_\phi; \theta)$$

Reparameterization

- For continuous, high dimensional z, we cannot evaluate the reconstruction part of the ELBO criterion

$$\int Q(z \mid x, \phi) \log P(x \mid z, \theta) dz = E_{z \sim Q_{z|x,\phi}} \left\{ \log P(x \mid z, \theta) \right\} \approx \log P(x \mid \hat{z}, \theta)$$

- We can sample z's from Q to approximate it but such samples would not retain any dependence on ϕ, making it hard to optimize the encoder

Reparameterization

- For continuous, high dimensional z, we cannot evaluate the reconstruction part of the ELBO criterion

$$\int Q(z \mid x, \phi) \log P(x \mid z, \theta) dz = E_{z \sim Q_{z|x,\phi}} \left\{ \log P(x \mid z, \theta) \right\} \approx \log P(x \mid \hat{z}, \theta)$$

- We can sample z's from Q to approximate it but such samples would not retain any dependence on ϕ, making it hard to optimize the encoder
- Instead, we reparameterize $z_\phi(x, \epsilon) = \mu(x; \phi) + \sigma(x; \phi) \odot \epsilon$ where $\epsilon \sim N(0,I)$. Sampling a z from Q is now equivalent to sampling $\epsilon \sim N(0,I)$ and calculating $z_\phi(x, \epsilon)$

Reparameterization

- For continuous, high dimensional z, we cannot evaluate the reconstruction part of the ELBO criterion

$$\int Q(z \mid x, \phi) \log P(x \mid z, \theta) dz = E_{z \sim Q_{z|x,\phi}} \left\{ \log P(x \mid z, \theta) \right\} \approx \log P(x \mid \hat{z}, \theta)$$

- We can sample z's from Q to approximate it but such samples would not retain any dependence on ϕ, making it hard to optimize the encoder
- Instead, we reparameterize $z_\phi(x, \epsilon) = \mu(x; \phi) + \sigma(x; \phi) \odot \epsilon$ where $\epsilon \sim N(0,I)$. Sampling a z from Q is now equivalent to sampling $\epsilon \sim N(0,I)$ and calculating $z_\phi(x, \epsilon)$
- As a result,

$$E_{z \sim Q_{z|x,\phi}} \left\{ \log P(x \mid z, \theta) \right\} = E_{\epsilon \sim N(0,I)} \left\{ \log P(x \mid z_\phi(x, \epsilon), \theta) \right\}$$

Reparameterization

- For continuous, high dimensional z, we cannot evaluate the reconstruction part of the ELBO criterion

$$\int Q(z \mid x, \phi) \log P(x \mid z, \theta) dz = E_{z \sim Q_{z|x,\phi}} \left\{ \log P(x \mid z, \theta) \right\} \approx \log P(x \mid \hat{z}, \theta)$$

- We can sample z's from Q to approximate it but such samples would not retain any dependence on ϕ, making it hard to optimize the encoder
- Instead, we reparameterize $z_\phi(x, \epsilon) = \mu(x; \phi) + \sigma(x; \phi) \odot \epsilon$ where $\epsilon \sim N(0,I)$. Sampling a z from Q is now equivalent to sampling $\epsilon \sim N(0,I)$ and calculating $z_\phi(x, \epsilon)$
- As a result,

$$E_{z \sim Q_{z|x,\phi}} \left\{ \log P(x \mid z, \theta) \right\} = E_{\epsilon \sim N(0,I)} \left\{ \log P(x \mid z_\phi(x, \epsilon), \theta) \right\} \approx \log P(x \mid z_\phi(x, \hat{\epsilon}), \theta)$$

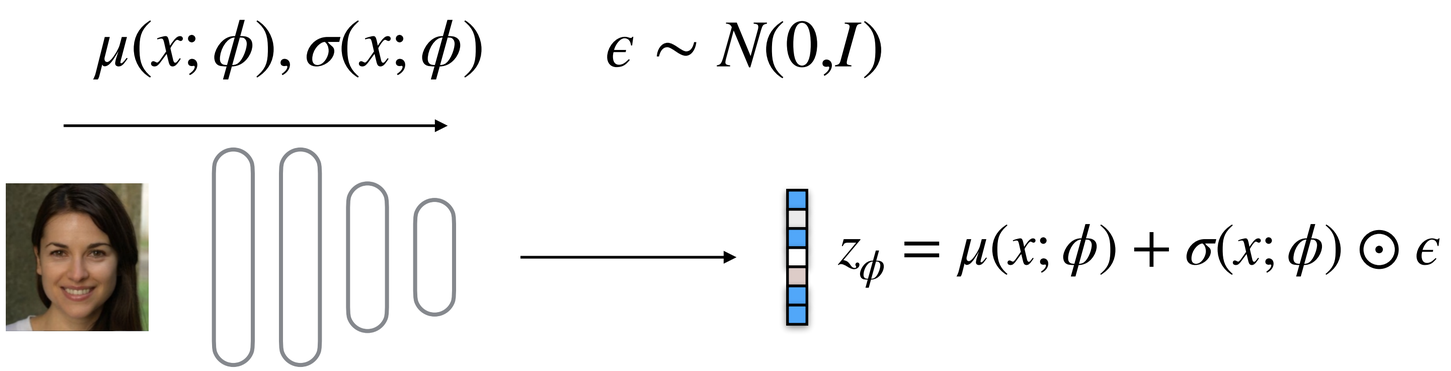
- where even a single sample approximation to the expectation retains its dependence on ϕ and allows us to calculate gradients to optimize Q as well

VAE update steps

- (1) Given x, pass x through the encoder to get $\mu(x;\phi), \sigma(x;\phi)$ that specify $Q(z|x,\phi)$

VAE update steps

- (1) Given x, pass x through the encoder to get $\mu(x;\phi), \sigma(x;\phi)$ that specify $Q(z|x,\phi)$
- (2) Sample $\epsilon \sim N(0,I)$ so that $z_\phi = \mu(x;\phi) + \sigma(x;\phi) \odot \epsilon$ corresponds to a sample from the encoder distribution $Q(z|x,\phi)$ (reparameterization)



$$z_\phi \sim N(z;\mu(x;\phi), \text{diag}(\sigma(x;\phi)^2))$$

VAE update steps

- (1) Given x, pass x through the encoder to get $\mu(x;\phi), \sigma(x;\phi)$ that specify $Q(z|x,\phi)$
- (2) Sample $\epsilon \sim N(0,I)$ so that $z_\phi = \mu(x;\phi) + \sigma(x;\phi) \odot \epsilon$ corresponds to a sample from the encoder distribution $Q(z|x,\phi)$ (reparameterization)
- (3: **E-step**) update the encoder to max one-sample stochastic ELBO criterion

$$\phi = \phi + \eta \nabla_\phi \left\{ \log P(x|z_\phi,\theta) - KL(Q_{z|x;\phi} \| P_z) \right\}$$

VAE update steps

- (1) Given x, pass x through the encoder to get $\mu(x;\phi), \sigma(x;\phi)$ that specify $Q(z|x,\phi)$
- (2) Sample $\epsilon \sim N(0,I)$ so that $z_\phi = \mu(x;\phi) + \sigma(x;\phi) \odot \epsilon$ corresponds to a sample from the encoder distribution $Q(z|x,\phi)$ (reparameterization)
- (3: **E-step**) update the encoder to max one-sample stochastic ELBO criterion

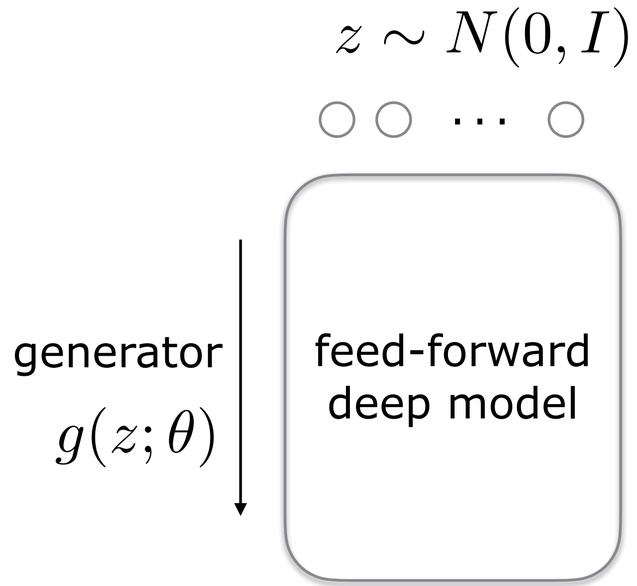
$$\phi = \phi + \eta \nabla_\phi \left\{ \log P(x|z_\phi,\theta) - KL(Q_{z|x;\phi} \| P_z) \right\}$$

- (4: **M-step**) update the generator (decoder) to max the stochastic ELBO

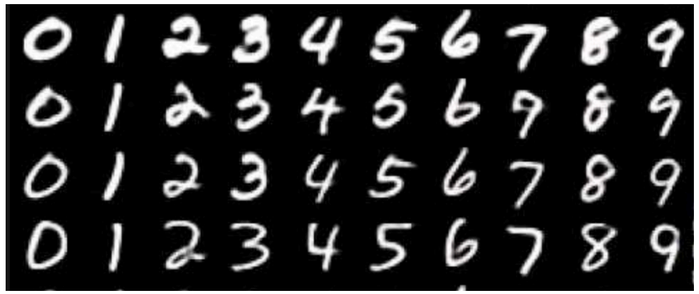
$$\theta = \theta + \eta \nabla_\theta \log P(x|z_\phi,\theta)$$

Variational autoencoder (VAE)

- Once trained, we can easily sample from the generative model



training set

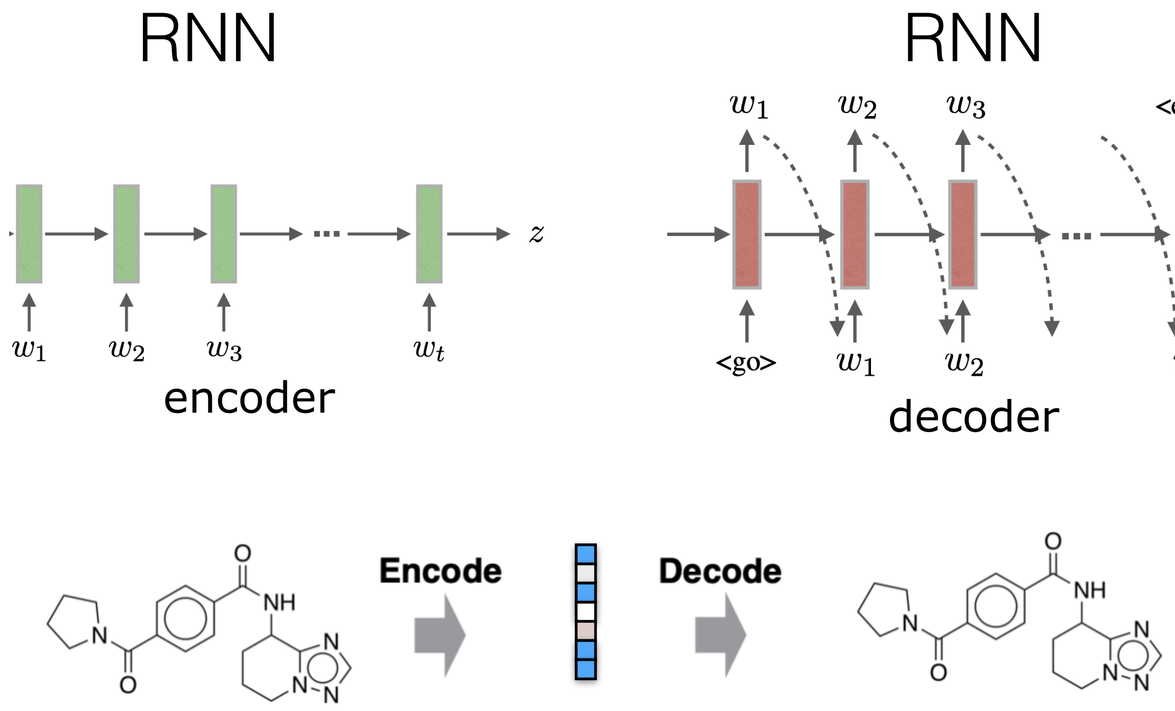


...

...

Variational autoencoder (VAE)

- VAEs are generally useful for latent variable models, not limited to images



- VAEs are useful not only for learning generative models but they also allow us to learn good representations (encodings) of complex examples (= non-linear dimensionality reduction)

VAEs: what you need to know

- How deep generative models realize images or other objects from latent randomization
- Optimization of the architecture would be easy (why?) if we had the latent random vector z together with the image as pairs
- EM is not tractable for these complex, implicit distribution problems; we need to use another encoder network to predict the result of the E-step (infer latent z)
- Variational auto-encoder uses an encoder network to infer z from x, and the generator maps this (random) z back to x
- VAE architecture is learned by maximizing a (stochastic) lower bound on the log-likelihood of the data (x samples) via the ELBO criterion
- Also important to understand why re-parameterization is needed