Introduction to Data Mining

CS 145

Lecture 9:

Unsupervised Learning: EM Algorithm & VAE

Discriminative Model

Generative Model

Discriminative Model:

Learn the probability distribution $p(y|x)$

Generative Model:

Learn the probability distribution $p(x)$

Conditional Generative Model:

Learn the probability distribution $p(x|y)$

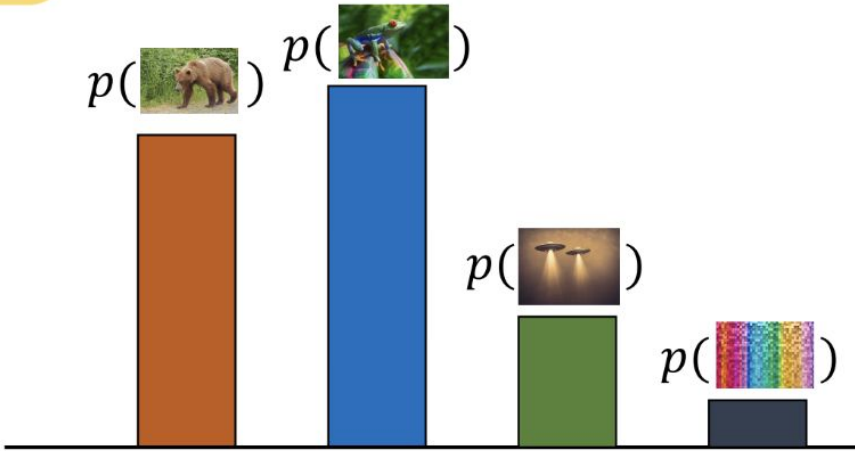
Generative Model

Generative Model:

Learn the probability distribution $p(x)$

Images compete with each other for probability mass.

The model needs to understand and represent the space of natural images. The model assigns small values to inputs that are “unreasonable”.

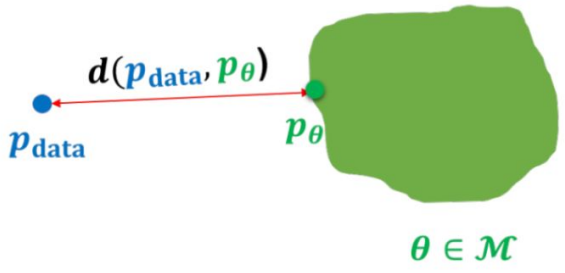


Generative Models

We are interested in parametric approximations to the data distribution p_{data} which summarize all the information about $\mathcal{D}$.

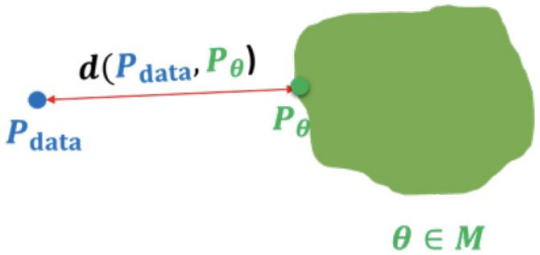
Our goal is to learn the parameters θ of a generative model within a model family $\mathcal{M}$ such that p_θ is close to p_{data}.

$\mathbf{x}^{(j)} \sim p_{\text{data}}$
$j = 1, 2, \ldots, |\mathcal{D}|$



- We are given a training set of examples, e.g., images of dogs

$x_i \sim P_{\text{data}}$
$i = 1, 2, \ldots, n$



- We want to learn a probability distribution $p(x)$ over images x such that
  - **Generation:** If we sample $x_{new} \sim p(x)$, x_{new} should look like a dog (*sampling*)
  - **Density estimation:** $p(x)$ should be high if x looks like a dog, and low otherwise (*anomaly detection*)
  - **Unsupervised representation learning:** We should be able to learn what these images have in common, e.g., ears, tail, etc. (*features*)
- First question: how to represent $p_\theta(x)$. Second question: **how to learn it**.

Maximum Likelihood Learning

- Approximate the expected log-likelihood

$$\mathbf{E}_{\mathbf{x}\sim P_{\text{data}}}\left[\log P_\theta(\mathbf{x})\right]$$

  with the *empirical log-likelihood*:

$$\mathbf{E}_{\mathcal{D}}\left[\log P_\theta(\mathbf{x})\right] = \frac{1}{|\mathcal{D}|}\sum_{\mathbf{x}\in\mathcal{D}} \log P_\theta(\mathbf{x})$$

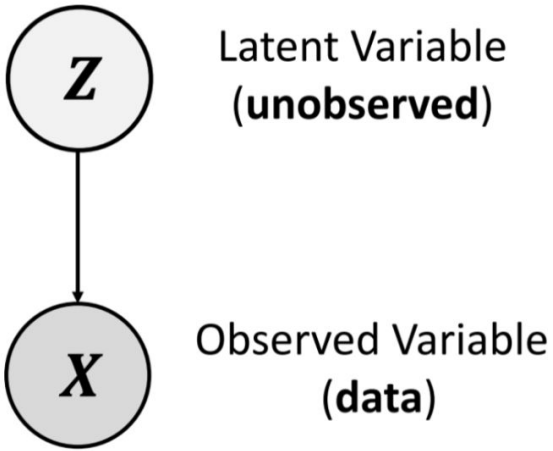
- **Maximum likelihood learning** is then:

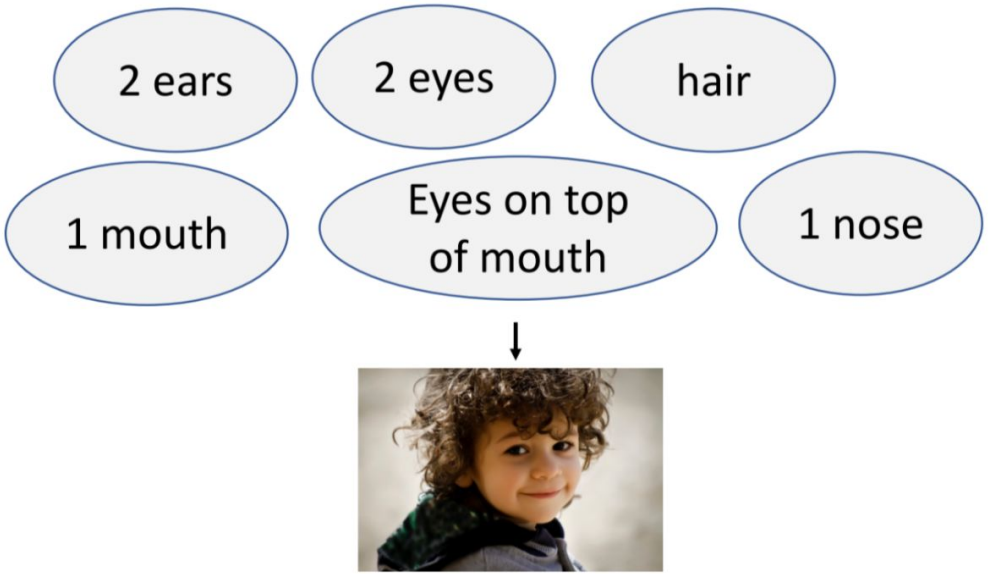
$$\max_{P_\theta} \frac{1}{|\mathcal{D}|}\sum_{\mathbf{x}\in\mathcal{D}} \log P_\theta(\mathbf{x})$$

- Equivalently, maximize likelihood of the data $P_\theta(\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)}) = \prod_{\mathbf{x}\in\mathcal{D}} P_\theta(\mathbf{x})$

Latent Variable Model

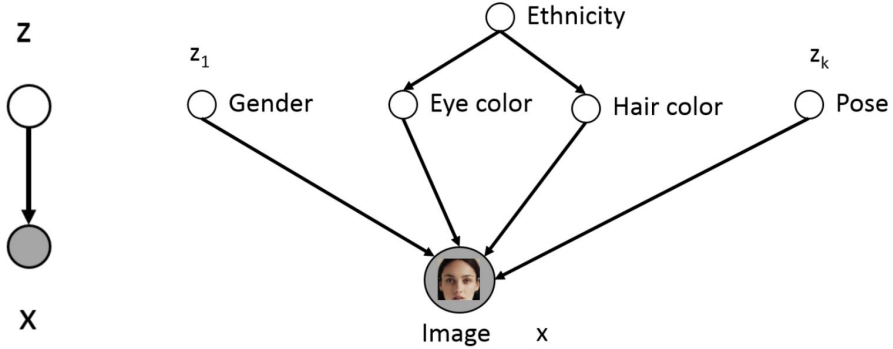
- Recall: variables which are always unobserved are called **latent variables** or sometimes hidden variables





Latent Variable Model

- Recall: variables which are always unobserved are called **latent variables** or sometimes hidden variables



1. Only shaded variables $\mathbf{x}$ are observed in the data (pixel values)
2. Latent variables $\mathbf{z}$ correspond to high level features
   - If $\mathbf{z}$ chosen properly, $p(\mathbf{x}|\mathbf{z})$ could be much simpler than $p(\mathbf{x})$
   - If we had trained this model, then we could identify features via $p(\mathbf{z} \mid \mathbf{x})$, e.g., $p(EyeColor = Blue|\mathbf{x})$
3. **Challenge:** Very difficult to specify these conditionals by hand

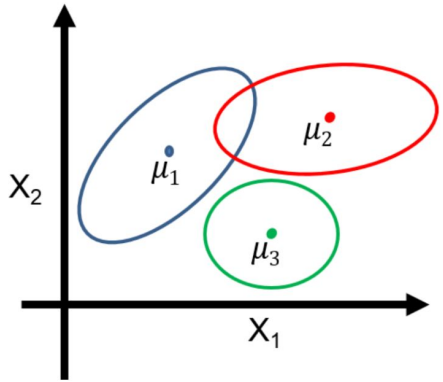
Latent Variable Model

- Recall: variables which are always unobserved are called **latent variables** or sometimes hidden variables
- In a mixture model, the identity of the component that generated a given datapoint is a latent variable
- Why use latent variables if introducing them complicates learning?
  - We can build a complex model out of simple parts - this can simplify the description of the model
  - We can sometimes use the latent variables as a representation of the original data (e.g. cluster assignments in a GMM model)

GMM as Latent Variable Model

Mixture of Gaussians. Bayes net: $\mathbf{z} \rightarrow \mathbf{x}$.

1. $\mathbf{z} \sim \text{Categorical}(1, \cdots, K)$
2. $p(\mathbf{x} \mid \mathbf{z} = k) = \mathcal{N}(\mu_k, \Sigma_k)$



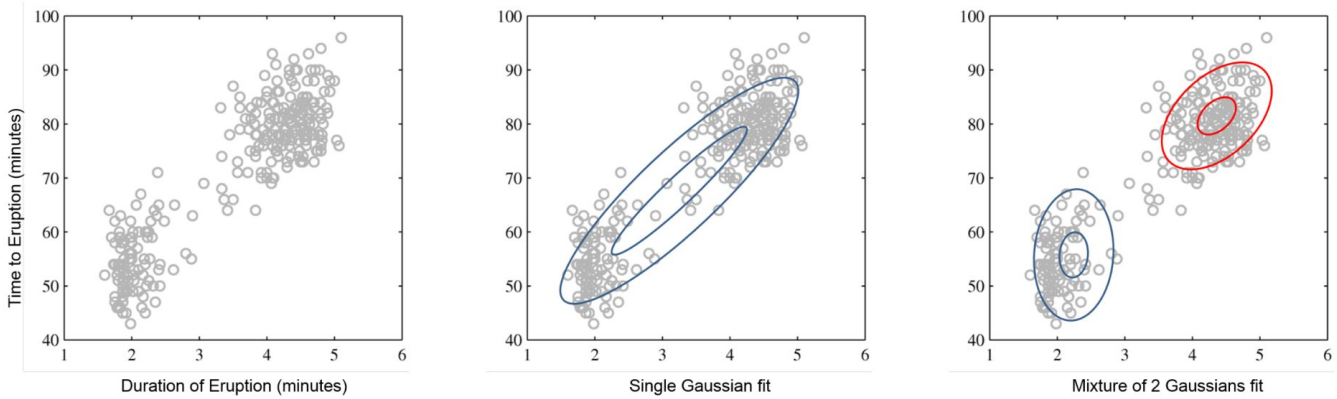
Generative process

1. Pick a mixture component k by sampling z
2. Generate a data point by sampling from that Gaussian

GMM as Latent Variable Model

Mixture of Gaussians:

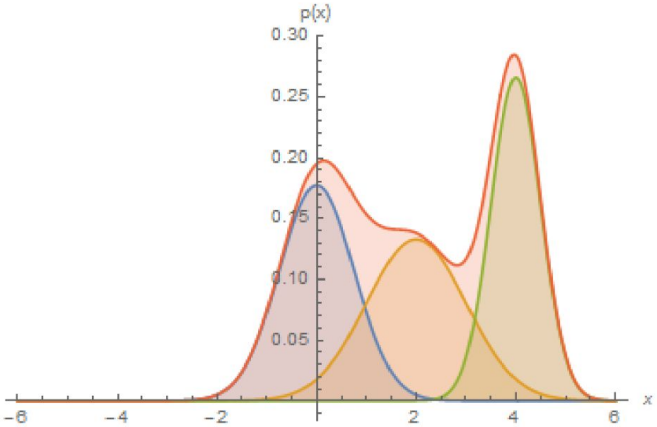
1. $\mathbf{z} \sim \text{Categorical}(1, \cdots, K)$
2. $p(\mathbf{x} \mid \mathbf{z} = k) = \mathcal{N}(\mu_k, \Sigma_k)$



- **Clustering:** The posterior $p(\mathbf{z} \mid \mathbf{x})$ identifies the mixture component
- **Unsupervised learning:** We are hoping to learn from unlabeled data

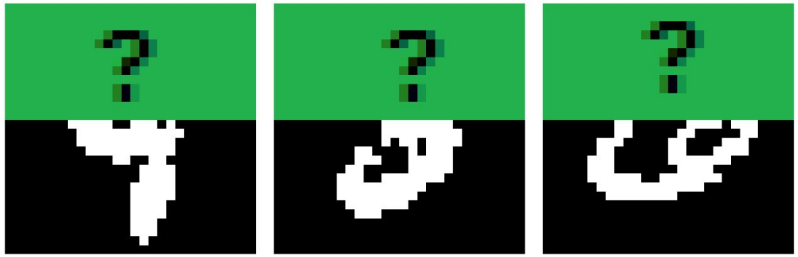
GMM as Latent Variable Model

Alternative motivation: Combine simple models into a more complex and expressive one



$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}) = \sum_{\mathbf{z}} p(\mathbf{z}) p(\mathbf{x} \mid \mathbf{z}) = \sum_{k=1}^{K} p(\mathbf{z} = k) \underbrace{\mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k)}_{\text{component}}$$

Likelihood with Latent Variable

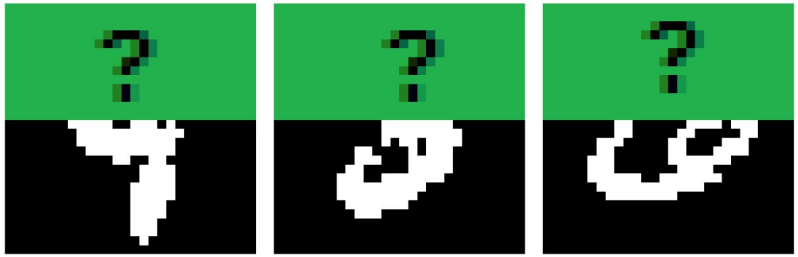


- Suppose some pixel values are missing at train time (e.g., top half)
- Let $\mathbf{X}$ denote observed random variables, and $\mathbf{Z}$ the unobserved ones (also called hidden or latent)
- Suppose we have a model for the joint distribution (e.g., PixelCNN)

$$p(\mathbf{X}, \mathbf{Z}; \theta)$$

What is the probability $p(\mathbf{X} = \bar{\mathbf{x}}; \theta)$ of observing a training data point $\bar{\mathbf{x}}$?

Likelihood with Latent Variable



- Suppose some pixel values are missing at train time (e.g., top half)
- Let $\mathbf{X}$ denote observed random variables, and $\mathbf{Z}$ the unobserved ones (also called hidden or latent)
- Suppose we have a model for the joint distribution (e.g., PixelCNN)

$$p(\mathbf{X}, \mathbf{Z}; \theta)$$

What is the probability $p(\mathbf{X} = \bar{\mathbf{x}}; \theta)$ of observing a training data point $\bar{\mathbf{x}}$?

$$\sum_{\mathbf{z}} p(\mathbf{X} = \bar{\mathbf{x}}, \mathbf{Z} = \mathbf{z}; \theta) = \sum_{\mathbf{z}} p(\bar{\mathbf{x}}, \mathbf{z}; \theta)$$

- Need to consider all possible ways to complete the image (fill green part)

Likelihood with Latent Variable

- Suppose that our joint distribution is

$$p(\mathbf{X}, \mathbf{Z}; \theta)$$

- We have a dataset $\mathcal{D}$, where for each datapoint the $\mathbf{X}$ variables are observed (e.g., pixel values) and the variables $\mathbf{Z}$ are never observed (e.g., cluster or class id.). $\mathcal{D} = \{\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(M)}\}$.
- Maximum likelihood learning:

$$\log \prod_{\mathbf{x} \in \mathcal{D}} p(\mathbf{x}; \theta) = \sum_{\mathbf{x} \in \mathcal{D}} \log p(\mathbf{x}; \theta) = \sum_{\mathbf{x} \in \mathcal{D}} \log \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}; \theta)$$

Likelihood with Latent Variable

- Suppose that our joint distribution is

$$p(\mathbf{X}, \mathbf{Z}; \theta)$$

- We have a dataset $\mathcal{D}$, where for each datapoint the $\mathbf{X}$ variables are observed (e.g., pixel values) and the variables $\mathbf{Z}$ are never observed (e.g., cluster or class id.). $\mathcal{D} = \{\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(M)}\}$.
- Maximum likelihood learning:

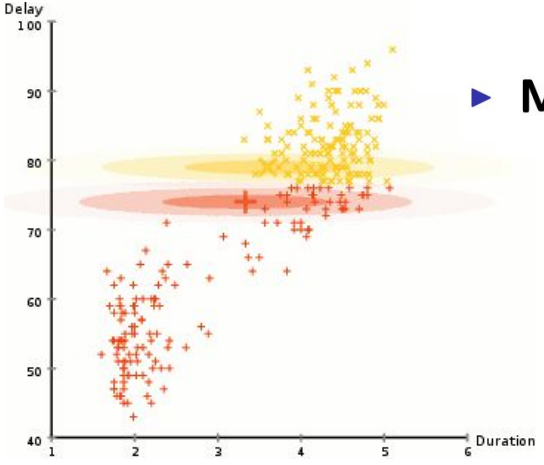
$$\log \prod_{\mathbf{x} \in \mathcal{D}} p(\mathbf{x}; \theta) = \sum_{\mathbf{x} \in \mathcal{D}} \log p(\mathbf{x}; \theta) = \sum_{\mathbf{x} \in \mathcal{D}} \log \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}; \theta)$$

- Evaluating $\log \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}; \theta)$ can be intractable. Suppose we have 30 binary latent features, $\mathbf{z} \in \{0, 1\}^{30}$. Evaluating $\sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}; \theta)$ involves a sum with 2^{30} terms. For continuous variables, $\log \int_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}; \theta) d\mathbf{z}$ is often intractable. Gradients ∇_θ also hard to compute.
- Need **approximations**. One gradient evaluation per training data point $\mathbf{x} \in \mathcal{D}$, so approximation needs to be cheap.

EM Algorithm for GMM

- **Initialize** the means μ_k, covariances Σ_k and mixing coefficients π_k
- Iterate until convergence:
  - **E-step**: Evaluate the responsibilities given current parameters
    - Let the current parameters be $\theta^{\text{old}} = \{\mu_k^{\text{old}}, \pi_k^{\text{old}}, \Sigma_k^{\text{old}}\}_{k=1}^K$

$$r_k^{(n)} := p(z^{(n)} = k | \mathbf{x}^{(n)}; \theta^{\text{old}}) = \frac{\pi_k^{\text{old}} \mathcal{N}(\mathbf{x}^{(n)} | \mu_k^{\text{old}}, \Sigma_k^{\text{old}})}{\sum_{j=1}^K \pi_j^{\text{old}} \mathcal{N}(\mathbf{x}^{(n)} | \mu_j^{\text{old}}, \Sigma_j^{\text{old}})}$$



  - **M-step**: Re-estimate the parameters given current responsibilities

Assume z assignment (r) is fixed:

$$\theta^{\text{new}} = \underset{\theta}{\text{argmax}} \sum_{n=1}^{N} \mathbb{E}_{r_k^{(n)}} \left[\log p(z^{(n)}, \mathbf{x}^{(n)}; \theta) \right]$$

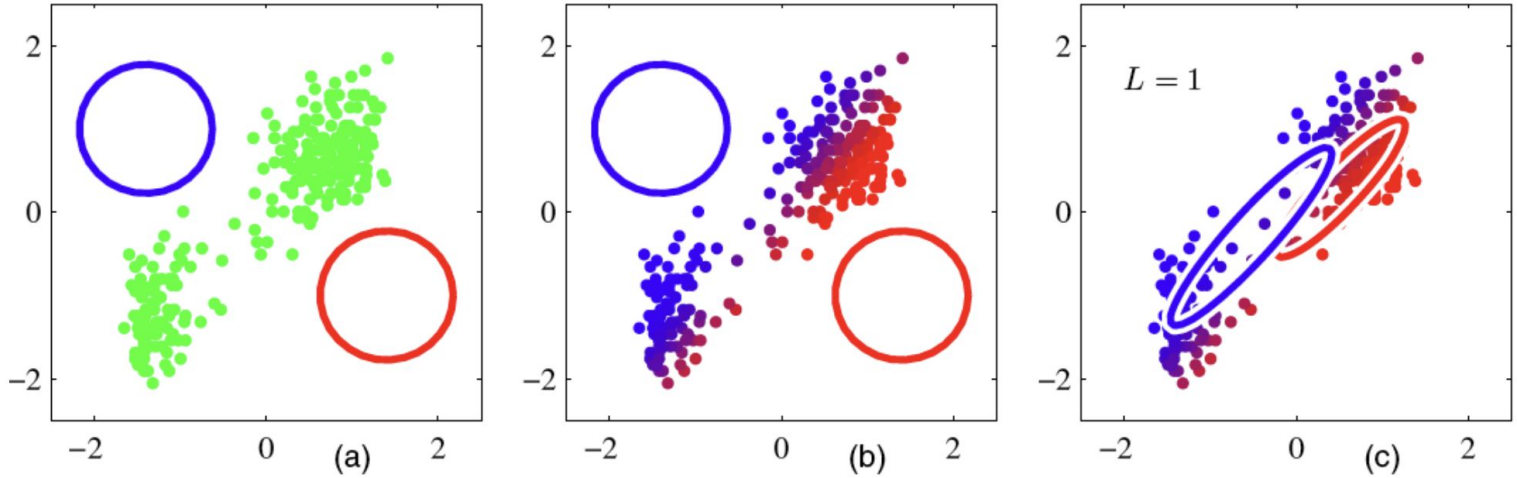
EM Algorithm for GMM

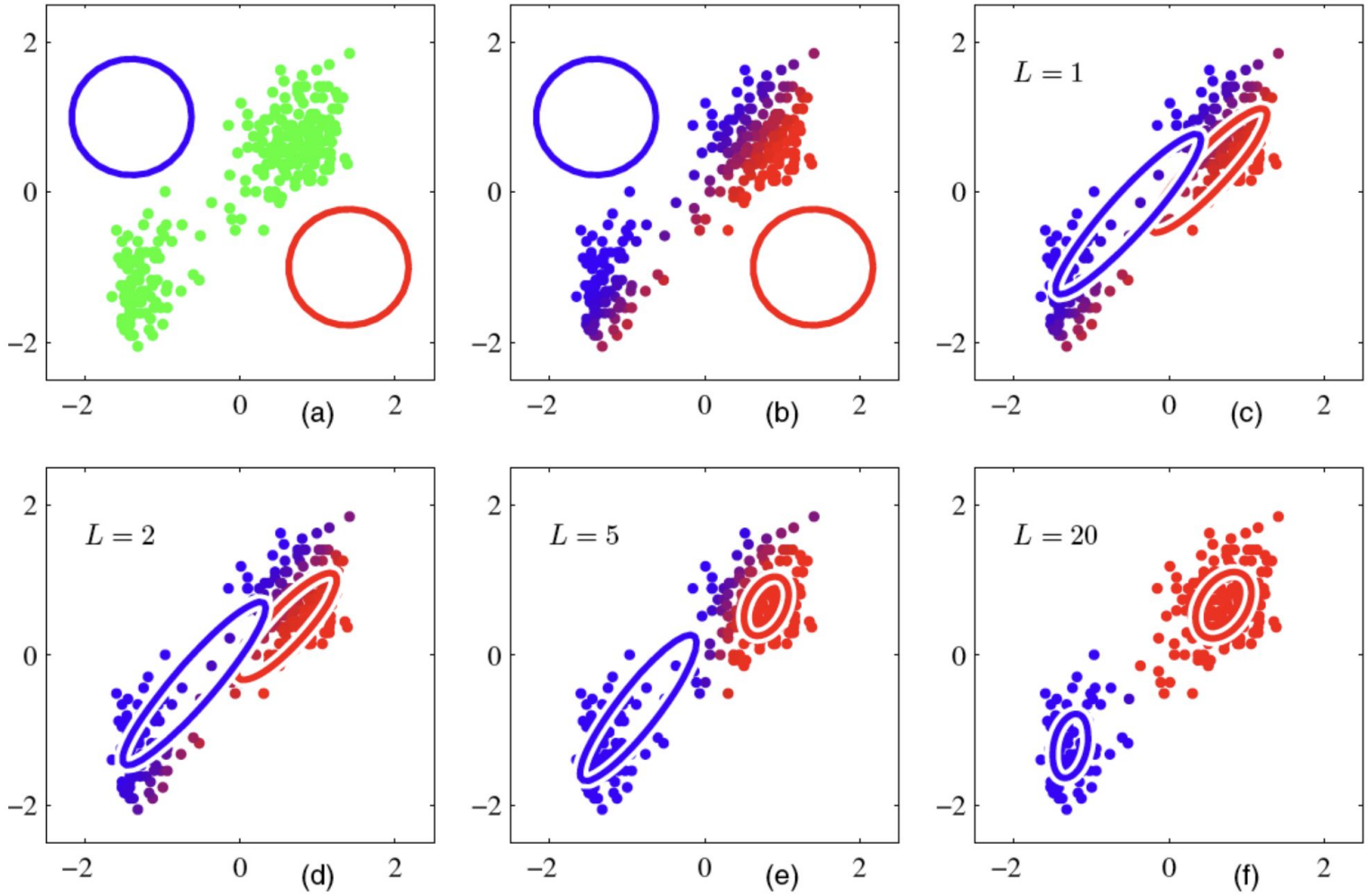
- **Initialize** the means μ_k, covariances Σ_k and mixing coefficients π_k
- Iterate until convergence:
  - **E-step**:

```
def e_step(self, X):
    self.weights = self.predict_proba(X)
    self.phi = self.weights.mean(axis=0)
```

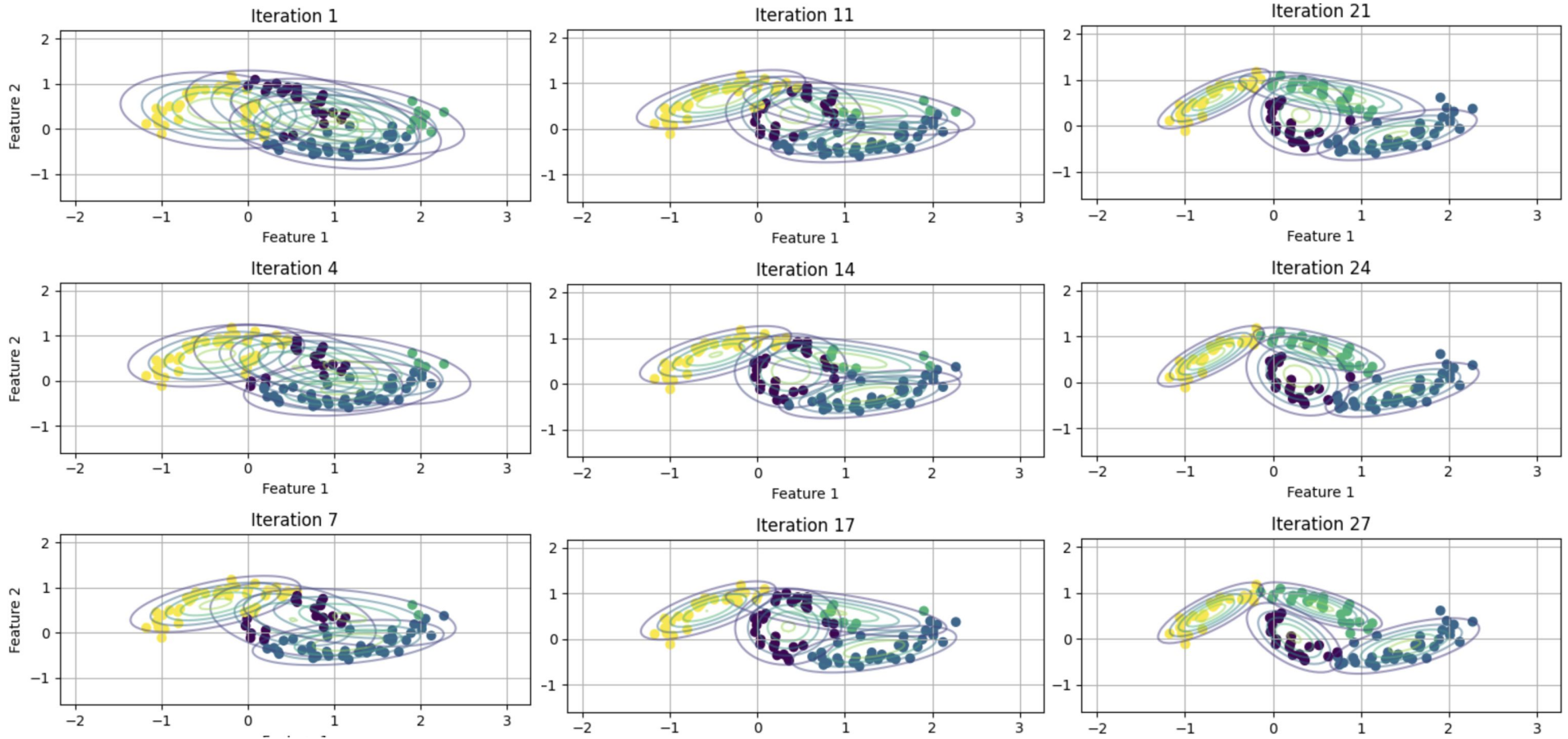
  - **M-step**: Re-estimate the parameters given current responsibilities

```
def m_step(self, X):
    for i in range(self.k):
        weight = self.weights[:, i]  # Simplified to use direct index
        total_weight = weight.sum()
        self.mu[i] = np.dot(weight, X) / total_weight  # Adjusted for
        X_mu = X - self.mu[i]
        self.sigma[i] = np.dot(weight * X_mu.T, X_mu) / total_weight
```

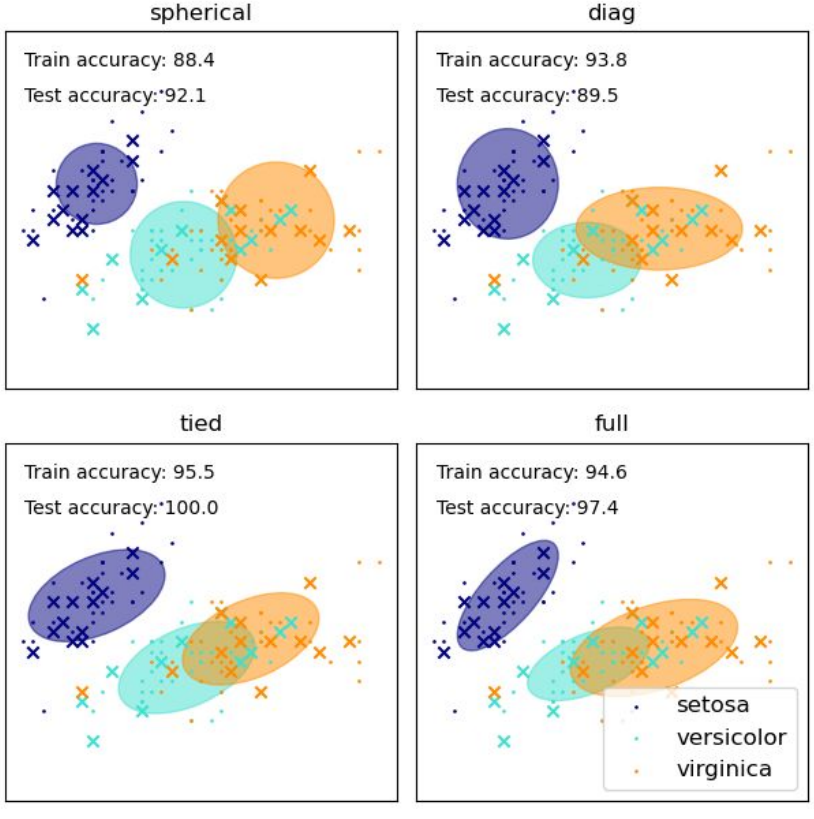

2
0
−2
−2
0
2
(a)
(b)
$L = 1$
(c)


2
0
−2
−2
0
2
(a)
(b)
$L = 1$
(c)
$L = 2$
(d)
$L = 5$
(e)
$L = 20$
(f)

Demo: https://colab.research.google.com/drive/1xcDd9SK0Toi8uQPsWocD--8jb-iankVA?usp=sharing



Covariance is the key to change shape



covariance_type : *{'full', 'tied', 'diag', 'spherical'}, default='full'*

String describing the type of covariance parameters to use. Must be one of:

- 'full': each component has its own general covariance matrix.
- 'tied': all components share the same general covariance matrix.
- 'diag': each component has its own diagonal covariance matrix.
- 'spherical': each component has its own single variance.

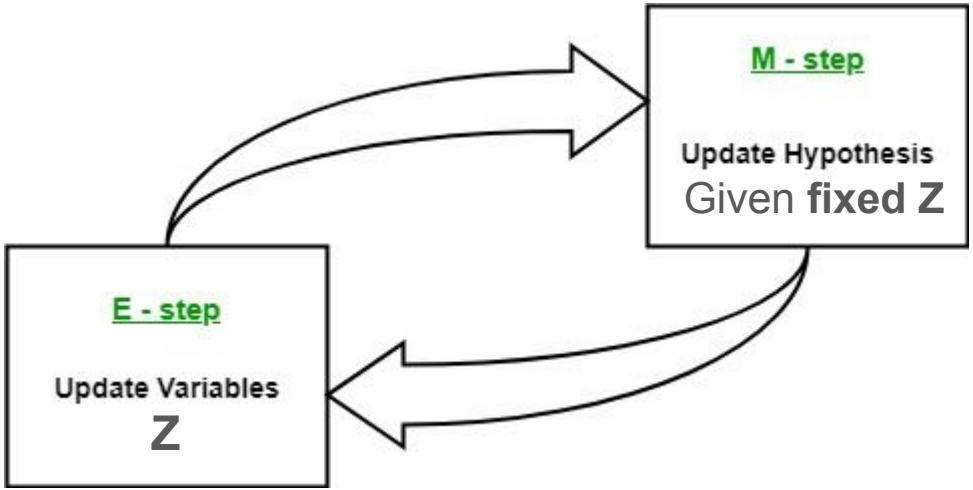
```
if gmm.covariance_type == "full":
    covariances = gmm.covariances_[n][:2, :2]
elif gmm.covariance_type == "tied":
    covariances = gmm.covariances_[:2, :2]
elif gmm.covariance_type == "diag":
    covariances = np.diag(gmm.covariances_[n][:2])
elif gmm.covariance_type == "spherical":
    covariances = np.eye(gmm.means_.shape[1]) * gmm.covariances_[n]
```

GMM vs K-Means

- EM for mixtures of Gaussians is just like a soft version of K-means, with **fixed priors and covariance**
- Instead of hard assignments in the E-step, we do **soft assignments** based on the softmax of the squared Mahalanobis distance from each point to each cluster.
- Each center moved by **weighted means** of the data, with weights given by soft assignments
- In K-means, weights are 0 or 1

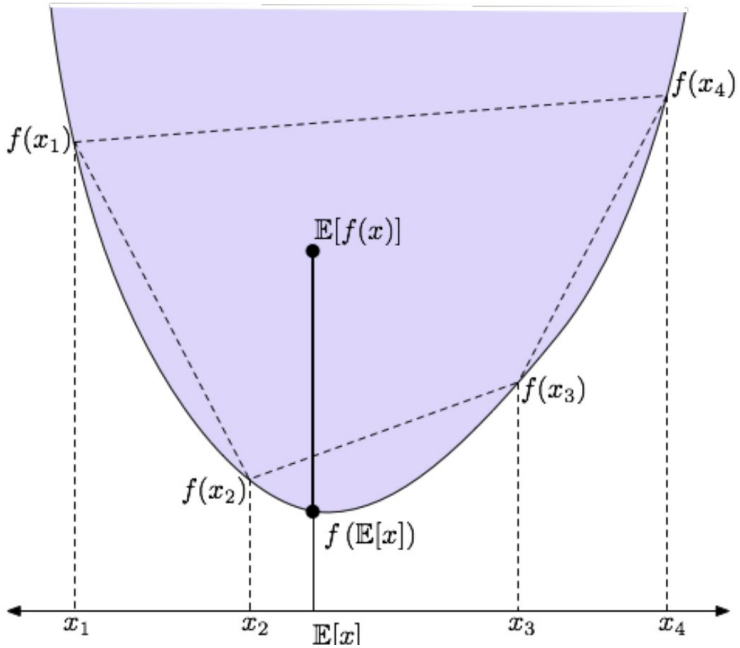
EM Algorithm

Optimize Model with Latent Variable



Jensen's Inequality: For convex f:

$$f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)]$$



- If g is **concave**, the inequality changes directions:

$$g(\mathbb{E}[X]) \geq \mathbb{E}[g(X)]$$

- In this lecture, we'll be using $\mathbf{x}$ to denote **observed data** and z to denote **the latent variables**
- We'll let $p(z, \mathbf{x}; \theta)$ denote the probabilistic model we've defined
  - Anything following a semicolon denotes a parameter of the distribution
  - We're not treating the parameters as random variables

- In this lecture, we'll be using $\mathbf{x}$ to denote **observed data** and z to denote **the latent variables**
- We'll let $p(z, \mathbf{x}; \boldsymbol{\theta})$ denote the probabilistic model we've defined
  - Anything following a semicolon denotes a parameter of the distribution
  - We're not treating the parameters as random variables
- We assume we have an observed dataset $\mathcal{D} = \{\mathbf{x}^{(n)}\}_{n=1}^{N}$ and would like to fit $\boldsymbol{\theta}$ using maximum likelihood:

$$\log p(\mathcal{D}; \boldsymbol{\theta}) = \sum_{n=1}^{N} \log p(\mathbf{x}^{(n)}; \boldsymbol{\theta})$$

- To compute $p(\mathbf{x}; \boldsymbol{\theta})$, we have to **marginalize** over z:

$$p(\mathbf{x}; \boldsymbol{\theta}) = \sum_{z} p(z, \mathbf{x}; \boldsymbol{\theta})$$

- To compute $p(\mathbf{x}; \boldsymbol{\theta})$, we have to **marginalize** over z:

$$p(\mathbf{x}; \boldsymbol{\theta}) = \sum_{z} p(z, \mathbf{x}; \boldsymbol{\theta})$$

- Typically no closed form solution to the maximum likelihood problem

$$\log p(\mathcal{D}; \boldsymbol{\theta}) = \sum_{n=1}^{N} \log p(\mathbf{x}^{(n)}; \boldsymbol{\theta}) = \sum_{n=1}^{N} \log \left(\sum_{z^{(n)}} p(z^{(n)}, \mathbf{x}^{(n)}; \boldsymbol{\theta}) \right)$$

- Typically no closed form solution to the maximum likelihood problem

$$\log p(\mathcal{D};\boldsymbol{\theta}) = \sum_{n=1}^{N} \log p(\mathbf{x}^{(n)};\boldsymbol{\theta}) = \sum_{n=1}^{N} \log \left(\sum_{z^{(n)}} p(z^{(n)}, \mathbf{x}^{(n)};\boldsymbol{\theta}) \right)$$

- Evaluating $\log \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}; \theta)$ can be intractable. Suppose we have 30 binary latent features, $\mathbf{z} \in \{0,1\}^{30}$. Evaluating $\sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}; \theta)$ involves a sum with 2^{30} terms. For continuous variables, $\log \int_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}; \theta) d\mathbf{z}$ is often intractable. Gradients ∇_θ also hard to compute.
- Need **approximations**. One gradient evaluation per training data point $\mathbf{x} \in \mathcal{D}$, so approximation needs to be cheap.

$$\log p(\mathcal{D};\boldsymbol{\theta}) = \sum_{n=1}^{N} \log p(\mathbf{x}^{(n)};\boldsymbol{\theta}) = \sum_{n=1}^{N} \log \left(\sum_{z^{(n)}} p(z^{(n)}, \mathbf{x}^{(n)};\boldsymbol{\theta}) \right)$$

- We introduce auxilliary distributions $q_n(z^{(n)})$ over each of the latent variables

$$\begin{aligned} \sum_{n=1}^{N} \log \left(\sum_{z^{(n)}} p(z^{(n)}, \mathbf{x}^{(n)};\boldsymbol{\theta}) \right) &= \sum_{n=1}^{N} \log \left(\sum_{z^{(n)}} q_n(z^{(n)}) \frac{p(z^{(n)}, \mathbf{x}^{(n)};\boldsymbol{\theta})}{q_n(z^{(n)})} \right) \\ &= \sum_{n=1}^{N} \log \left(\mathbb{E}_{q_n(z^{(n)})} \left[\frac{p(z^{(n)}, \mathbf{x}^{(n)};\boldsymbol{\theta})}{q_n(z^{(n)})} \right] \right) \end{aligned}$$

- We introduce auxilliary distributions $q_n(z^{(n)})$ over each of the latent variables

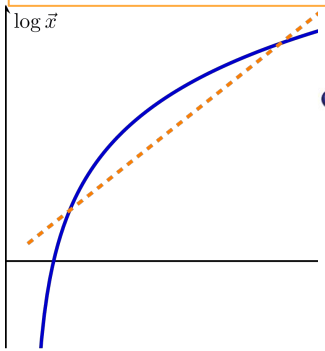
$$\sum_{n=1}^{N} \log \left(\sum_{z^{(n)}} p(z^{(n)}, \mathbf{x}^{(n)}; \boldsymbol{\theta}) \right) = \sum_{n=1}^{N} \log \left(\sum_{z^{(n)}} q_n(z^{(n)}) \frac{p(z^{(n)}, \mathbf{x}^{(n)}; \boldsymbol{\theta})}{q_n(z^{(n)})} \right)$$

$$= \sum_{n=1}^{N} \log \left(\mathbb{E}_{q_n(z^{(n)})} \left[\frac{p(z^{(n)}, \mathbf{x}^{(n)}; \boldsymbol{\theta})}{q_n(z^{(n)})} \right] \right)$$

$$\geq \sum_{n=1}^{N} \mathbb{E}_{q_n(z^{(n)})} \left[\log \frac{p(z^{(n)}, \mathbf{x}^{(n)}; \boldsymbol{\theta})}{q_n(z^{(n)})} \right]$$

If g is **concave**, the inequality changes directions:

$$g(\mathbb{E}[X]) \geq \mathbb{E}[g(X)]$$



- In the last step, we use Jensen's Inequality. Since log is concave:

$$\log \left(\mathbb{E}_{q_n(z^{(n)})} \left[\frac{p(z^{(n)}, \mathbf{x}^{(n)}; \boldsymbol{\theta})}{q_n(z^{(n)})} \right] \right) \geq \mathbb{E}_{q_n(z^{(n)})} \left[\log \frac{p(z^{(n)}, \mathbf{x}^{(n)}; \boldsymbol{\theta})}{q_n(z^{(n)})} \right]$$

Called Evidence Lower Bound (**ELBO**)

Evidence lower bound (ELBO)

- Suppose $q(\mathbf{z})$ is **any** probability distribution over the hidden variables
- **Evidence lower bound** (ELBO) holds for any q

$$\log p(\mathbf{x};\theta) = \log\left(\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right]\right) \geq \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\log\left(\frac{p_\theta(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right)\right]$$

$$= \sum_{\mathbf{z}} q(\mathbf{z}) \log\left(\frac{p_\theta(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right)$$

Evidence lower bound (ELBO)

- Suppose $q(\mathbf{z})$ is **any** probability distribution over the hidden variables
- **Evidence lower bound** (ELBO) holds for any q

$$\log p(\mathbf{x};\theta) = \log\left(\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right]\right) \geq \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\log\left(\frac{p_\theta(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right)\right]$$

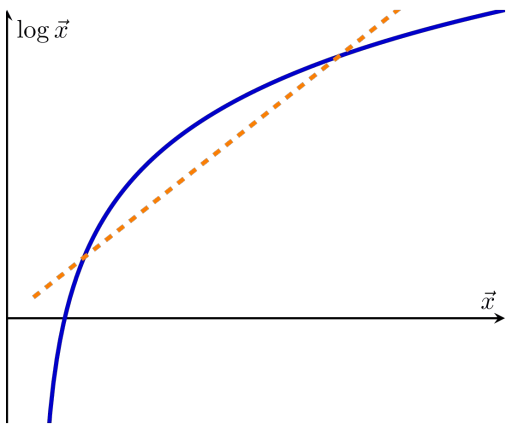
$$= \sum_{\mathbf{z}} q(\mathbf{z}) \log\left(\frac{p_\theta(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right)$$

$$= \sum_{\mathbf{z}} q(\mathbf{z}) \log p_\theta(\mathbf{x},\mathbf{z}) \underbrace{- \sum_{\mathbf{z}} q(\mathbf{z}) \log q(\mathbf{z})}_{\text{Entropy } H(q) \text{ of } q}$$

$$= \sum_{\mathbf{z}} q(\mathbf{z}) \log p_\theta(\mathbf{x},\mathbf{z}) + H(q)$$

$$\sum_{n=1}^{N} \log p(\mathbf{x}^{(n)}; \boldsymbol{\theta}) \geq \sum_{n=1}^{N} \mathbb{E}_{q_n(z^{(n)})} \left[\log \frac{p(z^{(n)}, \mathbf{x}^{(n)}; \boldsymbol{\theta})}{q_n(z^{(n)})} \right]$$
$$\equiv \mathcal{L}(q, \boldsymbol{\theta}) \text{ where } q = \{q_1, \ldots, q_N\}$$

- We expect $\mathcal{L}(q, \boldsymbol{\theta})$ might be easier to optimize w.r.t. $\boldsymbol{\theta}$, since it only appears in $\log p(z^{(n)}, \mathbf{x}^{(n)}; \boldsymbol{\theta})$, so we'll use this as our new objective
- For **any** auxilliary distributions q_n, we obtain a lower bound on the log likelihood
- Which q_n should we choose? Want to make the bound as tight as possible



Equality holds if x is constant (list of points all the same)

$$\log\left(\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right]\right) \geq \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\log\left(\frac{p_\theta(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right)\right]$$

- We know this bound is tight (i.e. the inequality becomes an equality) if there are constants c_n such that:

$$\frac{p(z^{(n)},\mathbf{x}^{(n)};\boldsymbol{\theta})}{q_n(z^{(n)})} = \text{constant} \implies q_n(z^{(n)}) = c_n p(z^{(n)},\mathbf{x}^{(n)};\boldsymbol{\theta})$$

$$\log\left(\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right]\right) \geq \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z})}\left[\log\left(\frac{p_\theta(\mathbf{x},\mathbf{z})}{q(\mathbf{z})}\right)\right]$$

- We know this bound is tight (i.e. the inequality becomes an equality) if there are constants c_n such that:

$$\frac{p(z^{(n)},\mathbf{x}^{(n)};\boldsymbol{\theta})}{q_n(z^{(n)})} = \text{constant} \implies q_n(z^{(n)}) = c_n p(z^{(n)},\mathbf{x}^{(n)};\boldsymbol{\theta})$$

- Using $\sum_{z^{(n)}} q_n(z^{(n)}) = 1$, we have:

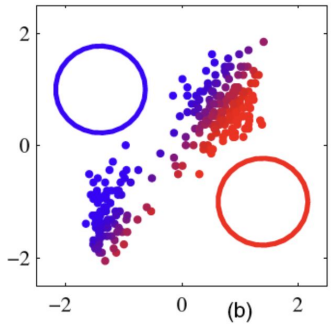
$$1 = \sum_{z^{(n)}} q_n(z^{(n)}) = c_n \sum_{z^{(n)}} p(z^{(n)},\mathbf{x}^{(n)};\boldsymbol{\theta}) = c_n p(\mathbf{x}^{(n)};\boldsymbol{\theta})$$

$$\implies c_n = \frac{1}{p(\mathbf{x}^{(n)};\boldsymbol{\theta})}$$

- Hence:

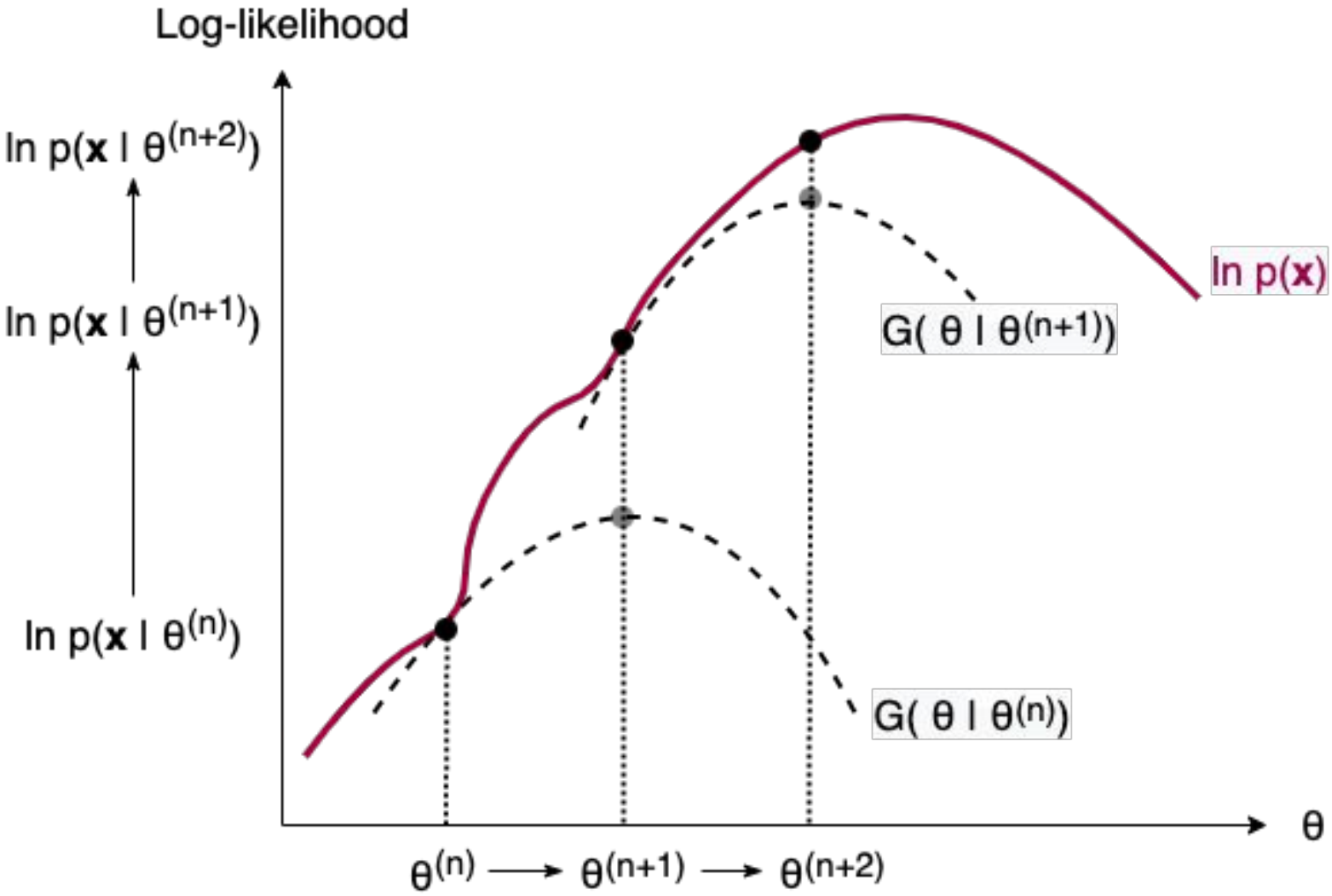
"classification"

$$q_n(z^{(n)}) = \frac{p(z^{(n)},\mathbf{x}^{(n)};\boldsymbol{\theta})}{p(\mathbf{x}^{(n)};\boldsymbol{\theta})} = p(z^{(n)}|\mathbf{x}^{(n)};\boldsymbol{\theta})$$



- The EM algorithm alternates between making the bound tight at the current parameter values and then optimizing the lower bound
- If the current parameter value is $\boldsymbol{\theta}^{\text{old}}$:
  - **E-step**: For all n, set $q_n(z^{(n)}) = p(z^{(n)}|\mathbf{x}^{(n)}; \boldsymbol{\theta}^{\text{old}})$ and form the lower bound $\mathcal{L}(q; \boldsymbol{\theta})$
    - Remember: $\log p(\mathcal{D}; \boldsymbol{\theta}^{\text{old}}) = \mathcal{L}(q; \boldsymbol{\theta}^{\text{old}})$ after this step
  - **M-step**: Optimize the lower bound:

$$\begin{aligned}\boldsymbol{\theta}^{\text{new}} &= \operatorname*{argmax}_{\boldsymbol{\theta}} \mathcal{L}(q, \boldsymbol{\theta}) \\ &= \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{n=1}^{N} \mathbb{E}_{q_n(z^{(n)})} \left[\log \frac{p(z^{(n)}, \mathbf{x}^{(n)}; \boldsymbol{\theta})}{q_n(z^{(n)})} \right]\end{aligned}$$


Log-likelihood
ln p(x | θ(n+2))
ln p(x | θ(n+1))
ln p(x | θ(n))
ln p(x)
G(θ | θ(n+1))
G(θ | θ(n))
θ
θ(n) → θ(n+1) → θ(n+2)

- **M-step**: Optimize the lower bound:

$$\sum_{n=1}^{N} \mathbb{E}_{q_n(z^{(n)})} \left[\log \frac{p(z^{(n)}, \mathbf{x}^{(n)}; \boldsymbol{\theta})}{q_n(z^{(n)})} \right] =$$

$$\sum_{n=1}^{N} \mathbb{E}_{q_n(z^{(n)})} \left[\log p(z^{(n)}, \mathbf{x}^{(n)}; \boldsymbol{\theta}) \right] - \underbrace{\mathbb{E}_{q_n(z^{(n)})} \left[\log q_n(z^{(n)}) \right]}_{\text{constant w.r.t.} \boldsymbol{\theta}}$$

- Substitute in $q_n(z^{(n)}) = p(z^{(n)} | \mathbf{x}^{(n)}; \boldsymbol{\theta}^{\text{old}})$:

$$\boldsymbol{\theta}^{\text{new}} = \operatorname*{argmax}_{\boldsymbol{\theta}} \sum_{n=1}^{N} \mathbb{E}_{p(z^{(n)} | \mathbf{x}^{(n)}; \boldsymbol{\theta}^{\text{old}})} \left[\log p(z^{(n)}, \mathbf{x}^{(n)}; \boldsymbol{\theta}) \right]$$

- This is the expected complete data log-likelihood.

You have two coins, A and B, with unknown probabilities of heads, p_A and p_B, respectively. You perform a series of experiments where you randomly choose one of the two coins, flip it a few times, but you don't record which coin was flipped, only the outcome of the flips.

Experiment Details:

1. **Select a coin** randomly.
2. **Flip the selected coin** 10 times.
3. **Record the number of heads** observed, but not the identity of the coin.
4. Repeat this process for a number of trials (e.g., 5 trials).

Data Example:

- Trial 1: 6 heads
- Trial 2: 9 heads
- Trial 3: 8 heads
- Trial 4: 3 heads
- Trial 5: 7 heads

Goal:

Estimate p_A and p_B, the probabilities of getting heads for coins A and B, respectively.

For each iteration with observation x, $r(A) = P(Z = A \mid x)$ means the tossed coin in this experiment belongs to coin A; $r(B) = P(Z = B \mid x) = 1 - r_A$

Let's assume $P_A = 0.55$, $P_B = 0.5$ (add a bit randomness in initialization), $P(z=A) = P(z=B) = 0.5$

E-step: estimate which coin is assigned to each data:

$$\theta = \left[\pi_{\{A/B\}} = P(z = A/B);\ \ \theta_{\{A/B\}} = P(x = \text{head} | z = A/B)\right]$$

$$\begin{aligned} r_A &= P(z = A \mid x; \theta^{old}) \\ &= \frac{P(z = A) \cdot P(x \mid z = A)}{\sum_z P(z) \cdot P(x \mid z)} \\ &= \frac{\pi_A \cdot P(x \mid z = A)}{\pi_A \cdot P(x \mid z = A) + \pi_B \cdot P(x \mid z = B)} \end{aligned}$$

For each iteration with observation x, $r(A) = P(Z = A \mid x)$ means the tossed coin in this experiment belongs to coin A; $r(B) = P(Z = B \mid x) = 1 - r_A$

Let's assume $P_A = 0.55$, $P_B = 0.5$ (add a bit randomness in initialization), $P(z=A) = P(z=B) = 0.5$

E-step: estimate which coin is assigned to each data:

$$r_A(x) = \frac{\pi_A P(x|A)}{\pi_A P(x|A) + \pi_B P(x|B)}$$

$$r_B(x) = \frac{\pi_B P(x|B)}{\pi_A P(x|A) + \pi_B P(x|B)}$$

where:

$$P(x|A) = \theta_A^{\#Head(x)}(1-\theta_A)^{\#Tail(x)}$$

$$P(x|B) = \theta_B^{\#Head(x)}(1-\theta_B)^{\#Tail(x)}$$

For each iteration with observation x, $r(A) = P(Z = A \mid x)$ means the tossed coin in this experiment belongs to coin A; $r(B) = P(Z = B \mid x) = 1 - r_A$

Let's assume $P_A = 0.55$, $P_B = 0.5$ (add a bit randomness in initialization), $P(z=A) = P(z=B) = 0.5$

E-step: estimate which coin is assigned to each data:

$$
\begin{aligned}
P(x_1 \mid z = A) &= \prod_i P(x_1^{(i)} \mid z = A) \\
&= (\theta_A)^{\text{Num. of Head}} \cdot (1 - \theta_A)^{\text{Num. of Tail}} \\
&= 0.55^6 * 0.45^4
\end{aligned}
$$

$$
\begin{aligned}
P(x_1 \mid z = B) &= \prod_i P(x_1^{(i)} \mid z = B) \\
&= (\theta_B)^{\text{Num. of Head}} \cdot (1 - \theta_B)^{\text{Num. of Tail}} \\
&= 0.5^6 * 0.5^4
\end{aligned}
$$

$$
\begin{aligned}
r_A(x_1) &= P(z = A \mid x_1; \theta^{old}) \\
&= \frac{\pi_A \cdot P(x_1 \mid z = A)}{\pi_A \cdot P(x_1 \mid z = A) + \pi_B \cdot P(x_1 \mid z = B)} \\
&= \frac{0.5 * 0.55^6 * 0.45^4}{0.5 * 0.55^6 * 0.45^4 + 0.5 * 0.5^6 * 0.5^4} \\
&= 0.54
\end{aligned}
$$

For each iteration with observation x, P(Z = A | x) means the tossed coin in this experiment belongs to coin A; P(Z = B | x) = 1 - P(Z = A | x)

Let's assume $P_A = 0.55$, $P_B = 0.5$ (add a bit randomness in initialization), P(z=A) = P(z=B) = 0.5

Iteration 1: 6 heads, 4 tails

M-step: maximizing the likelihood based on current assignment

$$\theta^{new} = \arg\max_{\theta} \sum_{x} \sum_{z} r_z(x) \cdot \log P(x, z)$$

$$= \arg\max_{\theta} \sum_{z} \sum_{x} r_z(x) \cdot (\log P(z) + \log P(x|z))$$

$$= \arg\max_{\theta} \sum_{z} \sum_{x} \Big(r_z(x) \cdot \big(\log \pi_z + \#\mathrm{Head}(x) \log \theta_z + \#\mathrm{Tail}(x) \log(1 - \theta_z) \big) \Big)$$

$$\theta^{new} = \arg\max_{\theta} \sum_{z} \sum_{x} \left(r_z(x) \cdot \left(\log \pi_z + \#\text{Head}(x) \log \theta_z + \#\text{Tail}(x) \log(1 - \theta_z) \right) \right)$$

Focusing only on the terms involving π and considering $\pi_A = \pi$ and $\pi_B = 1 - \pi$, the log-likelihood function simplifies to:

$$Q(\pi) = \sum_{x} r_A(x) \cdot \log \pi + \sum_{x} r_B(x) \cdot \log(1 - \pi)$$

Maximization for π

To find the optimal value of π, we take the derivative of this function with respect to π, set it to zero, and solve for π.

1. **Take the derivative:**

$$\frac{dQ}{d\pi} = \frac{\sum_x r_A(x)}{\pi} - \frac{\sum_x r_B(x)}{1 - \pi}$$

2. **Set the derivative to zero to find the critical points:**

$$\frac{\sum_x r_A(x)}{\pi} - \frac{\sum_x r_B(x)}{1 - \pi} = 0$$

3. **Solve for π:**

$$\sum_{x} r_A(x)(1 - \pi) = \sum_{x} r_B(x)\pi$$

$$\sum_{x} r_A(x) - \pi \sum_{x} r_A(x) = \pi \sum_{x} r_B(x)$$

Since $\sum_x r_A(x) + \sum_x r_B(x)$ sums to the total number of trials (or data points) N, where N is the total responsibility accounted across all data points, we get:

$$\pi = \frac{\sum_x r_A(x)}{N}$$

Maximization for θ_A

Focusing only on the terms involving θ_A (ignoring constants and terms involving other parameters like π):

$$Q(\theta_A) = \sum_x r_A(x) \cdot (\#\text{Head}(x) \log \theta_A + \#\text{Tail}(x) \log(1 - \theta_A))$$

To find the value of θ_A that maximizes this expression, we take its derivative with respect to θ_A, set the derivative to zero, and solve for θ_A:

1. **Take the derivative:**

$$\frac{dQ}{d\theta_A} = \sum_x r_A(x) \left(\frac{\#\text{Head}(x)}{\theta_A} - \frac{\#\text{Tail}(x)}{1 - \theta_A} \right)$$

2. **Set the derivative to zero:**

$$\sum_x r_A(x) \left(\frac{\#\text{Head}(x)}{\theta_A} - \frac{\#\text{Tail}(x)}{1 - \theta_A} \right) = 0$$

3. **Solve for θ_A:**

$$\sum_x r_A(x)\#\text{Head}(x) - \theta_A \sum_x r_A(x)\#\text{Head}(x) = \theta_A \sum_x r_A(x)\#\text{Tail}(x)$$

$$\theta_A(\sum_x r_A(x)\#\text{Head}(x) + \sum_x r_A(x)\#\text{Tail}(x)) = \sum_x r_A(x)\#\text{Head}(x)$$

$$\theta_A = \frac{\sum_x r_A(x)\#\text{Head}(x)}{\sum_x r_A(x)(\#\text{Head}(x) + \#\text{Tail}(x))}$$

Revisit GMM

- Let's revisit the mixture of Gaussians example from last lecture and derive the updates using our general EM algorithm
- Recall our model was:

$$p(z = k; \boldsymbol{\theta}) = \pi_k$$

$$p(\mathbf{x}|z = k; \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k)$$

- In this scenario, we have $\boldsymbol{\theta} = \{\mu_k, \pi_k, \Sigma_k\}_{k=1}^K$

E-Step for GMM

- Let the current parameters be $\boldsymbol{\theta}^{\text{old}} = \{\mu_k^{\text{old}}, \pi_k^{\text{old}}, \Sigma_k^{\text{old}}\}_{k=1}^K$
- **E-step**: For all n, set $q_n(z^{(n)}) = p(z^{(n)}|\mathbf{x}^{(n)}; \boldsymbol{\theta}^{\text{old}})$

$$r_k^{(n)} := q_n(z^{(n)} = k) = p(z^{(n)} = k|\mathbf{x}^{(n)}; \boldsymbol{\theta}^{\text{old}}) = \frac{\pi_k^{\text{old}} \mathcal{N}(\mathbf{x}^{(n)}|\mu_k^{\text{old}}, \Sigma_k^{\text{old}})}{\sum_{j=1}^K \pi_j^{\text{old}} \mathcal{N}(\mathbf{x}^{(n)}|\mu_j^{\text{old}}, \Sigma_j^{\text{old}})}$$

M-Step for GMM

M-step:

$$\boldsymbol{\theta}^{\text{new}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{n=1}^{N} \mathbb{E}_{q_n(z^{(n)})} \left[\log p(z^{(n)}, \mathbf{x}^{(n)}; \boldsymbol{\theta}) \right]$$

- Substitute in:
  - $\log p(z^{(n)}, \mathbf{x}^{(n)}; \boldsymbol{\theta}) = \sum_{k=1}^{K} \mathbb{I}[z^{(n)} = k] \left(\log \pi_k + \log \mathcal{N}(\mathbf{x}^{(n)}; \mu_k, \Sigma_k) \right)$
  - $q_n(z^{(n)}) = p(z^{(n)} | \mathbf{x}^{(n)}; \boldsymbol{\theta}^{\text{old}})$:

$$\begin{aligned}
\boldsymbol{\theta}^{\text{new}} &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{n=1}^{N} \mathbb{E}_{q_n(z^{(n)})} \left[\sum_{k=1}^{K} \mathbb{I}[z^{(n)} = k] \left(\log \pi_k + \log \mathcal{N}(\mathbf{x}^{(n)}; \mu_k, \Sigma_k) \right) \right] \\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{n=1}^{N} \sum_{k=1}^{K} r_k^{(n)} \left(\log \pi_k + \log \mathcal{N}(\mathbf{x}^{(n)}; \mu_k, \Sigma_k) \right)
\end{aligned}$$

M-Step for GMM

$$\boldsymbol{\theta}^{\text{new}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{n=1}^{N} \sum_{k=1}^{K} r_k^{(n)} \left(\log \pi_k + \mathcal{N}(\mathbf{x}^{(n)}; \mu_k, \Sigma_k) \right)$$

- Taking derivatives and setting to zero, we get the updates from last lecture:

$$\begin{aligned}
\mu_k &= \frac{1}{N_k} \sum_{n=1}^{N} r_k^{(n)} \mathbf{x}^{(n)} \\
\Sigma_k &= \frac{1}{N_k} \sum_{n=1}^{N} r_k^{(n)} (\mathbf{x}^{(n)} - \mu_k)(\mathbf{x}^{(n)} - \mu_k)^T \\
\pi_k &= \frac{N_k}{N} \quad \text{with} \quad N_k = \sum_{n=1}^{N} r_k^{(n)}
\end{aligned}$$

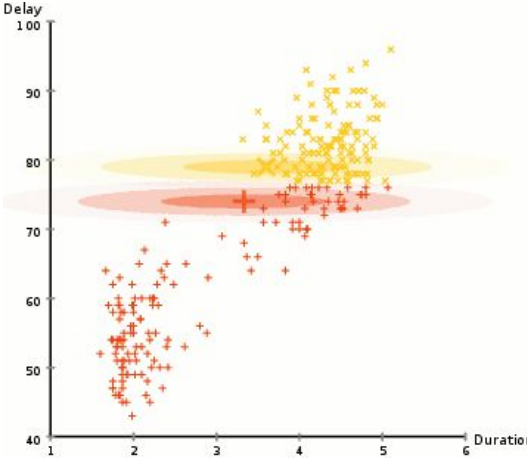
EM Convergence

- We can deduce that an iteration of EM will improve the log-likelihood by using the fact that the bound is tight at $\boldsymbol{\theta}^{\text{old}}$ after the E-step
- Let q denote the q_n's after the E-step i.e. $q_n(z^{(n)}) = p(z^{(n)}|\mathbf{x}^{(n)}; \boldsymbol{\theta}^{\text{old}})$

$$\begin{aligned}
\log p(\mathcal{D}; \boldsymbol{\theta}^{\text{new}}) &\geq \mathcal{L}(q, \boldsymbol{\theta}^{\text{new}}) && \text{since } \log p(\mathcal{D}; \boldsymbol{\theta}) \geq \mathcal{L}(q, \boldsymbol{\theta}) \text{ always} \\
&\geq \mathcal{L}(q, \boldsymbol{\theta}^{\text{old}}) && \text{since } \boldsymbol{\theta}^{\text{new}} = \operatorname*{argmax}_{\boldsymbol{\theta}} \mathcal{L}(q, \boldsymbol{\theta}) \\
&= \log p(\mathcal{D}; \boldsymbol{\theta}^{\text{old}}) && \text{since } \log p(\mathcal{D}; \boldsymbol{\theta}^{\text{old}}) = \mathcal{L}(q; \boldsymbol{\theta}^{\text{old}})
\end{aligned}$$

Summary of EM & GMM

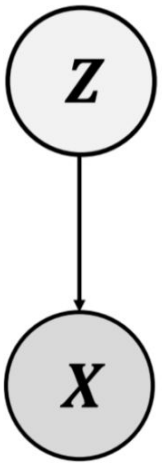
- A general algorithm for optimizing many latent variable models.
- Iteratively computes a lower bound then optimizes it.
- Converges but maybe to a local minima.
- Can use multiple restarts.
- Can initialize from k-means for mixture models
- Limitation - need to be able to compute $p(z|\mathbf{x};\boldsymbol{\theta})$, not possible for more complicated models.



(Variational) AutoEncoder

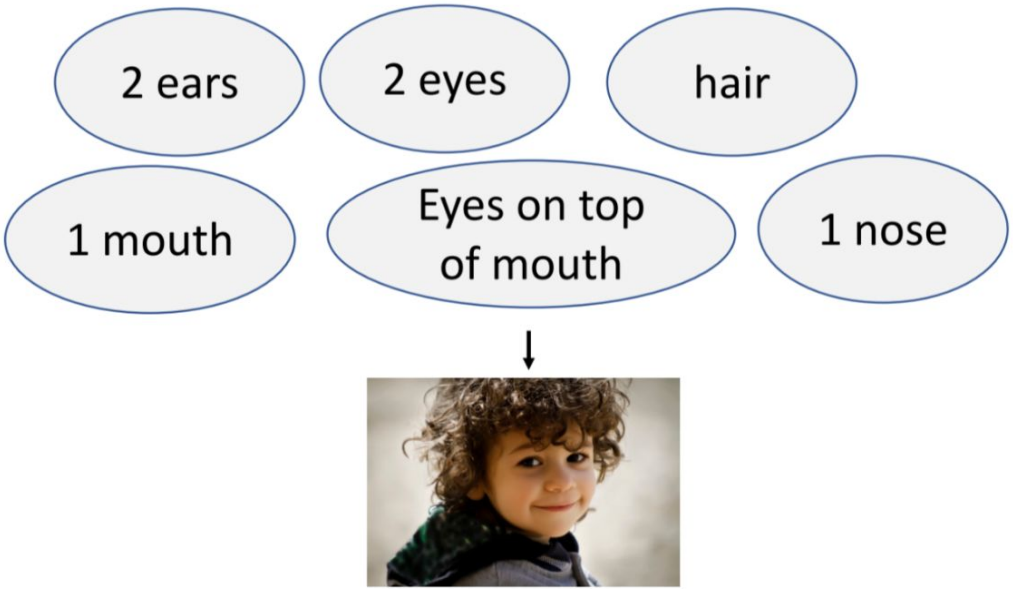
Variational Autoencoders

Belong in the family of latent variable probabilistic models that can infer the hidden structure in the underlying data



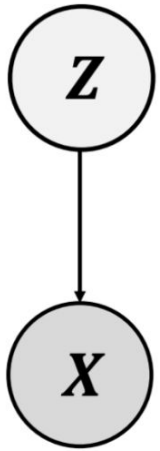
Latent Variable
(**unobserved**)

Observed Variable
(**data**)



Variational Autoencoders

Belong in the family of latent variable probabilistic models that can infer the hidden structure in the underlying data



Latent Variable
(**unobserved**)

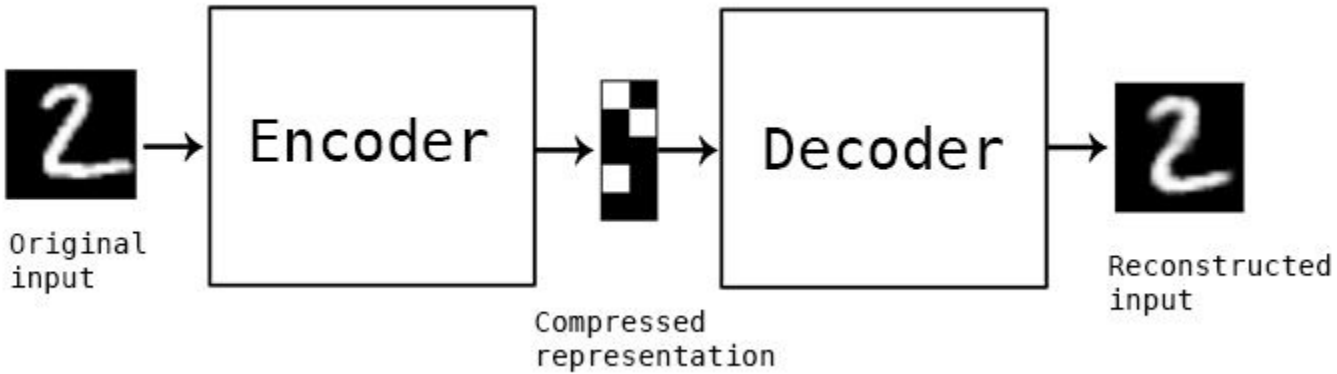
Observed Variable
(data)

The goal of variational autoencoders is twofold:

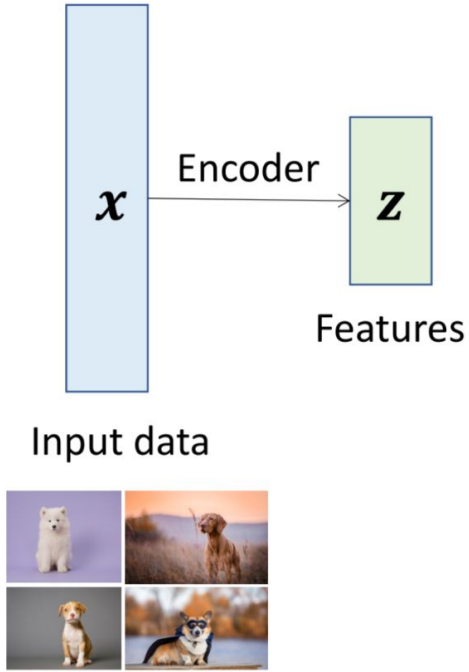
1. Model the data distribution $p(\boldsymbol{x})$ – as is the case with all generative models

2. Extract a latent representation $\boldsymbol{Z}$ that "describes" the data.

Variational Autoencoders

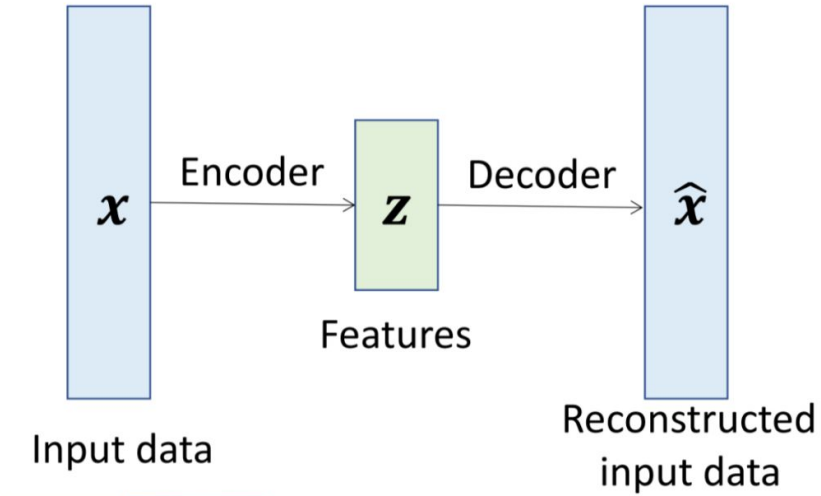


~~Variational~~ Autoencoders



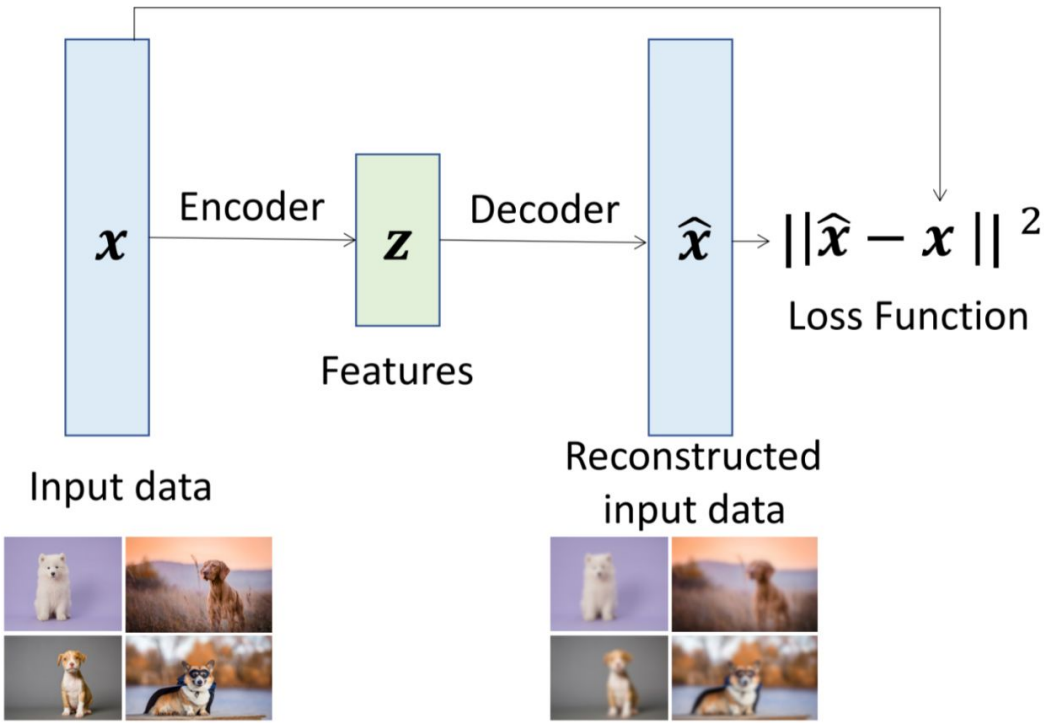
- The encoder extracts features which contain useful information from the data and can be used for downstream tasks
- We don't have labels, only data. So, we want to use unsupervised methods to learn these features.
- How?

~~Variational~~ Autoencoders



- Use the features z to reconstruct the input data x with a decoder

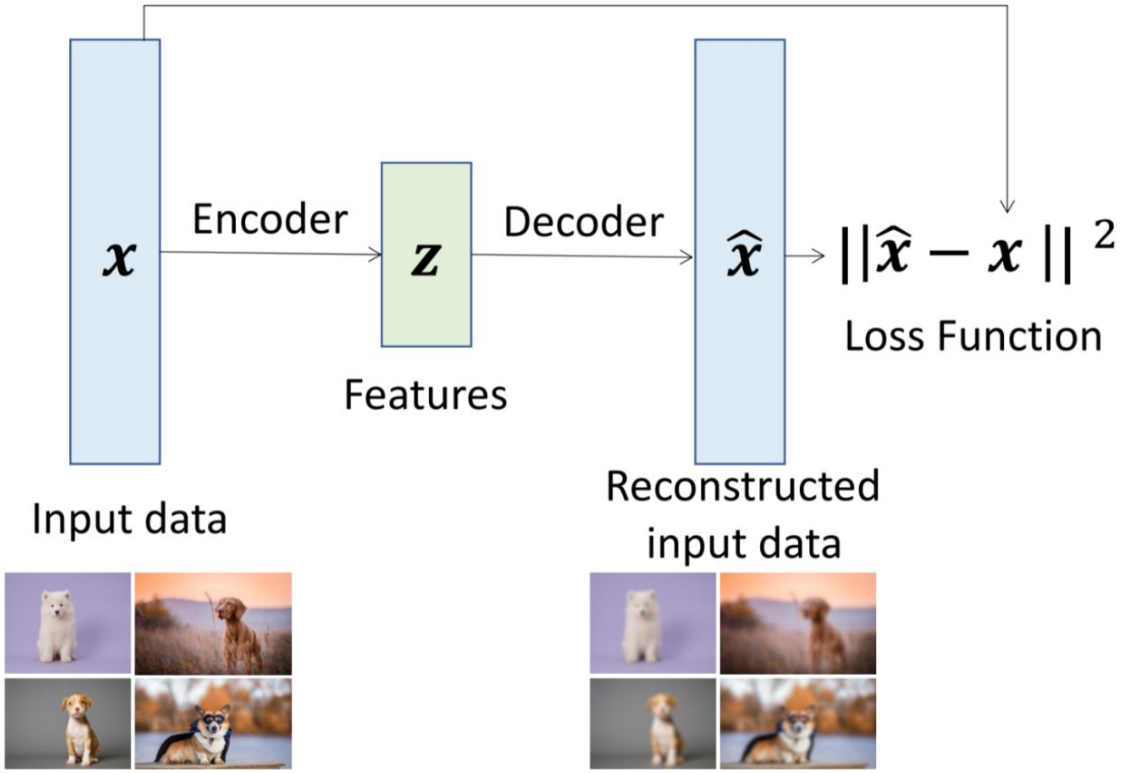
~~Variational~~ Autoencoders



During training:

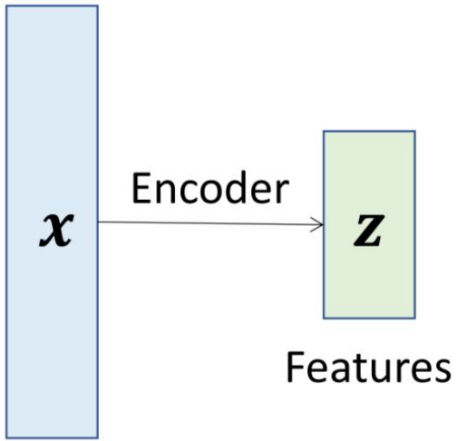
- We minimize an L2 distance between the input and the reconstructed data
- No labels used. Just data!

Autoencoders



The features z need to be lower dimensional than the data (→ dimensionality reduction, data compression)

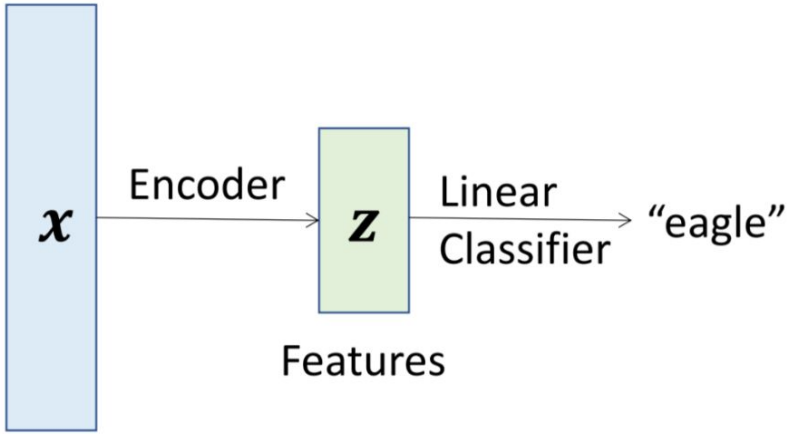
Autoencoders



After training:

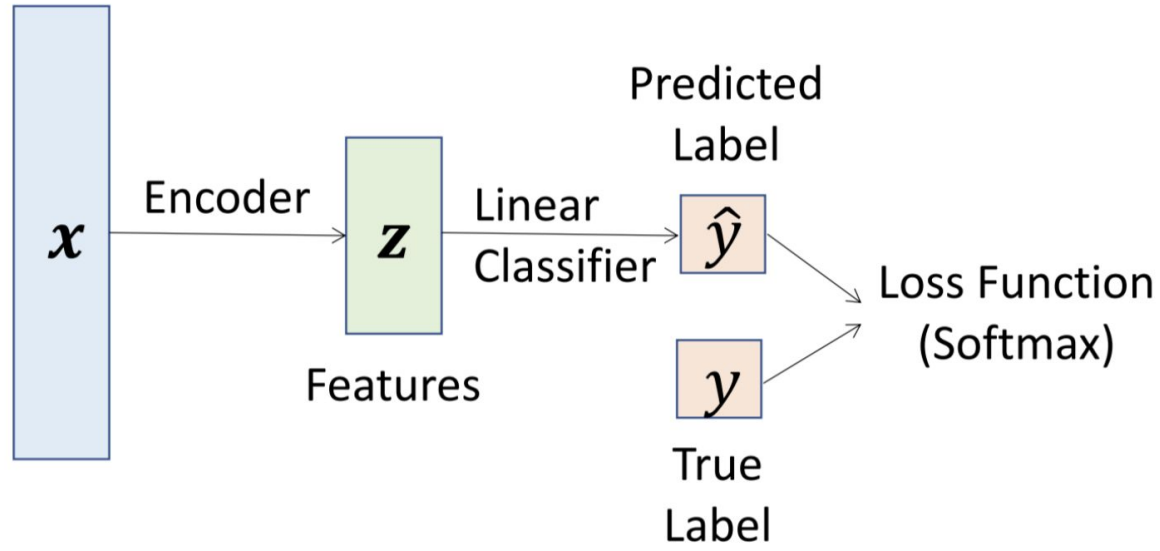
- Throw away the decoder
- Use encoder to encode input images to “useful” features
- These features can be used for downstream tasks because they encode useful bits about the input (since one can reconstruct the input from them)

Autoencoders: Downstream Tasks



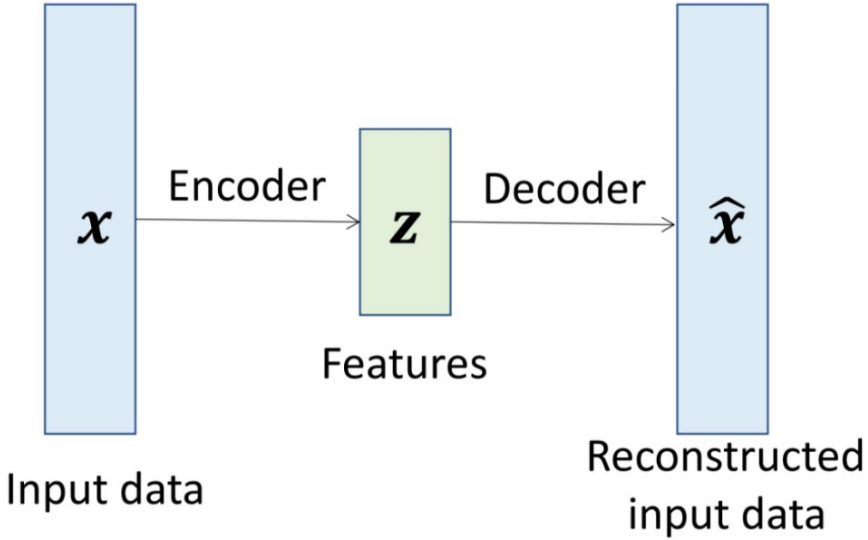
- A linear classifier inputs the features of the input image
- We train on a downstream supervised task with few annotated data points

Autoencoders: Downstream Tasks



Finetune the encoder and the linear classifier

Autoencoders

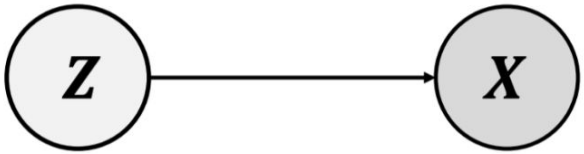


✓ Autoencoders learn latent features for data without any labels

✓ The features can be used to initialize a supervised task

But they are not probabilistic and cannot be used to sample new data from. Remember we want to capture $p(x)$ and sample from it.

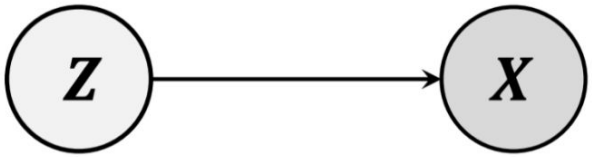
Variational Autoencoders



Assume we have training data $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^{N}$ which are generated by an unobserved (latent) representation $\boldsymbol{z}$.

For example, $\boldsymbol{x}$ is a dog image, and $\boldsymbol{z}$ is a latent vector which captures all the information one might need to generate an image of a dog, e.g. it has 2 eyes, 2 ears, 4 legs, it's cute etc.

Variational Autoencoders



$p(z)$ $p(x|z)$

The joint distribution

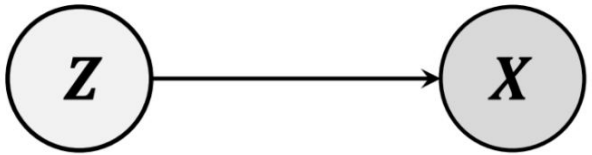
$$p(x, z) = p(x|z)p(z)$$

After training,

1. Sample from prior: $\hat{z} \sim p(z)$
2. Sample data from conditional: $\hat{x} \sim p(x|\hat{z})$

Intuitively, first generate the “high-level” semantic information about the data. Then generate the data.

Variational Autoencoders



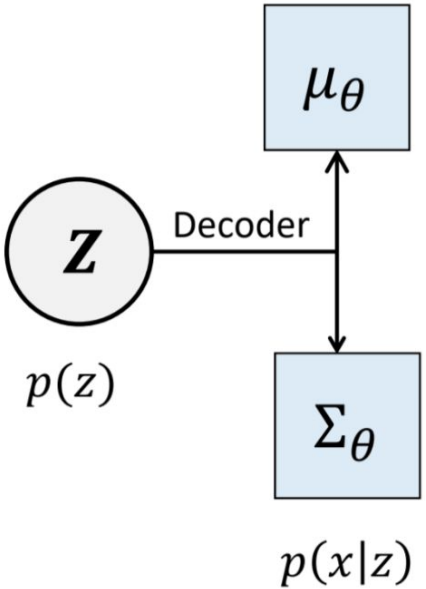
$p(z)$ $p(x|z)$

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

Represent $p(x|z)$ with a neural network

- The conditional distribution is parametrized, $p_\theta(x|z)$
- This neural network is the decoder

Variational Autoencoders

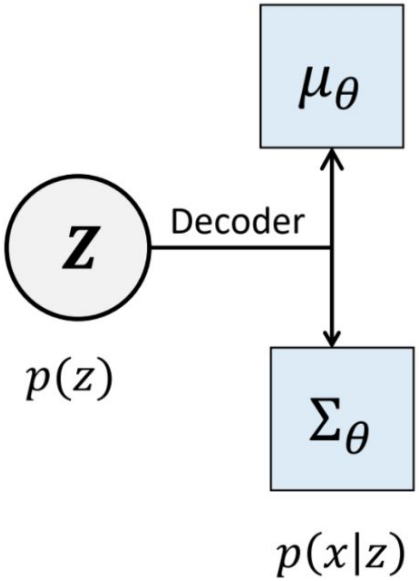


Unlike in original autoencoders, the decoder in variational autoencoders is probabilistic. It inputs z and outputs μ_θ and (diagonal) covariance Σ_θ

$$p(x|z) = N(\mu_\theta(z), \Sigma_\theta(z))$$

One can sample the decoder by sampling from a normal distribution with mean μ_θ and covariance Σ_θ

Variational Autoencoders

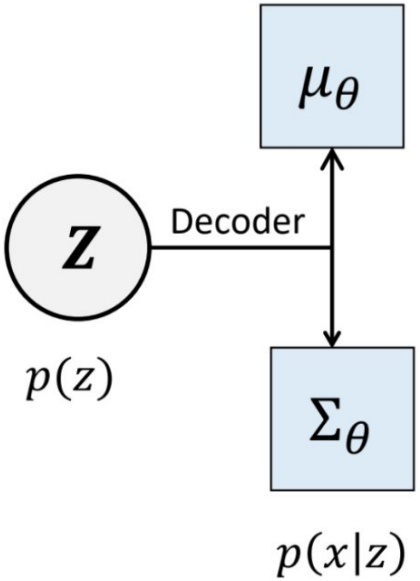


Density Estimation: We want to be able to estimate the density $p_\theta(x)$ of each datapoint x.

Note that we don't observe z, so to get the density we need to marginalize

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

Variational Autoencoders



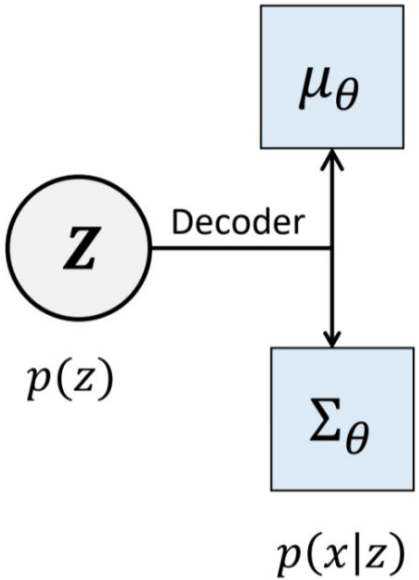
Density Estimation: We want to be able to estimate the density $p_\theta(x)$ of each datapoint x.

Note that we don't observe z, so to get the density we need to marginalize

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

Impossible to integrate over all z. Intractable!

Variational Autoencoders

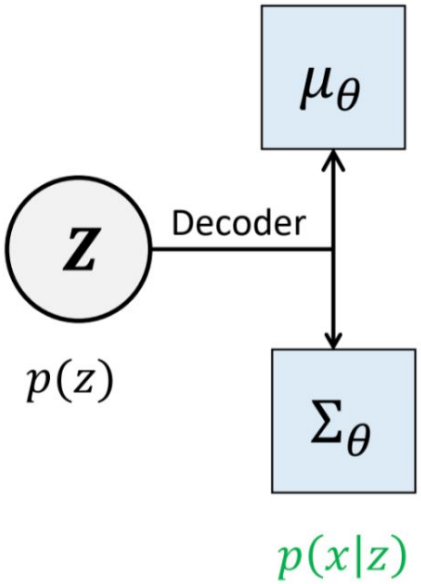


Density Estimation: We want to be able to estimate the density $p_\theta(x)$ of each datapoint x.

Bayes' rule says:

$$p(x) = \frac{p(x|z)p(z)}{p(z|x)}$$

Variational Autoencoders



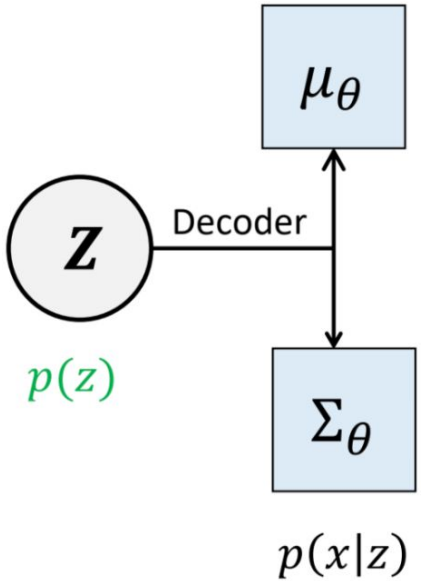
Density Estimation: We want to be able to estimate the density $p_\theta(x)$ of each datapoint x.

Bayes' rule says:

$$p(x) = \frac{p(x|z)p(z)}{p(z|x)}$$

Easy! My decoder network outputs $p(x|z)$ parametrized as a gaussian with μ_θ, Σ_θ

Variational Autoencoders



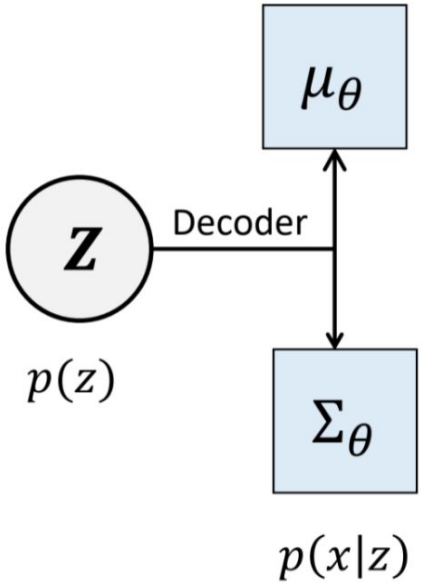
Density Estimation: We want to be able to estimate the density $p_\theta(x)$ of each datapoint x.

Bayes' rule says:

$$p(x) = \frac{p(x|z)p(z)}{p(z|x)}$$

Easy! I have assumed a Gaussian prior $N(0, I)$.

Variational Autoencoders



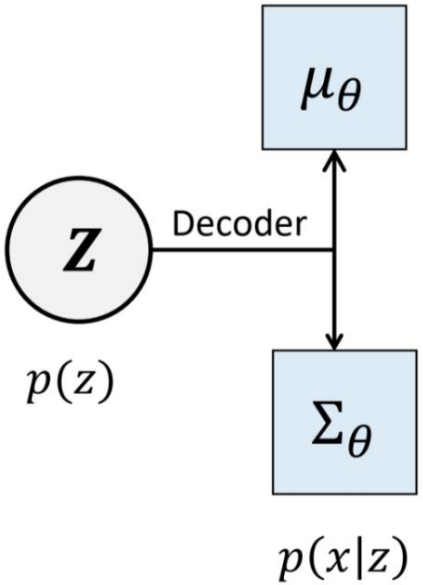
Density Estimation: We want to be able to estimate the density $p_\theta(x)$ of each datapoint x.

Bayes' rule says:

$$p(x) = \frac{p(x|z)p(z)}{p(z|x)}$$

Not easy! I have no way of computing this.

Variational Autoencoders

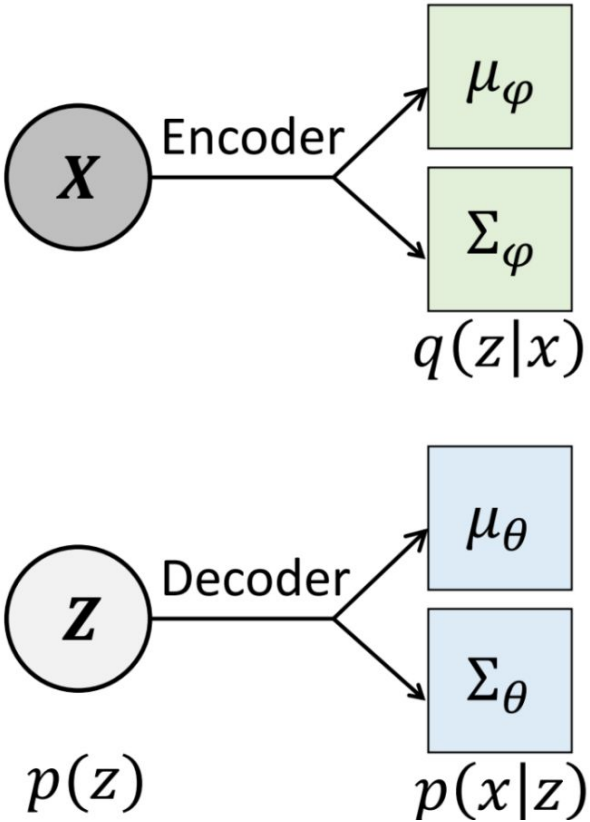


$$p(x) = \frac{p(x|z)p(z)}{p(z|x)}$$

We introduce a new parametric distribution $q(z|x)$which will approximate the unknown $\mathrm{p}(z|x)$

$$q(z|x) \approx \mathrm{p}(z|x)$$

Variational Autoencoders



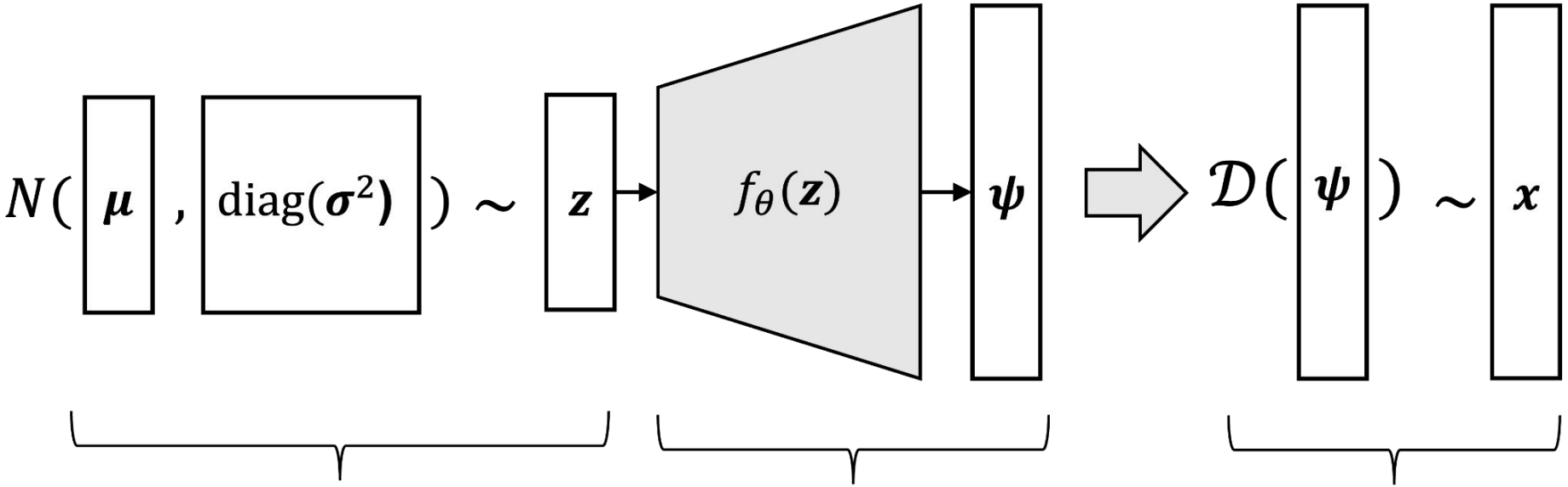
$$p(x) = \frac{p(x|z)\,p(z)}{p(z|x)} \approx \frac{p(x|z)p(z)}{q(z|x)}$$

$q(z|x)$ is parametrized by an encoder network with parameters φ which maps the input data to mean μ_φ and a **diagonal** covariance Σ_φ such that

$$q_\varphi(z|x) = \mathrm{N}(\mu_\varphi, \Sigma_\varphi)$$

Why diagonal Σ_φ?

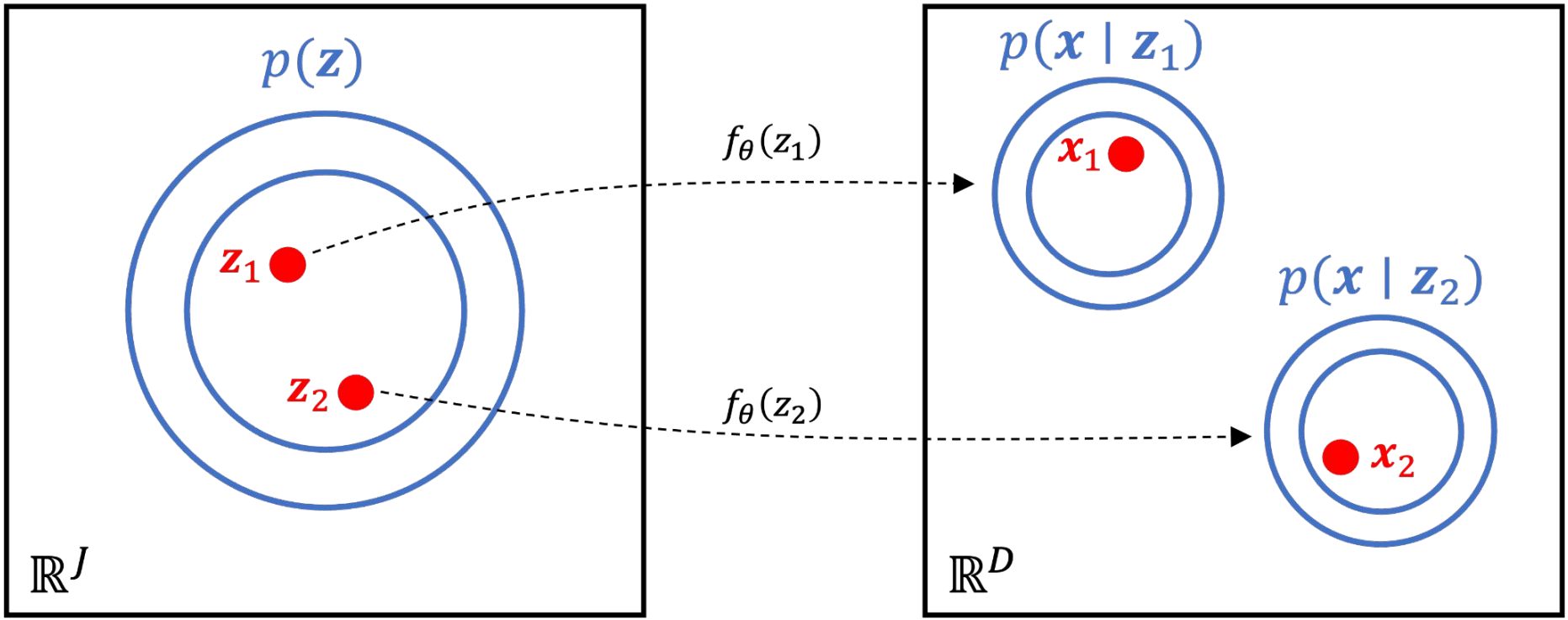
It makes computations easy.



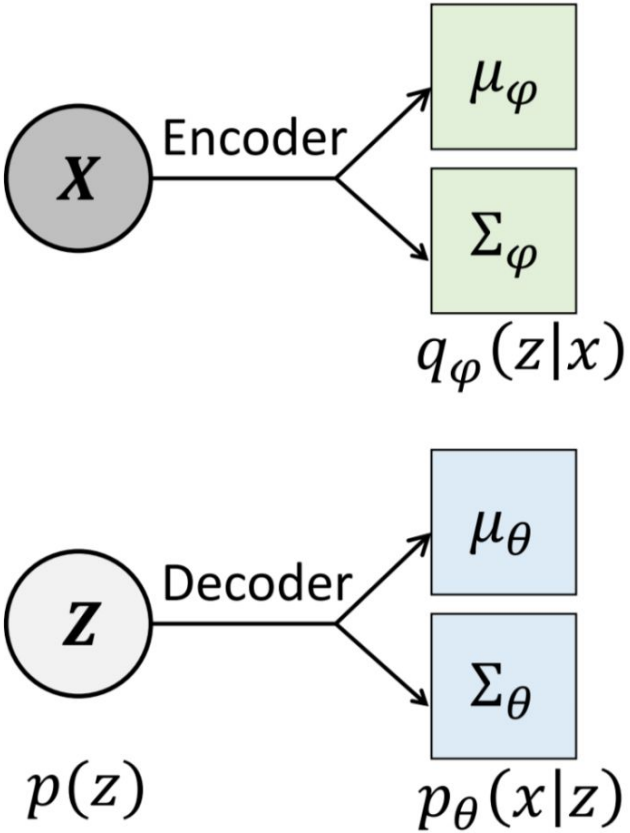
Sample $\boldsymbol{z}$ from a normal distribution parameterized by $\boldsymbol{\mu}$ and a diagonal covariance matrix with the values in $\boldsymbol{\sigma}^2$ along the diagonal

Decode $\boldsymbol{z}$ into the parameters, $\boldsymbol{\psi}$, used to sample $\boldsymbol{x}$

Sample $\boldsymbol{x}$


$p(z)$
z_1
z_2
$\mathbb{R}^J$
$f_\theta(z_1)$
$f_\theta(z_2)$
$p(x \mid z_1)$
x_1
$p(x \mid z_2)$
x_2
$\mathbb{R}^D$

Variational Autoencoders

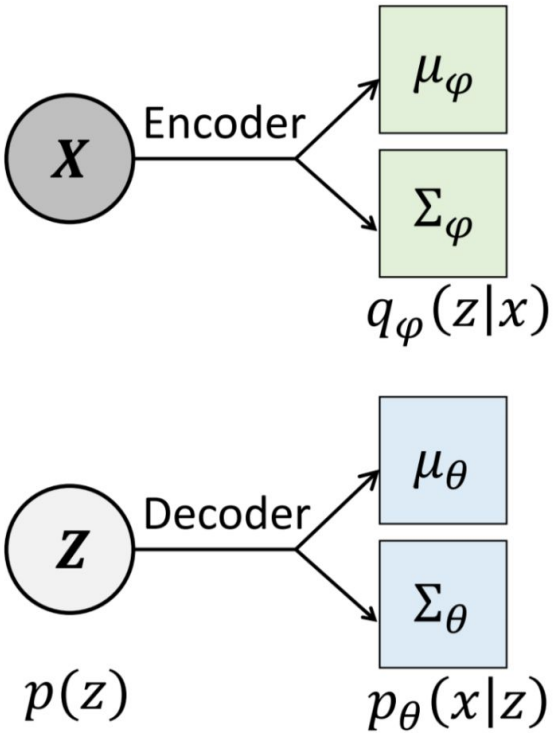


Optimization in VAEs involves jointly optimizing

1. The generative model parameters θ (decoder) to reduce the reconstruction error between input and output

2. The encoder parameters φ to model the posterior $q_\varphi(z|x)$

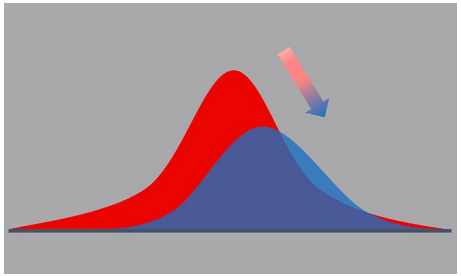
$$max_{\theta,\varphi}\, \mathcal{L}_{\theta,\varphi}(x) = \mathbb{E}_{z\sim q_\varphi}[\log p_\theta(x|z)] - d_{KL}(q_\varphi(z|x)|p(z))$$



Reconstruction Loss: This term measures given encoded latent code (z), how well we can reconstruct data (x)

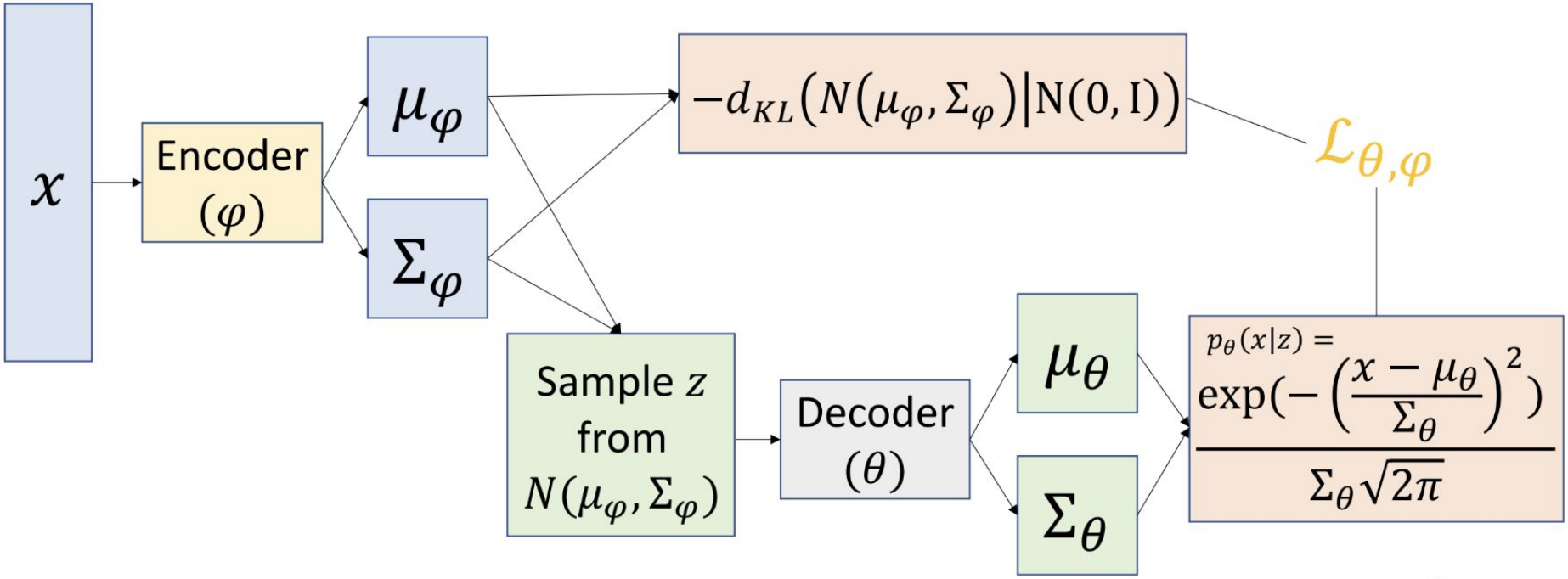
Encoder Prior Loss: This term measures the distance between encoded latent distribution q(z|x) to prior gaussian p(z).

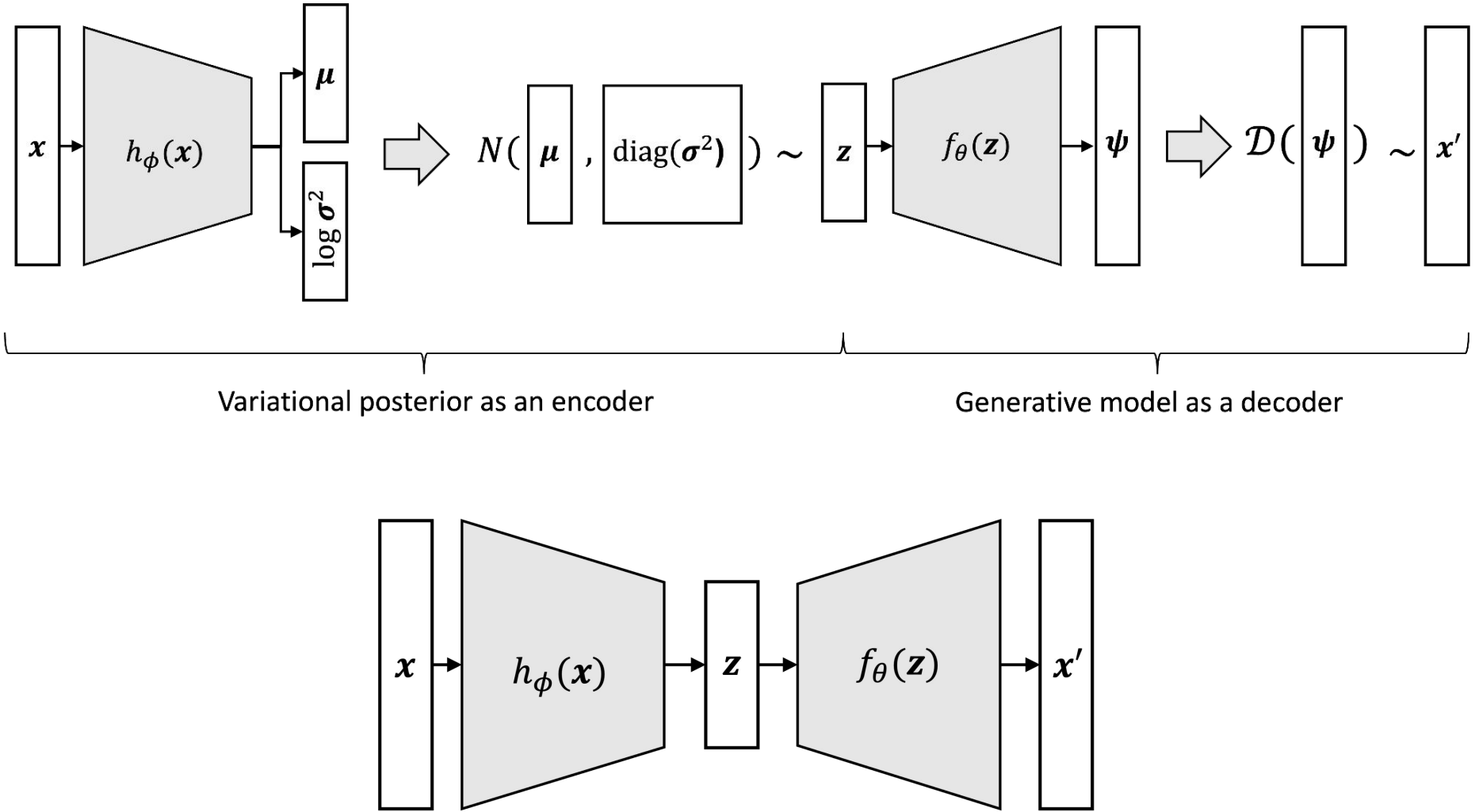
After this term becomes small



$$\mathcal{L}_{\theta,\varphi}(x) = \mathbb{E}_{z\sim q_\varphi}[\log p_\theta(x|z)] \; - \; d_{KL}(q_\varphi(z|x)|p(z))$$

Training VAEs

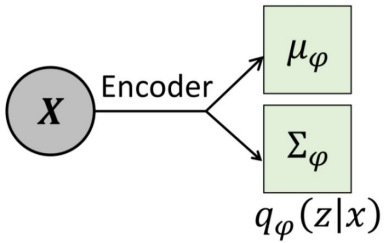


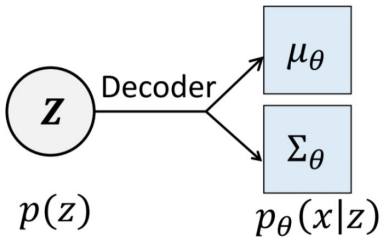

x
$h_\phi(x)$
μ
$\log \sigma^2$
$N(\ \mu\ ,\ \mathrm{diag}(\sigma^2)\ ) \sim z$
$f_\theta(z)$
ψ
$\mathcal{D}(\ \psi\ ) \sim x'$
Variational posterior as an encoder
Generative model as a decoder
x
$h_\phi(x)$
z
$f_\theta(z)$
x'

Optimizing the VAE objective

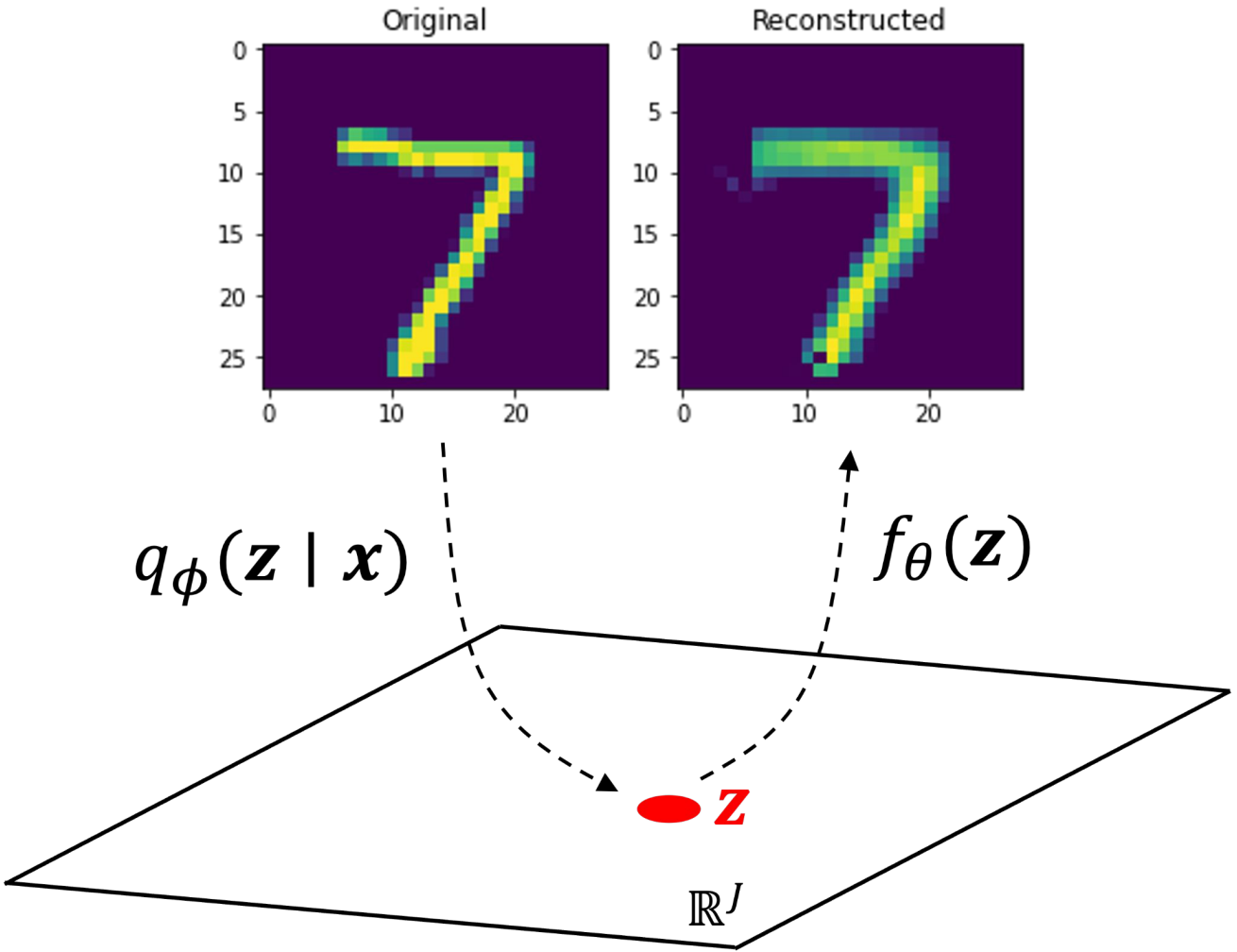
This term only depend on □

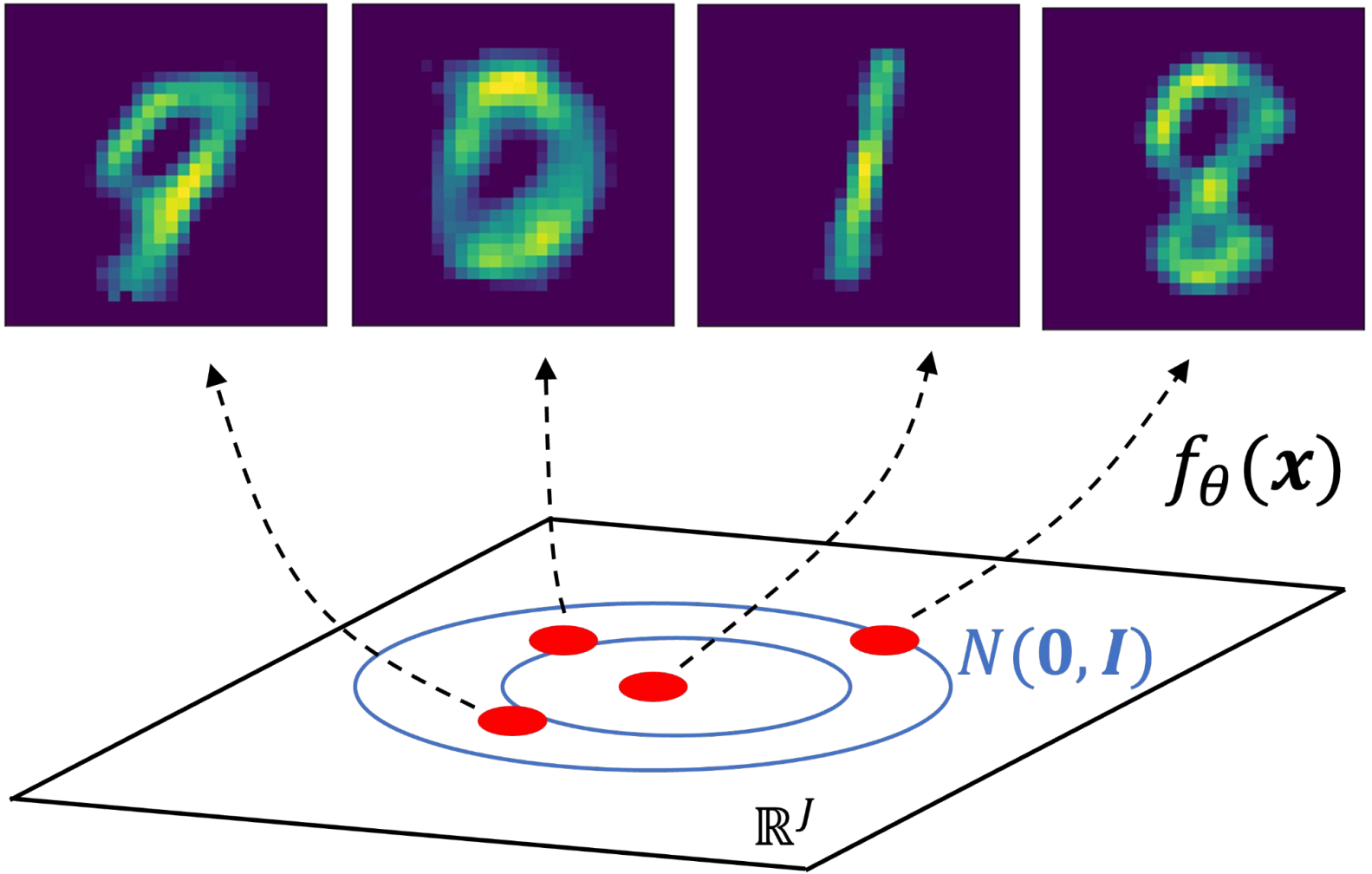
$$max_{\theta,\varphi}\, \mathcal{L}_{\theta,\varphi}(x) = \underbrace{\mathbb{E}_{z\sim q_\varphi}[\log p_\theta(x|z)]}_{\text{This term depends on both } \theta \text{ and } \varphi} - d_{KL}(q_\varphi(z|x)|p(z))$$

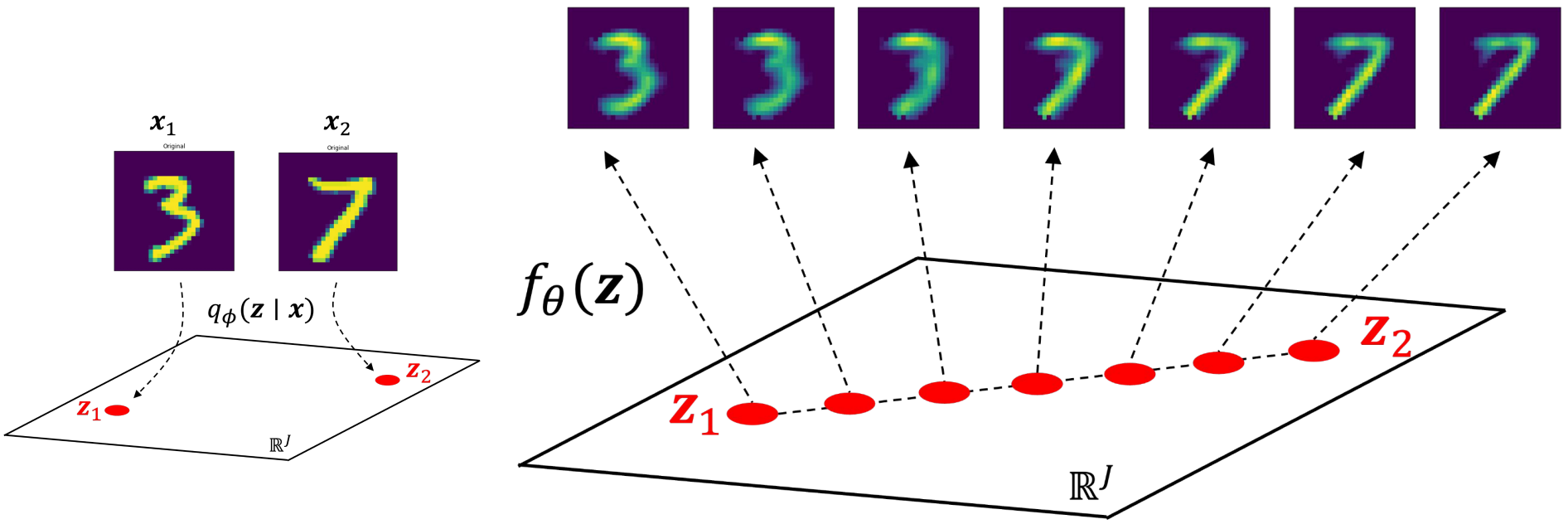




We will introduce how we get this in next lecture.


Original
Reconstructed
0
5
10
15
20
25
0
10
20
$q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$
$f_\theta(\boldsymbol{z})$
$\boldsymbol{z}$
$\mathbb{R}^J$


$f_\theta(\boldsymbol{x})$
$N(\mathbf{0}, \boldsymbol{I})$
$\mathbb{R}^J$


x_1
Original
x_2
Original
$q_\phi(z \mid x)$
z_1
z_2
$\mathbb{R}^J$
$f_\theta(z)$
z_1
z_2
$\mathbb{R}^J$

VAE Generations

The diagonal covariance constraint on the posterior forces the model to disentangle factors of variation in z.

MLE given fixed z (diagonal case)

- **Observation: if we knew** $z^{(n)}$ for every $\mathbf{x}^{(n)}$, (i.e. our dataset was $\mathcal{D}_{\text{complete}} = \{(z^{(n)}, \mathbf{x}^{(n)})\}_{n=1}^{N}$) the maximum likelihood problem is easy:

$$\begin{aligned}
\log p(\mathcal{D}_{\text{complete}}) &= \sum_{n=1}^{N} \log p(z^{(n)}, \mathbf{x}^{(n)}) \\
&= \sum_{n=1}^{N} \log p(\mathbf{x}^{(n)} | z^{(n)}) + \log p(z^{(n)}) \\
&= \sum_{n=1}^{N} \sum_{k=1}^{K} \mathbb{I}[z^{(n)} = k](\log \mathcal{N}(\mathbf{x}^{(n)} | \mu_k, I) + \log \pi_k)
\end{aligned}$$

MLE given fixed z (diagonal case)

$$\log p(\mathcal{D}_{\text{complete}}) = \sum_{n=1}^{N} \sum_{k=1}^{K} \mathbb{I}[z^{(n)} = k](\log \mathcal{N}(x^{(n)}|\mu_k, I) + \log \pi_k)$$

- We have been optimizing something similar for Gaussian bayes classifiers
- We would get this:

$$\mu_k = \frac{\sum_{n=1}^{N} \mathbb{I}[z^{(n)} = k]\, \mathbf{x}^{(n)}}{\sum_{n=1}^{N} \mathbb{I}[z^{(n)} = k]}$$

$$\pi_k = \frac{1}{N} \sum_{n=1}^{N} \mathbb{I}[z^{(n)} = k]$$

How to Estimate z (diagonal case)?

- We don't know $z^{(n)}$ for every $\mathbf{x}^{(n)}$, but we can compute $p(z^{(n)}|\mathbf{x}^{(n)})$ using Bayes rule
- Conditional probability (using Bayes rule) of $\mathbf{z}$ given $\mathbf{x}$

$$
\begin{aligned}
p(z = k|\mathbf{x}) &= \frac{p(z = k)p(\mathbf{x}|z = k)}{p(\mathbf{x})} \\
&= \frac{p(z = k)p(\mathbf{x}|z = k)}{\sum_{j=1}^{K} p(z = j)p(\mathbf{x}|z = j)} \\
&= \frac{\pi_k \mathcal{N}(\mathbf{x}|\mu_k, I)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}|\mu_j, I)}
\end{aligned}
$$

$$\log p(\mathcal{D}_{\text{complete}}) = \sum_{n=1}^{N} \sum_{k=1}^{K} \mathbb{I}[z^{(n)} = k](\log \mathcal{N}(x^{(n)}|\mu_k, I) + \log \pi_k)$$

- If we plug in $r_k^{(n)} = p(z^{(n)} = k|\mathbf{x}^{(n)})$ for $\mathbb{I}[z^{(n)} = k]$, we get:

$$\sum_{n=1}^{N} \sum_{k=1}^{K} r_k^{(n)}(\log \mathcal{N}(\mathbf{x}^{(n)}|\mu_k, I) + \log \pi_k)$$

- This is still easy to optimize! Solution is similar to what we have seen:

$$\mu_k = \frac{\sum_{n=1}^{N} r_k^{(n)} \mathbf{x}^{(n)}}{\sum_{n=1}^{N} r_k^{(n)}}$$

$$\pi_k = \frac{\sum_{n=1}^{N} r_k^{(n)}}{N}$$

- Note: this only works if we treat $r_k^{(n)}$ as fixed – really, it depends on the model parameters as well