Variational Autoencoders

Probabilistic Decision Making — Lecture 10

3rd December 2025

Robert Peharz
Institute of Machine Learning and Neural Computation
Graz University of Technology

Evidence Lower Bound (ELBO)

Latent variable model:

$$\log p(\boldsymbol{x}^{(i)} \mid \boldsymbol{\theta}) = \log \int p(\boldsymbol{x}^{(i)}, \boldsymbol{z}^{(i)} \mid \boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{z}^{(i)}$$

Sample-wise evidence lower bound (ELBO):

$$\begin{aligned}
\log p(\boldsymbol{x}^{(i)} \mid \boldsymbol{\theta}) &= \log \int \overbrace{\frac{q_i(\boldsymbol{z}^{(i)})}{q_i(\boldsymbol{z}^{(i)})}}^{=1} p(\boldsymbol{x}^{(i)}, \boldsymbol{z}^{(i)} \mid \boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{z}^{(i)} \\
&= \log \int q_i(\boldsymbol{z}^{(i)}) \frac{p(\boldsymbol{x}^{(i)}, \boldsymbol{z}^{(i)} \mid \boldsymbol{\theta})}{q_i(\boldsymbol{z}^{(i)})} \, \mathrm{d}\boldsymbol{z}^{(i)} \\
&= \log \mathbb{E}_{q_i} \left[\frac{p(\boldsymbol{x}^{(i)}, \boldsymbol{Z}^{(i)} \mid \boldsymbol{\theta})}{q_i(\boldsymbol{Z}^{(i)})} \right] \\
&\overset{\text{Jensen}}{\geq} \mathbb{E}_{q_i} \left[\log \frac{p(\boldsymbol{x}^{(i)}, \boldsymbol{Z}^{(i)} \mid \boldsymbol{\theta})}{q_i(\boldsymbol{Z}^{(i)})} \right] =: \mathrm{ELBO}(\boldsymbol{\theta}, q_i)
\end{aligned}$$

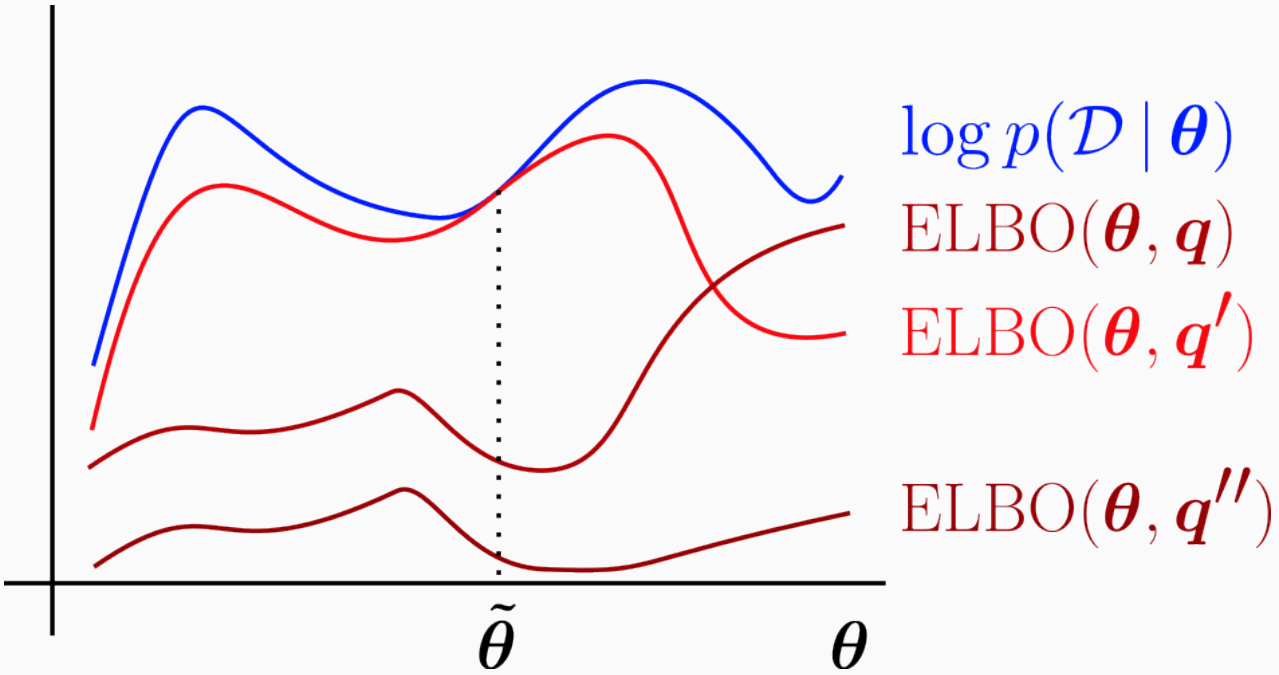
Lower bound approximation of the log-likelihood.

Evidence Lower Bound (ELBO)

(Total) ELBO:

$$\log p(\mathcal{D} \mid \boldsymbol{\theta}) = \sum_i \log p(\boldsymbol{x}^{(i)} \mid \boldsymbol{\theta}) \geq \sum_i \text{ELBO}(\boldsymbol{\theta}, q_i) =: \text{ELBO}(\boldsymbol{\theta}, \boldsymbol{q}),$$

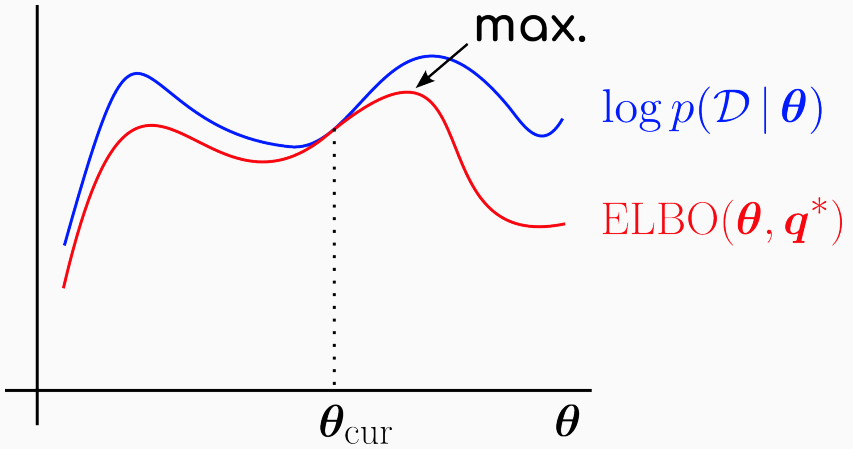
where $\boldsymbol{q} = \{q_i\}_{i=1}^N$ is the set of all **variational distributions.**



$$\log p(\mathcal{D} \mid \boldsymbol{\theta}) = \text{ELBO}(\boldsymbol{\theta}, \boldsymbol{q}) + \underbrace{\sum_i \text{KL}(q_i || p(\boldsymbol{z}^{(i)} | \boldsymbol{x}^{(i)}, \boldsymbol{\theta}))}_{\textbf{(total) variational gap}}$$

When $q_i^* \equiv p(\boldsymbol{z}^{(i)} | \boldsymbol{x}^{(i)}, \boldsymbol{\theta}_{\text{cur}})$ for all $\boldsymbol{q}^* = \{q_i^*\}_{i=1}^N$, then ELBO is tight at $\boldsymbol{\theta}_{\text{cur}}$.

Maximizing $\text{ELBO}(\boldsymbol{\theta}, \boldsymbol{q}^*)$ is a **minorization-maximization** method aka **expectation-maximization** (EM).



Variational Autoencoders

Variational Autoencoders (VAEs)

- **variational autoencoders** (**VAEs**) are "deep generative models," combining mixture models and neural networks
- the probabilistic model, the **density network**, was already proposed by MacKay (1995), but learning and inference were slow and hard
- Kingma and Welling (2014) as well as Rezende et al. (2014) used several tricks to make them work:
  - **amortized variational inference**
  - **"re-parametrization" trick**
  - stochastic optimization
- this combination of model and learning/inference techniques can be understood as an autoencoder structure, hence the name "variational autoencoder"

Let $\boldsymbol{Z}$ be a K-dimensional Gaussian with zero mean and identity covariance, a latent RV often called **code**:

$$p(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, I)$$

Let a neural network with parameters $\boldsymbol{\theta}$ be given, the so-called **decoder**, that takes a sample $\boldsymbol{z}$ as input and returns two D-dimensional vectors

$$\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}), \boldsymbol{\sigma}^2_{\boldsymbol{\theta}}(\boldsymbol{z})$$

They specify a Gaussian conditional distribution

$$p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}), \mathrm{diag}(\boldsymbol{\sigma}^2_{\boldsymbol{\theta}}(\boldsymbol{z}))),$$

where $\mathrm{diag}(\boldsymbol{v})$ is the diagonal matrix with vector $\boldsymbol{v}$ as diagonal.

$$p(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, I) \qquad \textbf{prior}$$

$$p(\boldsymbol{x} \mid \boldsymbol{z}, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}), \mathrm{diag}(\boldsymbol{\sigma}^2_{\boldsymbol{\theta}}(\boldsymbol{z}))) \qquad \textbf{decoder}$$

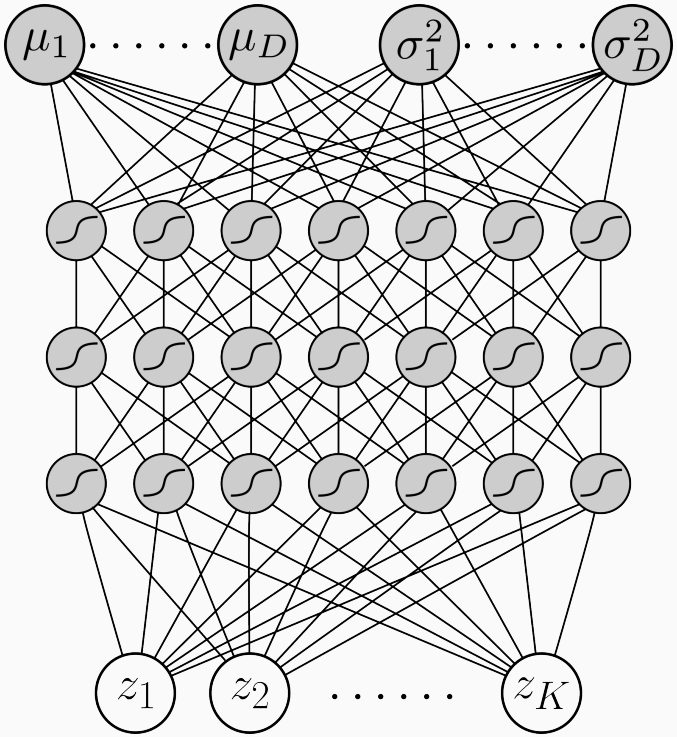
Prior and decoder give rise to a joint:

$$p(\boldsymbol{x}, \boldsymbol{z} \mid \boldsymbol{\theta}) = p(\boldsymbol{x} \mid \boldsymbol{z}, \boldsymbol{\theta})\, p(\boldsymbol{z})$$

Treating $\boldsymbol{Z}$ as latent variable we get the **density network** model:

$$\begin{aligned} p(\boldsymbol{x} \mid \boldsymbol{\theta}) &= \int p(\boldsymbol{x}, \boldsymbol{z} \mid \boldsymbol{\theta})\, \mathrm{d}\boldsymbol{z} \\ &= \int p(\boldsymbol{x} \mid \boldsymbol{z}, \boldsymbol{\theta})\, p(\boldsymbol{z})\, \mathrm{d}\boldsymbol{z} \end{aligned}$$

Note that the decoder parameters $\boldsymbol{\theta}$ (i.e. neural network weights) are the only parameters of the model.

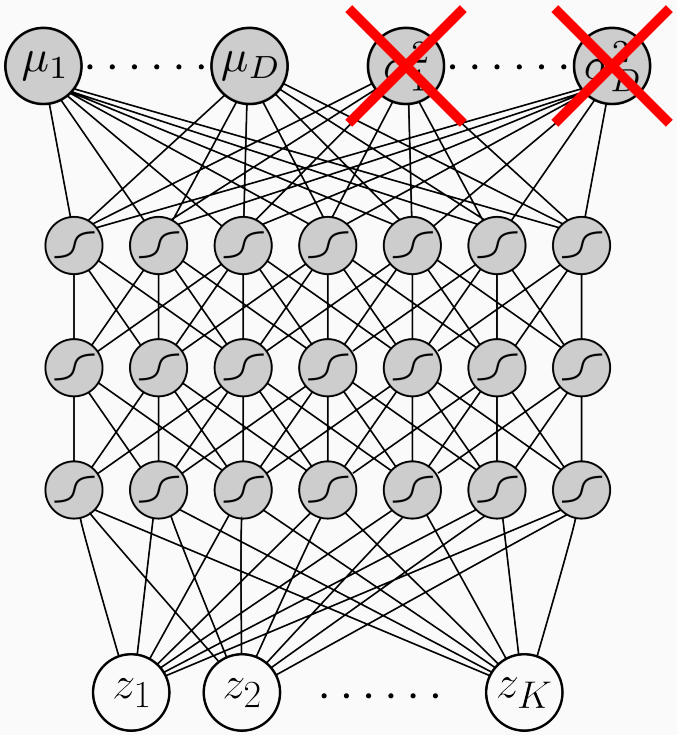


Density Network with Fixed Variances

Often, the density network is simplified to just output a mean, while the variances are fixed:

$$p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z}) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}), \sigma^2 I),$$

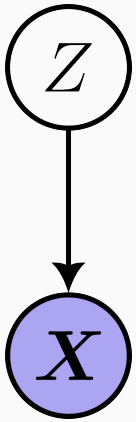
where σ^2 is some hyperparameter (might be fixed or trained as well).



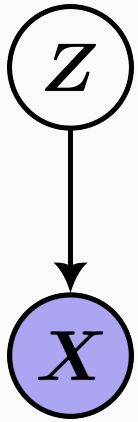
Density Networks vs. GMMs

Density networks and GMMs are close relatives. Both are Gaussian mixture models:

GMM

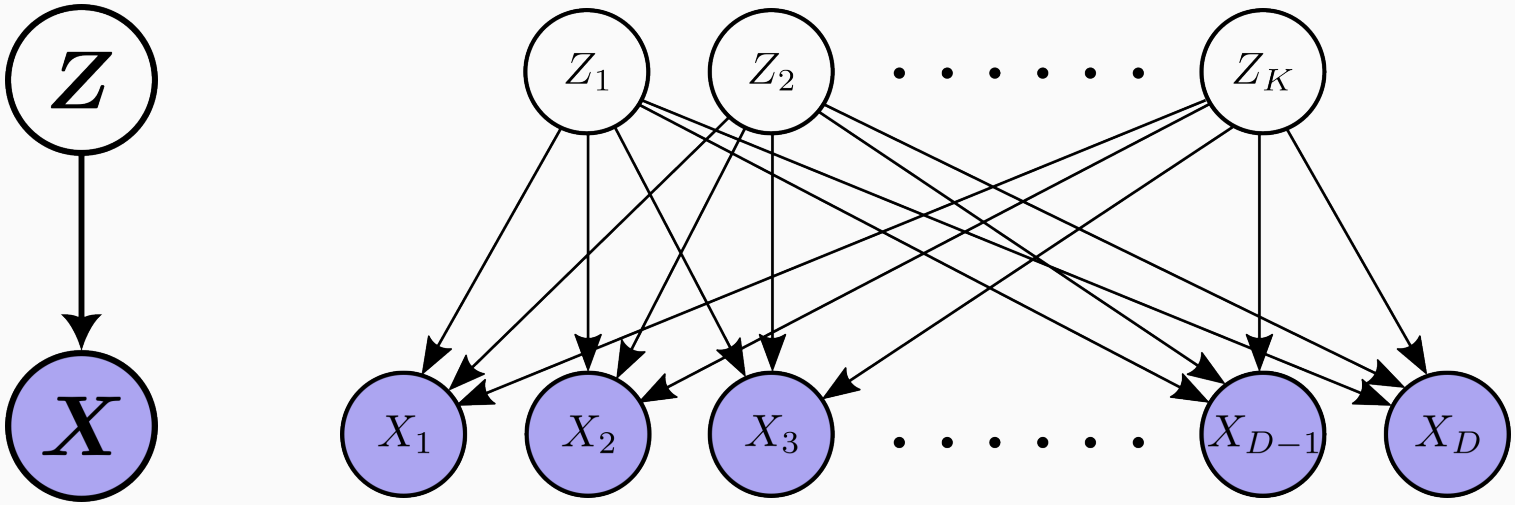


density network



The main difference is that GMMs are **finite** mixture models, while density networks are **infinite mixture models**.

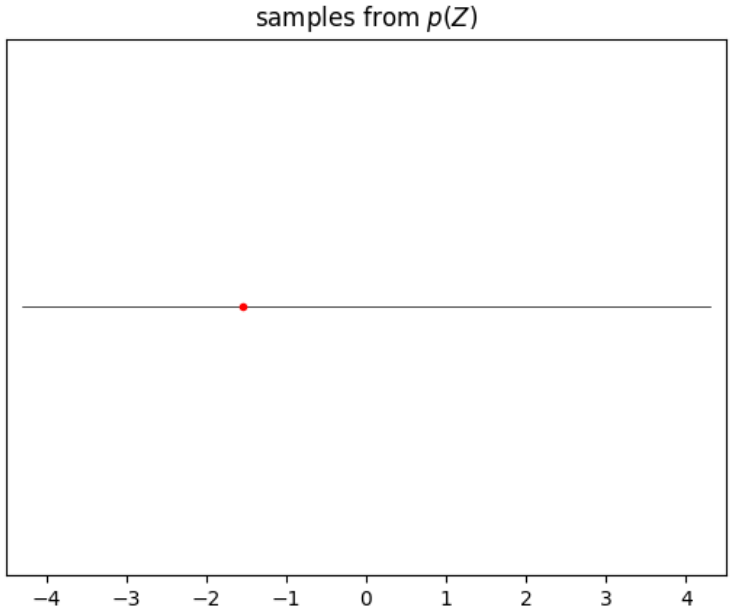
Density Networks as Bayesian Network

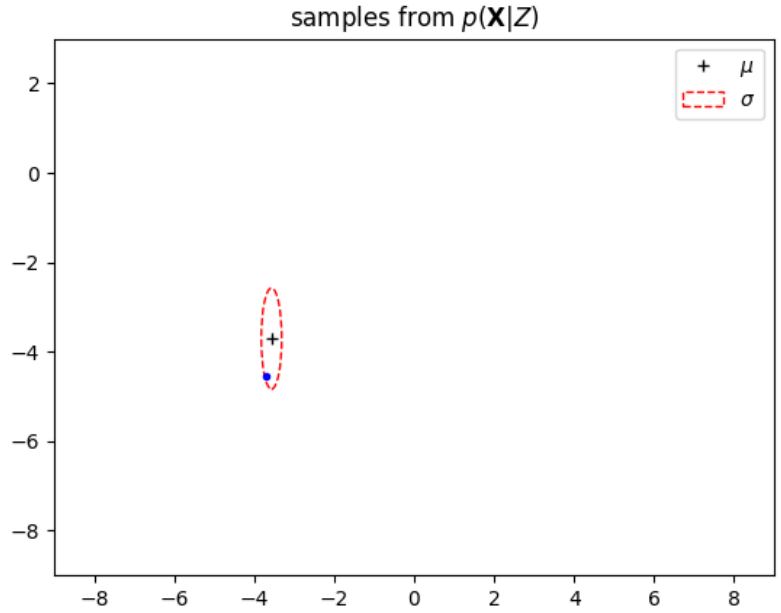


Left: density network as Bayesian network over vectors $\boldsymbol{Z}$ and $\boldsymbol{X}$

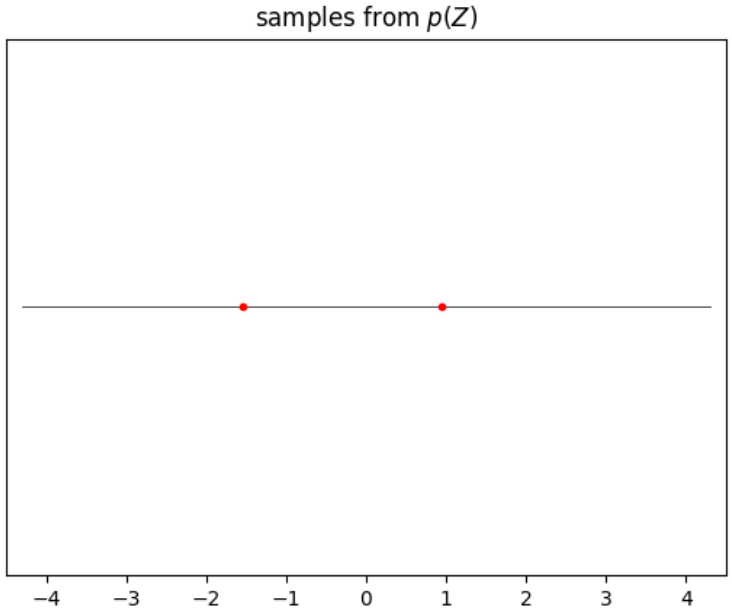
Right: density network as Bayesian network over single RVs

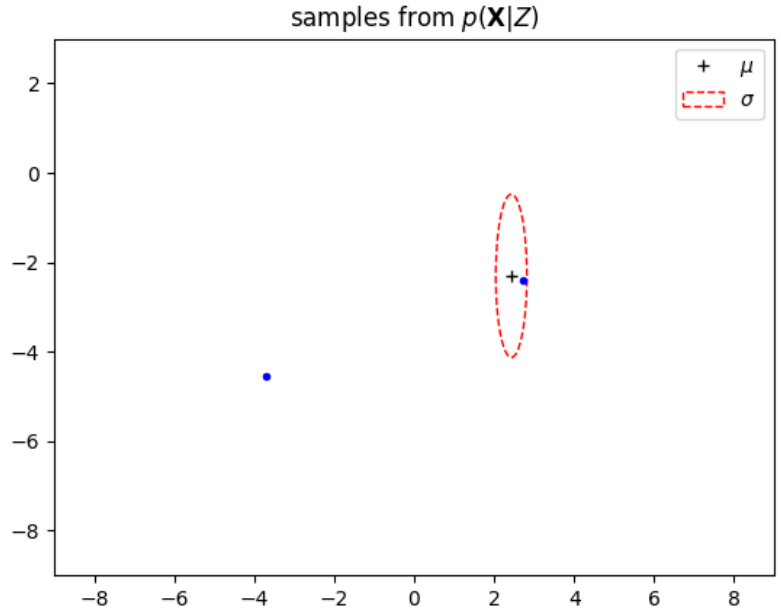
A toy density network with $K = 1$ (dimensionality of Z) and $D = 2$ (dimensionality of $\boldsymbol{X}$). Each sample of Z is transformed into a Gaussian over $\boldsymbol{X}$.



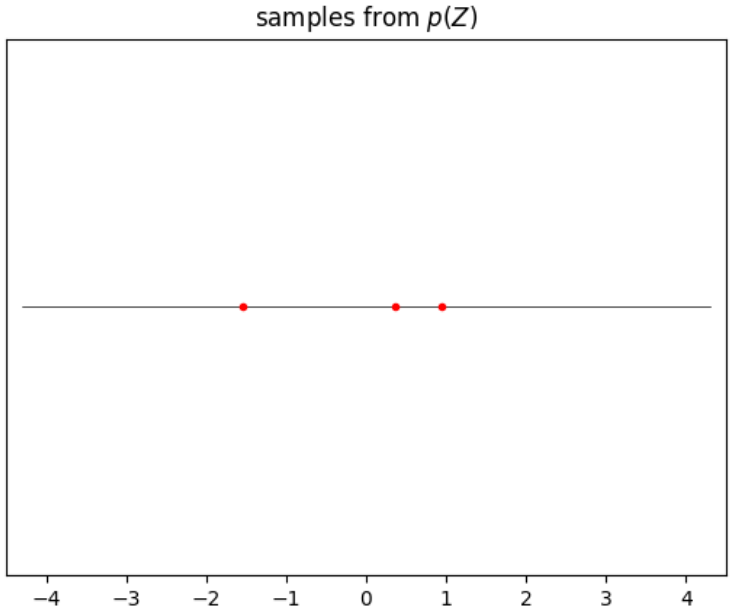


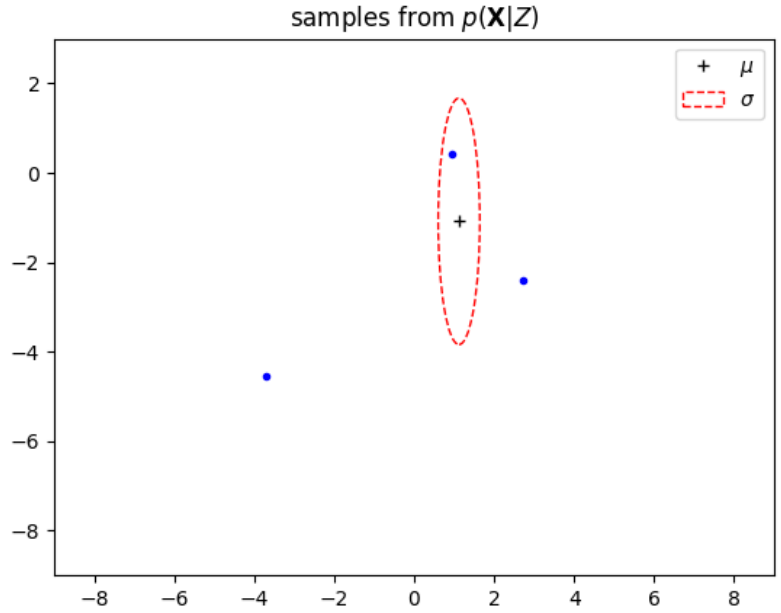
A toy density network with $K = 1$ (dimensionality of Z) and $D = 2$ (dimensionality of $\boldsymbol{X}$). Each sample of Z is transformed into a Gaussian over $\boldsymbol{X}$.



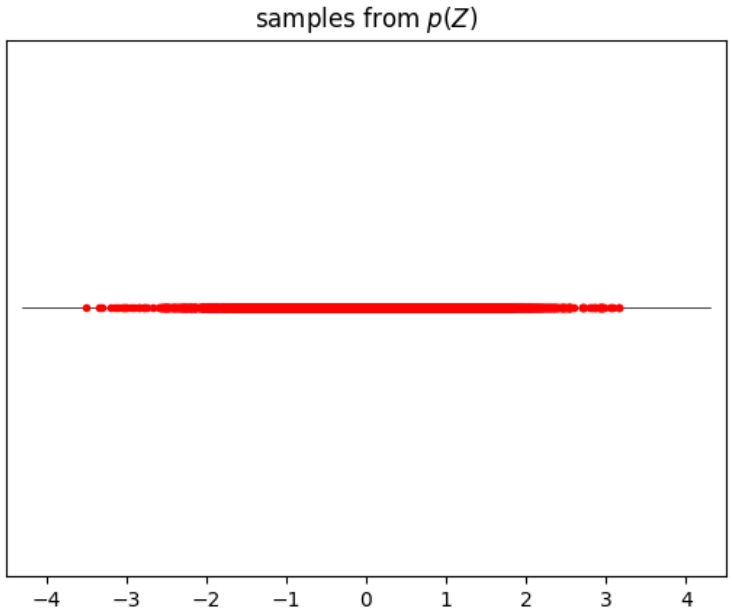


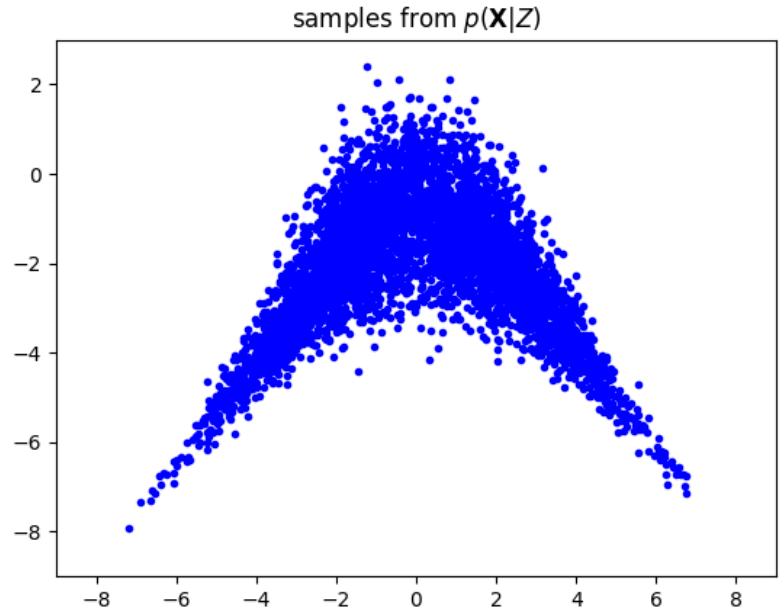
A toy density network with $K = 1$ (dimensionality of Z) and $D = 2$ (dimensionality of $\boldsymbol{X}$). Each sample of Z is transformed into a Gaussian over $\boldsymbol{X}$.





A toy density network with $K = 1$ (dimensionality of Z) and $D = 2$ (dimensionality of $\boldsymbol{X}$). Each sample of Z is transformed into a Gaussian over $\boldsymbol{X}$.





How to Learn Density Networks?

Ideally, we would like to train by maximum likelihood:

$$\arg\max_{\boldsymbol{\theta}} \log p(\mathcal{D} \mid \boldsymbol{\theta}) = \sum_{i=1}^{N} \log p(\boldsymbol{x}^{(i)} \mid \boldsymbol{\theta})$$

However, the sample-wise likelihoods are defined as

$$p(\boldsymbol{x} \mid \boldsymbol{\theta}) = \int \overbrace{p(\boldsymbol{x} \mid \boldsymbol{z}, \boldsymbol{\theta})}^{\mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}), \mathrm{diag}(\boldsymbol{\sigma}^2_{\boldsymbol{\theta}}(\boldsymbol{z})))} p(\boldsymbol{z}) \, \mathrm{d}\boldsymbol{z},$$

a K-dimensional integral over a non-linear function $\boldsymbol{\mu}_{\boldsymbol{\theta}}, \boldsymbol{\sigma}^2_{\boldsymbol{\theta}}$.

This integral is analytically completely intractable.

Expectation-Maximization?

Can we employ the EM algorithm?

The key step is computing for each sample the posterior over latent variables given observed variable, using the current model:

$$p(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{\theta}) = \frac{p(\boldsymbol{x} \mid \boldsymbol{z}, \boldsymbol{\theta})\, p(\boldsymbol{z})}{p(\boldsymbol{x} \mid \boldsymbol{\theta})} = \frac{p(\boldsymbol{x} \mid \boldsymbol{z}, \boldsymbol{\theta})\, p(\boldsymbol{z})}{\int p(\boldsymbol{x} \mid \boldsymbol{z}, \boldsymbol{\theta})\, p(\boldsymbol{z})\, \mathrm{d}\boldsymbol{z}}$$

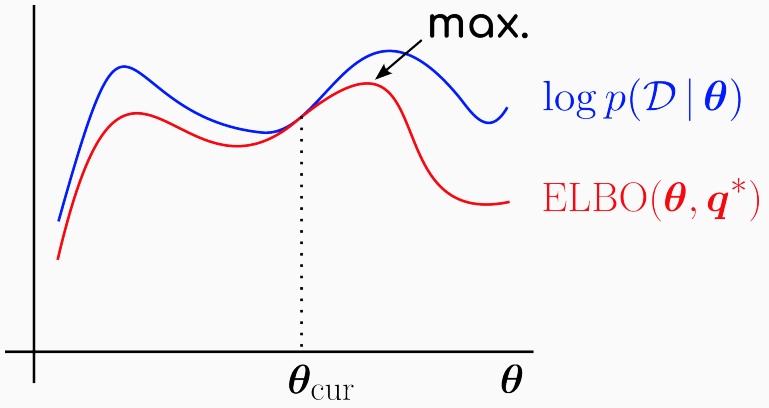
Same integral as before, intractable.

Variational EM

Recall that EM is just a minorization-maximization technique.

$$\log p(\mathcal{D} \mid \boldsymbol{\theta}) = \text{ELBO}(\boldsymbol{\theta}, \boldsymbol{q}) + \sum_i \text{KL}(q_i || p(\boldsymbol{z}^{(i)} | \boldsymbol{x}^{(i)}, \boldsymbol{\theta}))$$

Tight ELBO: $q_i^* \equiv p(\boldsymbol{z}^{(i)} | \boldsymbol{x}^{(i)}, \boldsymbol{\theta}_{\text{cur}})$



However, we might just give up on a tight ELBO and rather follow a **variational EM approach**:

$$\max_{\boldsymbol{\theta}, \boldsymbol{q}} \text{ELBO}(\boldsymbol{\theta}, \boldsymbol{q})$$

max w.r.t. $\boldsymbol{q}$: improve ELBO quality

max w.r.t. $\boldsymbol{\theta}$: learn model with ELBO as log-likelihood surrogate

How to represent the variational distributions?

$$\max_{\boldsymbol{\theta}, \boldsymbol{q}} \text{ELBO}(\boldsymbol{\theta}, \boldsymbol{q})$$

- $\boldsymbol{\theta}$ are the neural network weights of the decoder
- $\boldsymbol{q} = \{q_i\}_{i=1}^N$ are the variational distributions

We might use parametric q_i's, for example diagonal Gaussians.

However, we need **one q_i for each sample**—for large N this becomes quite a large number of variational parameters to be learned. Moreover, as soon as $\boldsymbol{\theta}$ is changed, the $\boldsymbol{q}$ need to be adapted as well.

Is there a more compact way to represent $\boldsymbol{q}$?

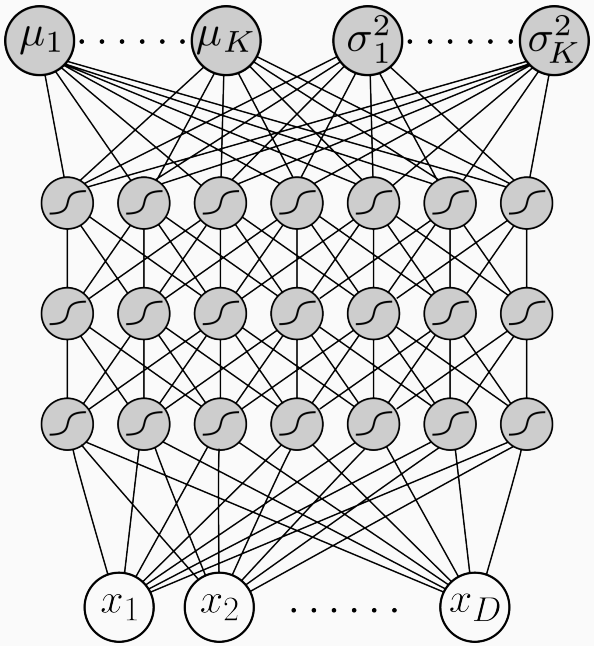
Inference Network, Encoder

Before, we introduced the **decoder**, that represents a conditional $p(\boldsymbol{x} \mid \boldsymbol{z})$. Why not do this "trick" again and use a neural network for representing $q_i(\boldsymbol{z}^{(i)}) := q(\boldsymbol{z}^{(i)} \mid \boldsymbol{x}^{(i)})$?

Specifically, we might construct a neural network with **variational parameters** ϕ that takes $\boldsymbol{x}$ as input and returns two K-dimensional vectors $\boldsymbol{\mu}_\phi(\boldsymbol{x}), \boldsymbol{\sigma}^2_\phi(\boldsymbol{x})$ specifiying

$$p(\boldsymbol{z} \mid \boldsymbol{x}, \phi) = \mathcal{N}(\boldsymbol{\mu}_\phi(\boldsymbol{x}), \text{diag}(\boldsymbol{\sigma}^2_\phi(\boldsymbol{x})))$$

We call this network the **inference network** or **encoder**. The fact that we represent **all** variational distributions with a **single** neural net is called **amortized inference**.



Overview

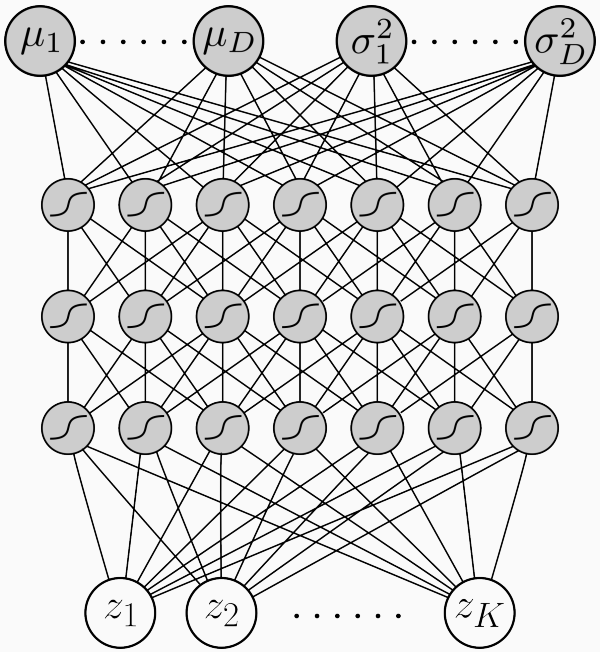
$p(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, I)$ **prior**

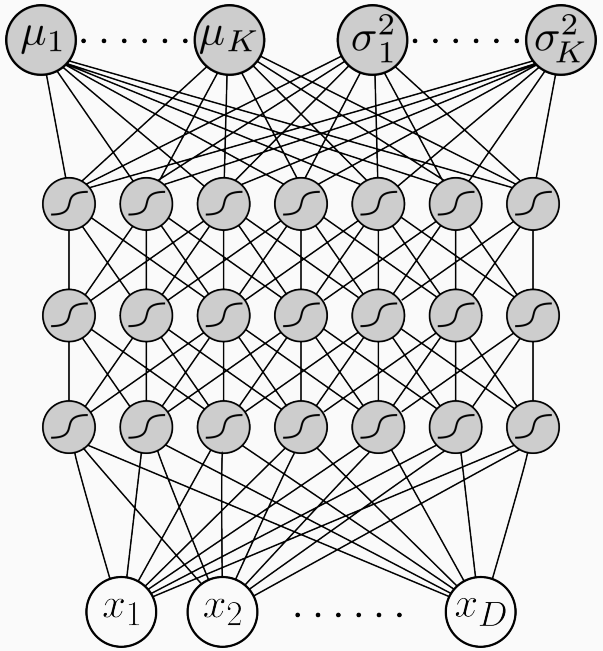
$\boldsymbol{\theta}$	**decoder params**	$\boldsymbol{\phi}$	**encoder params**
$\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}), \boldsymbol{\sigma}^2_{\boldsymbol{\theta}}(\boldsymbol{z})$	**decoder outputs**	$\boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}), \boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\boldsymbol{x})$	**encoder outputs**

$$p(\boldsymbol{x} \mid \boldsymbol{z}, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\theta}}(\boldsymbol{z}), \mathrm{diag}(\boldsymbol{\sigma}^2_{\boldsymbol{\theta}}(\boldsymbol{z})))$$

$$q(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{\phi}) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}), \mathrm{diag}(\boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\boldsymbol{x})))$$

$$p(\boldsymbol{x} \mid \boldsymbol{\theta}) = \int p(\boldsymbol{x} \mid \boldsymbol{z}, \boldsymbol{\theta})\, p(\boldsymbol{z})\, \mathrm{d}\boldsymbol{z}$$





Learning Algorithm

Variational EM

The overall learning problem is now

$$\max_{\boldsymbol{\theta},\boldsymbol{\phi}} \text{ ELBO}(\boldsymbol{\theta}, \boldsymbol{\phi})$$

- **maximization w.r.t. $\boldsymbol{\phi}$:** learn inference network, tighten lower bound
- **maximization w.r.t. $\boldsymbol{\theta}$:** learn generative model

Basic idea: apply gradient ascent simultaneously to $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$, meaning we need

$$\nabla_{\boldsymbol{\theta}} \text{ ELBO}(\boldsymbol{\theta}, \boldsymbol{\phi}) \qquad \text{and} \qquad \nabla_{\boldsymbol{\phi}} \text{ ELBO}(\boldsymbol{\theta}, \boldsymbol{\phi})$$

Spoiler: we will get only an (unbiased) estimate of the gradient.

Decomposing ELBO

In the following, let $\mathbb{E}_{q_i}[\cdot]$ be short for $\mathbb{E}_{q(\boldsymbol{z}^{(i)}|\boldsymbol{x}^{(i)},\boldsymbol{\phi})}[\cdot]$

Recall that the overall ELBO is the sum of sample-wise ELBOs:

$$\text{ELBO}(\boldsymbol{\theta},\boldsymbol{\phi}) = \sum_{i=1}^{N} \text{ELBO}_i(\boldsymbol{\theta},\boldsymbol{\phi}) = \sum_{i=1}^{N} \mathbb{E}_{q_i}\left[\log \frac{p(\boldsymbol{x}^{(i)}, \boldsymbol{Z}^{(i)} \mid \boldsymbol{\theta})}{q(\boldsymbol{Z}^{(i)}|\boldsymbol{x}^{(i)},\boldsymbol{\phi})}\right]$$

$$\begin{aligned}
\text{ELBO}_i(\boldsymbol{\theta},\boldsymbol{\phi}) &= \mathbb{E}_{q_i}\left[\log \frac{p(\boldsymbol{x}^{(i)}, \boldsymbol{Z}^{(i)} \mid \boldsymbol{\theta})}{q(\boldsymbol{Z}^{(i)}|\boldsymbol{x}^{(i)},\boldsymbol{\phi})}\right] \\
&= \mathbb{E}_{q_i}\left[\log \frac{p(\boldsymbol{x}^{(i)} \mid \boldsymbol{Z}^{(i)}, \boldsymbol{\theta})\, p(\boldsymbol{Z}^{(i)})}{q(\boldsymbol{Z}^{(i)}|\boldsymbol{x}^{(i)},\boldsymbol{\phi})}\right] \\
&= \mathbb{E}_{q_i}\left[\log p(\boldsymbol{x}^{(i)} \mid \boldsymbol{Z}^{(i)}, \boldsymbol{\theta})\right] + \mathbb{E}_{q_i}\left[\log \frac{p(\boldsymbol{Z}^{(i)})}{q(\boldsymbol{Z}^{(i)}|\boldsymbol{x}^{(i)},\boldsymbol{\phi})}\right] \\
&= \mathbb{E}_{q_i}\left[\log p(\boldsymbol{x}^{(i)} \mid \boldsymbol{Z}^{(i)}, \boldsymbol{\theta})\right] - \mathbb{E}_{q_i}\left[\log \frac{q(\boldsymbol{Z}^{(i)}|\boldsymbol{x}^{(i)},\boldsymbol{\phi})}{p(\boldsymbol{Z}^{(i)})}\right] \\
&= \mathbb{E}_{q_i}\left[\log p(\boldsymbol{x}^{(i)} \mid \boldsymbol{Z}^{(i)}, \boldsymbol{\theta})\right] - \text{KL}(q_i || p(\boldsymbol{z}^{(i)})
\end{aligned}$$

Interlude: Different ELBO Decompositions

The last time we saw

$$\text{ELBO}_i(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{q_i}\left[\log p(\boldsymbol{x}^{(i)}, \boldsymbol{Z}^{(i)} \mid \boldsymbol{\theta})\right] + H(q_i)$$

and

$$\text{ELBO}_i(\boldsymbol{\theta}, \boldsymbol{\phi}) = \log p(\boldsymbol{x}^{(i)} \mid \boldsymbol{\theta}) - \text{KL}(q_i || p(\boldsymbol{z}^{(i)} \mid \boldsymbol{x}^{(i)}))$$

We just derived:

$$\text{ELBO}_i(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{q_i}\left[\log p(\boldsymbol{x}^{(i)} \mid \boldsymbol{Z}^{(i)}, \boldsymbol{\theta})\right] - \text{KL}(q_i || p(\boldsymbol{z}^{(i)}))$$

These decompositions just differ whether and which chain rule we apply to the joint model.

Deriving ELBO Gradient Estimate

In the following, we will use the third decomposition

$$\text{ELBO}_i(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{q_i}\left[\log p(\boldsymbol{x}^{(i)} \mid \boldsymbol{Z}^{(i)}, \boldsymbol{\theta})\right] - \text{KL}(q_i || p(\boldsymbol{z}^{(i)}))$$

and derive gradient estimates w.r.t. $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$.

Since gradients of sums are the sum of the gradients, we can treat the two terms

- $\mathbb{E}_{q_i}\left[\log p(\boldsymbol{x}^{(i)} \mid \boldsymbol{Z}^{(i)}, \boldsymbol{\theta})\right]$
- $\text{KL}(q_i || p(\boldsymbol{z}^{(i)}))$

independently. We will start with the second term.

Deriving ELBO Gradient Estimate cont'd

We start with $\mathrm{KL}(q_i||p(\boldsymbol{z}^{(i)}))$

- q_i is Gaussian $\mathcal{N}(\boldsymbol{\mu}_\phi(\boldsymbol{x}^{(i)}), \mathrm{diag}(\boldsymbol{\sigma}^2_\phi(\boldsymbol{x}^{(i)})))$
- $p(\boldsymbol{z}^{(i)})$ is Gaussian $\mathcal{N}(\mathbf{0}, I)$

There is a closed-form solution (Kingma and Welling, 2013)

$$\mathrm{KL}(q_i||p(\boldsymbol{z}^{(i)})) = -\frac{1}{2}\sum_{j=1}^{K}\left[1 + \log\sigma^2_{\phi,j}(\mathbf{x}^{(i)}) - \mu^2_{\phi,j}(\mathbf{x}^{(i)}) - \sigma^2_{\phi,j}(\mathbf{x}^{(i)})\right]$$

Recall that $\mu_{\phi,j}(\mathbf{x}^{(i)})$ and $\sigma^2_{\phi,j}(\mathbf{x}^{(i)})$ are just the outputs of the encoder.

- no dependence on $\boldsymbol{\theta}$, hence $\nabla_{\boldsymbol{\theta}}\mathrm{KL}(q_i||p(\boldsymbol{z}^{(i)})) = \mathbf{0}$
- $\nabla_\phi\mathrm{KL}(q_i||p(\boldsymbol{z}^{(i)}))$ is easily obtained by automatic differentiation (backprop)

Deriving ELBO Gradient Estimate cont'd

Let's turn to

$$\mathbb{E}_{q_i}\left[\log p(\boldsymbol{x}^{(i)} \mid \boldsymbol{Z}^{(i)}, \boldsymbol{\theta})\right] = \int q(\boldsymbol{z}^{(i)}|\boldsymbol{x}^{(i)}, \boldsymbol{\phi}) \log p(\boldsymbol{x}^{(i)} \mid \boldsymbol{z}^{(i)}, \boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{z}^{(i)}$$

Note: depends on both $\boldsymbol{\theta}$ (through p) and on $\boldsymbol{\phi}$ (through q)
Bad news: intractable to compute this integral
However: we might draw samples from q_i (Gaussian) and do a Monte Carlo estimate. One often just uses a **single** sample—which is still an unbiased (but noisy) estimator:

$$\boldsymbol{z}^{(i)} \sim q(\boldsymbol{z}^{(i)}|\boldsymbol{x}^{(i)}, \boldsymbol{\phi}) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}^{(i)}), \mathrm{diag}(\boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\boldsymbol{x}^{(i)})))$$

$$\mathbb{E}_{q_i}\left[\log p(\boldsymbol{x}^{(i)} \mid \boldsymbol{Z}^{(i)}, \boldsymbol{\theta})\right] \approx \log p(\boldsymbol{x}^{(i)} \mid \boldsymbol{z}^{(i)}, \boldsymbol{\theta})$$

Deriving ELBO Gradient Estimate cont'd

$$\boldsymbol{z}^{(i)} \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\phi}}(\boldsymbol{x}^{(i)}), \text{diag}(\boldsymbol{\sigma}^2_{\boldsymbol{\phi}}(\boldsymbol{x}^{(i)})))$$

$$\mathbb{E}_{q_i}\left[\log p(\boldsymbol{x}^{(i)} \mid \boldsymbol{Z}^{(i)}, \boldsymbol{\theta})\right] \approx \log p(\boldsymbol{x}^{(i)} \mid \boldsymbol{z}^{(i)}, \boldsymbol{\theta})$$

Taking the gradient w.r.t. $\boldsymbol{\theta}$ is no problem.

$$\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}^{(i)} \mid \boldsymbol{z}^{(i)}, \boldsymbol{\theta}) =$$

$$\nabla_{\boldsymbol{\theta}} \left(-\frac{1}{2} \sum_{d=1}^{D} \log(2\pi) + \log \sigma^2_{\boldsymbol{\theta},d}(\boldsymbol{z}^{(i)}) + \frac{(x_d^{(i)} - \mu_{\boldsymbol{\theta},d}(\boldsymbol{z}^{(i)}))^2}{\sigma^2_{\boldsymbol{\theta},d}(\boldsymbol{z}^{(i)})} \right)$$

Recall that $\mu_{\boldsymbol{\theta},d}(\boldsymbol{z}^{(i)})$ and $\sigma^2_{\boldsymbol{\theta},d}(\boldsymbol{z}^{(i)})$ are just the outputs of the decoder—computing the gradient w.r.t. $\boldsymbol{\theta}$ is easy with automatic differentiation (backprop).

Deriving ELBO Gradient Estimate cont'd

$$\boldsymbol{z}^{(i)} \sim \mathcal{N}(\boldsymbol{\mu}_\phi(\boldsymbol{x}^{(i)}), \text{diag}(\boldsymbol{\sigma}^2_\phi(\boldsymbol{x}^{(i)})))$$

$$\mathbb{E}_{q_i}\left[\log p(\boldsymbol{x}^{(i)} \mid \boldsymbol{Z}^{(i)}, \boldsymbol{\theta})\right] \approx \log p(\boldsymbol{x}^{(i)} \mid \boldsymbol{z}^{(i)}, \boldsymbol{\theta})$$

However, $\boldsymbol{z}^{(i)}$ is sampled from a Gaussian whose parameters depend on ϕ—how to "backprop" through the "sampling operation $\sim$"?

The "Reparameterization Trick"

Note that sampling

$$\boldsymbol{z}^{(i)} \sim \mathcal{N}(\boldsymbol{\mu}_\phi(\boldsymbol{x}^{(i)}), \text{diag}(\boldsymbol{\sigma}^2_\phi(\boldsymbol{x}^{(i)})))$$

can be done like

$$\begin{aligned} \boldsymbol{\epsilon} &\sim \mathcal{N}(\mathbf{0}, I) \\ \boldsymbol{z}^{(i)} &= \boldsymbol{\mu}_\phi(\boldsymbol{x}^{(i)}) + \boldsymbol{\epsilon} \cdot \boldsymbol{\sigma}_\phi(\boldsymbol{x}^{(i)}) \end{aligned}$$

where $\cdot$ is element-wise multiplication.

$\boldsymbol{\epsilon}$ is helper noise, sampled from a fixed distribution, independent of ϕ. Now differentiating $\boldsymbol{z}^{(i)}$ w.r.t. ϕ is straight-forward. ✓

This is called the **reparametrization trick**.

For a given sample $\boldsymbol{x}^{(i)}$ do

- pass $\boldsymbol{x}^{(i)}$ through encoder, yielding $\boldsymbol{\mu}_\phi$ and $\boldsymbol{\sigma}^2_\phi$
- compute $\mathrm{KL} = -\frac{1}{2}\sum_{j=1}^{K}\left[1 + \log\sigma^2_{\phi,j} - \mu^2_{\phi,j} - \sigma^2_{\phi,j}\right]$
- sample $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, I)$
- let $\boldsymbol{z}^{(i)} = \boldsymbol{\mu}_\phi(\boldsymbol{x}^{(i)}) + \boldsymbol{\epsilon} \cdot \boldsymbol{\sigma}_\phi(\boldsymbol{x}^{(i)})$
- pass $\boldsymbol{z}^{(i)}$ through decoder, yielding $\boldsymbol{\mu}_{\boldsymbol{\theta}}$ and $\boldsymbol{\sigma}^2_{\boldsymbol{\theta}}$
- compute $\log p = -\frac{1}{2}\sum_{d=1}^{D}\left[\log(2\pi) + \log\sigma^2_{\boldsymbol{\theta},d} + \frac{(x^{(i)}_d - \mu_{\boldsymbol{\theta},d})^2}{\sigma^2_{\boldsymbol{\theta},d}}\right]$
- $\widehat{\mathrm{ELBO}}_i = \log p - \mathrm{KL}$
- compute $\nabla_{\boldsymbol{\theta}}\widehat{\mathrm{ELBO}}_i$ and $\nabla_{\boldsymbol{\phi}}\widehat{\mathrm{ELBO}}_i$ via autodiff

Final Learning Algorithm: Stochastic Gradient Ascent

In each iteration:

- draw mini-batch $\mathcal{B} = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(L)}\}$ of samples
- for each $\boldsymbol{x}^{(i)} \in \mathcal{B}$, compute

$$\nabla_{\boldsymbol{\theta}} \widehat{\text{ELBO}}_i \quad \text{and} \quad \nabla_{\boldsymbol{\phi}} \widehat{\text{ELBO}}_i$$

  using the procedure on the previous slide. Note that this can be parallelized over the mini-batch

- compute

$$\nabla_{\boldsymbol{\theta}} \widehat{\text{ELBO}} = \sum_{l=1}^{L} \nabla_{\boldsymbol{\theta}} \widehat{\text{ELBO}}_i \quad \text{and} \quad \nabla_{\boldsymbol{\phi}} \widehat{\text{ELBO}} = \sum_{l=1}^{L} \nabla_{\boldsymbol{\phi}} \widehat{\text{ELBO}}_i$$

  and perform a gradient ascent step.

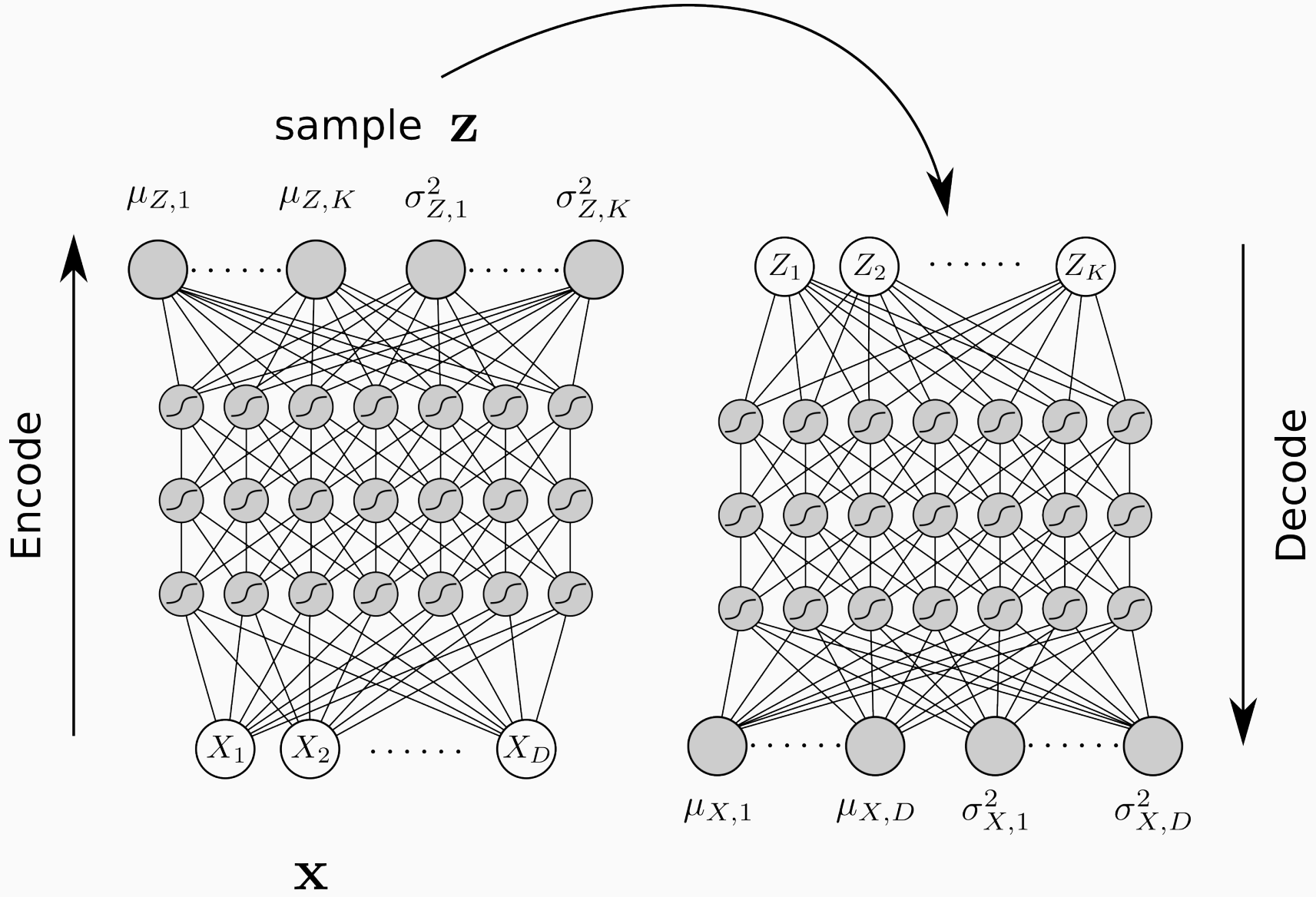
Note that this learning scheme resembles **classical autoencoders**

- encode sample $\boldsymbol{x}^{(i)}$ into a latent representation $\boldsymbol{z}^{(i)}$
- decode $\boldsymbol{z}^{(i)}$ into a (probabilistic) reconstruction $\log p(\boldsymbol{x}^{(i)} \mid \boldsymbol{z}^{(i)})$
- in this interpretation the ELBO can be understood as

$$\underbrace{\mathbb{E}_{q_i}\left[\log p(\boldsymbol{x}^{(i)} \mid \boldsymbol{Z}^{(i)}, \boldsymbol{\theta})\right]}_{\text{reconstruction loss}} - \underbrace{\mathrm{KL}(q_i || p(\boldsymbol{z}^{(i)})}_{\text{regularizer}}$$

- nice side effect: encoder delivers a **representation** (code, embedding) $\boldsymbol{z}^{(i)}$ for data $\boldsymbol{x}^{(i)}$
- **but: only the decoder specifies the probabilistic model—the encoder is merely an inference tool!**

Variational Autoencoder



VAE on MNIST Data

MNIST Data

28×28 gray scale images (784-dimensional vectors)

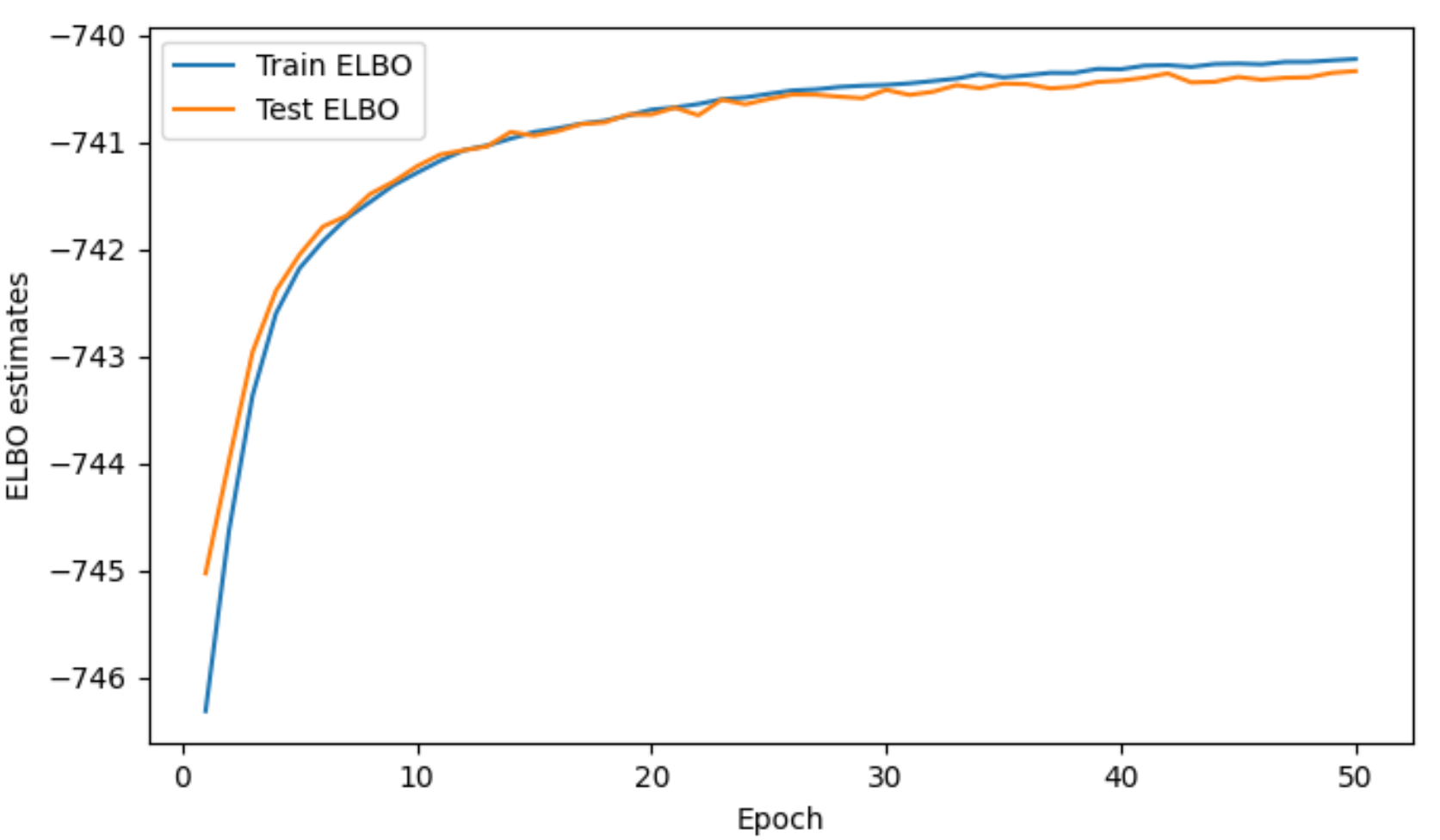
Training Data: 60,000 samples

Test Data: 10,000 samples

VAE Architecture and Training Details

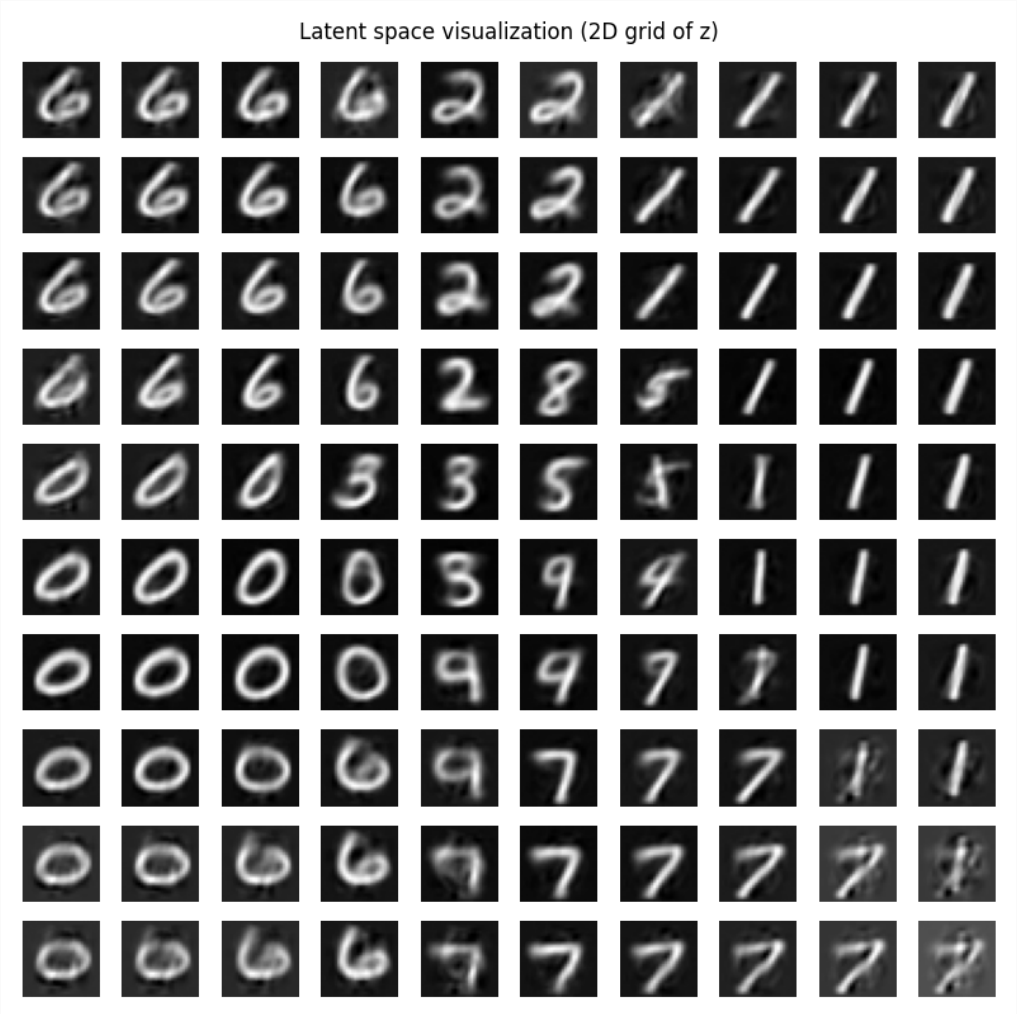
- $K = 2$ latent dimensions (for simplicity and visualization)
- encoder and decoder are both 2-layer MLPs with 200 units, using relu-activation functions
- train for 50 epochs with Adam as optimizer
- stepsize 0.001 and batch-size 128
- the decoder used a fixed variance of 1, i.e. only means are predicted

Training Curves



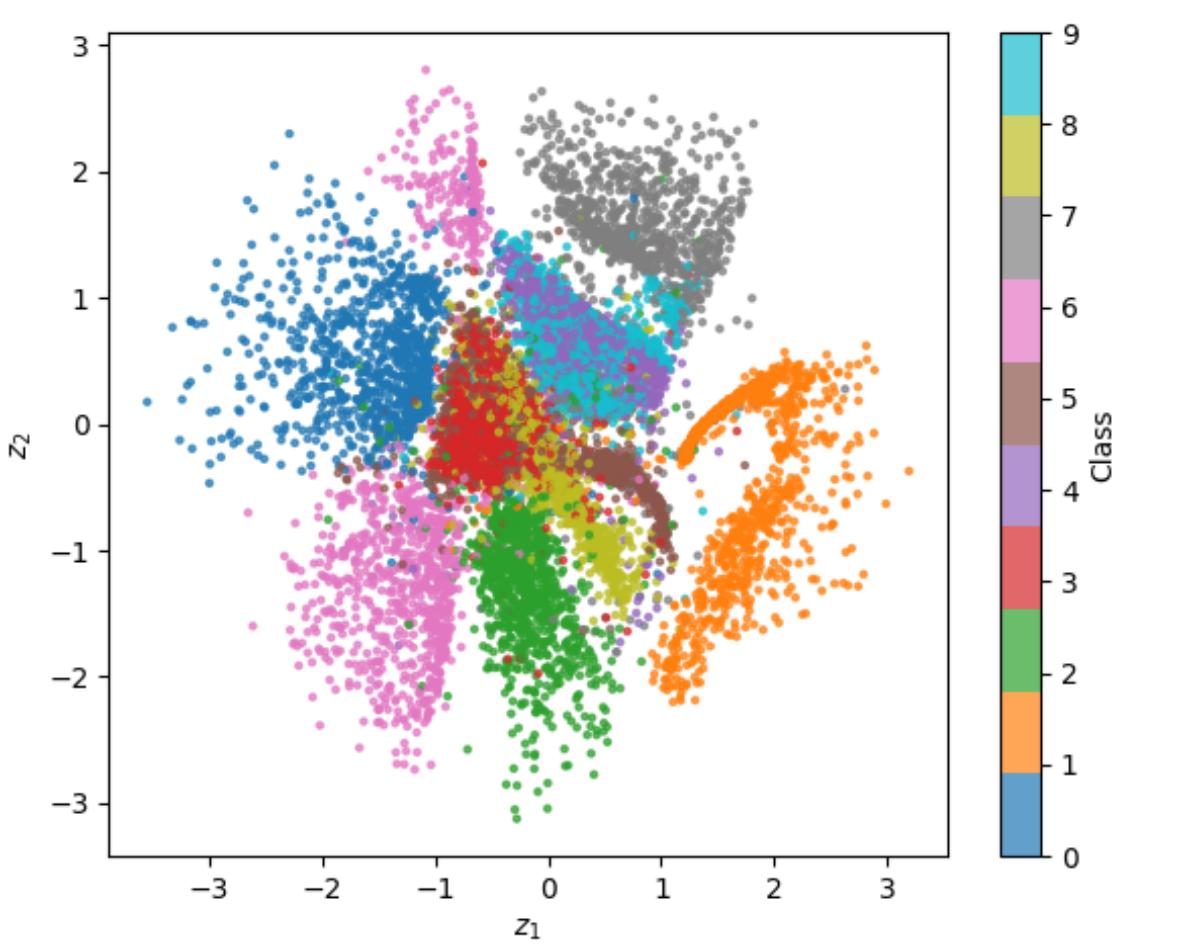
Latent Space Visualization

Sampling $\boldsymbol{z}$ on a grid from -3 to 3, followed by decoding, we can visualize the learned latent space (only decoded means are shown)



Embedding

The encoder delivers for each sample an embedding vector $\boldsymbol{z}^{(i)}$, that can be used as **dimensionality reduction** technique. Here, we color-code the classes (class information was not used in training):



VAE: Remarks and Extensions

- VAEs are one of **the** most prominent deep generative models
- Kingma & Welling: cited over **50,000 times today**
- **historical note:** actually the model is a density network (MacKay, 1995), but the VAE tricks made them fly (stochastic gradient, amortized inference, etc.)
- many extensions exist:
  - combinations with convnets for image data
  - combinations with graph networks, e.g. for molecule data
  - using multiple samples for $\boldsymbol{Z}$ (importance weighted autoencoder)
  - applications to time series and dynamical systems
  - trading off reconstruction and KL term (β-VAE)
  - learning the prior (VampPrior)