Tutorial
Unsupervised Learning: EM and Variational Auto-Encoder

Recommended Reading: For EM, Bishop chapter 9 is the best. As for the Variational Auto Encoder, watch the excellent video by Maximilian Sölch. Further reading is the two recommended Arxiv papers on the first page of the lecture slides

1 The EM algorithm

Problem 1: Consider a general mixture model with K components. Assume that the distribution of components $p(\mathbf{z})$ is uniform.

Derive a simplified posterior inference (assignment of responsibilities) algorithm for this scenario.

1 The EM algorithm

Problem 1: Consider a general mixture model with K components. Assume that the distribution of components $p(\mathbf{z})$ is uniform.

Derive a simplified posterior inference (assignment of responsibilities) algorithm for this scenario.

As we have seen multiple times since the Maximum Likelihood lecture, the uniformity assumption implies

$$p(\mathbf{z} \mid \mathbf{x}, \theta) \propto p(\mathbf{x} \mid \mathbf{z}, \theta).$$

Since $p(\mathbf{z} \mid \mathbf{x}, \theta)$ is discrete, we simply need to renormalise the distribution to obtain the responsibilities

$$r_{nk}(\theta) = \frac{p(\mathbf{x}_n \mid \mathbf{z} = k, \theta)}{\sum_{j=1}^{K} p(\mathbf{x}_n \mid \mathbf{z} = j, \theta)}.$$

(Notice that the crucial argument is that the mixture components are discrete. It applies even for non-uniform mixture distributions, as you can see with, e.g., mixture of Gaussians EM.)

Problem 2: Revisit the slides. Extract an explicit EM algorithm for the Mixture of Gaussians.

Problem 2: Revisit the slides. Extract an explicit EM algorithm for the Mixture of Gaussians.

We have two abstract representations of the EM algorithm in the slides. The first is the following figure:

$$\begin{tikzcd} r_{nk}\left(\theta^{(t)}\right) \arrow[rr, bend left, "\text{M-step: fix } r_{nk}\text{, optimise } \theta"] & & \theta^{(t+1)} \arrow[ll, bend left, "\text{E-step: fix } \theta\text{, update } r_{nk}"] \end{tikzcd}$$

(Remember that the *responsibilities* are defined as $r_{nk}(\theta) \equiv p(\mathbf{z}_n = k \mid \mathbf{x}_n, \theta)$.)

The second is coordinate ascent in the evidence lower bound.

The first step is understanding that these are the same *for MoGs*: The maximization in the q coordinate of the ELBO—the E-step—is a KL minimisation, which is optimally solved by the posterior. In the case of MoGs, this posterior is known explicitly.

Why do we know the posterior? We know the generative model $p(\mathbf{x}, \mathbf{z} \mid \theta) = p(\mathbf{x} \mid \mathbf{z}, \theta)p(\mathbf{z} \mid \theta)$ and the integral in the normalizing constant is feasible. Hence the E-step is comparably easy:

$$p(\mathbf{z} = k \mid \mathbf{x}, \theta) = \frac{p(\mathbf{x} \mid \mathbf{z} = k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)p(\mathbf{z} = k \mid \pi_k)}{\int p(\mathbf{x} \mid \mathbf{z})p(\mathbf{z})\,\mathrm{d}\mathbf{z}} = \frac{\pi_k \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \tag{1}$$

For the M-step, instead of optimising $p(\mathcal{D} \mid \theta)$ or $\ln p(\mathcal{D} \mid \theta)$, which we would like to do, we optimise the approximation that is as close as we can get:

$$\mathbb{E}_{p(\mathbf{z} \mid \mathcal{D}, \theta^{(t)})}[\ln p(\mathcal{D}, \mathbf{z} \mid \theta)] = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk}\left(\theta^{(t)}\right) \ln\left(\pi_k \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\right) \tag{2}$$

What are the differences to $\ln p(\mathcal{D} \mid \theta)$?

- The logarithm is *inside* the integral. (Nota bene: This is an implicit application of Jensen's inequality. It can be used as a second way to prove that the ELBO is an actual lower bound to the likelihood. It is implicitly applied in the lecture slides when assuming that the KL is non-negative.)
- $\theta^{(t)}$ and θ are distinct: The latter is the free variable, the former is fixed in our iterative procedure. Hence, $r_{nk}\left(\theta^{(t)}\right)$ is fixed by eq. (1), while $\pi_k, \boldsymbol{\mu}_k$, and $\boldsymbol{\Sigma}_k$ may vary.

Observing eq. (2) closely, we see that it decomposes into two independent summands, one for the mixture weights π_k and one for the component parameters $\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$:

$$(2) = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk}\left(\theta^{(t)}\right) \ln(\pi_k) + \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk}\left(\theta^{(t)}\right) \ln\left(\mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\right)$$

This simplifies optimization, as one of the summands disappears completely when taking the respective partial derivatives. The second summand almost looks like maximum likelihood for regular Gaussian distributions. The optimal values are very similar:

$$\boldsymbol{\mu}_k^{(t+1)} \longleftarrow \sum_{n=1}^{N} \frac{r_{nk}\left(\theta^{(t)}\right)\mathbf{x}_n}{\sum_{m=1}^{N} r_{mk}\left(\theta^{(t)}\right)} \tag{3}$$

$$\boldsymbol{\Sigma}_k^{(t+1)} \longleftarrow \sum_{n=1}^{N} \frac{r_{nk}\left(\theta^{(t)}\right)\left(\mathbf{x}_n - \boldsymbol{\mu}_k^{(t+1)}\right)\left(\mathbf{x}_n - \boldsymbol{\mu}_k^{(t+1)}\right)^T}{\sum_{m=1}^{N} r_{mk}\left(\theta^{(t)}\right)} \tag{4}$$

(Nota bene: The temporal indices $t+1$ are *not* typos.)

The only change from standard MLE is that each summand (i.e., each sample) is weighted by the normalised responsibilities! That's fairly intuitive, remember the slides: If we knew the true class for each, we would just update each component with the data samples from this class. Now we update each class's parameters with all samples, but only as much as we think they belong to the respective class, which is expressed by the responsibilities. Equation (3) is for instance an approximation of $\boldsymbol{\mu}_k^{\text{MLE}} = \frac{1}{N_k}\sum_{n=1}^{N} \mathbb{1}(\mathbf{z}_n = k)\mathbf{x}_n$, where $N_k = \sum_{n=1}^{N} \mathbb{1}(\mathbf{z}_n = k)$ (cf. the slides).

The only change from standard MLE is that each summand (i.e., each sample) is weighted by the normalised responsibilities! That's fairly intuitive, remember the slides: If we knew the true class for each, we would just update each component with the data samples from this class. Now we update each class's parameters with all samples, but only as much as we think they belong to the respective class, which is expressed by the responsibilities. Equation (3) is for instance an approximation of $\boldsymbol{\mu}_k^{\text{MLE}} = \frac{1}{N_k}\sum_{n=1}^{N} \mathbb{1}(\mathbf{z}_n = k)\mathbf{x}_n$, where $N_k = \sum_{n=1}^{N} \mathbb{1}(\mathbf{z}_n = k)$ (cf. the slides).

Similarly, we update the component weights:

$$N_k^{(t+1)} \longleftarrow \sum_{n=1}^{N} r_{nk}\left(\theta^{(t)}\right) \tag{5}$$

$$\pi_k^{(t+1)} \longleftarrow \frac{N_k^{(t+1)}}{\sum_{k=1}^{K} N_k^{(t+1)}} \tag{6}$$

Instead of counting the number of samples in class k and normalizing, we count (sum up) how much each sample *effectively* contributes to class k. The N_k variables are sometimes referred to as the *number of effective samples.*

(Nota bene: Deriving eqs. (5) and (6) technically requires some insight into constrained optimization that we did not cover. A Lagrange multiplier is necessary to ensure that the optimal π_k's sum up to 1. Given that the resulting solution eq. (6) normalizes naturally, it is safe to omit the exact steps.)

In total, the EM algorithm for a Mixture of Gaussians is as follows:

- Initialize $\theta^{(0)} = \{\pi_k^{(0)}, \boldsymbol{\mu}_k^{(0)}, \boldsymbol{\Sigma}_k^{(0)} \mid k = 1, \ldots, K\}$.
- For $t = 0, 1, \ldots$ until convergence, repeat
    1. E-step: Calculate responsibilities $r_{nk}\left(\theta^{(t)}\right)$ according to eq. (1) for all samples and classes.
    2. M-step: Update to $\theta^{(t+1)}$ according to eqs. (3) to (6).

Problem 3: Consider a mixture model where the components are given by independent Bernoulli variables. This is useful when modelling, e.g., binary images, where each of the D dimensions of the image $\mathbf{x}$ corresponds to a different pixel that is either black or white. More formally, we have

$$p(\mathbf{x} \mid \mathbf{z} = k) = \prod_{d=1}^{D} \theta_{kd}^{x_d} (1 - \theta_{kd})^{1 - x_d}.$$

That is, for a given mixture index $\mathbf{z} = k$, we have a product of independent Bernoullis, where θ_{kd} denotes the Bernoulli parameter for component k at pixel d.

Derive the EM algorithm for the parameters $\theta = \{\theta_{kd} \mid k = 1, \ldots, K, d = 1, \ldots, D\}$ of a mixture of Bernoullis.

Problem 3: Consider a mixture model where the components are given by independent Bernoulli variables. This is useful when modelling, e.g., binary images, where each of the D dimensions of the image $\mathbf{x}$ corresponds to a different pixel that is either black or white. More formally, we have

$$p(\mathbf{x} \mid \mathbf{z} = k) = \prod_{d=1}^{D} \theta_{kd}^{x_d} (1 - \theta_{kd})^{1-x_d}.$$

That is, for a given mixture index $\mathbf{z} = k$, we have a product of independent Bernoullis, where θ_{kd} denotes the Bernoulli parameter for component k at pixel d.

Derive the EM algorithm for the parameters $\theta = \{\theta_{kd} \mid k = 1, \ldots, K, d = 1, \ldots, D\}$ of a mixture of Bernoullis.

Given that we do not optimize for the component probabilities and we have no prior information given, we can assume uniformity and easily apply the first exercise for the E-step.

It remains to derive the M-step. Similiar to mixture of Gaussians:

$$\begin{aligned}
\mathbb{E}_{p(\mathbf{z}|\mathcal{D},\theta^{(t)})}[\ln p(\mathcal{D}, \mathbf{z} \mid \theta)] &= \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk}\left(\theta^{(t)}\right) \ln\left(\frac{1}{K} \prod_{d=1}^{D} \theta_{kd}^{x_d^{(n)}} (1-\theta_{kd})^{1-x_d^{(n)}}\right) \\
&= C + \sum_{n=1}^{N} \underbrace{\sum_{k=1}^{K} r_{nk}\left(\theta^{(t)}\right) \sum_{d=1}^{D} \left(x_d^{(n)} \ln \theta_{kd} + \left(1 - x_d^{(n)}\right) \ln(1-\theta_{kd})\right)}_{=:\mathcal{L}_n}
\end{aligned}$$

The constant C is independent of θ and hence irrelevant for further optimization. (Nota bene: We already left out a part of the evidence lower bound that is independent of θ, the entropy of the approximate posterior.)

We now need to take derivatives w.r.t. θ. We observe that the θ_{kd} do not interfere nonlinearly, i.e., we can handle the gradients individually:

$$\begin{aligned}
\frac{\partial \mathcal{L}_n}{\partial \theta_{k',d'}} &= \sum_{k=1}^{K} r_{nk}\left(\theta^{(t)}\right) \sum_{d=1}^{D} \left(x_d^{(n)} \frac{\partial \ln \theta_{kd}}{\partial \theta_{k',d'}} + \left(1 - x_d^{(n)}\right) \frac{\partial \ln(1-\theta_{kd})}{\partial \theta_{k',d'}} \right) \qquad \text{lots of zeros} \\
&= r_{nk'}\left(\theta^{(t)}\right) \left(\frac{x_{d'}^{(n)}}{\theta_{k',d'}} - \frac{1 - x_{d'}^{(n)}}{1-\theta_{k',d'}} \right)
\end{aligned}$$

Setting this to zero, we obtain the optimal update in a similar fashion as in the standard Bernoulli MLE:

$$\begin{aligned}
\frac{\partial \, \mathbb{E}_{p(\mathbf{z} \mid \mathcal{D}, \theta^{(t)})}[\ln p(\mathcal{D}, \mathbf{z} \mid \theta)]}{\partial \theta_{kd}} = \sum_{n=1}^{N} \frac{\partial \mathcal{L}_n}{\partial \theta_{kd}} = \sum_{n=1}^{N} r_{nk}\left(\theta^{(t)}\right) \left(\frac{x_d^{(n)}}{\theta_{kd}} - \frac{1 - x_d^{(n)}}{1-\theta_{kd}} \right) \overset{!}{=} 0 \\
\Leftrightarrow \theta_{kd} = \frac{\sum_{n=1}^{N} r_{nk}\left(\theta^{(t)}\right) x_d^{(n)}}{\sum_{n=1}^{N} r_{nk}\left(\theta^{(t)}\right)}
\end{aligned}$$

2 Variational Auto-Encoders

Problem 4: A crucial trick when implementing Variational Auto-Encoders (VAEs) is the *reparameterisation trick*,

$$\nabla_\phi \mathbb{E}_{q(\mathbf{z}|\phi)}[f(\mathbf{z})] = \mathbb{E}_{q(\epsilon)}\big[\nabla_\phi f(g(\epsilon, \phi))\big].$$

It is based on the fairly weak assumptions that

- f is differentiable,
- and we can draw samples from the distribution of $q(\mathbf{z} \mid \phi)$ by first drawing a sample $\epsilon \sim q(\epsilon)$ and then applying a deterministic, differentiable transform $g(\epsilon, \phi)$.

Prove that this trick is valid. Why is it so important?

2 Variational Auto-Encoders

Problem 4: A crucial trick when implementing Variational Auto-Encoders (VAEs) is the *reparameterisation trick*,

$$\nabla_\phi \mathbb{E}_{q(\mathbf{z}|\phi)}[f(\mathbf{z})] = \mathbb{E}_{q(\epsilon)}\big[\nabla_\phi f(g(\epsilon, \phi))\big].$$

It is based on the fairly weak assumptions that

- f is differentiable,
- and we can draw samples from the distribution of $q(\mathbf{z} \mid \phi)$ by first drawing a sample $\epsilon \sim q(\epsilon)$ and then applying a deterministic, differentiable transform $g(\epsilon, \phi)$.

Prove that this trick is valid. Why is it so important?

An explanantion of the trick is originally found on the highly recommended blog *The Spectator* `http://blog.shakirm.com/2015/10/machine-learning-trick-of-the-day-4-reparameterisation-tricks/` by one of the authors (Shakir Mohamed) of one of the two original VAE papers.

A sample ϵ is drawn from a base distribution $q(\epsilon)$, then mixed with R and added to μ. Eventually, you yield a transformed sample z of your target distribution. When doing backprop, the gradient can flow backwards into the parameters without being stuck in random samples.

Proving it is not as hard as it seems:

$$\begin{aligned}
\nabla_\phi \mathbb{E}_{q(\mathbf{z}|\phi)}[f(\mathbf{z})] &= \nabla_\phi \int q(\mathbf{z} \mid \phi) f(\mathbf{z}) \, \mathrm{d}z && \text{definition} \\
&= \nabla_\phi \int q(\epsilon) f(\mathbf{z}) \, \mathrm{d}\epsilon && \text{change of differential} \\
&= \nabla_\phi \int q(\epsilon) f(g(\epsilon, \phi)) \, \mathrm{d}\epsilon && \text{full change to reparameterization} \\
&= \nabla_\phi \mathbb{E}_{q(\epsilon)}[f(g(\epsilon, \phi))] && \text{definition} \\
&= \mathbb{E}_{q(\epsilon)}\big[\nabla_\phi f(g(\epsilon, \phi))\big] && \text{gradient can be moved into independent integral}
\end{aligned}$$

It is extremely useful as the integral in the first line is typically intractable. The last line, though, admits for Monte Carlo estimates of the gradient with low variance:

$$\mathbb{E}_{q(\epsilon)}\big[\nabla_\phi f(g(\epsilon, \phi))\big] = 1/S \sum_{s=1}^{S} \nabla_\phi f(g(\epsilon_s, \phi))$$

$$\ln p(\mathbf{x} \mid \theta) = \underbrace{\int q(\mathbf{z}) \ln \frac{p(\mathbf{x}, \mathbf{z} \mid \theta)}{q(\mathbf{z})} \, d\mathbf{z}}_{= \mathcal{L}_{\text{ELBO}}(q, \theta) \begin{cases} = \mathbb{E}_{q(\mathbf{z})}[\ln p(\mathbf{x}, \mathbf{z} \mid \theta)] \\ \quad + \ \mathrm{H}(q) \\ = \mathbb{E}_{q(\mathbf{z})}[\ln p(\mathbf{x} \mid \mathbf{z}, \theta)] \\ \quad - \mathrm{KL}(q(\mathbf{z}) \,||\, p(\mathbf{z})) \end{cases}} + \underbrace{\int q(\mathbf{z}) \ln \frac{q(\mathbf{z})}{p(\mathbf{z} \mid \mathbf{x}, \theta)} \, d\mathbf{z}}_{= \mathrm{KL}(q(\mathbf{z}) \,||\, p(\mathbf{z} \mid \mathbf{x}, \theta))}$$

$$\underset{\textcircled{\scriptsize θ}}{\arg\max}\, \mathcal{L}_{\text{ELBO}}(q, \theta) = \arg\max_{\theta} \mathbb{E}_{q(\mathbf{z})}[\ln p(\mathbf{x}, \mathbf{z} \mid \theta)] + \underbrace{\mathrm{H}(q)}_{\text{const. wrt. } \theta} = \arg\max_{\theta} \mathbb{E}_{q(\mathbf{z})}[\ln p(\mathbf{x}, \mathbf{z} \mid \theta)]$$

M-step

$$\begin{tikzcd} r_{nk}\left(\theta^{(t)}\right) \arrow[rr, bend left, "\text{fix } r_{nk}\text{, optimise } \theta"] & & \theta^{(t+1)} \arrow[ll, bend left, "\text{fix } \theta\text{, update } r_{nk}"] \end{tikzcd}$$

E-step

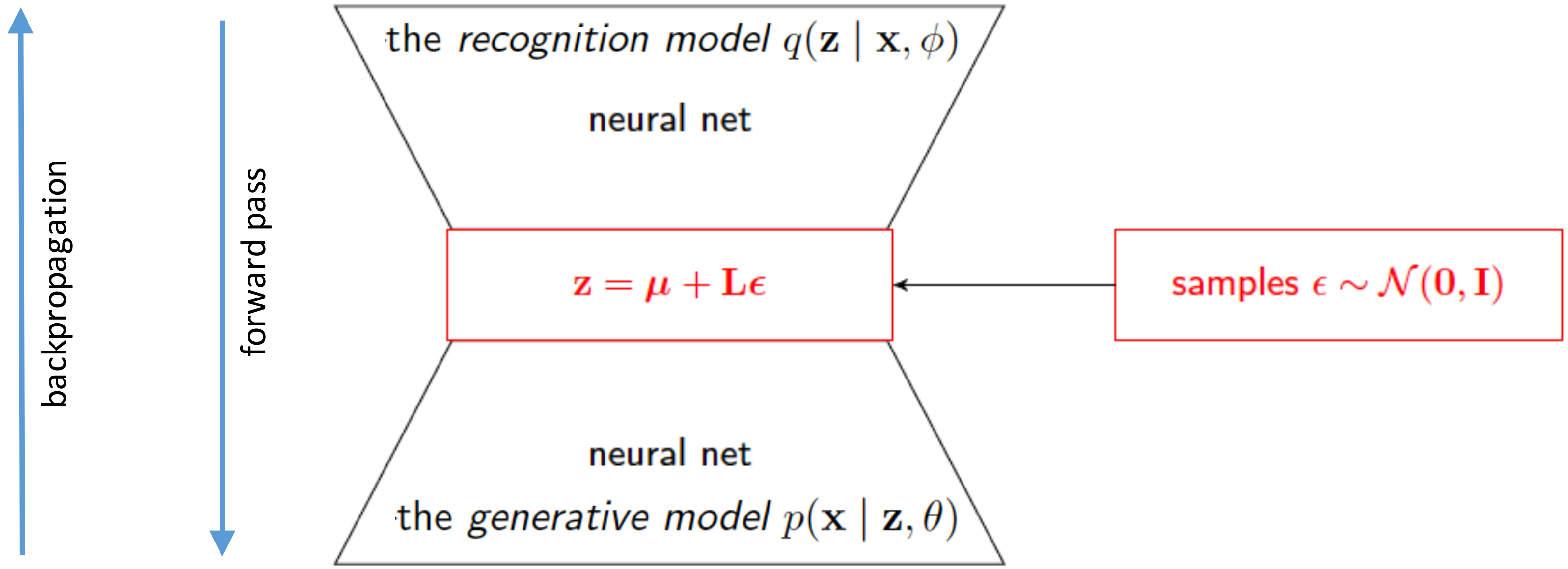
$r_{nk}\left(\theta^{(t)}\right) = p\left(\mathbf{z}_n = k \mid \mathbf{x}_n, \theta^{(t)}\right)$ is the (trivial) optimal solution to

$$\arg\min_{q} \mathrm{KL}\left(q(\mathbf{z}) \,||\, p\left(\mathbf{z} \mid \mathbf{x}, \theta^{(t)}\right)\right) = \underset{\textcircled{\scriptsize q}}{\arg\max}\, \mathcal{L}_{\text{ELBO}}\left(q, \theta^{(t)}\right)$$

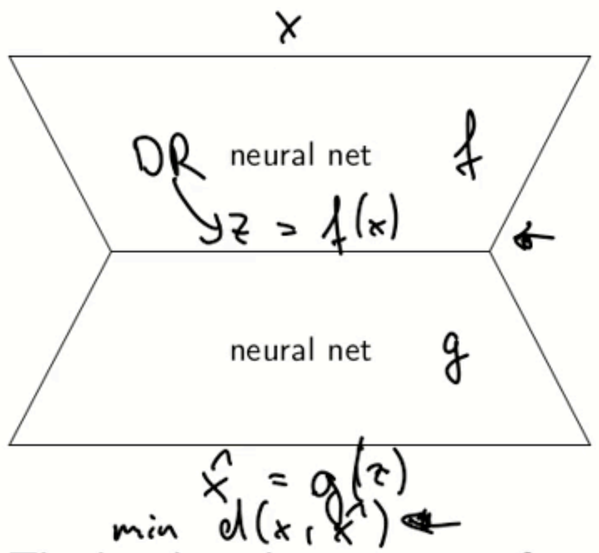
$$\mathcal{L}_{\text{ELBO}}(\phi,\theta) = \mathcal{L}_{\text{ELBO}}(q,\theta) = \mathbb{E}_{q(\mathbf{z})}[\ln p(\mathbf{x} \mid \mathbf{z},\theta)] - \text{KL}(q(\mathbf{z}) \,||\, p(\mathbf{z}))$$

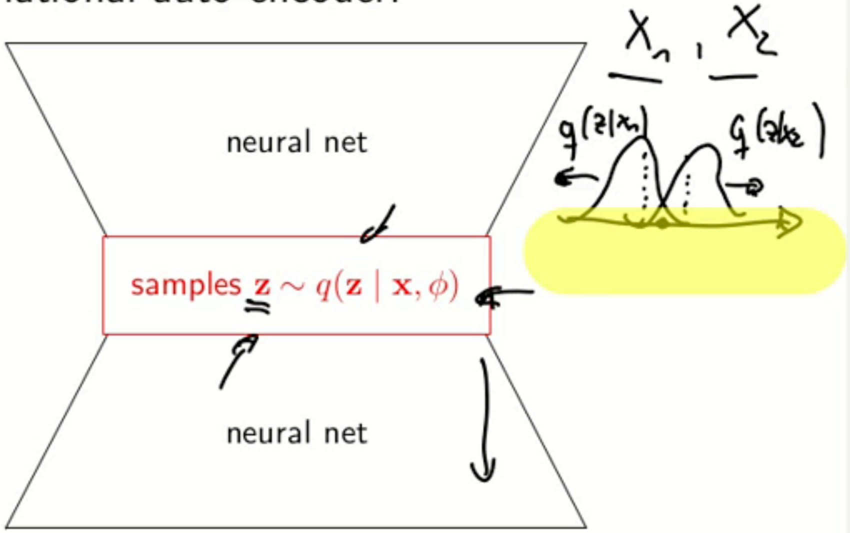
where $q(\mathbf{z}) \equiv q(\mathbf{z} \mid \mathbf{x},\phi) = \mathcal{N}(\mathbf{z} \mid \boldsymbol{\mu}_\phi(\mathbf{x}), \boldsymbol{\Sigma}_\phi(\mathbf{x}))$

and e.g. $p(\mathbf{x} \mid \mathbf{z},\theta) = \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_\theta(\mathbf{z}), \boldsymbol{\Sigma}_\theta(\mathbf{z}))$


backpropagation
forward pass
the *recognition model* $q(\mathbf{z} \mid \mathbf{x},\phi)$
neural net
$\mathbf{z} = \boldsymbol{\mu} + \mathbf{L}\boldsymbol{\epsilon}$
samples $\epsilon \sim \mathcal{N}(\mathbf{0},\mathbf{I})$
neural net
the *generative model* $p(\mathbf{x} \mid \mathbf{z},\theta)$

Deterministic auto-encoder vs. variational auto-encoder:





The bottleneck now comes from stochasticity through sampling rather than dimensionality reduction.

$$\mathcal{L}_{\text{ELBO}}(q,\theta) = \mathbb{E}_{q(\mathbf{z})}[\ln p(\mathbf{x} \mid \mathbf{z},\theta)] - \text{KL}(\underset{\mathcal{N}(\mu_\theta(z),\Sigma_\theta(z))}{q(\mathbf{z})} \,||\, \underset{\mathcal{N}(0,I)}{p(\mathbf{z})})$$

The ELBO rewards good reconstruction (first term), and comes with an inbuilt regulariser (the KL) that prevents collapsing to the deterministic autoencoder.

One more obstacle: Backpropagation through random sampling?

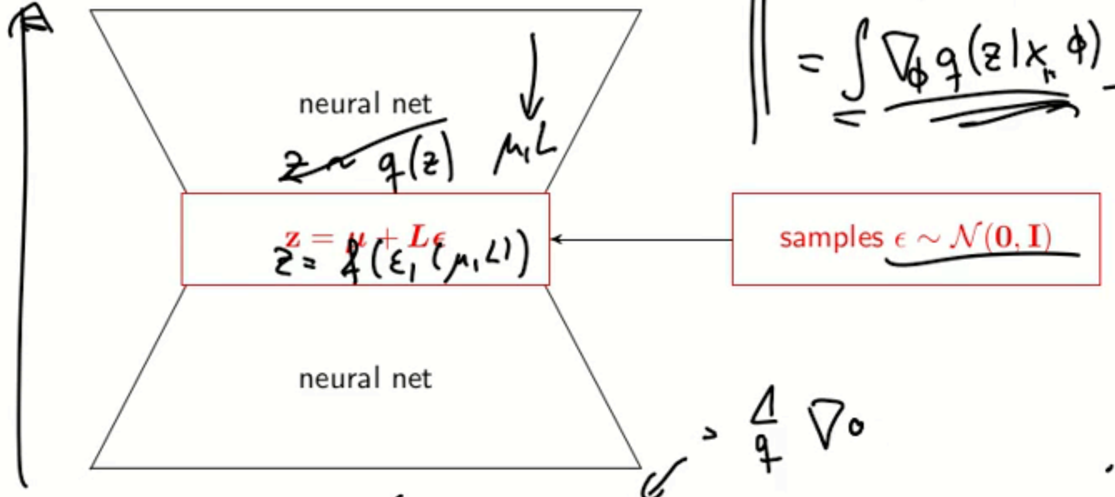
Solution: Reparametrisation.

$$\mathbf{z} \sim \underline{\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})} \quad \rightsquigarrow \quad \mathbf{z} = \underline{\boldsymbol{\mu} + \boldsymbol{L}\boldsymbol{\epsilon}}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \quad \boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}^T$$

This allows taking partial derivatives w. r. t. $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$.

$$\nabla_\phi E_{q(z|x,\phi)}[f(z,\theta)]$$

$$= \int \underline{\nabla_\phi q(z|x_n \phi)} \, \underline{f(z,\theta)} \, dz$$



$$= \frac{1}{q} \nabla_0$$

$$\nabla_\phi q_\phi = (\nabla_\phi \log q_\phi) \cdot q_\phi$$

Problem 5: Prove

$$
\begin{aligned}
\mathcal{L}_{\text{ELBO}}(\phi, \theta) &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\ln p(\mathbf{x} \mid \mathbf{z}, \theta)] - \mathrm{KL}\big(q_\phi(\mathbf{z} \mid \mathbf{x}) \,\|\, p(\mathbf{z})\big) && (1) \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\ln p(\mathbf{x}, \mathbf{z} \mid \theta)] + \mathbb{H}\left(q_\phi(\mathbf{z} \mid \mathbf{x})\right) && (2) \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\big[\ln p(\mathbf{x}, \mathbf{z} \mid \theta) - \ln q_\phi(\mathbf{z} \mid \mathbf{x})\big]. && (3)
\end{aligned}
$$

$\mathbb{H}(p(\mathbf{x})) = \mathbb{E}_{p(\mathbf{x})}[-\ln p(\mathbf{x})]$ denotes the entropy.

All three variants are found in practice when training variational inference models. Can you guess reasons?

Problem 5: Prove

$$\begin{aligned}
\mathcal{L}_{\text{ELBO}}(\phi,\theta) &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\ln p(\mathbf{x}\mid\mathbf{z},\theta)] - \mathrm{KL}\big(q_\phi(\mathbf{z}\mid\mathbf{x}) \,\|\, p(\mathbf{z})\big) && (1)\\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\ln p(\mathbf{x},\mathbf{z}\mid\theta)] + \mathbb{H}\left(q_\phi(\mathbf{z}\mid\mathbf{x})\right) && (2)\\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\big[\ln p(\mathbf{x},\mathbf{z}\mid\theta) - \ln q_\phi(\mathbf{z}\mid\mathbf{x})\big]. && (3)
\end{aligned}$$

$\mathbb{H}(p(\mathbf{x})) = \mathbb{E}_{p(\mathbf{x})}[-\ln p(\mathbf{x})]$ denotes the entropy.

All three variants are found in practice when training variational inference models. Can you guess reasons?

First, we chunk the last equation into three atoms by applying the multiplication rule on the joint, logarithm rules and linearity of expectation:

$$\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\big[\ln p(\mathbf{x},\mathbf{z}\mid\theta) - \ln q_\phi(\mathbf{z}\mid\mathbf{x})\big] = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\ln p(\mathbf{x}\mid\mathbf{z},\theta)] + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\ln p(\mathbf{z})] + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\big[-\ln q_\phi(\mathbf{z}\mid\mathbf{x})\big]$$

By recombining the first and second summand, we get the entropy variant. By combining the second and third summand, we arrive at the original ELBO from the slides.

To understand why these three variants differ, one needs to understand how a VAE is trained. Typically, the expectations under the approximate posterior $q_\phi(\mathbf{z}\mid\mathbf{x})$ are intractable, so we approximate them by Monte Carlo sampling.

The equations, from top to bottom, include more and more parts into these expectations. When the KL is analytically tractable, eq. (1) is used to reduce variance of the estimator. Sometimes only the entropy is known, then eq. (2) can still be used. If $q_\phi(\mathbf{z}\mid\mathbf{x})$ is known, then eq. (3) is used, a full Monte Carlo approximation of the ELBO.

Problem 6: Remember the results of a VAE on MNIST, cf. Figure 1.

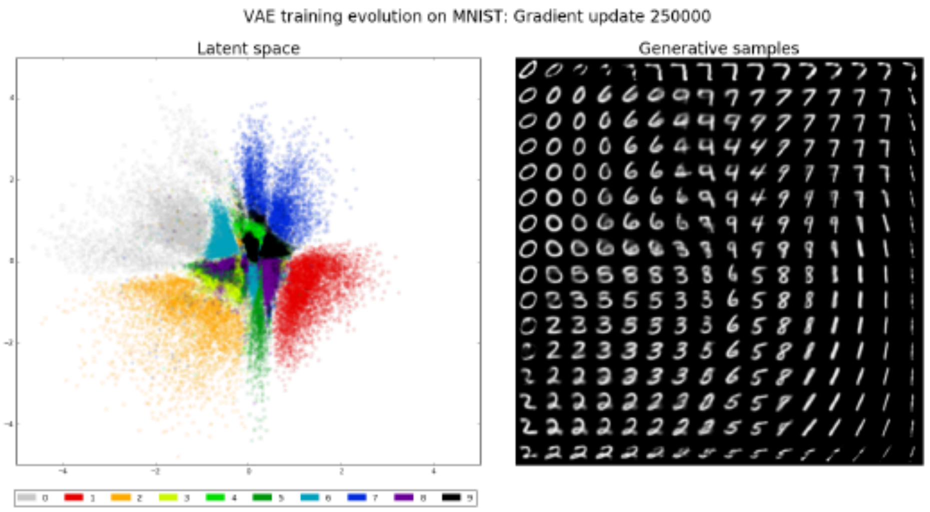


Figure 1: Left: Latent space samples from the approximate posterior $q_\phi(\mathbf{z} \mid \mathbf{x}_n)$ for $\mathbf{x}_n \in \mathcal{D}$. Right: Samples from the generative model with latent samples $\mathbf{z}$ randomly drawn from the prior.

Answer the following questions:

- Does the total distribution of the latent samples remind you of something?
- Can you justify your claim from the previous question?
- The latent space looks nicely ordered. Can you identify problems?

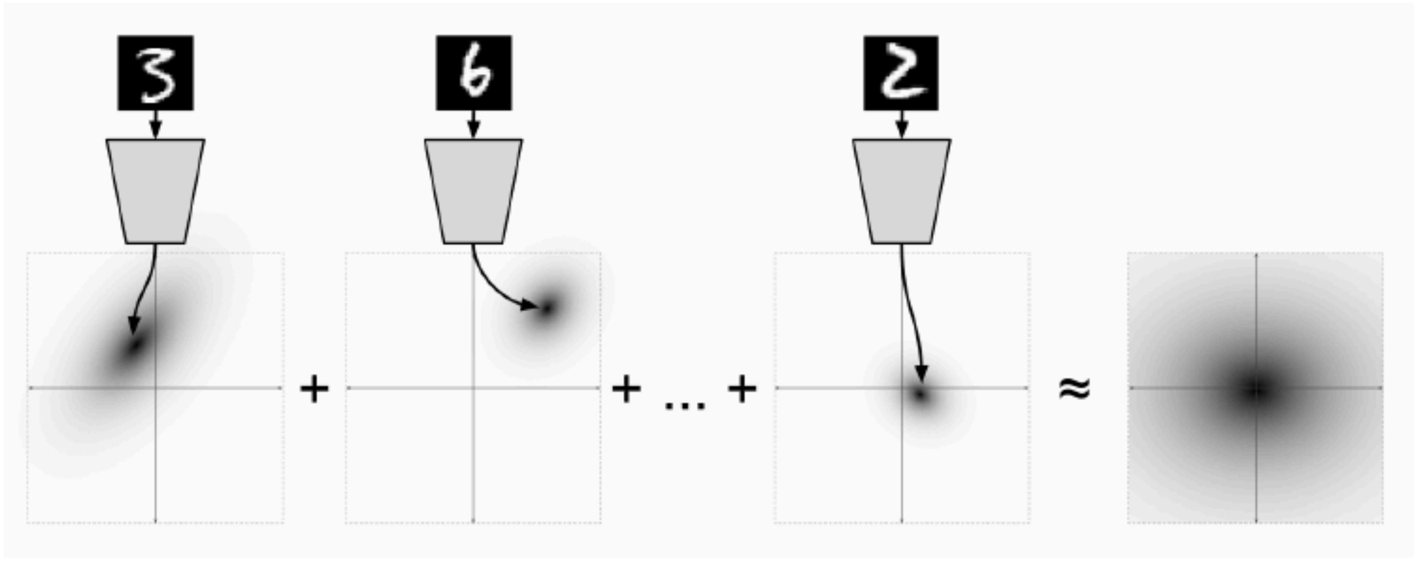
The samples resemble samples from a standard Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$. This can be theoretically justified. Each sample in the left plot is drawn from a distribution $q_\phi(\mathbf{z} \mid \mathbf{x}_n)$ for some data point $\mathbf{x}_n$. Hence the overall plot shows samples from the mixture distribution

$$\sum_{\mathbf{x}_n \in \mathcal{D}} q_\phi(\mathbf{z} \mid \mathbf{x}_n).$$

Analysing this mixture distribution will answer the second bullet:

$$\begin{aligned}
\sum_{\mathbf{x}_n \in \mathcal{D}} q_\phi(\mathbf{z} \mid \mathbf{x}_n) &\approx \int q_\phi(\mathbf{z} \mid \mathbf{x}) p(\mathbf{x}) \,\mathrm{d}\mathbf{x} && \text{Monte Carlo sampling of the integral} \\
&\approx \int p(\mathbf{z} \mid \mathbf{x}) p(\mathbf{x}) \,\mathrm{d}\mathbf{x} && \text{by training } q_\phi(\mathbf{z} \mid \mathbf{x}) \approx p(\mathbf{z} \mid \mathbf{x}) \\
&= \int p(\mathbf{z}, \mathbf{x}) \,\mathrm{d}\mathbf{x} && \text{multiplication rule} \\
&= p(\mathbf{z}) && \text{marginalization}
\end{aligned}$$

Put in words: Under the assumption that the approximate posterior is trained sufficiently well, the latent sample plot should resemble the prior distribution $p(\mathbf{z})$ in shape. The following figure found at `http://ijdykeman.github.io/ml/2016/12/21/cvae.html` illustrates this:



The standard Gauss distribution is a common choice as a prior: As you learned, it has the highest entropy with bounded variance on infinite support. That means it's the least informative and can save least information. Additionally, it has computational advantages: The KL divergence between two Gaussian distributions can be expressed analytically.

One very typical problem of VAEs can be seen in the plots: While the overall shape fits a Gaussian distribution well, there are valleys of no probability mass (and consequently no samples), e.g., between the grey and the orange area, or between the red and the blue area. The effects of this can be seen in the right plot of generative samples: When drawing $\mathbf{z}$ from the prior rather than the approximate posterior, there is a fairly high chance to end up in one of these blank spots of $\sum_{\mathbf{x}_n \in \mathcal{D}} q_\phi(\mathbf{z} \mid \mathbf{x}_n)$. This results in unnatural samples, some of which can be seen in the plot.

Mitigating these effects by, e.g., enhancing the approximate posterior distribution, is ongoing research.

(Nota bene: It is not necessary, but certainly helpful to understand the sampling technique for the right plot. We would like to preserve the structure of the latent space—notice how the samples are ordered similarly as the coloured patches in the latent space. However, the latent samples are unbounded, while our space for plotting is bounded. We could simply put a bounded mesh into the Gaussian latent space, but we would never get samples from outside this box.

Instead, we use inverse transform sampling: We sample a mesh in the unit box $[0, 1]^2$, and transform them into Gaussian samples via the inverse CDF. This way, we get representative Gaussian samples, and the corresponding uniform samples give us an ordering for plotting. Notice that this yields an (inverse) fish-eye effect, which is also present in the right plot.)