A Tutorial on Variational Autoencoders

Elia Bronzo, Shoichi Yip

5 March 2024

These notes are not intended to be distributed to anyone, but only a path trough the main concepts we intend to explain during the final presentation, using slides. Presentation may differ due to the limited time at our disposal.

1 An introduction to probabilistic models and DLVMs

1.1 Probabilistic models and variational inference

The problem we are interested in is learning a *probabilistic model* over the space of data. To fix some notation we define a point in the space where our data live as $\mathbf{x}$. We assume it to be a random sample from an underlying unknown process that we denote $p^*(\mathbf{x})$, where the star indicates the "true" distribution that generates the data that we are interested in computing/approximating.

In particular we try to approximate the true generating process with a parametric model $p_\theta(\mathbf{x})$ where θ is the set of parameters of our model.
Learning in this context means finding the best value θ of our model so that the $p_\theta(x)$ best approximate $p^*(x)$

$$p_\theta(x) \approx p^*(x) \tag{1}$$

The model can also be a differentiable feed-forward neural-network aka a DNN and this has many benefits since:

- DNNs have an high representation capacity
- DNNs can be easly trained with SGD

We can optimize the likelihood (log-likelihood) over the dataset ($\mathcal{D}$) of the (assumed) i.i.d. datapoints using gradient descent. In particular, using Stochastic gradient descent, which consists in using as unbiased estimator of the gradient of the log-likelihood of the dataset, the gradient over the log-likelihood of a random subset of the data called Mini-Batch ($\mathcal{M}$):

$$\frac{1}{N_\mathcal{D}} \boldsymbol{\nabla}_\theta \log p_\theta(\mathcal{D}) \approx \frac{1}{N_\mathcal{M}} \boldsymbol{\nabla}_\theta \log p_\theta(\mathcal{M}) \tag{2}$$

To be read as: the RHS is the estimator of the LHS

1.2 Deep Latent Variables Models

What we are interested in are models with latent variables, denoted by $\mathbf{z}$. Latent variables are part of the model but we do not observe them in the dataset. The probabilistic model then becomes $p_\theta(\mathbf{x}, \mathbf{z})$, hence it is defined both over the observed and latent variables.

A quantity of particular interest is:

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, \mathbf{z}) \mathrm{d}\mathbf{z} = \int p_\theta(\mathbf{z}) p_\theta(\mathbf{x}|\mathbf{z}) \mathrm{d}\mathbf{z} \tag{3}$$

where we used the definition of conditional probability to decompose the jPDF. $p_\theta(\mathbf{z})$ is called also *prior* over $\mathbf{z}$ since it is not conditioned over any observations. $p_\theta(\mathbf{x})$ is also called *marginal likelihood* or *model evidence*.
Examples: $p_\theta(\mathbf{x}|\mathbf{z})$ is a Gaussian and the prior over z is discrete $\rightarrow$ $p_\theta(\mathbf{x})$ is a Gaussian mixture.
If $p_\theta(\mathbf{x}, \mathbf{z})$ is parameterized with a DNN the model is called *Deep Latent Variable Model* or DLVM. The advantage of this so is due to the expressivity of the DNN: even with simple (for example Gaussian) priors the marginal distribution of equation (3) can contain almost arbitrary dependencies.

Example of Gaussian prior and factorized Bernoulli posterior when we describe the code.

1.2.1 Problem of intractabilities

If our model is a deep neural network why we cannot just use SGD to optimize the Log Likelihood of the dataset?

We do not have $\mathbf{z}$ and hence we cannot optimize the likelihood of the model because "we don't know the values of the latent variables associated to each data-point". What we really want to optimize is the marginal likelihood over the observed variables only: $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, \mathbf{z}) \mathrm{d}\mathbf{z}$. This problem, as can be easily guessed, is intractable: we do not have an analytic solution in case of a generic DNN used as probabilistic model neither an efficient estimator of it. This is why we cannot optimize it easily with SGD.

Where does the intractability come from?
the marginal likelihood and the prior are related by the following relation which is just the definition of conditional probability:

$$p_\theta(\mathbf{z}|\mathbf{x}) = \frac{p_\theta(\mathbf{x}, \mathbf{z})}{p_\theta(\mathbf{x})} \tag{4}$$

$p_\theta(\mathbf{x}, \mathbf{z})$ is a DNN, which imply that is tractable. $p_\theta(\mathbf{x})$ would be tractable if the posterior $p_\theta(\mathbf{z}|\mathbf{x})$ was.

$$\forall i = 1, ..., N: \quad \boldsymbol{\phi}^{(i)} \leftarrow \underset{\boldsymbol{\phi}}{\operatorname{argmax}}\, \mathcal{L}(\mathbf{x}^{(i)}; \boldsymbol{\theta}, \boldsymbol{\phi}) \qquad \text{(E-step)}$$

$$\boldsymbol{\theta} \leftarrow \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{i=1}^{N} \mathcal{L}(\mathbf{x}^{(i)}; \boldsymbol{\theta}, \boldsymbol{\phi}) \qquad \text{(M-step)}$$

Figure 1: Expectation Maximization for ELBO (from Kigma 2019)

In principle standard Bayesian inference techniques can be used to approximate the posterior. An option that is the one used for GMM is basically doing Expectation-Maximization which requires a data-point-wise optimization loop in the expectation step which is not very efficient.
the previous discussion may be moved after the introdution of ELBO while resenting so that we can show what is ELBO optimization with E-M and have a clear notation. To me is interesting comparing optimizing ELBO as done in VAE vs EM, but I don't know if it fits with the time

2 VAEs from a probabilistic

Variational Autoencoders are a framework to optimize efficiently DLVM while *at the same time* "optimizing" also the inference model, that is the posterior over the latent variables we introduced above. All this using SGD!

2.1 Decoder

The decoder is the DLVM discussed above in the following just $p_\theta(\mathbf{x}, \mathbf{z})$.

2.2 Encoder

In VAE, to turn the intractable problem of inferring a posterior over the latent variable into a tractable problem we introduce a *parametric inference model* $q_\phi(\mathbf{z}|\mathbf{x})$ also called **Encoder**. ϕ are the variational parameters to be obtimazed in order for the encoder to approximate the true posterior:

$$q_\phi(\mathbf{z}|\mathbf{x}) \approx p_\theta(\mathbf{z}|\mathbf{x}) \tag{5}$$

The inference model can be almost any directed graphical model, what is used and we used is a single deep neural network, hence ϕ are the weigths and biases of the net.
In particular, to fix the ideas, we can imagine that we want to learn a Gaussian posterior in the latent space This is the basic chices but others are also possible. The neural network will learn the mean and the covariance matrix of such Gaussian, its outputs will be $\boldsymbol{\mu}$, $\log \boldsymbol{\sigma}$.
An important difference with normal expectation-maximization is that the parameters of the neural networks are shared for the inference of all points. A single function/model q_ϕ to perform inference over all data points, in the sense

that a single set of parameters ϕ are responsible for the inference of all dataset at difference with expectation maximization where during the expectation step an optimal set is computed at each data point. This sharing of parameters across data points is called *amortized* variational inference, and using SGD is very efficient.

2.3 Optimization objective: ELBO

The optimization objective of the VAE is the Evidence Lower Bound (ELBO) aka variational lower bound. This is the derivation: For any q_ϕ:

$$\log p_\theta(\mathbf{x}) = \mathbf{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}] \tag{6}$$

$$= \mathbf{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_\theta(\mathbf{x},\mathbf{z}}{p_\theta(\mathbf{z}|\mathbf{x})}\right]\right] \tag{7}$$

$$= \mathbf{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_\theta(\mathbf{x},\mathbf{z}}{q_\phi(\mathbf{z}|\mathbf{x})}\frac{q_\phi(\mathbf{z}|\mathbf{x}}{p_\theta(\mathbf{z}|\mathbf{x})}\right]\right] \tag{8}$$

$$= \mathbf{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_\theta(\mathbf{x},\mathbf{z}}{q_\phi(\mathbf{z}|\mathbf{x}}\right]\right] + \mathbf{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{q_\phi(\mathbf{z}|\mathbf{x}}{p_\theta(\mathbf{z}|\mathbf{x}))}\right]\right] \tag{9}$$

The first step is taking the expected value of a quantity which is constant with respect to the distribution.

This is the main relation in understanding VAE. On the LHS we have what we would like to optimize (the marginal likelihood). On the RHS instead we have two terms. The first is the so called ELBO that we will denote by $\mathcal{L}_{\theta,\phi}(\mathbf{x})$ and the second is the Kullback-Leibler divergence between the encoder and the true posterior $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x}))$. This relation can be rewritten in a more intuitive way to understand what the VAE does:

$$\log p_\theta(\mathbf{x}) - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})) = \mathbf{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_\theta(\mathbf{x},\mathbf{z}}{q_\phi(\mathbf{z}|\mathbf{x}}\right]\right] = \mathcal{L}_{\theta,\phi}(\mathbf{x}) \tag{10}$$

With a VAE we optimize the ELBO which, apart from an important detail that we will discuss in a moment, is composed only by tractable parts: both the encoder and the decoder are tractable since they are Deep neural networks.
Maximizing the ELBO isn't exactly equivalent to maximize our real target, the marginal likelihood, but to both maximizing the marginal likelihood and minimizing the DKL between the true posterior and the encoder: this is the essence of what we cited above, with a VAE we both optimize a DLVM and the inference model.
maybe we need a small part on what is the DKL so that in principle one can understand that is zero when the two distribution are the same but all people in the class know that so maybe we would just recall the properties I just mentioned and how it is kind of a "distance" between distribution.

$$D_{KL} >= 0$$

This can be proved with Jensen. This implies that ELBO, as the name says, is a lower bound for the marginal likelihood:

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) = \log p_\theta(\mathbf{x}) - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})) \tag{11}$$
$$\leq \log p_\theta(\mathbf{x}) \tag{12}$$

Being a lower bound is a good choice to maximize the ELBO.

3 VAE from the perspective of neural networks

We have discussed the ideas of Variational Autoencoders from the perspective of variational inference in probability models. But we know that their implementation is also of great interest to deep learning researchers. Let us understand why.

3.1 What are autoencoders

Autoencoders belong in general terms to the family of Artificial Neural Networks that allow for the learning of so-called **latent representations** of data in an **unsupervised** setting. By latent representation we mean low dimensional embeddings of the input data.

In particular, **autoencoders** perform this learning process trying to reproduce their outputs as close as possible to the inputs. They have a very simple structure: the **encoder** maps the data inputs into a latent representation space, and the **decoder** maps the data back from the latent space to the original space.

This would seem like a trivial task for an MLP, for example, that of learning the identity. However one of the main feature of autoencoders is that the latent representation space is a lower-dimensional space with respect to the original one and may impose additional constraints. This means that information flowing from the input to the output is subject to a bottleneck, hence forcing the autoencoder to learn the most important features and drop the others. This "regularisation ability" is due to what is called **undercompleteness**.

Autoencoders come in many flavours:

- the basic implementation, which might be an MLP with one lower dimensional hidden layer, linear activation functions and MSE loss is as simple as a **PCA**. We can hence think of autoencoders as generalizations of simple regularisation strategies like the PCA.

- Other variants may involve the use of **multiple hidden layers**, and the architecture can be tweaked (say, by concatenating weights or by training one layer at a time). These multilayer autoencoders are often used in the context of semi-supervised pre-training: when few labels are available, the whole autoencoder is trained for an unsupervised task and then only the encoder weights are kept for fine-tuning.

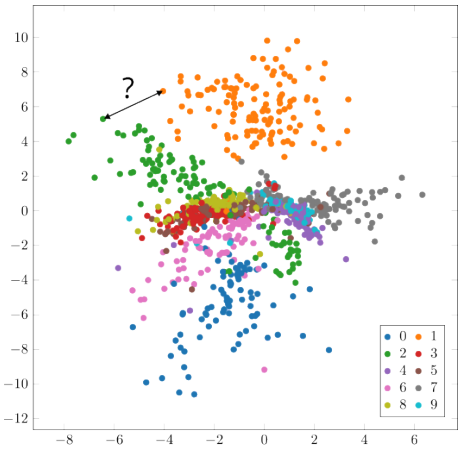

Figure 2: Latent space representation of a MNIST dataset for an autoencoder. Originally from Gertjan van den Burg.

- For tasks of computer vision **convolutional autoencoders** are used, and for tasks of noise removal we can train also **denoising autoencoders** by imposing the constraint of a certain Gaussian noise in the autoencoder.

In summary, we have mentioned many types of **detetrministic autoencoders**, that learn features of data thanks to the undercompleteness property of the neural network structure. Since we want to reconstruct the input image in the output of the autoencoder, we will try to minimize a **reconstruction loss**.

Though deterministic autoencoders learn hidden features, they are still not generative because they do not provide *per se* a schema to interpolate between points in the latent space. The best we can do is to replicate the data points that we have trained the model with. This is due to the fact that the fact that our latent space is discrete.

3.2 Variational Autoencoder

Let us introduce the star of our show: the **Variational Autoencoder** (VAE). First introduced by Diederik Kingma and Max Welling in 2013, VAEs are very different from the other autoencoder siblings. In fact VAEs are **probabilistic** by nature, since they imply a sampling procedure in both the learning and prediction process. The second main feature is that they are **generative**, since they not only learn but also generate instances that look similar to the data in the training set. This is because now the latent space is continuous by design, and we can easily sample and interpolate in this continuous space.

These are two features that VAEs and RBMs have in common, but VAEs are much easier to train since they do not require a thermodynamic equilibrium to be reached for training purposes.

3.3 How is the VAE made

The setting of our problem is the following: we have the $\boldsymbol{x}$ local observed variable and $\boldsymbol{z}$ the local latent variable.

In the **decoder** part we would like to get the **likelihood** $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ that, given a vector of the latent space $\boldsymbol{z}$, we would get a value of the output $\boldsymbol{x}$.

In the **encoder** part we would like to do the opposite process, which is to compute the **posterior** $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ and to optimize the parameters θ with respect to the **marginal likelihood** $p_\theta(\boldsymbol{x})$.

For very simple and discrete cases the computation of $p_\theta(\boldsymbol{z}|\boldsymbol{x})$ is feasible by explicit enumeration. However, as soon as the size of the dataset grows or when latent variables are continuous, the computation becomes intractable.

As it was already pointed out in the previous section, we can tackle the intractability by replacing the posterior $p_\theta(\boldsymbol{z}|vecx)$ with $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ where ϕ are the variational parameters. Our goal is to attain a $\phi = \phi^*$ such that

$$\phi^* = \arg\min_\phi \left\{ \mathcal{D}_{\mathrm{KL}} \Big[q_\phi(\boldsymbol{z}|\boldsymbol{x}) \;\Big|\Big|\; p_\theta(\boldsymbol{z}|\boldsymbol{x}) \Big] \right\} \tag{13}$$

We do not know how to minimize eq. 13 as well: so we resort to maximising the Evidence Lower Bound, defined as

$$\mathrm{ELBO}(x;\theta,\phi) = \langle \log p_\theta(\boldsymbol{x}|\boldsymbol{z}) \rangle_{q_\phi(\boldsymbol{z}|\boldsymbol{x})} - \mathcal{D}_{\mathrm{KL}} \Big[q_\phi(\boldsymbol{z}|\boldsymbol{x}) \;\Big|\Big|\; p_\theta(\boldsymbol{z}|\boldsymbol{x}) \Big] \tag{14}$$

3.4 Why use a neural network?

Now we may ask ourselves what inference scheme should we use. One trivial choice would be to just compute the exact posterior, but as we stated it is often the case that the computation is intractable. So we formulated a variational problem by supposing that there exists a tentative posterior $q_\phi(\boldsymbol{x}|\boldsymbol{z})$ that can sufficiently well approximate the real one.

This problem can be tackled in two ways:

- We can straightforwardly minimize following an Expectation-Maximisation scheme, and if EM relies on gradient-based optimisation we call this **stochastic variational inference** (SVI).
- We can rather train a neural network to predict the variational parameters λ using a **deep inference** technique. For the purpose, an **inference network** (aka the encoder) is asked to predict the parameters of the approximate posterior. Instead of training the network to optimize a separate $\lambda^{(n)}$ value of the variational parameter for each $\boldsymbol{x}^{(n)}$ that belongs to the dataset, we may well have just one encoder network. This technique is called **amortized variational inference** (AVI) because the task of performing inference is amortized across the entire dataset.

The second type of inference is deemed to be much faster and less computationally expensive with respect to any other technique, and this is why neural networks are the most popular choice in this setting.

So instead of looking for an n-wise parametrization $\lambda_n = (\boldsymbol{\mu}_n, \boldsymbol{\sigma}_n)$ we will look for ϕ such that we will have for each data point $\boldsymbol{x}_n$ a tuple $(\boldsymbol{\mu}_\phi(\boldsymbol{x}_n), \boldsymbol{\sigma}_\phi(\boldsymbol{x}_n))$. We replaced *local parameters* λ_n with *global variables* ϕ.

3.5 Computing the loss function

Now we have to attempt to maximize the loss function of choice, which in our case is the ELBO, with respect to the parameter sets of the encoder and of the decoder (respectively θ and ϕ). We will suppose that the prior is Gaussian

$$p_\theta(\boldsymbol{z}) \sim \mathcal{N}(\boldsymbol{z}; \mathbf{0}, \mathbf{1})$$

and also the approximated posterior is Gaussian

$$q_\phi(\boldsymbol{z}|\boldsymbol{x}) \sim \mathcal{N}(\boldsymbol{z}; \boldsymbol{\mu}, \Sigma)$$

where $\Sigma = \mathrm{diag}(\sigma^2)$ is the covariance matrix having only nonzero diagonal elements given by σ_l^2 the l-th variance where l goes from 1 to L the dimension of the latent space. Then

$$\int q_\phi(\boldsymbol{z}) \log p(\boldsymbol{z}) \, \mathrm{d}\boldsymbol{z} = \int \mathrm{d}\boldsymbol{z} \, \frac{1}{\sqrt{\det(\Sigma)}\sqrt{2\pi}^L} \prod_{l=1}^{L} \exp\left\{ -\frac{(z_l - \mu_l)^2}{2\sigma_l^2} \right\} \log \frac{1}{\sqrt{2\pi}^L} e^{-\frac{|\boldsymbol{z}|^2}{2}} \tag{15}$$

$$= \int \mathrm{d}\boldsymbol{z} \, \frac{1}{\sqrt{\det(\Sigma)}\sqrt{2\pi}^L} \prod_{l=1}^{L} \exp\left\{ -\frac{(z_l - \mu_l)^2}{2\sigma_l^2} \right\} \left[-\frac{L}{2} \log 2\pi - \frac{|\boldsymbol{z}|^2}{2} \right] \tag{16}$$

$$= -\frac{L}{2} \log 2\pi - \frac{1}{2} \sum_{l=1}^{L} (\sigma_l^2 + \mu_l^2) \tag{17}$$

and

$$\int q_\phi(\boldsymbol{z}) \log q_\phi(\boldsymbol{z}) \, \mathrm{d}\boldsymbol{z} = \int \mathrm{d}\boldsymbol{z} \, \frac{1}{\sqrt{\det(\Sigma)}\sqrt{2\pi}^L} \prod_{l=1}^{L} \exp\left\{ -\frac{(z_l - \mu_l)^2}{2\sigma_l^2} \right\} \log \frac{1}{\sqrt{\det(\Sigma)}\sqrt{2\pi}^L} \exp\left\{ -\frac{(z_l - \mu_l)^2}{2\sigma_l^2} \right\} \tag{18}$$

$$= \int \mathrm{d}\boldsymbol{z} \, \frac{1}{\sqrt{\det(\Sigma)}\sqrt{2\pi}^L} \prod_{l=1}^{L} \exp\left\{ -\frac{(z_l - \mu_l)^2}{2\sigma_l^2} \right\} \left[-\frac{L}{2} \log 2\pi - \log \sigma_l^2 - \frac{(z_l - \mu_l)^2}{2\sigma_l^2} \right] \tag{19}$$

$$= -\frac{L}{2} \log 2\pi - \frac{1}{2} \sum_{l=1}^{L} (1 + \log \sigma_l^2) \tag{20}$$

hence the Kullback-Leibler divergence is

$$\mathcal{D}_{\mathrm{KL}}\Big[q_\phi(\boldsymbol{z}|\boldsymbol{x}) \;\Big|\Big|\; p_\theta(\boldsymbol{z})\Big] = \int \mathrm{d}z\, q_\phi(\boldsymbol{z})[\log p_\theta(\boldsymbol{z}) - \log q_\phi(\boldsymbol{z})] \tag{21}$$

$$= \frac{1}{2}\sum_{l=1}^{L}\left[1 + \log\sigma_l^2 - \sigma_l^2 - \mu_l^2\right] \tag{22}$$

where the "conditioned on $\boldsymbol{x}$" just implies that the μ's and the σ's have a dependency on $\boldsymbol{x}$.

Recall that our loss function is

$$\mathcal{L}_{\theta,\phi}(\boldsymbol{x}) = \left\langle \log \frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \right\rangle_{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \tag{23}$$

$$= \langle \log p_\theta(\boldsymbol{x}|\boldsymbol{z})\rangle_{q_\phi(\boldsymbol{z}|\boldsymbol{x})} - \mathcal{D}_{\mathrm{KL}}\Big[q_\phi(\boldsymbol{z}|\boldsymbol{x}) \;\Big|\Big|\; p_\theta(\boldsymbol{z}|\boldsymbol{x})\Big] \tag{24}$$

We can compute the gradient with respect to θ by using eq. 23

$$\nabla_\theta \mathcal{L}_{\theta,\phi}(\boldsymbol{x}) = \nabla_\theta \langle \log p_\theta(\boldsymbol{x},\boldsymbol{z})\rangle_{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \tag{25}$$

$$= \langle \nabla_\theta \log p_\theta(\boldsymbol{x}|\boldsymbol{z})\rangle_{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \tag{26}$$

since we are extremising, we do not care about terms that do not depend on θ.

The gradient with respect to ϕ is a little less trivial. Use eq. 24 and eq. 22

$$\nabla_\phi \mathcal{L}_{\theta,\phi}(\boldsymbol{x}) = \nabla_\phi \langle \log p_\theta(\boldsymbol{x}|\boldsymbol{z})\rangle_{q_\phi(\boldsymbol{z}|\boldsymbol{x})} - \nabla_\phi \mathcal{D}_{\mathrm{KL}}\Big[q_\phi(\boldsymbol{z}|\boldsymbol{x}) \;\Big|\Big|\; p_\theta(\boldsymbol{z}|\boldsymbol{x})\Big] \tag{27}$$

$$= \nabla_\phi \langle \log p_\theta(\boldsymbol{x}|\boldsymbol{z})\rangle_{q_\phi(\boldsymbol{z}|\boldsymbol{x})} + \frac{1}{2}\sum_{l=1}^{L}\nabla_\phi[1 + \log\sigma_l^2 - \sigma_l^2 - \mu_l^2] \tag{28}$$

The second term is trivial since we have the closed formed available. The first term, however, is not at all trivial since we have a dependency on ϕ in the measure of the expectation value. In practical terms, we are unable to commute the gradient with the expectation value. This is strictly related to the fact that even if we are able to make a forward pass in the neural network, we are unable to backpropagate over the random sampling.

3.6 The reparametrization trick

Let us pick up the last concept we exposed: we are unable to backpropagate because we have a probabilistic step that makes the pipeline non differentiable. An idea would be to make the step *deterministic*.

This is done through what is called **reparametrization trick**: we replace the probabilistic variable $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0},\boldsymbol{1})$ with a deterministic "surrogate" map

$$\boldsymbol{z} = g_\phi(\epsilon,\boldsymbol{x}) \tag{29}$$

parametrized by ϕ, where $\boldsymbol{\epsilon}$ is an auxiliary variable with probability $p(\boldsymbol{\epsilon})$.

Because of eq. 29 we know that

$$q_\phi(\boldsymbol{z}|\boldsymbol{x})\,\mathrm{d}\boldsymbol{z} = p(\boldsymbol{\epsilon})\,\mathrm{d}\boldsymbol{\epsilon} \tag{30}$$

then the expectation value of the first term in eq. 28 becomes

$$\begin{aligned}
\langle \log p_\theta(\boldsymbol{x}|\boldsymbol{z})\rangle &= \int \mathrm{d}\boldsymbol{z}\, q_\phi(\boldsymbol{z}|\boldsymbol{x}) f(\boldsymbol{z}) && (31)\\
&= \int \mathrm{d}\boldsymbol{\epsilon}\, p(\boldsymbol{\epsilon}) f(\boldsymbol{z}) && (32)\\
&= \int \mathrm{d}\boldsymbol{\epsilon}\, p(\boldsymbol{\epsilon}) f(g_\phi(\boldsymbol{\epsilon}, \boldsymbol{x})) && (33)\\
&\simeq \frac{1}{N}\sum_{i=1}^{N} f(g_\phi(\boldsymbol{\epsilon}^{(i)}, \boldsymbol{x}^{(i)})) && (34)
\end{aligned}$$

where $\epsilon^{(n)}$ is the new parameter set for the n-th data point.

Then we can **reparametrize** the random variable $\boldsymbol{z}$. Since we are dealing with Gaussian distributions, we can for example choose

$$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{1}) \qquad \boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} \tag{35}$$

In the case of other distributions it is not necessarily straightforward to reparametrize. Refer to Kingma and Welling.

At this point we can conclude the differentiation left in eq. 28

$$\nabla_\phi \mathcal{L}_{\theta,\phi}(\boldsymbol{x}) = \nabla_\phi \left\langle \log p_\theta(\boldsymbol{x}|\boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon})\right\rangle_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{1})} + \frac{1}{2}\sum_{l=1}^{L} \nabla_\phi [1 + \log \sigma_l^2 - \sigma_l^2 - \mu_l^2] \tag{36}$$

$$= \left\langle \nabla_\phi \log p_\theta(\boldsymbol{x}|\boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon})\right\rangle_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{1})} + \frac{1}{2}\sum_{j=1}^{L} \nabla_\phi [1 + \log \sigma_l^2 - \sigma_l^2 - \mu_l^2] \tag{37}$$

3.7 The loss function and the neural network structure

Now we can hence write the expression for a loss function that we know we can backpropagate. If we assume the approximate posterior to be Gaussian, the final expression for the loss function will be:

$$\begin{aligned}
\mathcal{L}_{\theta,\phi}(\boldsymbol{x}_i) &= \frac{1}{2}\sum_{l=1}^{L}[1 + \log {\sigma^{(i)}}_l^2 - {\sigma^{(i)}}_l^2 - {\mu^{(i)}}_l^2] + \frac{1}{N}\sum_{i=1}^{N} \log p_\theta(\boldsymbol{x}^{(i)}|z^{(i)}) && (38)\\
&= \frac{1}{2}\sum_{l=1}^{L}[1 + \log {\sigma^{(i)}}_l^2 - {\sigma^{(i)}}_l^2 - {\mu^{(i)}}_l^2] + \frac{1}{N}\sum_{i=1}^{N} (y^{(i)} - x^{(i)})^2 && (39)
\end{aligned}$$

C.2 Gaussian MLP as encoder or decoder

In this case let encoder or decoder be a multivariate Gaussian with a diagonal covariance structure:

$$\begin{aligned} \log p(\mathbf{x}|\mathbf{z}) &= \log \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2 \mathbf{I}) \\ \text{where } \boldsymbol{\mu} &= \mathbf{W}_4 \mathbf{h} + \mathbf{b}_4 \\ \log \boldsymbol{\sigma}^2 &= \mathbf{W}_5 \mathbf{h} + \mathbf{b}_5 \\ \mathbf{h} &= \tanh(\mathbf{W}_3 \mathbf{z} + \mathbf{b}_3) \end{aligned} \tag{12}$$

where $\{\mathbf{W}_3, \mathbf{W}_4, \mathbf{W}_5, \mathbf{b}_3, \mathbf{b}_4, \mathbf{b}_5\}$ are the weights and biases of the MLP and part of $\boldsymbol{\theta}$ when used as decoder. Note that when this network is used as an encoder $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, then $\mathbf{z}$ and $\mathbf{x}$ are swapped, and the weights and biases are variational parameters $\boldsymbol{\phi}$.

Figure 3: We are replacing this with our expressions.

We would also like to explicitly state the forward chain in the one hidden layer per encoder and decoder case. The decoder is:

$$\log p(\boldsymbol{x}|\boldsymbol{z}) = \log \mathcal{N}() \tag{40}$$

$$\text{where } \boldsymbol{\mu} = \tag{41}$$

4 Conclusion

1. complication: got stuck at the beginning of training in a minimum due to minimization of DKL. Solution: parameter beta in front of DKL to be annealed from 0 to 1 in the first few epochs. examples of use: "Generating Sentences from a Continuous Space", and "Ladder VAE" to be cited
2. VAE vs GAN: very high level GANs are complicated
3. VAE for MSA as Weigt project, using code to train for our dataset and comparing results with the generative model I already have.

For the conclusion there is an high prpbability only one or two point can be discussed.