Estimating the gradient of the ELBO

Feb 8, 2021 • Massimiliano Patacchiola

This is the second post of the series on variational inference. In the previous post I have introduced the variational framework, and the three main characters at play: the evidence, the Kullback-Leibler (KL) divergence, and the Evidence Lower BOund (ELBO). I have showed how the ELBO can be considered as a surrogate objective for finding the posterior distribution $p(\mathbf{z}|\mathbf{x})$ when the evidence $p(\mathbf{x})$ is intractable. By maximizing the ELBO we can find the parameters $\boldsymbol{\theta}$ of a variational distribution $q_\theta(\mathbf{z})$ that better fit the posterior. However, I did not mention how the ELBO itself can by maximized in practice. One approach is to estimate the gradient of the ELBO w.r.t. $\boldsymbol{\theta}$ and then update $\boldsymbol{\theta}$ in the direction of steepest ascent. Depending on the particular problem at hand and the choice of variational distribution, this could be challenging.

In this post I will focus on this particular problem, showing how we can estimate the gradients of the ELBO by using two techniques: the score function estimator (a.k.a. REINFORCE) and the pathwise estimator (a.k.a. reparametrization trick).

Definition of the problem

We are interested in a system with two components: a stochastic component named *measure* and a *cost function*. Training consists of two phases: a simulation phase and an optimisation phase. The entire system is stochastic since one of the element is stochastic. However, in many cases the system is said to be *doubly-stochastic* if for instance we are using stochastic gradient descent in the optimization component. In a doubly-stochastic system one source of randomness arises from the simulation phase (e.g. using Monte Carlo estimators) and a second source arises in the optimization phase (e.g. sampling datapoints in the mini-batch gradient descent).

Let's try now to better formalize our setting. As usual, I will use the bold notation $\mathbf{x}$ to represent a collection of random variables. Consider a generic density function $p_\theta(\mathbf{x})$ parameterized by a vector $\boldsymbol{\theta}$, representing the stochastic component (measure), and a function $f(\mathbf{x})$, representing the *cost function*. We assume $p_\theta(\mathbf{x})$ to be differentiable, but the cost function $f(\mathbf{x})$ is not necessarily differentiable, for instance it could be discrete or a black-

box (e.g. only the outputs are given). Since we are dealing with expectations (integrals) we can use Monte Carlo to get an unbiased approximation of the expected value. Monte Carlo numerically evaluates the integral by drawing samples $\hat{\mathbf{x}}^{(1)},\dots,\hat{\mathbf{x}}^{(N)}$ from the distribution $p_\theta(\mathbf{x})$ and computing the average of the function evaluated at these points. In our case, the Monte Carlo approximation of $\mathbb{E}_{p_\theta}[f(\mathbf{x})]$ corresponds to

$$\mathbb{E}_{p_\theta}[f(\mathbf{x})] \approx \frac{1}{N}\sum_{n=1}^{N} f(\hat{\mathbf{x}}^{(n)}) \quad \text{with} \quad \hat{\mathbf{x}}^{(n)} \sim p_\theta(\mathbf{x}),$$

where N is the to total number of samples. However, in many applications we are interested in something slightly different: we wish to get the *gradient of the expectation*. This raises an issue, the gradient of an expectation cannot be approximated via Monte Carlo as we usually do. Let's see why:

$$\nabla_\theta \mathbb{E}_{p_\theta}[f(\mathbf{x})] = \nabla_\theta \int p_\theta(\mathbf{x}) f(\mathbf{x}) d\mathbf{x} \qquad (1.1)$$

$$= \int \underbrace{\nabla_\theta p_\theta(\mathbf{x})}_{\text{issue}} f(\mathbf{x}) d\mathbf{x} \qquad (1.2)$$

In the last step I have used the Leibniz's rule to move the gradient inside the integral. Here is the issue, most of the time the gradient of a density function is not itself a density function, since it may have negative values and could not integrate to one. In our particular case, the quantity $\nabla_\theta p(\mathbf{x}_\theta)$ may not be a proper density. Why? Well a density must be positive and sum up to one, while the gradient can be negative and does not sum up to one. It follows that the integral in (1.2) cannot be casted into an expectation and therefore it cannot be approximated as usual via Monte Carlo.

When we try to get the gradient of the ELBO we incur in the same issue, since the gradient of the ELBO corresponds to the gradient of an expectation:

$$\nabla_\theta \text{ELBO}(q) = \nabla_\theta \mathbb{E}_q\big[\log p(\mathbf{x},\mathbf{z}) - \log q(\mathbf{z})\big].$$

Note that, the gradient of the ELBO can also be expressed in terms of measure and cost function: the variational distribution q represents the measure and the term inside the brackets the cost function.

In the next sections I will present the score function estimator and the pathwise gradient estimator, which are part of two different groups, the former differentiate through the measure and the latter through the cost function.

The score function estimator

The score function estimator is also known as the likelihood-ratio estimator and as REINFORCE in the reinforcement learning literature. This estimator

gets the gradient by computing the derivatives of the measure, meaning by direct differentiation through $p_\theta(\mathbf{x})$. The estimator is particularly popular because it is unbiased, flexible, and it does not impose any restriction over $p_\theta(\mathbf{x})$ or $f(\mathbf{x})$. However, a major issue is that is has large variance. Before showing the derivation of the estimator, it is important to provide a definition of score and introduce some tools like the log-derivative trick.

What is the score? In statistical terms, the score (or score function) is just the gradient of the log-likelihood function with respect to the parameter vector. In our case this correspond to

$$\text{score} = \nabla_\theta \log p_\theta(\mathbf{x}).$$

The log-derivative trick. The log-derivative trick just consists of applying the chain rule to a composite function in which the outermost term is a logarithm. In our case the score function happens to be in such a form, with $\log$ being the external term and $p_\theta(\mathbf{x})$ the internal one. It follows that we can apply the log-derivative trick straight away to get

$$\nabla_\theta \log p_\theta(\mathbf{x}) = \frac{1}{p_\theta(\mathbf{x})} \cdot \nabla_\theta p_\theta(\mathbf{x}) = \frac{\nabla_\theta p_\theta(\mathbf{x})}{p_\theta(\mathbf{x})}.$$

The last term is called the *score ratio*. The log-derivative trick can be quite helpful. For instance, exploiting this trick we can notice an interesting property of the score function: its expected value is equal to zero

$$\mathbb{E}_{p_\theta}[\nabla_\theta \log p_\theta(\mathbf{x})] = \mathbb{E}_{p_\theta}\left[\frac{\nabla_\theta p_\theta(\mathbf{x})}{p_\theta(\mathbf{x})}\right] = \int p_\theta(\mathbf{x}) \frac{\nabla_\theta p_\theta(\mathbf{x})}{p_\theta(\mathbf{x})} d\mathbf{x} = \nabla_\theta \int p_\theta(\mathbf{x}) d\mathbf{x} = \nabla_\theta 1 = 0.$$

This property will be helpful when we derive the two gradient estimators of the ELBO; it is also fundamental in the context of control variates, and it has been exploited by Roeder et al. (2017) for variance reduction. Armed with the log-derivative trick, we are ready to derive the score function estimator.

Derivation of the estimator. We want to overcome the problem described in the previous section, where we saw that the gradient of an expectation cannot be approximated via Monte Carlo. Our goal here will be to exploit the log-derivative trick to bypass this issue. In particular, we want to reach a friendly form of the integral and turn it into a proper expectation. Let's break it down

$$\nabla_\theta \mathbb{E}_{p_\theta}[f(\mathbf{x})] = \nabla_\theta \int p_\theta(\mathbf{x}) f(\mathbf{x}) d\mathbf{x} \tag{2.1}$$

$$= \int \nabla_\theta p_\theta(\mathbf{x}) f(\mathbf{x}) d\mathbf{x} \tag{2.2}$$

$$= \int \frac{p_\theta(\mathbf{x})}{p_\theta(\mathbf{x})} \nabla_\theta p_\theta(\mathbf{x}) f(\mathbf{x}) d\mathbf{x} \tag{2.3}$$

$$= \int p_\theta(\mathbf{x}) \frac{\nabla_\theta p_\theta(\mathbf{x})}{p_\theta(\mathbf{x})} f(\mathbf{x}) d\mathbf{x} \tag{2.4}$$

$$= \int p_\theta(\mathbf{x}) \nabla_\theta \log p_\theta(\mathbf{x}) f(\mathbf{x}) d\mathbf{x} \tag{2.5}$$

$$= \mathbb{E}_{p_\theta}\big[\underbrace{\nabla_\theta \log p_\theta(\mathbf{x})}_{\text{score}} \underbrace{f(\mathbf{x})}_{\text{cost}}\big] \tag{2.6}$$

Note that, (2.5) is a proper expectation and we can now get the Monte Carlo approximation of the gradient as we wanted

$$\mathbb{E}_{p_\theta}\big[\nabla_\theta \log p_\theta(\mathbf{x}) f(\mathbf{x})\big] \approx \frac{1}{N} \sum_{n=1}^{N} \nabla_\theta \log p_\theta(\hat{\mathbf{x}}^{(n)}) f(\hat{\mathbf{x}}^{(n)}) \quad \text{with} \quad \hat{\mathbf{x}}^{(n)} \sim p_\theta(\mathbf{x}).$$

Here is a step-by-step description of the procedure I have followed:

$(2.1 \text{ and } 2.2)$ Those two steps are identical to the one described in the previous section. In the first step apply the definition of expectation and in the second the Leibniz's rule to move the gradient inside the integral.

$(2.2 \to 2.3)$ We use the identity trick by adding a new term at our convenience. The value of the new term is equal to 1, therefore it does not have any effect on the product.

$(2.3 \to 2.4)$ Rearranging the terms by switching the denominator.

$(2.4 \to 2.5)$ After rearranging the terms we notice that it is possible to use the log-derivative trick.

$(2.5 \to 2.6)$ The new form of the integral corresponds to a proper expectation, therefore we can rewrite in the equivalent form.

The score function estimator of the ELBO. Applying the score function estimator to the ELBO has been known as Black Box Variational Inference (BBVI). The derivation for this case follows the generic derivation given above with some minor differences:

$$\nabla_\theta \text{ELBO}(q) = \nabla_\theta \mathbb{E}_q\big[\log p(\mathbf{x},\mathbf{z}) - \log q_\theta(\mathbf{z})\big] = \nabla_\theta \mathbb{E}_q\big[f_\theta(\mathbf{z})\big] \tag{3.1}$$

$$= \int \nabla_\theta \big(q_\theta(\mathbf{z}) f_\theta(\mathbf{z})\big) dz \tag{3.2}$$

$$= \int \nabla_\theta q_\theta(\mathbf{z}) f_\theta(\mathbf{z}) + q_\theta(\mathbf{z}) \nabla_\theta f_\theta(\mathbf{z}) dz \tag{3.3}$$

$$= \int q_\theta(\mathbf{z}) \nabla_\theta \log q_\theta(\mathbf{z}) f_\theta(\mathbf{z}) + q_\theta(\mathbf{z}) \nabla_\theta f_\theta(\mathbf{z}) dz \tag{3.4}$$

$$= \mathbb{E}_q\big[\nabla_\theta \log q_\theta(\mathbf{z}) f_\theta(\mathbf{z}) + \nabla_\theta f_\theta(\mathbf{z})\big] \tag{3.5}$$

$$= \mathbb{E}_q\big[\nabla_\theta \log q_\theta(\mathbf{z}) f_\theta(\mathbf{z})\big] + \mathbb{E}_q\big[\nabla_\theta f_\theta(\mathbf{z})\big] \tag{3.6}$$

$$= \mathbb{E}_q\big[\nabla_\theta \log q_\theta(\mathbf{z}) f_\theta(\mathbf{z})\big] + \mathbb{E}_q\big[\nabla_\theta \log p(\mathbf{x},\mathbf{z}) - \nabla_\theta \log q_\theta(\mathbf{z})\big] \tag{3.7}$$

$$= \mathbb{E}_q\big[\underbrace{\nabla_\theta \log q_\theta(\mathbf{z})}_{\text{score}} \underbrace{\big(\log p(\mathbf{x},\mathbf{z}) - \log q_\theta(\mathbf{z})\big)}_{\text{cost}}\big] \tag{3.8}$$

(3.1) To avoid cluttering I used $f_\theta(\mathbf{z})$ to represent the cost function (the quantity inside the brackets). Note that, in our particular case $f_\theta(\mathbf{z})$ depends on θ the variational parameters, since it contains the variational distribution $q_\theta(\mathbf{z})$.

$(3.1 \rightarrow 3.2)$ Applying the definition of expectation to get the integral and moving the gradient using Leibniz's rule.

$(3.2 \rightarrow 3.3)$ Unlike Equation (2.2) in the general derivation, here it is not possible to directly apply the log-derivative trick because both $f_\theta(\mathbf{z})$ and $q_\theta(\mathbf{z})$ are functions of the target parameters $\boldsymbol{\theta}$. However, it is possible to use the product rule of derivatives to split the gradient.

$(3.3 \rightarrow 3.4)$ Exploiting the log-derivative trick to rewrite $\nabla_\theta \log q_\theta(\mathbf{z})$.

$(3.4 \rightarrow 3.5)$ Switched back to expectation notation.

$(3.5 \rightarrow 3.6)$ Using the linearity of expectation to split the sum into two separate terms.

$(3.6 \rightarrow 3.7)$ Plugging back the original terms associated to $f_\theta(\mathbf{z})$.

$(3.7 \rightarrow 3.8)$ We can remove the second term in (3.7), because

$$\mathbb{E}_q\big[\underbrace{\nabla_\theta \log p(\mathbf{x},\mathbf{z})}_{=0} - \underbrace{\nabla_\theta \log q_\theta(\mathbf{z})}_{\text{score}}\big] = \mathbb{E}_q[0] = 0,$$

that is, the gradient of $p(\mathbf{x},\mathbf{z})$ w.r.t θ is zero, because the distribution does not depend on those parameters (those are the parameters of the variational distribution); the expectation of the score function is also equal to zero, as showed in the previous section.

(3.8) The expectation can be estimated using Monte Carlo if two requisites are satisfied: (i) it must be possible to sample $\hat{\mathbf{z}} \sim q_\theta(\mathbf{z})$, and (ii) $\log q_\theta(\mathbf{z})$

must be differentiable w.r.t. $\boldsymbol{\theta}$. Since $q_\theta(\mathbf{z})$ is our variational distribution we can choose a family that satisfies both those requisites.

The pathwise gradient estimator

Another way to estimate the gradient consists of differentiating the cost function $f(\mathbf{x})$ through the random variable $\mathbf{x}$, which encodes the pathway from the target parameters $\boldsymbol{\theta}$. This approach is the one used in the pathwise gradient estimator. The pathwise estimator also appears under several names, such as process derivative, pathwise derivative, and more recently as the reparameterisation trick. A well know application of the reparameterization trick is in the context of Variational Auto-Encoders (VAEs), where is used for backpropagating through a stochastic hidden layer.

In the previous section we noted that the score function estimator has a large variance, meaning that the resulting signal does a poor job at finding the parameters we are interested in. The pathwise estimator solve this issue, since it has low variance, but on the other hand it requires a differentiable cost function and the possibility to reparameterize the measure $p_\theta(\mathbf{x})$. The idea behind the pathwise estimator is to exploit the structural property of the system. In particular, here we care about the sequence of transformations that are applied to the stochastic component affecting the overall objective. Those transformations pass through the measure and into the cost function.

Sampling path (or sampling process). In continuous probability distributions samples can be drawn directly and indirectly as

$$\hat{\mathbf{x}} \sim p_\theta(\mathbf{x}) \quad \text{and} \quad \hat{\mathbf{x}} = t_\theta(\hat{\boldsymbol{\epsilon}}) \quad \hat{\boldsymbol{\epsilon}} \sim p(\boldsymbol{\epsilon}).$$

The indirect approach is also known as the *sampling path* or *sampling process* and it consists of first picking an intermediate sample $\hat{\boldsymbol{\epsilon}}$ from a known distribution $p(\boldsymbol{\epsilon})$, which is independent from $\boldsymbol{\theta}$, and then transform the intermediate sample through a deterministic path $t_\theta(\hat{\boldsymbol{\epsilon}})$ to get $\hat{\mathbf{x}}$. We will refer to the intermediate distribution $p(\boldsymbol{\epsilon})$ as the *base distribution*, and to the function $t_\theta(\hat{\boldsymbol{\epsilon}})$ as the *sampling path*. Let's see a practical example. For simplicity, I assume that the distribution we want to approximate $q_\theta(\mathbf{x})$ is a univariate Gaussian $\mathcal{N}(x|\mu, \sigma)$ (in this case x, μ, σ are scalars) and that we are interested in finding the parameters of this Gaussian $\boldsymbol{\theta} = \{\mu, \sigma\}$. In a similar way the estimator can be adapted to many other continuous univariate distributions such as Gamma, Beta, von Mises, and Student. We need a valid path and a base distribution, obvious choices are the *location-scale* transform and the *standard* Gaussian

$$x = t_\theta(\epsilon) = \mu + \sigma\epsilon \quad \text{and} \quad \mathcal{N}(\epsilon|0, 1).$$

Note that, this path is reversible, meaning that we can get $\epsilon \to x$ through t_θ and $x \to \epsilon$ through the inverse path t_θ^{-1} as follows

$$\epsilon = t_\theta^{-1}(x) = \frac{x - \mu}{\sigma}.$$

Expectation via LOTUS. We need to reconcile sampling paths with the formalism of our main objective that is, estimating an expected value given the measure $p_\theta(\mathbf{x})$ and the cost function $f(\mathbf{x})$. Comparing our target expectation with the expectation derived from the sampling path, we get

$$\mathbb{E}_{p_\theta(\mathbf{x})}[f(\mathbf{x})] \quad \text{and} \quad \mathbb{E}_{p(\boldsymbol{\epsilon})}[f(t_\theta(\boldsymbol{\epsilon}))].$$

The question is: are those expectations equivalent? To answer this question we invoke the *law of the unconscious statistician* or LOTUS, which states that we can compute the expectation of a function of a random variable (in our case the cost function f) without the need of knowing its distribution (in our case the measure p_θ) whenever we use a valid sampling path and a base distribution. In other words, we can replace the original expectation with the expectation over the base distribution $p(\boldsymbol{\epsilon})$ using the transformation $t_\theta(\boldsymbol{\epsilon})$. Therefore, LOTUS implies that the two expectations are equivalent.

Derivation of the estimator. We can exploit sampling paths and LOTUS to derive the pathwise estimator. All we need is a differentiable sampling path $t_\theta(\boldsymbol{\epsilon})$ and a base distribution $p(\boldsymbol{\epsilon})$. The derivation goes like this:

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{p_\theta(\mathbf{x})}[f(\mathbf{x})] &= \nabla_\theta \mathbb{E}_{p(\boldsymbol{\epsilon})}[f(t_\theta(\boldsymbol{\epsilon}))] && (4.1)\\
&= \nabla_\theta \int p(\boldsymbol{\epsilon}) f(t_\theta(\boldsymbol{\epsilon})) d\boldsymbol{\epsilon} && (4.2)\\
&= \int p(\boldsymbol{\epsilon}) \nabla_\theta f(t_\theta(\boldsymbol{\epsilon})) d\boldsymbol{\epsilon} && (4.3)\\
&= \int p(\boldsymbol{\epsilon}) \nabla_x f(\mathbf{x}) \nabla_\theta t_\theta(\boldsymbol{\epsilon}) d\boldsymbol{\epsilon} && (4.4)\\
&= \mathbb{E}_{p(\boldsymbol{\epsilon})}[\underbrace{\nabla_\theta t_\theta(\boldsymbol{\epsilon})}_{\nabla \text{path}} \underbrace{\nabla_x f(\mathbf{x})}_{\nabla \text{cost}}] && (4.5)
\end{aligned}$$

(4.1) Applying LOTUS replacing the original expectation based on $\mathbf{x}$ with the surrogate expecation based on $\boldsymbol{\epsilon}$.

$(4.1 \rightarrow 4.2)$ Rewriting in integral form by applying the definition of expectation.

$(4.2 \rightarrow 4.3)$ Moving the gradient inside the integral, this keeps intact the distribution $p(\boldsymbol{\epsilon})$ which is independent from $\boldsymbol{\theta}$.

$(4.3 \rightarrow 4.4)$ Applying the chain rule to unpack the gradient of the composite function.

$(4.4 \rightarrow 4.5)$ Moving the gradient inside the expectation, reorganizing the terms, and switching back to the expectation form. Note that, unlike the

derivation of the score estimator we did not need to apply the log-derivative trick.

(4.5) It is now possible to apply the Monte Carlo estimator to approximate the gradient

$$\nabla_\theta \mathbb{E}_{p_\theta}[f(\mathbf{x})] \approx \frac{1}{N}\sum_{n=1}^{N} \nabla_\theta t_\theta(\hat{\boldsymbol{\epsilon}}^{(n)}) \nabla_x f(\mathbf{x}^{(n)}) \quad \text{with} \quad \hat{\boldsymbol{\epsilon}}^{(n)} \sim p(\boldsymbol{\epsilon}) \quad \text{and} \quad \mathbf{x}^{(n)} = t_\theta(\hat{\boldsymbol{\epsilon}}^{(n)}).$$

It is important to notice that unlike the score function estimator, here we are taking the gradient (or derivative) of the cost function, therefore the cost must always be differentiable (no black-box allowed).

The pathwise estimator of the ELBO. Let's apply the pathwise estimator to our original problem: estimating the gradient of the ELBO.

$$\begin{aligned}
\nabla_\theta \text{ELBO} &= \nabla_\theta \mathbb{E}_{q_\theta(\mathbf{z})}\big[\log p(\mathbf{x},\mathbf{z}) - \log q_\theta(\mathbf{z})\big] && (5.\\
&= \mathbb{E}_{p(\boldsymbol{\epsilon})}\big[\nabla_\theta \log p(\mathbf{x}, t_\theta(\boldsymbol{\epsilon})) - \nabla_\theta \log q_\theta(t_\theta(\boldsymbol{\epsilon}))\big] && (5.\\
&= \mathbb{E}_{p(\boldsymbol{\epsilon})}\big[\nabla_z \log p(\mathbf{x},\mathbf{z})\nabla_\theta t_\theta(\boldsymbol{\epsilon}) - \nabla_z \log q_\theta(\mathbf{z})\nabla_\theta t_\theta(\boldsymbol{\epsilon}) - \nabla_\theta \log q_\theta(t_\theta(\boldsymbol{\epsilon}))\big] && (5.\\
&= \mathbb{E}_{p(\boldsymbol{\epsilon})}\big[\nabla_z \log p(\mathbf{x},\mathbf{z})\nabla_\theta t_\theta(\boldsymbol{\epsilon}) - \nabla_z \log q_\theta(\mathbf{z})\nabla_\theta t_\theta(\boldsymbol{\epsilon})\big] - \mathbb{E}_{p(\boldsymbol{\epsilon})}\big[\nabla_\theta \log q_\theta(t_\theta(\boldsymbol{\epsilon}))\big] && (5.\\
&= \mathbb{E}_{p(\boldsymbol{\epsilon})}\big[\nabla_z \log p(\mathbf{x},\mathbf{z})\nabla_\theta t_\theta(\boldsymbol{\epsilon}) - \nabla_z \log q_\theta(\mathbf{z})\nabla_\theta t_\theta(\boldsymbol{\epsilon})\big] && (5.\\
&= \mathbb{E}_{p(\boldsymbol{\epsilon})}\big[\underbrace{\nabla_\theta t_\theta(\boldsymbol{\epsilon})}_{\nabla \text{path}} \underbrace{\nabla_z \big(\log p(\mathbf{x},\mathbf{z}) - \log q_\theta(\mathbf{z})\big)}_{\nabla \text{cost}}\big] && (5.
\end{aligned}$$

$(5.1 \to 5.2)$ Applying the reparameterization using $p(\boldsymbol{\epsilon})$ as base function and $\mathbf{z} = t(\boldsymbol{\epsilon})$ as transformation. It is possible to move the gradient inside the integral since $p(\boldsymbol{\epsilon})$ does not depend on $\boldsymbol{\theta}$. As part of the reparameterization, we need to replace all the instances of $\mathbf{z}$ with the alternative form $t_\theta(\boldsymbol{\epsilon})$.

$(5.2 \to 5.3)$ Applying the chain rule to the two composite functions. Decomposition of the first term is straightforward and just consists of multiplying the derivative of the three functions: the outermost $(\log)$, the intermediate (p), and the innermost (t). Formally we have

$$\nabla_\theta \log p(\mathbf{x}, t_\theta(\boldsymbol{\epsilon})) = \frac{1}{p(\mathbf{x},\mathbf{z})}\nabla_z p(\mathbf{x},\mathbf{z})\nabla_\theta t_\theta(\boldsymbol{\epsilon}) = \nabla_z \log p(\mathbf{x},\mathbf{z})\nabla_\theta t_\theta(\boldsymbol{\epsilon}),$$

note that I have used the log-derivative trick to get $\log p(\mathbf{x},\mathbf{z})$ from the ratio. The decomposition of the second term $\nabla_\theta \log q_\theta(t_\theta(\boldsymbol{\epsilon})$ is tricky since the function depends on $\boldsymbol{\theta}$ in two ways: (i) through the parameters of the variational distribution q_θ and (ii) through the parameters of the transformation t_θ. In this case we can still apply the chain rule, but we need to account for the total derivative, meaning that we end up with the sum of two terms, the first given by the chain rule (see above) and the second being the derivative of the function itself:

$$\nabla_\theta \log q_\theta(t_\theta(\boldsymbol{\epsilon})) = \nabla_z \log q_\theta(\mathbf{z})\nabla_\theta t_\theta(\boldsymbol{\epsilon}) + \nabla_\theta \log q_\theta(t_\theta(\boldsymbol{\epsilon}))$$

$(5.3 \rightarrow 5.4)$ Exploiting the linearity of the expectation to isolate a term.

$(5.4 \rightarrow 5.5)$ The term isolated in the previous step is the score and, as showed before, the expectation of the score is zero. The term can be safely removed. Well, to be honest removing this term is a delicate matter and can have an impact on the variance. In some condition the score term acts as a control variate: a term with zero expectation added to the estimator to reduce variance (see Roeder et al., 2017).

$(5.5 \rightarrow 5.6)$ Collecting the terms. The final form of the pathwise estimator of the ELBO in (5.6) is pretty similar to the generic form we have derived in (4.5). Note that, both the marginal distribution and the variational distribution need to be differentiable.

Comparing estimators

Following Mohamed et al. (2019) I will use *four properties* to quantify the quality of an estimator:

1. *Consistency.* Increasing N, the number of samples, the estimate should converge to the true expected value (see law of large numbers). Typically, our estimators are consistent, meaning we will not focus too much on this property.
2. *Unbiasedness.* Repeating the estimate multiple times we see that *on average* the estimate is centered around the true expected value. This property is preferred because it guarantees the convergence of stochastic optimisation procedures.
3. *Minimum variance.* Estimators are random variables since they depend on a random variable. Given the same number of samples N, we prefer the estimator with lower variance. Low variance means that the gradient estimates are more accurate, therefore we have a higher chances of converging to useful local minima and we can use larger step sizes (faster convergence).
4. *Efficiency.* We prefer estimators that needs a low number of samples and can be easily parallelized.

Both the score function estimator and the pathwise estimator are consistent (property 1) and unbiased (property 2), therefore here I will focus on comparing them in terms of their varirance (property 3) and efficiency (property 4).

Variance of the score function estimator. Let's give a look at the variance of the score function estimator

$$\mathbb{V}_{p_\theta}[\nabla_\theta \log p_\theta(\mathbf{x}) f(\mathbf{x})] = \mathbb{E}_{p_\theta}[\big(\nabla_\theta \log p_\theta(\mathbf{x}) f(\mathbf{x})\big)^2] - \mathbb{E}_{p_\theta}[\nabla_\theta \log p_\theta(\mathbf{x}) f(\mathbf{x})]^2,$$

where I have just applied the definition of variance: the variance of a random variable is equal to the mean of the square of the random variable

minus the square of the mean of the random variable. In general, when using the score function estimator we are not really concerned about the cost function $f(\mathbf{x})$, which can even be a black-box, but the variance depends on the cost function $f(\mathbf{x})$ which is a multiplicative term over the gradient. In other words, since the cost function tipically does not depend on the parameters $\boldsymbol{\theta}$ it will not directly affect the estimation of the gradient $\nabla_\theta \log p_\theta(\mathbf{x})$ but its multiplicative nature will affect the variance. For example, if $\mathbf{x} \in \mathbb{R}^D$ and the cost function is a sum of those D terms, $f(\mathbf{x}) = \sum f(\mathbf{x}_d)$, then the variance of the estimator will be of order $O(D^2)$. In this case the variance depends quadratically from the dimensionality of the input.

Variance of the pathwise estimator. The variance of the pathwise estimator has been consistently found to be lower than the variance of the score estimator. The lower variance may be attributed to the estimation the gradient of the log-joint distribution (cost function) which provides richer information about the direction of the maximum posterior mode. There are some interesting theoretical guarantees for the variance of this estimator. The variance is bounded by the squared Lipschitz constant of the cost function (see Mohamed et al., 2019). Importantly, the bounds are independent of the dimensionality of the parameter space and we can get low-variance gradient estimates in the high-dimensional setting. Differently from the score function case, here we are able to directly differentiate through the cost function itself. Therefore, additional terms play no role in the variance of the pathwise estimator, since only the path influencing the parameters is included in the gradient. The lower variance

Efficiency. Here, the term "efficiency" means computationally efficiency. The first thing to notice is that variance is directly connected to efficiency, since estimators with high variance require more samples to better approximate the gradient, as a consequence of the law of the large numbers. A recent analysis has been provided by Kucukelbir et al. (2017), who found that the score function estimator can require up to two orders of magnitude more samples to arrive at the same variance as a pathwise estimator. Intuitively, the variance increases with the number of terms to be estimated via Monte Carlo methods. This is why exact integration is desirable wherever possible, as it reduces the number of those terms.

Conclusion

In this post I have introduced two estimators of the gradient, the score function estimator and the pathwise estimator, showing how they can be applied to our particular problem: estimating the gradient of the ELBO. Which one is better? Difficult to say. The score function estimator is quite flexible and can be used when we do not have access to the cost function (e.g. black-box model) but it has high variance (this problem can be reduced using control variates and Rao-Blackwellization, see Ranganath et al., 2014).

On the other hand the pathwise estimator has low variance but it cannot be used when the cost function is not differentiable. As usual the right choice is application dependant and requires knowledge of the problem at hand.

Resources

- Shakir Mohamed's blog in particular [link-1] and [link-2]
- Shakir Mohamed's tutorials, e.g. [PDF-1] and [PDF-2]
- Marc Deisenroth's slides in particular [PDF]
- Yuge Shi's blog in particular [link]
- Andrew Miller's blog in particular [link]
- Yarin Gal's thesis [PDF]
- NeurIPS 2016 tutorial on variational inference [PDF]
- *"Pattern Recognition and Machine Learning"*, Chapter 10, C. Bishop

References

Kucukelbir, A., Tran, D., Ranganath, R., Gelman, A., & Blei, D. M. (2017). Automatic differentiation variational inference. The Journal of Machine Learning Research, 18(1), 430-474.

Mohamed, S., Rosca, M., Figurnov, M., & Mnih, A. (2019). Monte Carlo gradient estimation in machine learning. arXiv preprint:1906.10652 [arXiv]

Ranganath, R., Gerrish, S., & Blei, D. (2014). Black box variational inference. In Artificial intelligence and statistics (pp. 814-822). PMLR. [arXiv]

Roeder, G., Wu, Y., & Duvenaud, D. K. (2017). Sticking the landing: Simple, lower-variance gradient estimators for variational inference. In Advances in Neural Information Processing Systems (pp. 6925-6934). [arXiv]

```
> find_me_on(Github, Linkedin, Twitter, YouTube);

> return_copyright(2021, MassimilianoPatacchiola, AllRightsReserved);
```