Chapter 15

Variational Inference

Reading

1. Sections 1-2 of "Variational Inference: A Review for Statisticians" by Blei et al. (2017).
2. Sections 1-5 of "Auto-Encoding Variational Bayes" by Kingma and Welling (2013)
3. Chapter 2 of Durk Kingma's thesis: https://pure.uva.nl/ws/files/17891313/Thesis.pdf.
4. Bishop Chapter 11.5-11.6
5. Bishop Chapter 10-10.3
6. Lots of great intuition at http://ruishu.io/2018/03/14/vae/

We have been primarily concerned with models for classification and regression as yet in this course. The task there is to match the target (a class identity or a real-valued outcome). We now change tracks to consider generative modeling, these are models that are trained to synthesize new data. Effectively, the task here is not *match* a target datum, but given a training dataset of images/text, create a model that outputs similar images/text at test time. We will first take a look at variational methods and generative modeling using these methods in this chapter and do implicit generative models such as Generative Adversarial Networks in the next chapter.

15.1 The model

Imagine how you would draw the image of a dog x on paper. First, you would decide in your mind, its breed, its age, the color of its fur etc. Let

us call these quantities "latent factors". Latent factors can also include things that are not specific to the dog, e.g., the background of your painting (grass, house, beach etc.), the weather on that day (cloudy, sunny etc.), the viewpoint (zoomed in/far away). We will denote all such quantities by

$$z := \text{latent factors.}$$

Having decided upon all these factors, you realize your painting x. The painting x is not unique given latent factors z, e.g., two people can start off with the same latent factors and draw two totally different pictures.

We therefore model the generative process as a obtaining samples from a probability distribution

$$p(x \mid z).$$

Given a latent factor z and an image x, the quantity $p(x \mid z)$ denotes the likelihood of the sample. Given the painting image x, we do not know what the latent factors are. For instance, it is not easy to say whether the following image is that of a cat or a dog.

> In other words, the latent factors of data x are not known to us if we do not take part in the generative process. Nature is in charge of generating the data and our goal here is to guess the parameters of this generative model to be able to synthesize new samples that look as if Nature generated them.

There can be lots of latent factors z. So let us control this complexity and assume that we know a prior over the latent factors

$$\text{prior } p(z)$$

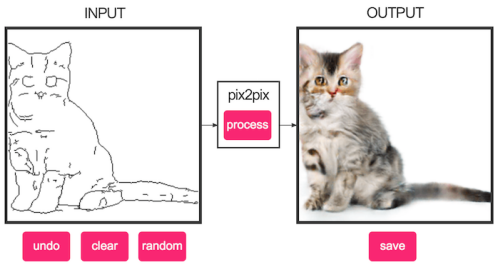
that models our belief of how likely a factor "dog with color blue" is in Nature.

Let us imagine Nature's generative model as running in two steps

1. First, sample a latent factor z from some distribution, and then
2. sample a datum $x \sim p(x \mid z)$.

The central point to appreciate is that we know neither Nature's distribution for sampling latents z nor its generative model $p(x \mid z)$. We will need to fit both these quantities using a training dataset of images/text.

The purpose of doing so can be many-fold, e.g., we may want to generate new data to amplify the size of our training set, given a part of the input image (say due to occlusions, or image corruption) we may want to complete the rest of it.



Most such applications require the knowledge of the latent factors that generated the data. Therefore, formally, we are interested in computing the posterior distribution of the latents and Nature's distribution of the latents

$$\text{posterior } p(z \mid x)$$
$$\text{prior } p(z)$$

using samples in a training dataset $D = \left\{x^i\right\}_{i=1}^n$. Notice that we do not need labels for this problem, effectively labels $y^i = x^i$ itself because our generative model should of course be very good at generating samples from the training data.

15.2 Some technical basics

15.2.1 Variational calculus

We will first take a brief look at what is called variational calculus.

A function is something takes in a variable as input and returns the value of the function as the output, e.g., $\mathbb{R} \ni f(x) = 5\ x^2$ for $x \in \mathbb{R}$. Similarly, a *functional* is an object that takes in a *function* as an input and returns a real number as the output. An example of this is entropy

$$\mathbb{R} \ni H[p] = -\int p(x) \log p(x)\ \mathrm{d}x$$

which takes in a probability density p as the input and returns a real number. Entropy is therefore a *functional*. Just like standard calculus where we take derivatives/minimize over functions, we can also take derivatives of the functional.

The functional derivative $\frac{\delta H[p]}{\delta p}(x)$ is defined in a funny way as

$$\int \frac{\delta H[p]}{\delta p}(x)\ \varphi(x)\ \mathrm{d}x = \lim_{\epsilon \to 0} \frac{H[p + \epsilon\varphi] - H[p]}{\epsilon}$$

for any arbitrary function φ. Essentially, you perturb the argument to the functional p by some epsilon and see how much the functional changes. The change in the functional is measured using the test function φ by integrating its changes $\frac{\delta H(p)}{\delta p}(x)$ at each point x in the domain. There may be certain conditions that the perturbation φ needs to satisfy depending upon the problem, e.g., since $p + \epsilon\varphi$ should also be legitimate probability density, the functional derivative above should only consider test functions φ such that

$$\forall \epsilon\ \int (p(x) + \epsilon\varphi(x))\,\mathrm{d}x = 1 \Rightarrow \int \varphi(x)\,\mathrm{d}x = 0.$$

The KL-divergence between two probability densities,

$$\mathrm{KL}(p\ ||\ q) = \int p(x) \log \frac{p(x)}{q(x)}\ \mathrm{d}x\,,$$

is another such functional; it has two arguments p and q.

Variational optimization is concerned with minimizing functionals.

For instance, while a problem looks like

$$w^* = \operatorname*{argmin}_{w \in \mathbb{R}^p} \ell(w)$$

in standard optimization, a variational optimization problem with KL-divergence as the loss given a fixed density p looks like

$$q^* = \operatorname*{argmin}_{q \in \mathcal{Q}} \mathrm{KL}(q\ ||\ p). \tag{15.1}$$

The variable of optimization is the probability density q and we will denote the domain of the variable by $\mathcal{Q}$. Since we want q to be a legitimate

probability density, we should choose

$$\mathcal{Q} \subseteq \mathcal{P}(\mathcal{X})$$

where $\mathcal{P}(\mathcal{X})$ denotes the set of all probability densities on some domain $\mathcal{X}$.

Picking the domain and objective in variational optimization Picking a good domain $\mathcal{Q}$ to minimize over is important. It is similar to the notion of the a hypothesis class in machine learning. If $\mathcal{Q}$ is too big, it is difficult to solve the optimization problem but we obtain a better value to $\mathrm{KL}(q||p)$. If $\mathcal{Q}$ is too small, the optimization problem may be easy but we may not match the desired distribution p very well. Imagine if p is a mixture of two Gaussians and we pick $\mathcal{Q}$ to be a family of uni-modal Gaussian distributions. Since the KL-divergence is zero if and only if the two distributions are equal, we are never going to be able to minimize it completely. On the other hand, if we pick $\mathcal{Q}$ to be the family of distributions with 2 or more Gaussian modes, then we can perfectly match p. Essentially, the crux of variational inference boils down to picking a good family of distributions $\mathcal{Q}$ that makes solving (15.1) easy.

What functional should we use to measure the distance between q and p? The KL-divergence is popular and easy to use in practice but there are many others. For example, when we studied the Gibbs distribution we briefly talked about something called "Wassserstein metric": if one imagines a mountain of dirt given by distribution q and another mountain of dirt p, the Wassserstein distance $W_2(q, p)$ is the amount of work done in transporting the dirt from q to p; it is also called the "earth mover's distance". The Wassserstein metric is as legitimate a distance between two distributions, just like the Kullback-Leibler divergence.

15.2.2 Laplace approximation

Laplace approximation is a very useful trick to solve variational optimization problems approximately. Here is how it works. Suppose we have to estimate an expectation of our random variable $\varphi(w)$

$$\mathop{\mathrm{E}}_{w \sim e^{-n\ell(w)}}[\varphi(w)] = \int e^{-nf(w)}\, \varphi(w)\, \mathrm{d}w$$

over draws $w \sim$ from some probability distribution $e^{-n\ell(w)}$ for some large value of n. The above integral takes many values, some have small $\ell(w)$ and some have large $\ell(w)$. The values of w where $\ell(w)$ is small are the ones that have the highest $e^{-n\ell(w)}$, especially as $n \to \infty$, and therefore the ones that count for most in the integral. The Laplace approximation is a trick to estimate the integral for large n. It replaces the integral by

taking a Taylor series expansion of the exponent as follows.

$$\int e^{-n\ell(w)}\ \varphi(w)\ \mathrm{d}w \approx \int \varphi(w)\ e^{-n\left(\ell(w^*)+\frac{1}{2}(w-w^*)^\top \nabla^2\ell(w^*)(w-w^*)\right)}\ \mathrm{d}w$$
$$= e^{-n\ell(w^*)} \int \varphi(w)\ e^{-\frac{n}{2}(w-w^*)^\top \nabla^2\ell(w^*)(w-w^*)}\ \mathrm{d}w \tag{15.2}$$

where $w^* = \operatorname{argmin} \ell(w)$ is the global minimum of $\ell(w)$. The integral is now with respect to a Gaussian distribution and can be done more easily.

How does a Laplace approximation look? Let us look at an example.

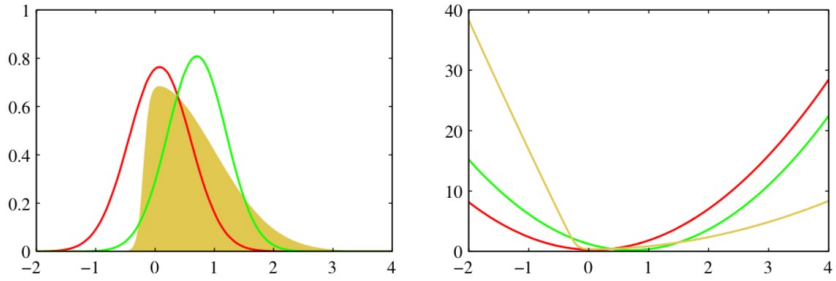


Figure 10.1 Illustration of the variational approximation for the example considered earlier in Figure 4.14. The left-hand plot shows the original distribution (yellow) along with the Laplace (red) and variational (green) approximations, and the right-hand plot shows the negative logarithms of the corresponding curves.

Although the Laplace approximation trick is reasonable only for very large values of n, it is a quick way to estimate what the correct domain of the a variational optimization problem should be. For example, if we are approximating a probability distribution with a Gaussian family, the Laplace approximation tells us what the mean of the family should be and we can only consider the variance as the variable in a variational optimization problem.

15.2.3 Digging deeper into KL-divergence

Let us take an example to understand KL-divergence better.

Figure 15.1 compares two forms of KL-divergence. The green contours represent equi-probability lines (1,2,3 standard deviations) for a two-dimensional correlated Gaussian $p(z_1, z_2)$. Red contours represent similar equi-probability lines for the variational approximation of this distribution using an uncorrelated Gaussian distribution

$$q(z) = q_1(z_1)q_2(z_2)$$

where both q_1, q_2 are one-dimensional Gaussians. The variational family $q \in \mathcal{Q}$ thus consists of factored uncorrelated Gaussians and we are trying to find the best member of this family that approximates the *correlated* true distribution $p(z)$.

Left panel (a) in Figure 15.1 shows the result using the forward KL-divergence minimization

$$q^* = \operatorname*{argmin}_{q\in\mathcal{Q}} \mathrm{KL}(q \,||\, p).$$

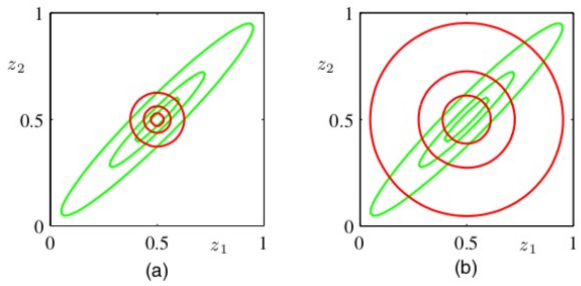


Figure 15.1: Comparison between the variational approximation of a correlated Gaussian using forward and reverse KL divergence and a factored Gaussian family.

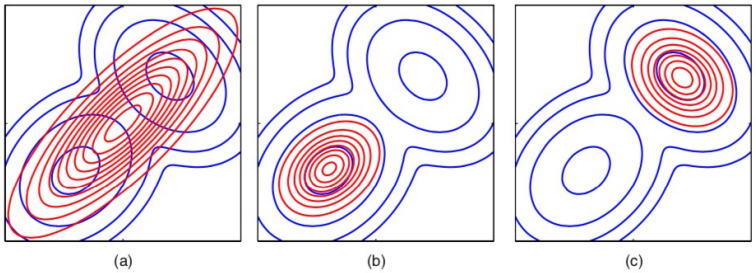


Figure 15.2: Approximating a multi-modal distribution using a uni-modal variational family.

while the right panel (b) shows the result for the reverse KL-divergence minimization

$$q^* = \underset{q \in \mathcal{Q}}{\text{argmin}}\, \text{KL}(p \,||\, q).$$

We see that both these forms capture the mean of the true distribution $p(z)$ correctly. The variance of the two approximations is quite different depending upon which form we employ.

We next consider the case when a multi-modal probability distribution $p(z)$ is approximated using a unimodal Gaussian distribution. Both these examples are very often seen in practice, the distribution of true data/latent factors is often correlated and multi-modal. We have seen one instance of this: the distribution of weights of a deep network in the Gibbs distribution is multi-modal because of multiple global minima.

The distribution p is bi-modal and the variational problem is no longer convex in this case; depending upon the initial condition using q, one may get different solutions shown in panels (a), (b) or (c). You should also think about the fact that the solution in panel (a) could be the solution of optimizing the reverse KL divergence; in contrast, the solutions in panels (b) and (c) have to be the ones obtained from optimizing the forward KL divergence.

KL-divergence is not the only distance used in variational inference and there are many many other ones. You should think of these different

❓ Use the expression of the KL-divergence to convince yourself why the forward KL under-estimates the variance while the reverse KL over-estimates the variance in Figure 15.1.

ways to measure distances between probability distributions in variational inference as different surrogate losses; which one we use is highly problem dependent although the forward KL-divergence $\mathrm{KL}(q \mid\mid p)$ is the most common.

15.3 Evidence Lower Bound (ELBO)

We now go back to the generative model.

> We will formalize our goal in generative modeling as computing Nature's posterior distribution of latent factors
>
> $$p(z \mid x).$$
>
> We have access to a training dataset $D = \left\{(x^i)\right\}_{i=1}^n$. We do not know (i) what form Nature's posterior distribution takes, e.g., Gaussian, multi-modal distribution etc. and (ii) we do not know the true latent factors z that Nature uses. So we are going to approximate the true posterior using some variational family of our choice
>
> $$\mathcal{Q} \ni q^*(z \mid x) \approx p(z \mid x).$$
>
> This is the basic idea of variational inference: to approximate a complex distribution $p(z \mid x)$ using a member of from a simpler family of our choosing $\mathcal{Q}$. In practice, this variational family $\mathcal{Q}$ will be parameterized by a deep network.

With this background, the mathematical process of executing the above program is quite simple. We will simply minimize the KL-divergence

$$q^*(z \mid x) = \underset{q \in \mathcal{Q}}{\operatorname{argmin}} \; \frac{1}{n} \sum_{i=1}^{n} \mathrm{KL}\left(q(z \mid x^i) \mid\mid p(z \mid x^i)\right). \tag{15.3}$$

We next rewrite this KL-divergence above in a special form.

$$\begin{aligned}
0 &\leq \mathrm{KL}\left(q(z \mid x^i) \mid\mid p(z \mid x^i)\right) \\
&= \mathop{\mathrm{E}}_{z \sim q(z|x^i)} \left[\log \frac{q(z \mid x^i)}{p(z \mid x^i)}\right] \\
&= - \mathop{\mathrm{E}}_{z \sim q(z|x^i)} \left[\log p(z \mid x^i)\right] + \mathop{\mathrm{E}}_{z \sim q(z|x^i)} \left[\log q(z \mid x^i)\right] \\
&= - \mathop{\mathrm{E}}_{z \sim q(z|x^i)} \left[\log p(z, x^i) - \log p(x^i)\right] + \mathop{\mathrm{E}}_{z \sim q(z|x^i)} \left[\log q(z \mid x^i)\right] \\
&= \log p(x^i) - \mathop{\mathrm{E}}_{z \sim q(z|x^i)} \left[\log p(z, x^i)\right] + \mathop{\mathrm{E}}_{z \sim q(z|x^i)} \left[\log q(z \mid x^i)\right]. \\
\Rightarrow \log p(x^i) &\geq \mathop{\mathrm{E}}_{z \sim q(z|x^i)} \left[\log p(z, x^i)\right] - \mathop{\mathrm{E}}_{z \sim q(z|x^i)} \left[\log q(z \mid x^i)\right]
\end{aligned}$$

This is quite interesting. The left-hand side of this inequality is the log-likelihood of the data under Nature's distribution, i.e., it is fixed and

independent of what we do. The left-hand side is also called the "evidence" in statistics (which is a bit ironic because we can never know the evidence). The right-hand side

$$\text{ELBO}(q, x^i) := \mathop{\text{E}}_{z\sim q(z|x^i)}\left[\log p(z, x^i)\right] - \mathop{\text{E}}_{z\sim q(z|x^i)}\left[\log q(z \mid x^i)\right]. \tag{15.4}$$

is a lower bound on the evidence and therefore called the Evidence Lower Bound (ELBO).

> Next comes a key step: a good generative model should be such that the evidence of the training data, i.e., the log-likelihood of this data under Nature's distribution, should be large under the model. We therefore want to maximize the ELBO on our training data
>
> $$q^*(z \mid x) = \mathop{\text{argmax}}_{q\in\mathcal{Q}} \frac{1}{n}\sum_{i=1}^{n} \text{ELBO}(q, x^i). \tag{15.5}$$
>
> to find the posterior distribution of the latent factors $q^*(z)$. Maximizing ELBO is equivalent to minimizing the average KL-divergence $\text{KL}(q(z \mid x^i) \,||\, p(z \mid x^i)$ over all training samples.

We will again solve the optimization problem in (15.5) using stochastic gradient descent. Before we study how to do that, let us consider what model we have developed so far. The solution to this problem

$$q^*(z \mid x) \approx p(z \mid x)$$

approximates Nature's posterior distribution. If we maximize ELBO well, given an input x, samples $z \sim q^*(z \mid x)$ are likely to be the latent factors that Nature could have chosen while rendering this image. But we still do not know how to synthesize an image x for these latent factors. We now rewrite ELBO in a different form to understand this.

$$\begin{aligned}
\text{ELBO}(q, x^i) &= \mathop{\text{E}}_{z\sim q(z|x^i)}\left[\log p(z, x^i)\right] - \mathop{\text{E}}_{z\sim q(z|x^i)}\left[\log q(z \mid x^i)\right] \\
&= \mathop{\text{E}}_{z\sim q(z|x^i)}\left[\log p(x^i \mid z) + \log p(z)\right] - \mathop{\text{E}}_{z\sim q(z|x^i)}\left[\log q(z \mid x^i)\right] \\
&= \mathop{\text{E}}_{z\sim q(z|x^i)}\left[\log p(x^i \mid z)\right] - \text{KL}(q(z \mid x^i) \,||\, p(z)).
\end{aligned}$$

This form of ELBO

$$\text{ELBO}(q, x^i) = \mathop{\text{E}}_{z\sim q(z|x^i)}\left[\log p(x^i \mid z)\right] - \text{KL}(q(z \mid x^i) \,||\, p(z)) \tag{15.6}$$

is very interesting. The first term is Nature's log-likelihood of datum x^i given the latent factor z sampled from *our* candidate posterior q. The second term is the discrepancy between our variational approximation of the posterior $q^*(z \mid x^i) \approx p(z \mid x^i)$ and Nature's true marginal distribution over latent factors $p(z)$. This alternative form of ELBO is

conceptually very similar to what we do in standard classification, e.g.,

$$\operatorname*{argmin}_{w} \left\{ \ell(w) + \frac{\alpha}{2}\|w\|^2 \right\}.$$

We would like our $q(z \mid x^i)$ to be close to Nature's prior distribution $p(z)$ but at the same time be such that samples from $q(z \mid x^i)$ have a high log-likelihood $p(x^i \mid z)$ of synthesizing images in the training set. The KL-term is therefore a regularizer for the first data-fitting term.

15.3.1 Parameterizing ELBO

What variational family $\mathcal{Q}$ should we choose? Say we parametrized each distribution $q(z \mid x^i)$ by its mean and diagonal of the covariance.

$$\mathbb{R}^m \ni z \sim q(z \mid x^i) = N(\mu(x^i), \sigma^2(x^i)I) \in \mathcal{Q}(x^i)$$

where $\mu(x^i), \sigma^2(x^i) \in \mathbb{R}^m$. The ELBO in (15.6) is totally independent for each x^i in the training dataset, so all $i \in \{1, \ldots, n\}$ we can solve for

$$\mu^*(x^i), \sigma^2(x^i) = \operatorname*{argmax}_{\mu,\sigma^2} \text{ ELBO}\left(N(\mu(x^i), \sigma^2(x^i)I), x^i\right).$$

But this is not a good idea: the parameters μ, σ^2 are distinct for each input x^i and effectively they are being trained using a dataset of only input image x^i.

> Amortized variational inference is a clever trick that ties together the variational families $\mathcal{Q}(x^i)$. We will be using a deep network with parameters $u \in \mathbb{R}^p$ that takes x^i as the input and gives $\mu(x^i; u), \sigma^2(x^i; u)$ as the outputs
>
> $$\text{Encoder} : x^i \underbrace{\mapsto}_{\text{parameters } u} \mu(x^i; u), \sigma^2(x^i; u).$$
>
> The variational family $\mathcal{Q}(x^i)$ that we are considering is therefore the set of distributions expressed by this deep network with p parameters. The family $\mathcal{Q}(x^i)$ is still distinct for each datum x^i but they are are all tied together by the same weights u.
>
> **Encoder.** We will call this deep network the encoder because it takes in an input x^i and encodes it into $\mu(x^i; u), \sigma^2(x^i; u)$ which parameterize the distribution of the latent factors.

Decoder. Observe that although we have now parameterized the distribution $q(z \mid x^i)$ using a deep network with weights u, we still do not know how to model the term $p(x^i \mid z)$. After all, this is Nature's log-likelihood.

We have a dataset $\left\{(x^i, z^i)\right\}_{i=1}^n$ that consists of the images x^i and their corresponding latents z^i sampled from our encoder. We are going to

model Nature's rendering process $p(x \mid z)$ using a deep network. This is a program that we have done many times in the past, e.g., we model the targets in classification y^i as samples from the softmax distribution with images x^i as the input and train the weights using maximum-likelihood (as you may recall, this is equivalent to the cross-entropy loss).

We can repeat that program here: we are going to learn a deep network

$$\text{Decoder} : p_v(x^i \mid z) \approx p(x^i \mid z).$$

with parameters $v \in \mathbb{R}^p$ that models Nature's likelihood $p(x^i \mid z)$.

Different possible decoders for MNIST Depending upon the type of data x^i, we will code up the deep network in different ways. For instance, if each pixel of $x^i \in \mathbb{R}^{28\times 28}$ is grayscale $[0, 255]$ like it is in MNIST, the output of the decoder is a multinomial with size $28 \times 28 \times 256$.

If we take the training dataset as binarized MNIST (if pixel jk is less than 128 set it to 0, else set it to 1), then the output of the decoder has size $28 \times 28 \times 2$ and we can fit this using a logistic distribution at each pixel

$$p_v(x^i \mid z) = \prod_{j,k=1}^{28} \underbrace{p_v(x^i_{jk} \mid z)}_{\text{logistic distribution for pixel } x^i_{jk} \in \{0,1\}}$$

The log-likelihood term in (15.6) will then correspond to the logistic loss as discussed in the Homework.

⚠ The distribution of labels y^i in classification was one-hot vectors, so the softmax layer created a multinomial distribution on the classes.

Using a mean-field prior $p(z)$. We do not know what the prior distribution $p(z)$ in (15.6) is. We will choose a simple prior

$$p(z) = \prod_{j=1}^{m} p_j(z_j) \tag{15.7}$$

where $p_i(z_i)$ is the distribution of the i^{th} latent factor z_i. Such distributions are called mean-field priors (where the distribution of a vector $z \in \mathbb{R}^m$ is modeled as independent distributions on its components). We will further choose each distribution

$$p_j(z_j) = N(0, 1)$$

to be a zero-mean standard Gaussian distribution. This is a Gaussian mean-field prior. Just like the choice of a regularizer is critical in machine learning for obtaining good generalization, the chose of a prior is critical in variational inference for synthesizing good images from the generative model.

15.4 Gradient of the ELBO

We now have all the ingredients in place for training a variational generative model. Let us summarize our setup.

1. Encoder parameters u are weights of a deep network that takes in x^i as input and outputs parameters $\mu(x^i), \sigma^2(x^i)$ of the latent distribution. We have tacitly assumed the latent posterior $p(z \mid x^i)$ to be a Gaussian here; if you have a problem where you wish to have a different latent, e.g., all the latent genes that could have caused a particular cancer, then you want to output the parameters of that distribution from the encoder.

2. The decoder models the likelihood $p_v(x^i \mid z)$ using parameters v.

3. The prior $p(z)$ will be a mean-field Gaussian distribution. The prior has no parameters in our case, although you may see research papers where the prior also has its own parameters. A popular choice is to use

$$\text{ELBO}_\beta(q, x^i) = \mathop{\text{E}}_{z\sim q(z|x^i)} \left[\log p(x^i \mid z)\right] - \beta^{-1}\,\text{KL}(q(z \mid x^i)\,||\,p(z))$$

in place of the standard ELBO. The hyper-parameter $\beta > 0$ gives more control over the strength of the prior; this is of course akin to picking the weight-decay coefficient.

⚠ The concept of variational inference and ELBO are much more general than generative models or the encoder-decoder structure that we have developed. Go through the assigned reading material to learn more.

The ELBO when rewritten in terms of the encoder and decoder parameters looks as follows.

$$\text{ELBO}(u, v; x^i) = \mathop{\text{E}}_{z\sim q_u(z|x^i)} \left[\log p_v(x^i \mid z)\right] - \text{KL}(q_u(z \mid x^i)\,||\,p(z)). \tag{15.8}$$

Our goal is to fit the weights u, v using

$$u^*, v^* = \mathop{\text{argmax}}_{u,v\in\mathbb{R}^p} \frac{1}{n}\sum_{i=1}^{n} \text{ELBO}(u, v; x^i). \tag{15.9}$$

The number of parameters of the encoder and decoder can be different but for clarity we imagine them to be the same.

(15.9) is an optimization problem and in this section, we will see how to compute the gradient of the objective so that we can solve the problem using SGD.

15.4.1 The Reparameterization Trick

Focus on the gradient with respect to u of the first term of ELBO

$$\nabla_u \mathop{\text{E}}_{z\sim q(z|x^i)} \left[\varphi(z)\right].$$

We have written $\log p_v(x^i \mid z) = \varphi(z)$ to keep the notation clear; we do not care about the exact form of the integrand in this section.

If we draw a computational graph for the forward propagation of this term, it looks as follows

$$u, x^i \rightarrow \text{sample } z \text{ from } q_u(z \mid x^i) \rightarrow \varphi(z).$$

The intermediate sampling step is troublesome, we do not really know how to use the chain rule of calculus across sampling, i.e., given

$$\overline{\varphi(z)} := \frac{\mathrm{d}}{\mathrm{d}_u}\varphi(z)$$

we need to compute $\overline{u} = \mathrm{d}\ell/\mathrm{d}_u\, u$. We only know how to apply the chain rule for *deterministic operations* of the form

$$u, x^i \rightarrow z = \text{some deterministic function } g(u, x^i) \rightarrow \varphi(z),$$

in which case we use the standard backprop across the function g.

> The Reparameterization Trick enables us to obtain backpropagation gradients across sampling operations via a creative use of the Laplace approximation of the distribution $q_u(z \mid x^i)$.

We known from the Laplace approximation that we can compute an expectation over z using a Gaussian centered at the global maximum of the distribution $q_u(z \mid x^i)$ with variance equal to the inverse Hessian at that maximum. Motivated by this, the Reparameterization Trick *rewrites* the random variable z as

$$z = \mu(x^i; u) + \sigma(x^i; u) \odot \epsilon$$

where

$$\epsilon \sim N(0, I_{m\times m})$$

is a sample from a standard multi-variate Gaussian distribution and the notation $\odot$ denotes element-wise product. Effectively, we imagine that the encoder outputs

$$\begin{aligned}
\mu(x^i; u) &= \operatorname*{argmax}_z q_u(z \mid x^i) \\
\sigma^2(x^i; u) &= \operatorname{diag}\left(\left[\nabla_z^2 q_u(z \mid x^i)\right]^{-1}\right).
\end{aligned}$$

Just like the integral in (15.2) was performed over the Gaussian, the integral over z can be rewritten as an integral over ϵ

$$\begin{aligned}
\nabla_u \mathop{\mathrm{E}}_{z\sim q_u(z|x^i)}[\varphi(z)] &= \nabla_u \mathop{\mathrm{E}}_{\epsilon\sim N(0,I)}\left[\varphi\left(\mu(x^i; u) + \sigma(x^i; u) \odot \epsilon\right)\right] \\
&= \mathop{\mathrm{E}}_{\epsilon\sim N(0,I)}\left[\nabla_u\varphi\left(\mu(x^i; u) + \sigma(x^i; u) \odot \epsilon\right)\right] \\
&\approx \frac{1}{N}\sum_{j=1}^{N} \nabla_u\varphi\left(\mu(x^i; u) + \sigma(x^i; u) \odot \epsilon^j\right), \text{ where } \epsilon^j \sim N(0, I).
\end{aligned}$$

We can take the gradient operator inside the expectation in this case because ϵ no longer depends on the weights u. The term $\nabla_u \varphi\left(\mu(x^i; u) + \sigma(x^i; u) \odot \epsilon^j\right)$ is a deterministic operation given a sample z^j and can be computed using standard backpropagation.

15.4.2 Score-function estimator of the gradient

Let us look at an alternative way to compute the same gradient.

$$\begin{aligned}
\nabla_u \mathop{\mathrm{E}}_{z\sim q_u(z|x^i)}[\varphi(z)] &= \nabla_u \int \varphi(z)\; q_u(z \mid x^i)\; \mathrm{d}z \\
&= \int \varphi(z)\; \nabla_u q_u(z \mid x^i)\; \mathrm{d}z \\
&= \int \varphi(z)\; \frac{\nabla_u q_u(z \mid x^i)}{q_u(z \mid x^i)}\; q_u(z \mid x^i)\; \mathrm{d}z \\
&= \int \varphi(z)\; \nabla_u \log q_u(z \mid x^i)\; q_u(z \mid x^i)\; \mathrm{d}z \\
&= \mathop{\mathrm{E}}_{z\sim q_u(z|x^i)}\left[\varphi(z) \nabla_u \log q_u(z \mid x^i)\right] \\
&\approx \frac{1}{N}\sum_{j=1}^{N} \varphi(z^j) \nabla_u \log q_u(z^j|x^i)\;, \text{with } z^j \sim q_u(z \mid x^i).
\end{aligned} \tag{15.10}$$

The term

$$\frac{\nabla_u q_u(z \mid x^i)}{q_u(z \mid x^i)} = \nabla_u \log q_u(z \mid x^i) \tag{15.11}$$

is called the score function of a probability distribution q_u. The above calculation is quite beautiful: calculating the gradient of the expectation of any quantity $\varphi(z)$ is equal to the expectation of the same quantity weighted by the score function

$$\nabla_u \mathop{\mathrm{E}}_{z\sim q_u}[\varphi(z)] = \mathop{\mathrm{E}}_{z\sim q_u}\left[\varphi(z) \nabla_u \log q_u\right].$$

Due to this trick, we can compute the gradient using N samples

$$z^j \sim p_u(z \mid x^i) \tag{15.12}$$

from the encoder; this is easy if, say, the encoder outputs the mean and standard-deviation of the distribution of the latents. Given z^j, the gradient

$$\nabla_u \log q_u(z^j \mid x^i)$$

is just the standard back-propagation gradient of the quantity $\log q_u(z^j \mid x^i)$ with respect to weights u of the deep network and can be computed using autograd.

The key difference between the Reparameterization Trick and

the score-function estimator is that in the latter, we do not need to make sure that the gradient $\mathrm{d}\ell/\mathrm{d}z^j$ can be back-propagated across the sampling operation. The score-function estimator directly computes the gradient of the entire expectation by a weighted average across the samples.

Having two different ways of computing the same gradient may seem redundant but they both are suited to very different applications. The Reparameterization Trick is not accurate in cases when the distribution $q_u(z \mid x^i)$ is multi-modal because we have only one mean $\mu(x^i)$ around which the samples are drawn. The score-function trick does not have this problem because so long as iid samples are drawn in (15.12) (using any method, e.g., importance sampling) we obtain true estimate of the gradient. The problem in score-function estimator lies in that the denominator $q_u(z \mid x^i)$ in (15.11) can take very small values if the particular sample z is unlikely. The summation (15.10) is a combination of many N, some very large in magnitude and some very small; the variance of score-function estimate of the gradient in (15.10) can therefore be quite large in most problems.

Typically, the Reparameterization Trick is commonly used in generative models while both the Reparameterization Trick and the score-function estimator are used widely in Reinforcement Learning.

15.4.3 Gradient of the remaining terms in ELBO

The gradient with respect to weights v of the decoder of the first term in ELBO

$$\nabla_v \mathop{\mathrm{E}}_{z\sim q_u(z|x^i)} \left[\log p_v(x^i \mid z)\right]$$

is simply the standard backpropagation gradient (the sampling distribution of the encoder does not depend on the weights of the decoder).

Let us focus on the second term

$$\mathrm{KL}\left(q_u(z \mid x^i) \mid\mid \prod_{j=1}^{m} p_j(z_j)\right). \tag{15.13}$$

where $p_j(z_j) = N(0,1)$ are terms of the mean-field prior. The gradient of this term with respect to weights of the decoder is zero

$$\nabla_v \mathrm{KL}\left(q_u(z \mid x^i) \mid\mid \prod_{j=1}^{m} p_j(z_j)\right) = 0.$$

Following the reasoning in the Reparameterization Trick, we are positing that $q_u(z \mid x^i)$ is a Gaussian distribution:

$$q_u(z \mid x^i) = N\left(\mu(x^i; u), \sigma^2(x^i; u)I\right).$$

Notice that $\sigma^2(x^i; u) \in \mathbb{R}^m$ is the diagonal of the covariance and therefore the individual marginals $q_u(z_j \mid x^i)$ and $q_u(z_{j'} \mid x^i)$ for two indices j, j' are independent. We can therefore write

$$q_u(z \mid x^i) = \prod_{j=1}^{m} N(\mu_j(x^i; u), \sigma_j^2(x^i; u)). \tag{15.14}$$

The KL-divergence of a univariate Gaussian $N(\mu_1, \sigma_1^2)$ with respect to the standard Gaussian is

$$\mathrm{KL}\left(N(\mu, \sigma^2) \,||\, N(0,1)\right) = \log \frac{1}{\sigma} + \frac{\sigma^2 + \mu^2}{2} - \frac{1}{2}. \tag{15.15}$$

The general formula is

$$\mathrm{KL}\left(N(\mu_1, \sigma_1^2) \,||\, N(\mu_2, \sigma_2^2)\right) = \log \frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2}.$$

Due to (15.14), the KL-divergence in (15.13) is a sum of the KL-divergences of the individual univariate Gaussians

$$\mathrm{KL}(q_u(z \mid x^i) \,||p(z)) = -\frac{1}{2} \sum_{j=1}^{m} \left(\log \sigma_j^2(x^i; u) - \sigma_j^2(x^i; u) - \mu_j^2(x^i; u) + 1\right). \tag{15.16}$$

The right-hand side of this equation is only a function of u and its gradient can be calculated using standard back-propagation.

This completes our development of ELBO. Using the gradient calculated in this section, we can use SGD to maximize the objective in (15.5) and train a generative model.

Prove that

$$\mathrm{KL}\left(\prod_{j=1}^{m} q_j(z_j) \,||\, \prod_{j=1}^{m} p_j(z_j)\right) = \sum_{j=1}^{m} \mathrm{KL}(q_j(z_j) \,||p_j(z_j)).$$

15.5 Some comments

Although the mathematics of ELBO seems complicated, it is quite easy to implement generative models using variational inference in practice. You did for a simple MNIST problem in the homework/recitation but if the encoder and decoder are convolutional and deconvolutional architectures respectively, we can get very sophisticated generative models.

Figure 15.3: Samples from a state-of-the-art VAE trained on ImageNet (Razavi et al., 2019)

Variational inference and information-theoretic methods are a rich (and old) area of research and there are many modifications/innovations to ELBO, e.g., read Alemi et al. (2018) for some simple yet deep modifications.

Bibliography

Alemi, A., Poole, B., Fischer, I., Dillon, J., Saurous, R. A., and Murphy, K. (2018). Fixing a broken elbo. In *International Conference on Machine Learning*, pages 159–168. PMLR.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Razavi, A., van den Oord, A., and Vinyals, O. (2019). Generating diverse high-fidelity images with vq-vae-2. In *Advances in Neural Information Processing Systems*, pages 14866–14876.