7

Discussion: The Variational Autoencoder

From the start of this tutorial, we have encouraged the reader to consider our three themes of representation, inference, and learning as interconnected endeavors. Now, we center our concluding discussion on an illustrative example of how these interconnections can play out to yield exceptionally powerful methods.

In this discussion, we present a highly influential deep probabilistic model: the *variational autoencoder* (VAE; Kingma and Welling 2014; Kingma and Welling 2019). The VAE is a neural architecture for latent representation learning. VAEs have been famously used for image generation (Gregor *et al.*, 2015), language modeling (Bowman *et al.*, 2016), molecular design (Kusner *et al.*, 2017), and semi-supervised learning tasks (Maaløe *et al.*, 2016). As a testament to its impact, the VAE was recognized with the Test of Time Award at the 2024 International Conference on Learning Representations (ICLR, 2024; Kingma and Welling, 2024). Using the VAE as a case study, we will draw connections among ideas from throughout this tutorial and demonstrate how these ideas are useful in machine learning research.

7.1 Deep Generative Latent Variable Models

Consider a directed LVM of the form

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}|\mathbf{z})p(\mathbf{z})$$

with observed variables $\mathbf{x} \in \mathcal{X}$ (where $\mathcal{X}$ can be continuous or discrete) and latent variables $\mathbf{z} \in \mathbb{R}^k$.

To make things concrete, you can think of $\mathbf{x}$ as an image (e.g., a human face), and $\mathbf{z}$ as latent factors that explain the features of the image. For example, in the case of a face, one coordinate of $\mathbf{z}$ could encode whether the face is happy or sad, while another one encodes whether the face is male or female, etc.

We may also be interested in models with many layers of latent variables, e.g.,

$$p(\mathbf{x} \mid \mathbf{z}_1)p(\mathbf{z}_1 \mid \mathbf{z}_2)p(\mathbf{z}_2 \mid \mathbf{z}_3) \cdots p(\mathbf{z}_{m-1} \mid \mathbf{z}_m)p(\mathbf{z}_m).$$

These models are often referred to as *deep latent variable models* and are capable of learning hierarchical latent representations. In this section, we will assume for simplicity that there is only one latent layer.

7.1.1 Learning in Deep Generative Latent Variable Models

Suppose that we are given a dataset $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ and an LVM $p(\mathbf{x}, \mathbf{z}; \theta)$ with observed variables $\mathbf{x}$, latent variables $\mathbf{z}$, and parameters θ. We are interested in the following inference and learning tasks:

1. *Learning the parameters θ of p.*
2. *Approximate posterior inference over* $\mathbf{z}$. Given an image $\mathbf{x}_i$, what are its associated latent factors?
3. *Approximate marginal inference over* $\mathbf{x}$. Given an image $\mathbf{x}_i$ with missing parts, how do we fill-in these parts?

We will also make the following assumptions:

1. *Intractability.* Computing the posterior probability $p(\mathbf{z} \mid \mathbf{x})$ is intractable.

2. *Big data*. The dataset $\mathcal{D}$ is too large to fit in memory, and so we can only work with small, subsampled batches of $\mathcal{D}$.

Many interesting models fall in this class, including the VAE.

7.1.2 The Standard Approaches

We have learned several techniques that could be used to solve our three tasks. Let's try them out.

As we have discussed, EM (Algorithm 6) can be used to learn LVMs. Recall, however, that performing the E-step requires computing the approximate posterior $p(\mathbf{z} \mid \mathbf{x})$, which we have assumed to be intractable. In the M-step, we learn parameters θ by looking at the entire dataset, which is going to be too large to hold in memory.

To perform approximate inference, we can use mean field variational inference (Section 5.2). Recall, however, that one step of mean field requires us to compute an expectation whose time complexity scales exponentially with the size of the Markov blanket of the target variable (Definition 3.7, 3.8).

What is the size of the Markov blanket for $\mathbf{z}$? If we assume that at least one component of $\mathbf{x}$ depends on each component of $\mathbf{z}$, then this introduces a v-structure (Table 3.1) into the graph of our model (the $\mathbf{x}$, which are observed, are explaining away the differences among the $\mathbf{z}$). Thus, we know that all the $\mathbf{z}$ variables will depend on each other and the Markov blanket of some $\mathbf{z}_i$ will contain all the other $\mathbf{z}$ variables. This will make mean field intractable. An equivalent (and simpler) explanation is that p will contain a factor $p(\mathbf{x}_i \mid \mathbf{z}_1, \ldots, \mathbf{z}_k)$, in which all the $\mathbf{z}$ variables are tied.

Another approach would be to use sampling-based methods (Section 5.1). In the seminal paper that first describes the VAE (Kingma and Welling, 2014), the authors compare the VAE against these kinds of algorithms. However, they find that these sampling methods do not scale well to large datasets. In addition, techniques such as Metropolis-Hastings require a hand-crafted proposal distribution, which might be difficult to choose.

7.2 Auto-Encoding Variational Bayes

We are now going to learn about *auto-encoding variational Bayes* (AEVB), an algorithm that can efficiently solve our three inference and learning tasks. The VAE is one instantiation of this algorithm.

In this section, following typical notation in the VAE literature, we will use $p_\theta(\mathbf{x}, \mathbf{z})$ to denote a probabilistic model of observed variables $\mathbf{x}$ and latent variables $\mathbf{z}$ with parameter θ (and similarly for variational distributions q_ϕ with parameter ϕ).

AEVB is based on ideas from variational inference (Section 5.2). Recall that in variational inference, we are interested in maximizing the ELBO, written here as

$$\mathcal{L}(p_\theta, q_\phi) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z} \mid \mathbf{x})\right],$$

over a family of distributions q_ϕ. The ELBO satisfies the equation

$$\log p_\theta(\mathbf{x}) = \mathrm{KL}(q_\phi(\mathbf{z} \mid \mathbf{x}) \,||\, p(\mathbf{z} \mid \mathbf{x})) + \mathcal{L}(p_\theta, q_\phi).$$

Note that the variational approximation $q_\phi(\mathbf{z} \mid \mathbf{x})$ is conditioned on $\mathbf{x}$. This means that we are effectively choosing a different density $q(\mathbf{z})$ for every $\mathbf{x}$, which will be used to produce an improved posterior approximation, rather than always choosing a single static $q(\mathbf{z})$.

How exactly do we compute and optimize over $q_\phi(\mathbf{z} \mid \mathbf{x})$? As noted above, mean field could be applied here, but it tends to be both computationally challenging and inaccurate, since the fully factorized assumption on q is too restrictive for our purposes.

7.2.1 Black-Box Variational Inference

The first important idea used in the AEVB algorithm is a general purpose approach for optimizing q_ϕ that works for a rich class of variational densities. Later, we will show special cases of this algorithm given specific choices of q_ϕ.

This approach — called *black-box variational inference* — consists of maximizing the ELBO using gradient descent over ϕ (instead of using, for example, a coordinate descent algorithm like in mean field). Hence, this procedure only assumes that q_ϕ is differentiable in its parameters ϕ.

The term *black-box variational inference* was first coined by Ranganath *et al.* (2014), while the core ideas were also inspired by earlier work, such as the Wake-Sleep algorithm (Hinton *et al.*, 1995).

Additionally, rather than treating inference and learning as separate steps, the AEVB algorithm performs both inference and learning simultaneously by jointly optimizing ϕ and θ via gradient descent. Optimization over ϕ will keep the ELBO tight around $\log p_\theta(\mathbf{x})$. Optimization over θ will push the lower bound (and hence $\log p_\theta(\mathbf{x})$) up. This is somewhat similar to how the EM algorithm optimizes a lower bound on the marginal likelihood (Section 6.4.1).

7.2.2 The Score Function Gradient Estimator

To perform black-box variational inference, we need to compute the gradient

$$\nabla_{\theta,\phi} \mathbb{E}_{q_\phi(\mathbf{z})} \left[\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z})\right],$$

where we use $q_\phi(\mathbf{z})$ to denote an arbitrary variational distribution. Taking the expectation with respect to q_ϕ in closed form will often not be possible. Instead, we may take Monte Carlo estimates of the gradient by sampling from q_ϕ. This is easy to do for the gradient with respect to θ. We can swap the gradient and the expectation and estimate the following expression via Monte Carlo:

$$\mathbb{E}_{q_\phi(\mathbf{z})} \left[\nabla_\theta \log p_\theta(\mathbf{x}, \mathbf{z})\right].$$

However, taking the gradient with respect to ϕ is more difficult. Since the expectation is taken with respect to q_ϕ, which itself depends on ϕ, we cannot simply move the gradient inside the expectation.

One way to estimate this gradient is via the so-called *score function estimator*:

$$\begin{aligned}
&\nabla_\phi \mathbb{E}_{q_\phi(\mathbf{z})} \left[\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z})\right] \\
&= \mathbb{E}_{q_\phi(\mathbf{z})} \left[\left(\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z})\right) \nabla_\phi \log q_\phi(\mathbf{z})\right].
\end{aligned}$$

This follows from some basic algebra and calculus and takes about half a page to derive. We leave it as an exercise to the reader, but for those interested, the full derivation may be found in Appendix B of Mnih and Gregor (2014).

The above identity places the gradient inside the expectation, which we may now evaluate using Monte Carlo. We refer to this as the score function estimator of the gradient.

Unfortunately, the score function estimator has an important shortcoming: it has a high variance. What does this mean? Suppose you are using Monte Carlo to estimate some quantity with expected value equal to 1. If your samples are all close to 1 (e.g., $\{0.9, 1.1, 0.96, 1.05, \dots\}$), then after a few samples you will get a good estimate of the true expectation. On the other hand, suppose that 99% of the time you sample 0, and 1% of the time you sample 100. Then the expectation is still correct, but you will have to take a very large number of samples to figure out that the true expectation is actually 1. We refer to the latter case as being *high variance*.

This is the kind of problem we often run into with the score function estimator. In fact, its variance is so high that we cannot use it to learn many models.

The key contribution of the VAE paper is to propose an alternative estimator that is much better behaved. This is done in two steps: we first reformulate the ELBO so that parts of it can be computed in closed form (without Monte Carlo sampling), and then we use an alternative gradient estimator based on the so-called *reparametrization trick*.

7.2.3 The Stochastic Gradient Variational Bayes Estimator

We will turn to a reformulation of the ELBO that takes the form

$$\log p(\mathbf{x}) \geq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x} \mid \mathbf{z})\right] - \mathrm{KL}(q_\phi(\mathbf{z} \mid \mathbf{x}) \,||\, p(\mathbf{z})).$$

This reparametrization is known as the *Stochastic Gradient Variational Bayes* (SGVB) estimator, and it has a very interesting interpretation. First, think of $\mathbf{x}$ as an observed data point. The right-hand side consists of two terms. Both terms involve taking a sample $\mathbf{z} \sim q_\phi(\mathbf{z} \mid \mathbf{x})$, which we can interpret as a code describing $\mathbf{x}$. We are therefore going to call q_ϕ the *encoder*.

In the first term, $\log p_\theta(\mathbf{x} \mid \mathbf{z})$ is the log-likelihood of the observed $\mathbf{x}$ given the code $\mathbf{z}$ that we have sampled. This term is maximized when $p_\theta(\mathbf{x} \mid \mathbf{z})$ assigns high probability to the original $\mathbf{x}$. It is thus trying to

reconstruct $\mathbf{x}$ given the code $\mathbf{z}$. For that reason, we call $p_\theta(\mathbf{x} \mid \mathbf{z})$ the *decoder* and call the first term the *reconstruction error*.

The second term is the divergence between $q_\phi(\mathbf{z} \mid \mathbf{x})$ and the prior $p(\mathbf{z})$, which we will fix to be a unit Normal distribution. This encourages the codes $\mathbf{z}$ to look Gaussian. We call this the *regularization* term. This prevents $q_\phi(\mathbf{z} \mid \mathbf{x})$ from simply encoding an identity mapping, and instead forces it to learn some more interesting representation (e.g., the facial features in our first example).

Thus, our optimization objective is trying to fit a $q_\phi(\mathbf{z} \mid \mathbf{x})$ that will map $\mathbf{x}$ into a useful latent space $\mathbf{z}$ from which we are able to reconstruct $\mathbf{x}$ via $p_\theta(\mathbf{x} \mid \mathbf{z})$. This type of objective is reminiscent of autoencoder neural networks. This is where the AEVB algorithm takes its name.

7.2.4 The Reparametrization Trick

As we have seen earlier, optimizing our objective requires a good estimate of the gradient. The main technical contribution of the VAE paper is a low-variance gradient estimator based on the *reparametrization trick*.

Under certain mild conditions, we may express the distribution $q_\phi(\mathbf{z} \mid \mathbf{x})$ as the following two-step generative process.

1. First, we sample a noise variable ϵ from a simple distribution $p(\epsilon)$ (e.g., the standard Normal $\mathcal{N}(0, 1)$):

$$\epsilon \sim p(\epsilon).$$

2. Then, we apply a deterministic transformation $g_\phi(\epsilon, \mathbf{x})$ that maps the random noise to a more complex distribution:

$$\mathbf{z} = g_\phi(\epsilon, \mathbf{x}).$$

For many interesting classes of q_ϕ, it is possible to choose a $g_\phi(\epsilon, \mathbf{x})$, such that $\mathbf{z} = g_\phi(\epsilon, \mathbf{x})$ will be distributed according to $q_\phi(\mathbf{z} \mid \mathbf{x})$.

Gaussian variables provide the simplest example of the reparametrization trick. Instead of writing $\mathbf{z} \sim q_{\mu,\sigma}(\mathbf{z}) = \mathcal{N}(\mu, \sigma)$, we can write

$$\mathbf{z} = g_{\mu,\sigma}(\epsilon) = \mu + \epsilon \cdot \sigma,$$

where $\epsilon \sim \mathcal{N}(0, 1)$. It is easy to check that the two ways of expressing the random variable $\mathbf{z}$ lead to the same distribution.

The biggest advantage of this approach is that we can now write the gradient of an expectation with respect to $q_\phi(\mathbf{z} \mid \mathbf{x})$ (for any f) as

$$\begin{aligned}\nabla_\phi \mathbb{E}_{z\sim q_\phi(\mathbf{z}|\mathbf{x})}\left[f(\mathbf{x},\mathbf{z})\right] &= \nabla_\phi \mathbb{E}_{\epsilon\sim p(\epsilon)}\left[f(\mathbf{x}, g_\phi(\epsilon,\mathbf{x}))\right] \\ &= \mathbb{E}_{\epsilon\sim p(\epsilon)}\left[\nabla_\phi f(\mathbf{x}, g_\phi(\epsilon,\mathbf{x}))\right].\end{aligned}$$

Since $p(\epsilon)$ does depend on ϕ, we can swap the gradient ∇_ϕ and the expectation $\mathbb{E}_{\epsilon\sim p(\epsilon)}$. As the gradient is now inside the expectation, we can take Monte Carlo samples to estimate the right-hand term. This approach has much lower variance than the score function estimator, and will enable us to learn models that we couldn't otherwise learn.

7.2.5 Choosing q and p

Until now, we did not specify the exact form of p_θ or q_ϕ, besides saying that these could be arbitrary functions. How should one parametrize these distributions? The best $q_\phi(\mathbf{z} \mid \mathbf{x})$ should be able to approximate the posterior $p_\theta(\mathbf{z} \mid \mathbf{x})$. Similarly, $p_\theta(\mathbf{x} \mid \mathbf{z})$ should be flexible enough to represent the richness of the data.

For these reasons, we are going to parametrize q and p by *neural networks*. These are extremely expressive function approximators that can be efficiently optimized over large datasets. This choice also draws a fascinating bridge between classical machine learning methods (approximate Bayesian inference in this case) and modern deep learning.

But what does it mean to parametrize a distribution with a neural network? Let's assume again that $q(\mathbf{z} \mid \mathbf{x})$ and $p(\mathbf{x} \mid \mathbf{z})$ are Normal distributions. We can express them as

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

where $\boldsymbol{\mu}(\mathbf{x}), \boldsymbol{\sigma}(\mathbf{x})$ are deterministic vector-valued functions of $\mathbf{x}$ parametrized by an arbitrary complex neural network.

More generally, the same technique can be applied to any exponential family distribution by parameterizing the sufficient statistics by a function of $\mathbf{x}$.

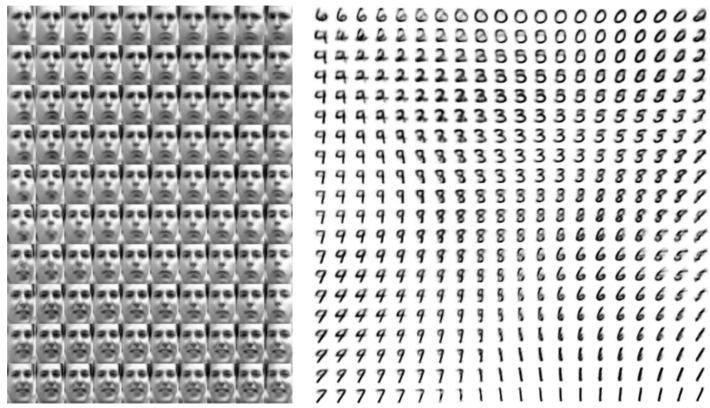

Figure 7.1: Data manifolds learned by AEVB for the Frey Face dataset (left) and MNIST (right). We can interpolate over human faces and written digits by interpolating over the latent variables. Figure from Kingma and Welling (2014).

7.3 The Variational Autoencoder

We are now ready to define the AEVB algorithm and the VAE, its most popular instantiation. The AEVB algorithm simply combines

1. the auto-encoding ELBO reformulation;
2. the black-box variational inference approach; and
3. the reparametrization-based low-variance gradient estimator.

It optimizes the auto-encoding ELBO using black-box variational inference with the reparametrized gradient estimator. This algorithm is applicable to any deep generative model p_θ with latent variables that is differentiable in θ.

A VAE uses the AEVB algorithm to learn a specific model p using a particular encoder q. The model p is parametrized as

$$\begin{aligned} p(\mathbf{x} \mid \mathbf{z}) &= \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}(\mathbf{z}), \operatorname{diag}(\boldsymbol{\sigma}(\mathbf{z}))^2) \\ p(\mathbf{z}) &= \mathcal{N}(\mathbf{z}; 0, I), \end{aligned}$$

where $\boldsymbol{\mu}(\mathbf{z}), \boldsymbol{\sigma}(\mathbf{z})$ are parametrized by a neural network (in the original formulation, two dense hidden layers of 500 units each). Model q is

similarly parametrized as

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

This choice of p and q allows us to further simplify the auto-encoding ELBO. In particular, we can use a closed form expression to compute the regularization term, and we only use Monte Carlo estimates for the reconstruction term. These expressions are given in the original paper.

We can interpret the VAE as a directed latent variable probabilistic graphical model. We can also view it as a particular objective for training an autoencoder neural network. Unlike previous approaches, this objective derives reconstruction and regularization terms from a more principled, Bayesian perspective.

7.3.1 Applications of the VAE

The VAE can be applied to images in order to learn interesting latent representations, as illustrated on the Frey Face dataset and the MNIST handwritten digits database[1] (Figure 7.1). On the face dataset, we can interpolate between facial expressions by interpolating between latent variables (e.g., we can generate smooth transitions between "angry" and "surprised"). On the MNIST dataset, we can similarly interpolate between numbers. The VAE has also been used as a design tool in the basic sciences and engineering. For example, VAEs have been used in chemistry for the automated design of novel molecules (Gómez-Bombarelli *et al.*, 2018) and in materials science to generate new stable materials (Xie *et al.*, 2022).

Further Reading

- D. P. Kingma and M. Welling. (2019). "An Introduction to Variational Autoencoders". *Foundations and Trends® in Machine Learning*. 12(4): 307–392. ISSN: 1935-8237, 1935-8245. DOI: 10.1561/2200000056.

[1] https://yann.lecun.com/exdb/mnist/