Advanced machine learning and data analysis for the physical sciences

Morten Hjorth-Jensen[1]

Department of Physics and Center for Computing in Science Education,
University of Oslo, Norway[1]

April 24, 2025

© 1999-2025, Morten Hjorth-Jensen. Released under CC Attribution-NonCommercial 4.0 license

Plans for the week April 21-25, 2025

Deep generative models

1. Variational Autoencoders (VAE), mathematics, basic mathematics
2. Writing our own codes for VAEs

Readings

1. Add VAE material

Mathematics of Variational Autoencoders

We have defined earlier a probability (marginal) distribution with hidden variables $\boldsymbol{h}$ and parameters $\boldsymbol{\Theta}$ as

$$p(\boldsymbol{x};\boldsymbol{\Theta}) = \int d\boldsymbol{h} p(\boldsymbol{x},\boldsymbol{h};\boldsymbol{\Theta}),$$

for continuous variables $\boldsymbol{h}$ and

$$p(\boldsymbol{x};\boldsymbol{\Theta}) = \sum_{\boldsymbol{h}} p(\boldsymbol{x},\boldsymbol{h};\boldsymbol{\Theta}),$$

for discrete stochastic events $\boldsymbol{h}$. The variables $\boldsymbol{h}$ are normally called the **latent variables** in the theory of autoencoders. We will also call then for that here.

Using the conditional probability

Using the the definition of the conditional probabilities $p(\boldsymbol{x}|\boldsymbol{h};\boldsymbol{\Theta})$, $p(\boldsymbol{h}|\boldsymbol{x};\boldsymbol{\Theta})$ and and the prior $p(\boldsymbol{h})$, we can rewrite the above equation as

$$p(\boldsymbol{x};\boldsymbol{\Theta}) = \sum_{\boldsymbol{h}} p(\boldsymbol{x}|\boldsymbol{h};\boldsymbol{\Theta})p(\boldsymbol{h},$$

which allows us to make the dependence of $\boldsymbol{x}$ on $\boldsymbol{h}$ explicit by using the law of total probability. The intuition behind this approach for finding the marginal probability for $\boldsymbol{x}$ is to optimize the above equations with respect to the parameters $\boldsymbol{\Theta}$. This is done normally by maximizing the probability, the so-called maximum-likelihood approach discussed earlier.

VAEs versus autoencoders

This trained probability is assumed to be able to produce similar samples as the input. In VAEs it is then common to compare via for example the mean-squared error or the cross-entropy the predicted values with the input values. Compared with autoencoders, we are now producing a probability instead of a functions which mimicks the input.
In VAEs, the choice of this output distribution is often Gaussian, meaning that the conditional probability is

$$p(\boldsymbol{x}|\boldsymbol{h};\boldsymbol{\Theta}) = N(\boldsymbol{x}|f(\boldsymbol{h};\boldsymbol{\Theta}), \sigma^2 \times \boldsymbol{I}),$$

with mean value given by the function $f(\boldsymbol{h};\boldsymbol{\Theta})$ and a diagonal covariance matrix multiplied by a parameter σ^2 which is treated as a hyperparameter.

Gradient descent

By having a Gaussian distribution, we can use gradient descent (or any other optimization technique) to increase $p(\boldsymbol{x}; \boldsymbol{\Theta})$ by making $f(\boldsymbol{h}; \boldsymbol{\Theta})$ approach $\boldsymbol{x}$ for some $\boldsymbol{h}$, gradually making the training data more likely under the generative model. The important property is simply that the marginal probability can be computed, and it is continuous in $\boldsymbol{\Theta}$..

Are VAEs just modified autoencoders?

The mathematical basis of VAEs actually has relatively little to do with classical autoencoders, for example the sparse autoencoders or denoising autoencoders discussed earlier.
VAEs approximately maximize the probability equation discussed above. They are called autoencoders only because the final training objective that derives from this setup does have an encoder and a decoder, and resembles a traditional autoencoder. Unlike sparse autoencoders, there are generally no tuning parameters analogous to the sparsity penalties. And unlike sparse and denoising autoencoders, we can sample directly from $p(\boldsymbol{x})$ without performing Markov Chain Monte Carlo.

Training VAEs

To solve the integral or sum for $p(\boldsymbol{x})$, there are two problems that VAEs must deal with: how to define the latent variables $\boldsymbol{h}$, that is decide what information they represent, and how to deal with the integral over $\boldsymbol{h}$. VAEs give a definite answer to both.

Kullback-Leibler relative entropy (notation to be updated)

When the goal of the training is to approximate a probability distribution, as it is in generative modeling, another relevant measure is the **Kullback-Leibler divergence**, also known as the relative entropy or Shannon entropy. It is a non-symmetric measure of the dissimilarity between two probability density functions p and q. If p is the unkown probability which we approximate with q, we can measure the difference by

$$\mathrm{KL}(p||q) = \int_{-\infty}^{\infty} p(\boldsymbol{x}) \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} d\boldsymbol{x}.$$

Kullback-Leibler divergence and RBMs

Thus, the Kullback-Leibler divergence between the distribution of the training data $f(\boldsymbol{x})$ and the model marginal distribution $p(\boldsymbol{x}; \boldsymbol{\Theta})$ from an RBM is

$$
\begin{aligned}
\mathrm{KL}(f(\boldsymbol{x})||p(\boldsymbol{x}|\boldsymbol{\Theta})) &= \int_{-\infty}^{\infty} f(\boldsymbol{x}) \log \frac{f(\boldsymbol{x})}{p(\boldsymbol{x}; \boldsymbol{\Theta})} d\boldsymbol{x} \\
&= \int_{-\infty}^{\infty} f(\boldsymbol{x}) \log f(\boldsymbol{x}) d\boldsymbol{x} - \int_{-\infty}^{\infty} f(\boldsymbol{x}) \log p(\boldsymbol{x}; \boldsymbol{\Theta}) d\boldsymbol{x} \\
&= \langle \log f(\boldsymbol{x}) \rangle_{f(\boldsymbol{x})} - \langle \log p(\boldsymbol{x}; \boldsymbol{\Theta}) \rangle_{f(\boldsymbol{x})} \\
&= \langle \log f(\boldsymbol{x}) \rangle_{data} + \langle E(\boldsymbol{x}) \rangle_{data} + \log Z.
\end{aligned}
$$

Maximizing log-likelihood

The first term is constant with respect to $\boldsymbol{\Theta}$ since $f(\boldsymbol{x})$ is independent of $\boldsymbol{\Theta}$. Thus the Kullback-Leibler divergence is minimal when the second term is minimal. The second term is the log-likelihood cost function, hence minimizing the Kullback-Leibler divergence is equivalent to maximizing the log-likelihood.

Back to VAEs

We want to train the marginal probability with some latent variables $\boldsymbol{h}$

$$p(\boldsymbol{x}; \boldsymbol{\Theta}) = \int d\boldsymbol{h} p(\boldsymbol{x}, \boldsymbol{h}; \boldsymbol{\Theta}),$$

for the continuous version (see previous slides for the discrete variant).

Using the KL divergence

In practice, for most $\boldsymbol{h}$, $p(\boldsymbol{x}|\boldsymbol{h};\boldsymbol{\Theta})$ will be nearly zero, and hence contribute almost nothing to our estimate of $p(\boldsymbol{x})$.
The key idea behind the variational autoencoder is to attempt to sample values of $\boldsymbol{h}$ that are likely to have produced $\boldsymbol{x}$, and compute $p(\boldsymbol{x})$ just from those.
This means that we need a new function $Q(\boldsymbol{h}|\boldsymbol{x})$ which can take a value of $\boldsymbol{x}$ and give us a distribution over $\boldsymbol{h}$ values that are likely to produce $\boldsymbol{x}$. Hopefully the space of $\boldsymbol{h}$ values that are likely under Q will be much smaller than the space of all $\boldsymbol{h}$'s that are likely under the prior $p(\boldsymbol{h})$. This lets us, for example, compute $E_{\boldsymbol{h}\sim Q}p(\boldsymbol{x}|\boldsymbol{h})$ relatively easily. Note that we drop $\boldsymbol{\Theta}$ from here and for notational simplicity.

Kullback-Leibler again

However, if $\boldsymbol{h}$ is sampled from an arbitrary distribution with PDF $Q(\boldsymbol{h})$, which is not $\mathcal{N}(0, I)$, then how does that help us optimize $p(\boldsymbol{x})$?
The first thing we need to do is relate $E_{\boldsymbol{h}\sim Q} P(\boldsymbol{x}|\boldsymbol{h})$ and $p(\boldsymbol{x})$. We will see where Q comes from later.
The relationship between $E_{\boldsymbol{h}\sim Q} p(\boldsymbol{x}|\boldsymbol{h})$ and $p(\boldsymbol{x})$ is one of the cornerstones of variational Bayesian methods. We begin with the definition of Kullback-Leibler divergence (KL divergence or $\mathcal{D}$) between $p(\boldsymbol{h}|\boldsymbol{x})$ and $Q(\boldsymbol{h})$, for some arbitrary Q (which may or may not depend on $\boldsymbol{x}$):

$$\mathcal{D}\left[Q(\boldsymbol{h}) \| p(\boldsymbol{h}|\boldsymbol{x})\right] = E_{\boldsymbol{h}\sim Q}\left[\log Q(\boldsymbol{h}) - \log p(\boldsymbol{h}|\boldsymbol{x})\right].$$

And applying Bayes rule

We can get both $p(\boldsymbol{x})$ and $p(\boldsymbol{x}|\boldsymbol{h})$ into this equation by applying Bayes rule to $p(\boldsymbol{h}|\boldsymbol{x})$

$$\mathcal{D}\left[Q(\boldsymbol{h})\|p(\boldsymbol{h}|\boldsymbol{x})\right] = E_{\boldsymbol{h}\sim Q}\left[\log Q(\boldsymbol{h}) - \log p(\boldsymbol{x}|\boldsymbol{h}) - \log p(\boldsymbol{h})\right] + \log p(\boldsymbol{x}).$$

Here, $\log p(\boldsymbol{x})$ comes out of the expectation because it does not depend on $\boldsymbol{h}$. Negating both sides, rearranging, and contracting part of $E_{\boldsymbol{h}\sim Q}$ into a KL-divergence terms yields:

$$\log p(\boldsymbol{x}) - \mathcal{D}\left[Q(\boldsymbol{h})\|p(\boldsymbol{h}|\boldsymbol{x})\right] = E_{\boldsymbol{h}\sim Q}\left[\log p(\boldsymbol{x}|\boldsymbol{h})\right] - \mathcal{D}\left[Q(\boldsymbol{h})\|P(\boldsymbol{h})\right].$$

Rearranging

Using Bayes rule we obtain

$$E_{\boldsymbol{h}\sim Q}\left[\log p(y_i|\boldsymbol{h}, x_i)\right] = E_{\boldsymbol{h}\sim Q}\left[\log p(\boldsymbol{h}|y_i, x_i) - \log p(\boldsymbol{h}|x_i) + \log p(y_i|x_i)\right]$$

Rearranging the terms and subtracting $E_{\boldsymbol{h}\sim Q} \log Q(\boldsymbol{h})$ from both sides gives

$$\log P(y_i|x_i) - E_{\boldsymbol{h}\sim Q}\left[\log Q(\boldsymbol{h}) - \log p(\boldsymbol{h}|x_i, y_i)\right] = E_{\boldsymbol{h}\sim Q}\left[\log p(y_i|\boldsymbol{h}, x_i) + \log p(\boldsymbol{h}|x_i) - \log Q(\boldsymbol{h})\right]$$

Note that $\boldsymbol{x}$ is fixed, and Q can be *any* distribution, not just a distribution which does a good job mapping $\boldsymbol{x}$ to the $\boldsymbol{h}$'s that can produce X.

Inferring the probability

Since we are interested in inferring $p(\boldsymbol{x})$, it makes sense to construct a Q which *does* depend on $\boldsymbol{x}$, and in particular, one which makes $\mathcal{D}\left[Q(\boldsymbol{h})\|p(\boldsymbol{h}|\boldsymbol{x})\right]$ small

$$\log p(\boldsymbol{x}) - \mathcal{D}\left[Q(\boldsymbol{h}|\boldsymbol{x})\|p(\boldsymbol{h}|\boldsymbol{x})\right] = E_{\boldsymbol{h}\sim Q}\left[\log p(\boldsymbol{x}|\boldsymbol{h})\right] - \mathcal{D}\left[Q(\boldsymbol{h}|\boldsymbol{x})\|p(\boldsymbol{h})\right].$$

Hence, during training, it makes sense to choose a Q which will make $E_{\boldsymbol{h}\sim Q}[\log Q(\boldsymbol{h}) - \log p(\boldsymbol{h}|x_i, y_i)]$ (a $\mathcal{D}$-divergence) small, such that the right hand side is a close approximation to $\log p(y_i|y_i)$.

Central equation of VAEs

This equation serves as the core of the variational autoencoder, and it is worth spending some time thinking about what it means.

1. The left hand side has the quantity we want to maximize, namely $\log p(\boldsymbol{x})$ plus an error term.
2. The right hand side is something we can optimize via stochastic gradient descent given the right choice of Q.

Setting up SGD

So how can we perform stochastic gradient descent?
First we need to be a bit more specific about the form that $Q(\boldsymbol{h}|\boldsymbol{x})$ will take. The usual choice is to say that
$Q(\boldsymbol{h}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{h}|\mu(\boldsymbol{x};\vartheta), \Sigma(;\vartheta))$, where μ and Σ are arbitrary deterministic functions with parameters ϑ that can be learned from data (we will omit ϑ in later equations). In practice, μ and Σ are again implemented via neural networks, and Σ is constrained to be a diagonal matrix.

More on the SGD

The name variational "autoencoder" comes from the fact that μ and Σ are "encoding" $\boldsymbol{x}$ into the latent space $\boldsymbol{h}$. The advantages of this choice are computational, as they make it clear how to compute the right hand side. The last term—$\mathcal{D}\left[Q(\boldsymbol{h}|\boldsymbol{x})\|p(\boldsymbol{h})\right]$—is now a KL-divergence between two multivariate Gaussian distributions, which can be computed in closed form as:

$$\mathcal{D}[\mathcal{N}(\mu_0, \Sigma_0)\|\mathcal{N}(\mu_1, \Sigma_1)] =$$
$$\frac{1}{2}\left(\mathrm{tr}\left(\Sigma_1^{-1}\Sigma_0\right) + (\mu_1 - \mu_0)^\top \Sigma_1^{-1}(\mu_1 - \mu_0) - k + \log\left(\frac{\det \Sigma_1}{\det \Sigma_0}\right.\right.$$

where k is the dimensionality of the distribution.

Simplification

In our case, this simplifies to:

$$\mathcal{D}[\mathcal{N}(\mu(X), \Sigma(X)) \| \mathcal{N}(0, I)] =$$
$$\frac{1}{2}\left(\mathrm{tr}\left(\Sigma(X)\right) + \left(\mu(X)\right)^{\top}\left(\mu(X)\right) - k - \log\det\left(\Sigma(X)\right)\right).$$

Terms to compute

The first term on the right hand side is a bit more tricky. We could use sampling to estimate $E_{z\sim Q}[\log P(X|z)]$, but getting a good estimate would require passing many samples of z through f, which would be expensive. Hence, as is standard in stochastic gradient descent, we take one sample of z and treat $\log P(X|z)$ for that z as an approximation of $E_{z\sim Q}[\log P(X|z)]$. After all, we are already doing stochastic gradient descent over different values of X sampled from a dataset D. The full equation we want to optimize is:

$$E_{X\sim D}[\log P(X) - \mathcal{D}[Q(z|X)\|P(z|X)]] = E_{X\sim D}[E_{z\sim Q}[\log P(X|z)] - \mathcal{D}[Q(z|X)\|P(z)]].$$

Computing the gradients

If we take the gradient of this equation, the gradient symbol can be moved into the expectations. Therefore, we can sample a single value of X and a single value of z from the distribution $Q(z|X)$, and compute the gradient of:

$$\log P(X|z) - \mathcal{D}\left[Q(z|X)\|P(z)\right]. \tag{1}$$

We can then average the gradient of this function over arbitrarily many samples of X and z, and the result converges to the gradient. There is, however, a significant problem $E_{z\sim Q}\left[\log P(X|z)\right]$ depends not just on the parameters of P, but also on the parameters of Q. In order to make VAEs work, it is essential to drive Q to produce codes for X that P can reliably decode.

$$E_{X\sim D}\left[E_{\epsilon\sim\mathcal{N}(0,I)}[\log P(X|z=\mu(X)+\Sigma^{1/2}(X)*\epsilon)] - \mathcal{D}\left[Q(z|X)\|P(z)\right]\right]$$