5.5. Deriving the autoencoder

The starting point for an autoencoder is generative modelling. Suppose we have a dataset $x_1, \ldots, x_n$ and we decide to model it as independent samples from a random variable X, where X is generated according to a latent variable model

$$Z \longrightarrow \boxed{f_\theta} \longrightarrow X$$

where Z is some standard random variable, f_θ is a neural network, and X is a parametric random variable whose distribution depends on $f_\theta(Z)$.

The obvious way to train this model is maximum likelihood estimation, i.e. picking θ to maximize the log likelihood of the dataset, call it $\mathcal{L}(\theta)$:

$$\begin{aligned}
\mathcal{L}(\theta) &= \log \Pr(x_1, \ldots, x_n \,;\, \theta) \\
&= \sum_i \log \Pr_X(x_i \,;\, \theta) \quad \text{since } x_i \text{ modelled as independent samples} \\
&= \sum_i \log \int_z \Pr_X(x_i \mid Z = z \,;\, \theta)\, \Pr_Z(z)\, dz \quad \text{by law of total prob.} \\
&= \sum_i \log \mathbb{E}\, h_i(Z) \quad \text{where} \quad h_i(z) = \Pr_X(x_i \mid Z = z \,;\, \theta) \\
&= \sum_i \log \mathbb{E}_{z \sim Z} \Pr_X(x_i \mid Z = z \,;\, \theta) \quad \text{in cleaner notation.}
\end{aligned}$$

We could approximate this expectation using Monte Carlo. So why not try importance sampling? Importance sampling is just a drop-in replacement for Monte Carlo.

Let's pick some arbitrary sampling distribution $\tilde{Z}$. Then the log likelihood of a datapoint can be rewritten

$$\log \Pr_X(x_i \mid Z = z \,;\, \theta) = \log \mathbb{E}_{z \sim \tilde{Z}} \left\{ \Pr_X\left(x_i \mid Z = z \,;\, \theta\right) \frac{\Pr_Z(z)}{\Pr_{\tilde{Z}}(z)} \right\}.$$

perfect sampling distribution: section 5.3 page 98

So, what sampling distribution should we use? We know from general principles that the perfect sampling distribution is

$$\Pr_{\tilde{Z}}(z) = \text{const} \times \Pr_X(x_i \mid Z = z \,;\, \theta) \Pr_Z(z).$$

In other words, the perfect sampling distribution is the posterior distribution $\tilde{Z} \sim (Z \mid X = x_i)$. If we were able to use this sampling distribution, then we'd only need one sample, because every sample from $z \sim \tilde{Z}$ would give exactly the same value—from which deduce that the constant would have to be $\Pr_X(x; \theta)$, and so finding this perfect sampling distribution is just as hard as calculating the log likelihood in the first place! Nevertheless, the importance sampling method works for any sampling distribution at all, and if we make some reasonable attempt at approximating the perfect sampling distribution, we should still do well. At the very least, we ought to choose the sampling distribution dependent on the datapoint x_i.

Idea 1. The first big idea of probabilistic autoencoders is that we can use a neural network to give us a sampling distribution:

$$x_i \longrightarrow \boxed{g_\phi} \longrightarrow \tilde{Z}^{(i,\phi)}$$

where g_ϕ is a neural network, and $\tilde{Z}^{(i,\phi)}$ is the sampling distribution we'll use for datapoint x_i, a parametric random variable whose distribution depends on $g_\phi(x_i)$. We're writing it as $\tilde{Z}^{(i,\phi)}$ to emphasize that the sampling distribution depends on both x_i and ϕ.

Idea 2. The second big idea of probabilistic encoders is that it's possible to train both neural networks jointly, both f_θ and g_ϕ, by optimizing a single objective function. The first step is to get a lower bound on the log likelihood:

Jensen's inequality: section 4.7 page 90

$$
\begin{aligned}
\mathcal{L}(\theta) &= \sum_i \log \mathbb{E}_{z\sim Z} \Pr_X(x_i \mid Z = z\,;\,\theta) \\
&= \sum_i \log \mathbb{E}_{z\sim \tilde{Z}^{(i,\phi)}} \Big\{ \Pr_X\big(x_i \mid Z = z\,;\,\theta\big) \frac{\Pr_Z(z)}{\Pr_{\tilde{Z}^{(i,\phi)}}(z)} \Big\} \quad \text{imp. samp. change of var.} \\
&\geq \sum_i \mathbb{E}_{z\sim \tilde{Z}^{(i,\phi)}} \log \Big\{ \Pr_X\big(x_i \mid Z = z\,;\,\theta\big) \frac{\Pr_Z(z)}{\Pr_{\tilde{Z}^{(i,\phi)}}(z)} \Big\} \quad \text{by Jensen's inequality} \\
&= \mathcal{L}_{\text{lb}}(\theta, \phi) \quad \text{let this define the lower-bound log likelihood function } \mathcal{L}_{\text{lb}}
\end{aligned}
$$

The importance sampling change-of-variable is 'safe'—it's an equality—so the right hand side of the importance sampling equation is insensitive to ϕ, so it has no 'force' to push towards a well-trained network g_ϕ. What about the expression we get after applying Jensen's inequality? We know that for the perfect sampling distribution the term in braces $\{\cdot\}$ is constant, hence the inequality is an equality. For an imperfect sampling distribution, the inequality is strict. Therefore $\mathcal{L}_{\text{lb}}(\theta, \phi)$ is sensitive[36] to ϕ.

To train the networks, we just find the $\hat{\theta}$ and $\hat{\phi}$ that maximize $\mathcal{L}_{\text{lb}}(\theta, \phi)$. Why does this work? We've shown that

$$\mathcal{L}(\theta) \geq \max_\phi \mathcal{L}_{\text{lb}}(\theta, \phi) \quad \text{for all } \theta$$

with equality if we have a perfectly expressive network g_ϕ, capable of achieving the perfect sampling distribution by appropriate choice of ϕ. So

$$
\begin{aligned}
\max_\theta \mathcal{L}(\theta) &\geq \mathcal{L}(\hat{\theta}) \quad \text{since } \hat{\theta} \text{ may not be optimal for } \mathcal{L} \\
&\geq \max_\phi \mathcal{L}_{\text{lb}}(\hat{\theta}, \phi) \quad \text{by the lower bound, above} \\
&= \mathcal{L}_{\text{lb}}(\hat{\theta}, \hat{\phi}) = \max_\theta \max_\phi \mathcal{L}_{\text{lb}}(\theta, \phi) \quad \text{since } \hat{\theta} \text{ is optimal for } \mathcal{L}_{\text{lb}}
\end{aligned}
$$

and, if g is perfectly expressive, this is

$$= \max_\theta \mathcal{L}(\theta) \quad \text{since the lower bound, above, is tight.}$$

If g is perfectly expressive, the inequalities make a 'sandwich' and hence $\hat{\theta}$ is a maximum likelihood estimator for our original generative network. If g isn't perfectly expressive, then $\hat{\theta}$ will be less than optimal; but by choosing a more expressive g we can reduce the shortfall.

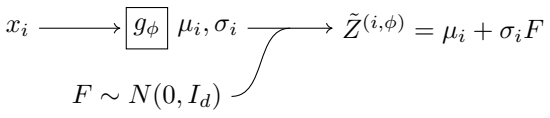
Idea 3. The third big idea is about how to turn $\mathcal{L}_{\text{lb}}(\theta, \phi)$ into something that we can actually maximize using gradient descent, and it's known as the *reparameterization trick*. To use gradient descent, we need to write the log likelihood lower-bound term

$$\mathbb{E}_{z\sim \tilde{Z}^{(i,\phi)}} \log \Big\{ \Pr_X\big(x_i \mid Z = z\,;\,\theta\big) \frac{\Pr_Z(z)}{\Pr_{\tilde{Z}^{(i,\phi)}}(z)} \Big\}$$

as a differentiable function of θ and ϕ. It's tempting to simply write it out as a Monte Carlo approximation

$$\mathbb{E}_{z\sim \tilde{Z}^{(i,\phi)}} h(z\,;\,\theta, \phi) \approx \frac{1}{m} \sum_{j=1}^m h(z_j\,;\,\theta, \phi) \quad \text{where } z_j \text{ sampled from } \tilde{Z}^{(i,\phi)}$$

and differentiate the terms in this sum with respect to θ and ϕ—but that's no good because how we *choose* z_j depends on ϕ, and choosing isn't a differentiable operation. The reparameterization trick consists in writing out the sampling distribution as an explicit latent variable model



and then using the Monte Carlo approximation

$$\mathbb{E}_{z\sim \tilde{Z}^{(i,\phi)}} h\big(z\,;\,\theta, \phi\big) \approx \frac{1}{m} \sum_{j=1}^m h\big(\mu_i + \sigma_i f_j\,;\,\theta, \phi\big) \quad \text{where } f_j \text{ sampled from } N(0, I_d).$$

[36] It can be shown with some delicate but uninteresting algebra that the error introduced by Jensen's inequality is $\mathrm{KL}(\Pr_{\tilde{Z}^{(i,\phi)}} \parallel \Pr_{(Z \mid X=x)})$. This is an algebraic way of saying "the closer our importance sampling distribution is to the perfect sampling distribution, the smaller the error."

This trick has turned the implicit dependency "z_j is sampled from a distribution that depends on ϕ" into an explicit dependency "z_j is a deterministic function of ϕ and of a sampled noise term f_j", and now it's safe to differentiate. It doesn't have to be based on a Normal distribution: we can use any distribution for F, and any function for making $\tilde{Z}$ as long as it's differentiable with respect to ϕ.

THE COMPLETE AUTOENCODER

Let's put everything together. The probabilistic autoencoder consists of two networks, an encoder network g_ϕ and a decoder network f_θ. Training consists in finding θ and ϕ to maximize

$$\mathcal{L}_{\text{lb}}(\theta, \phi) = \sum_i \mathbb{E}_F \log\Big\{ \Pr_X\big(x_i \mid \tilde{Z}^{(i,\phi)}\,;\,\theta\big) \frac{\Pr_Z(\tilde{Z}^{(i,\phi)})}{\Pr_{\tilde{Z}^{(i,\phi)}}(\tilde{Z}^{(i,\phi)})} \Big\}$$

where $\tilde{Z}^{(i,\phi)}$ is a function of $g_\phi(x_i)$ and F. This can be approximated using Monte Carlo sampling from F. Alternatively, we can rewrite it as

$$\mathcal{L}_{\text{lb}}(\theta, \phi) = \sum_i \Big\{ \Big[\mathbb{E}_F \log \Pr_X\big(x_i \mid \tilde{Z}^{(i,\phi)}\,;\,\theta\big) \Big] - \text{KL}(\Pr_{\tilde{Z}^{(i,\phi)}} \parallel \Pr_Z) \Big\}$$

where

$$\text{KL}\big(\Pr_{\tilde{Z}^{(i,\phi)}} \parallel \Pr_Z\big) = \mathbb{E}_{z \sim \tilde{Z}^{(i,\phi)}} \log\Big(\frac{\Pr_{\tilde{Z}^{(i,\phi)}}(z)}{\Pr_Z(z)} \Big).$$

This version is useful for simple sampling distributions where there's a closed-form solution for the KL term. There's no point using a computational approximation if we can calculate it exactly.

THE ENCODER AS A CERTIFICATE OF FIT

We set out to fit a generative model

$$Z \longrightarrow \boxed{f_\theta} \longrightarrow X$$

We derived an *approximating* autoencoder system, which doesn't find the perfect maximum likelihood for θ, but which can get close. If the encoder network is perfectly expressive, i.e. if it's capable of producing $\tilde{Z} \sim (Z \mid X = x)$, then the θ we find by training the autoencoder, call it $\hat{\theta}$, will be optimal:

$$\mathcal{L}(\hat{\theta}) = \max_\theta \mathcal{L}(\theta).$$

If the encoder network isn't very expressive, for example if it's just made up of a handful of nodes or if it's been inadequately trained, then there may be a large shortfall,

$$\mathcal{L}(\hat{\theta}) \ll \max_\theta \mathcal{L}(\theta).$$

There's another way to think of the autoencoder setup: as a *certificate for goodness of fit*. The whole point of generative modelling is to come up with a good model for the dataset, as measured by log likelihood. We've discussed how to use this as an objective during training—but more importantly we should use it as a metric for evaluation. In training we seek to maximize the log likelihood of the training dataset; in evaluation we measure the log likelihood of the holdout dataset.

If you simply provide a generator network as a black box, which takes in Z and spits out X, then your audience might not be able to evaluate it because they don't have any way to compute the log likelihood. But if you provide an (encoder, generator) pair, then your audience can use your encoder for computing a lower bound on the log likelihood. They can then take this lower bound *as the evaluation metric*.

- If your encoder is crummy, then the lower bound may be very poor. Your audience won't know whether it's your generator that's bad or your encoder, and they probably won't give you the benefit of the doubt. The burden of proof is on you, if you want to persuade people that your generator is great.
- If your encoder is great, then the lower bound will be very close to the true log likelihood of your generator. So, if the lower bound is great, your audience is forced to acknowledge that your generator is great.

In effect, the encoder is a certificate of goodness-of-fit for the generator.