Machine learning for Applied Physics and High Energy Physics

Lecture 23 Variational Autoencoders (VAEs)

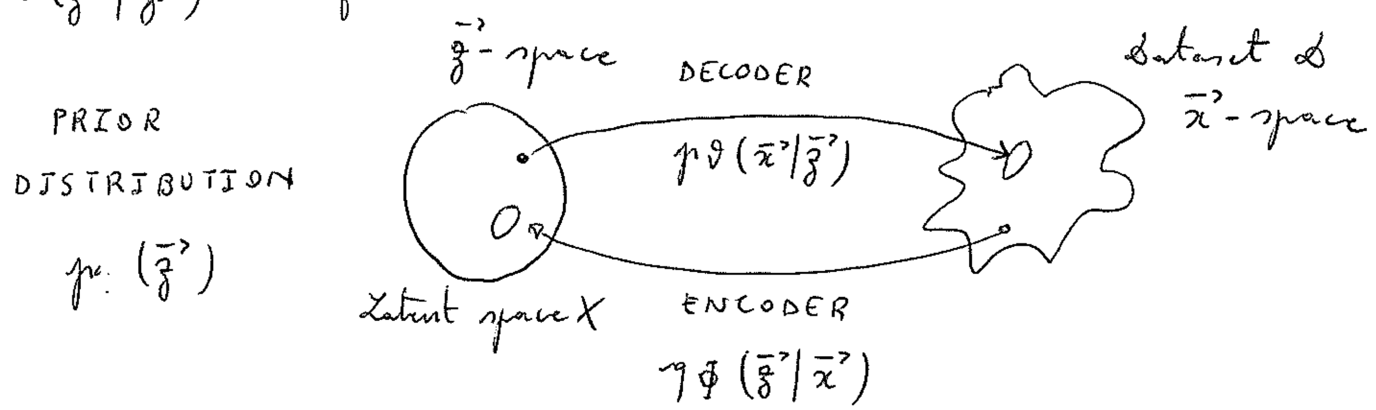
A class of powerful latent-variable generative models is represented by Variational Autoencoders (VAEs). The central idea is to represent the map from latent variables to observable variables using a deep neural network (DNN). The learning mapping is

$$p(\vec{x} \,|\, \vec{z}, \vec{\vartheta}) .$$

A VAE is a latent-variable model $p_\vartheta(\vec{x}, \vec{z})$ with

$\vec{x}$ observed variables
$\vec{z}$ latent variables

The latent variables are usually drawn from a (pre-specified) prior distribution $p(\vec{z})$, which is almost always taken as a multivariate Gaussian distribution. The conditional distribution $p_\vartheta(\vec{x} \,|\, \vec{z})$ maps points in the latent space to new examples in the data space. This is often called STOCHASTIC DECODER. and it defines the generative model for the data. The reverse mapping that gives the posterior over the latent variables $p_\vartheta(\vec{z} \,|\, \vec{x})$ is often called STOCHASTIC ENCODER.



The challenge is to infer the posterior distribution of the latent variables given a sample from the data. This can be done, in principle, using Bayes theorem

$$p_\theta(\vec{z}|\vec{x}) = \frac{p(\vec{z})\,p_\theta(\vec{x}|\vec{z})}{p_\theta(\vec{x})}$$

For some models, this can be computed analytically. In general, however, this is intractable given that the denominator requires to marginalise over all possible configurations of the latent variables, $p_\theta(\vec{x}) = \int p_\theta(\vec{x},\vec{z})\,d\vec{z} = \int p_\theta(\vec{x},\vec{z})\,p(\vec{z})\,d\vec{z}$. In VAEs, $p_\theta(\vec{x}|\vec{z})$ is a DNN, therefore marginalisation is impossible

A way to address the issue of computing $p_\theta(\vec{x})$ could be through importance sampling. That is, we choose a proposal distribution $q(\vec{z}|\vec{x})$ which is easy to sample from and write

$$p_\theta(\vec{x}) = \int p_\theta(\vec{x}|\vec{z})\,\frac{p(\vec{z})}{q_\Phi(\vec{z}|\vec{x})}\,q_\Phi(\vec{z}|\vec{x})\,d\vec{z}$$

By sampling from $q_\Phi(\vec{z}|\vec{x})$ we can get a Monte Carlo estimate of $p_\theta(\vec{x})$. However our estimates could become noisy if we do not sample a sufficiently high number of points.

We know, from the previous lecture, that we can write

$$\langle \ln p_\theta(\vec{x}) \rangle_{data} = - S[p_{data}] - D_{KL}(p_{data} \| p_\theta)$$

Let us similarly define the variational free energy

$$-F_{q_\Phi}(\vec{x}) = \mathbb{E}_{q_\Phi(\vec{z}|\vec{x})}\left[\ln p_\theta(\vec{x}|\vec{z})\right] - D_{KL}\left(q_\Phi(\vec{z}|\vec{x}) \| p(\vec{z})\right)$$

The log-likelihood is

$$\ln p(\vec{x}) = D_{KL}\left(q_\Phi(\vec{z}|\vec{x}) \| p_\theta(\vec{z}|\vec{x})\right) - F_{q_\Phi}(\vec{x})$$

Because the KL divergence is strictly positive, the (negative) variational free energy is a lower bound on the log-likelihood (evidence lower bound or ELBO)

The first term can be viewed as a "reconstruction error" where we start from data $\vec{x}$, encode it into the latent representation using an approximate posterior $q_\Phi(\vec{z}|\vec{x})$ and then evaluate the log probability of the original data given the inferred latents. The second term acts as a regulariser and encourages the posterior distributions to be close to $p(\vec{z})$. VAEs train models by minimising the variational free energy. Training a VAE is complicated because we must simultaneously learn two sets of parameters: ϑ (decoder) and Φ (encoder). The basic approach is the same as for all DNN models: we use gradient descent with the variational free energy as cost function:

$$C_{\vartheta,\Phi} = \sum_{\vec{x}} - F_{q_\Phi}(\vec{x})$$

Taking the gradient w.r.t. ϑ gives

$$C_{\vartheta,\Phi} = \mathbb{E}_{q_\Phi(\vec{z}|\vec{x})}\left[\nabla_\vartheta \ln p_\vartheta(\vec{x}|\vec{z})\right] \sim \nabla_\vartheta \ln p_\vartheta(\vec{x}|\vec{z})$$

where the approximation means that the expectation value was replaced by a single Monte Carlo sample z drawn from $q_\Phi(\vec{z}|\vec{x})$. When $p_\vartheta(\vec{x}|\vec{z})$ is approximated by a NN, this can be computed using backpropagation with the reconstruction error as the objective function.

Taking the gradient w.r.t. Φ is more complicated. Typically one uses the "reparametrisation trick". The idea is to change variables so that Φ no longer appears in the distribution we are taking an expectation value w.r.t. To this purpose, we express the random variable $\vec{z} \sim q_{\Phi}(\vec{z}|\vec{x})$ as some differentiable and invertible transformation of another random variable $\mathcal{E}$: 4

$$\vec{z} = g\,(\vec{\mathcal{E}}, \Phi, \vec{x})$$

where the distribution of $\vec{\mathcal{E}}$ is independent of $\vec{x}$ and Φ.
Then we can replace expectation values over $q_{\Phi}(\vec{z}|\vec{x})$ by expectation values over $p_{\mathcal{E}}$

$$\mathbb{E}_{q_{\Phi}(\vec{z}|\vec{x})}\left[f(\vec{z})\right] = \mathbb{E}_{p_{\vec{\mathcal{E}}}}\left[f(\vec{z})\right]$$

Evaluating the derivative then becomes quite straightforward since

$$\nabla_{\Phi}\,\mathbb{E}_{q_{\Phi}(\vec{z}|\vec{x})}\left[f(\vec{z})\right] \sim \mathbb{E}_{p_{\mathcal{E}}}\left[\nabla_{\Phi}\,f(\vec{z})\right]$$

Of course, when we do this, we still need to be able to calculate the Jacobian of this change of variables

$$d_{\Phi}(\vec{x}, \Phi) = \det\left|\frac{\partial\vec{z}}{\partial\vec{\mathcal{E}}}\right|$$

since

$$\ln q_{\Phi}(\vec{z}|\vec{x}) = \ln p\,(\vec{\mathcal{E}}) - \ln d_{\Phi}(\vec{x}, \Phi)\,.$$

Because we can compute gradients, we can deploy back-propagation.
One of the problems that commonly occur when training VAEs by stochastic optimisation of the variational free energy is that it often gets stuck in local minima, especially at the beginning of the training. The objective function can be improved in two ways:
1 reducing the reconstruction error

2 making the posterior distribution $q_\phi(\vec{z}|\vec{x})$ as close to $p(\vec{z})$ as possible.

For complex data sets, at the beginning of the training, when the reconstruction error is very poor, the model often quickly learns to make $q(\vec{z}|\vec{x}) \simeq p(\vec{z})$ and gets stuck in this local minimum. To overcome this issue, the cost function is modified as follows

$$\mathbb{E}_{q_\phi(\vec{z}|\vec{x})}\left[\ln p_\theta(\vec{x}|\vec{z})\right] - \beta\, D_{KL}\left(q_\phi(\vec{z}|\vec{x}) \,\|\, p(\vec{z})\right)$$

where β is slowly annealed from 0 to 1.

We now discuss one of the most widely used VAE architectures: a VAE with factorised Gaussian posteriors $q_\phi(\vec{z}|\vec{x}) = \mathcal{N}(\vec{z}, \vec{\mu}(\vec{x}), \mathrm{diag}(\sigma^2(\vec{x})))$ and standard normal latent variables $p(z) = \mathcal{N}(0, \mathbb{1})$

$$\begin{cases} q_\phi(\vec{z}|\vec{x}) = \mathcal{N}(\vec{z}, \vec{\mu}(\vec{x}), \mathrm{diag}(\sigma^2(\vec{x}))) \\ p(\vec{z}) = \mathcal{N}(0, \mathbb{1}) \end{cases}$$

The training and implementation simplify greatly because we can analytically work out the term

$$D_{KL}\left(q_\phi(\vec{z}|\vec{x}) \,\|\, p(\vec{z})\right)$$

Let us drop the $\vec{x}$ dependence and write

$$D_{KL}\left(q_\phi(\vec{z}|\vec{x}) \,\|\, p(\vec{z})\right) = \int d\vec{z}\, q_\phi \ln q_\phi - \int d\vec{z}\, q_\phi \ln p$$

The first term is

$$\int d\vec{z}\, q_\phi(\vec{z}) \ln p(\vec{z}) = \int \mathcal{N}(\vec{z}, \vec{\mu}(\vec{x}), \mathrm{diag}(\sigma^2(x))) \ln \mathcal{N}(\vec{0}, \mathbb{1})$$

$$= -\frac{J}{2}\ln 2\pi - \frac{1}{2}\sum_{j=1}^{J}\left(\mu_j^2 + \ln\sigma_j^2\right)$$

where J is the dimension of the latent space.
The other term is

$$\int d\vec{z}\, q_\Phi(\vec{z}) \ln q_\Phi(\vec{z}) = \int \mathcal{N}(\vec{z}, \vec{\mu}(\vec{x}), \mathrm{diag}(\sigma^2(\vec{x}))) \cdot \ln\left[\mathcal{N}(\vec{z}, \vec{\mu}(\vec{x}), \mathrm{diag}(\sigma^2(\vec{x})))\right]$$

$$= -\frac{J}{2}\ln 2\pi - \frac{1}{2}\sum_{j=1}^{J}\left(1 + \sigma_j^2\right)$$

Combining the results gives

$$-D_{KL}\left(q_\Phi(\vec{z}|\vec{x}) \,\|\, p(\vec{z})\right) = \frac{1}{2}\sum_{j=1}^{J}\left(1 + \ln\sigma_j^2 - \mu_j^2(\vec{x}) - \sigma_j^2(\vec{x})\right)$$

This analytic expression allows us to implement the Gaussian VAE in a straightforward way using neural networks. Since the parameters are all compositions of differentiable functions, we can use standard backpropagation.