Variational autoencoder

The purpose of this tutorial is to present the Variational Autoencoder and explain the mathematics behind it. Furthermore we would like to give insight to other variations of this model, to disentanglement and also to ELBO surgery.

Throughout this paper we will use the MNIST dataset of handwritten digits, to demonstrate how the model works. The MNIST dataset consists of 50000 grayscale pictures of 28×28 pixels.

Generative model

Generative models in general work in the following way. We assume there is an unknown distribution in some k-dimensional vectorspace which describes our dataset. We denote its PDF (probability density function) by p^*. By sampling from this distribution we receive a k-dimensional vector (in our case $k = 784$), which describes new data (in our case a handwritten digit). So if such a model is trained on the MNIST dataset, the p^* should assign high values to an image of a digit, and low values to a picture of random gibberish. The model receives the training data, and from that tries to learn the unknown PDF. Unfortunately this p^* isn't always learnable, so the goal is to approximate it as well as possible.

These models have many applications, e.g. generating fake human faces, or purely synthetic music, or filling a forest with 3D plants in a video game. One of the most popular generative models is the variational autoencoder.

Latent variable model

If the pixels were independent of each other, the task of learning the PDF is relatively simple, as we can learn the PDF of every pixel independently. Sampling from p^* would be easy as well, we can sample each pixel independently.

Unfortunately, in the MNIST dataset this is not the case, there are clear dependencies between the pixels. For example if on the left hand side of an image we see the left half of a four, then on the right hand side we cannot see the right half of a zero, because then the whole picture is clearly not a digit.

These dependencies are coded in the latent space. We can think of it as a d-dimensional vectorspace $\mathbb{R}^d$, where every vector contains d essential information about how to draw a digit. For example the first dimension could code the number represented by the digit, the second dimension the width, etc.

Problem scenario

Summerizing the previous paragraphs, we have a dataset, $\mathcal{D} = \{x^{(1)}, x^{(1)}, \ldots, x^{(n)}\}$, which we assume is i.i.d, from an unknown distribution p^*, this is the training set. Also, we assume that our data point x is generated by the following two-step process:

- $z^{(i)}$ is sampled from a simple prior distribution $p(z)$ on the latent space,
- $x^{(i)}$ is sampled from a complex conditional distribution $p(x|z)$.

So first the process decides what properties the digit will have, and then it draws the digit. The goal of the model is to maximize the marginal distribution at every $x^{(i)}$ in our dataset collectively. We are looking for our optimal distribution in a family of distributions, and wish to learn its parameters (the p^* distribution is not necessarily in the family).

Intractabilities

By definition, $p(x) = E_{z\sim p(z)}p(x|z)$, but in practice this integral is intractible for more interesting models. So tipically when doing maximum likelihood (ML) estimation, we cannot optimize the marginal with respect to its parameters.

We could use Monte Carlo methods, namely take N samples from the prior distribution: $\{z_1, \dots, z_N\}$ and approximate the marginal by

$$p(x) \approx \frac{1}{N}\sum_{i=1}^{N} p(x|z_i).$$

Unfortunately the dimension of the x random variable is huge, so N needs to be a very large number to achieve a relatively good approximation. So we would like to find a method which works in the case of intractability, and when sampling based methods are too slow.

The prior and the posterior

The first question is, how to define the latent variables. The decisions the model has to make are very complicated. It has to decide which digit to draw, what style it will have, etc. Also, these attributes can be correlated, e.g. if someone writes very fast, the number is usually slanted and slender at the same time.

Ideally we don't want to set these properties ourselves, also we would like to avoid describing the dependencies between the coordinates. The VAE does neither, instead we can get an element of the latent space by sampling from the standard normal distribution, so we set $p(z) = \mathcal{N}(0, I)$.

This works, because we can obtain any distribution in d dimensions by mapping the standard normal distribution in d dimensions through a sufficiently complicated function. So what happens is that the first few layers of the model learns the function which maps the normally distributed z values to whatever latent values are needed for the model, and then map those to a digit x.

We assume that the posterior is from a family of distributions with some θ unknown parameter. We'll now add a subscript θ to make this explicit, and write $p_\theta(x|z)$ from now on. Our goal is to learn this parameter vector so the above process maximizes the likelihood of the dataset $\mathcal{D}$. Instead of the marginal distribution, we will work with its logarithm, as the maximum point is the same. As the training set is i.i.d, we want to maximize the following sum: $\sum_{i=1}^{n} \log p_\theta(x^{(i)})$. So we can deal with the elements of the training set separately. With formula:

$$\arg\max_\theta \sum_{i=1}^{n} \log E_{z\sim p(z)} p_\theta(x^{(i)}|z).$$

We will denote the empirical data distribution by $\hat{P}$, which is simply the discrete uniform distribution over our dataset. The above formula using the $\hat{P}$ notation:

$$\arg\max_{\theta} E_{x\sim\hat{P}} \log E_{z\sim p(z)} p_\theta(x|z).$$

This can also be formulated as minimizing the KL-divergence between the $\hat{P}$ empirical data distribution and the $p(x)$ marginal distribution:

$$\mathrm{KL}(\hat{P}\|p_\theta(x)) = E_{x\sim\hat{P}}[\log \hat{P}(x)] - E_{x\sim\hat{P}}[\log p_\theta(x)]$$

which is a constant minus the log-likelihood.

In our case we choose the $p_\theta(x|z)$ posterior to be a normal distribution $\mathcal{N}(f(z,\theta),\sigma^2 I)$, where σ is a hyperparameter. The output of the model is $f(z,\theta)$. The good thing about these choices is that even though both the prior and the posterior are relatively simple, still the marginal can be very complex, so we can use it to approximate the real underlying distribution.

Shortcut

We said earlier, that we could approximate the marginal distribution by sampling, but we would need many samples. The question is whether we can make the process faster. In practice, for most z, the value $p_\theta(x^{(i)}|z)$ is very small, so it contributes almost nothing to our estimate.

The key idea behind the Variational Autoencoder is that we want to sample values of z that are likely to have produced $x^{(i)}$ and compute $p(x^{(i)})$ just from those. This is described exactly by the likelihood, $p(z|x^{(i)})$.

Unfortunately it is intractable, so we need another tractable distribution on the latent space, $q(z|x^{(i)})$, which approximates the likelihood. We hope that the set of z-s, for which this value is large is smaller, than thoses z-s, which are likely according to the prior. If this is true, then e.g. we can compute $E_{z\sim q} p(x^{(i)}|z)$ relatively easily, we will see why this is important.

We would like to note, that now we can see that we got the standard autoencoder structure, the q models the encoder and the posterior models the decoder.

Reforming the Kullback-Leibler divergence

What we will do is relate the marginal distribution with the $E_{z\sim q} p(x^{(i)}|z)$ expected value, we will see where q comes from later. First, we examine the KL-divergence between $q(z|x^{(i)})$ and $p_\theta(z|x^{(i)})$. By definition

$$D_{KL}(q(z|x^{(i)})\|p_\theta(z|x^{(i)})) = E_{q(z|x^{(i)})}(\log q(z|x^{(i)}) - \log p_\theta(z|x^{(i)})).$$

By Bayes' theorem

$$p_\theta(z|x^{(i)}) = \frac{p_\theta(x^{(i)} \mid z) p(z)}{p_\theta(x^{(i)})}.$$

So taking the logarithm of both sides:

$$\log p_\theta(z|x^{(i)}) = \log p_\theta(x^{(i)} \mid z) + \log p(z) - \log p_\theta(x^{(i)}).$$

Writing this in we get the following:

$$D_{KL}(q(z|x^{(i)})\|p_\theta(z|x^{(i)})) = E_{q(z|x^{(i)})}\Big(\log q(z|x^{(i)}) - \log p_\theta(x^{(i)}|z) - \log p(z) + \log p_\theta(x^{(i)})\Big).$$

As the last term in the expectation doesn't depend on z, we can take it out:

$$D_{KL}(q(z|x^{(i)})\|p_\theta(z|x^{(i)})) = E_{q(z|x^{(i)})}\Big(\log q(z|x^{(i)}) - \log p_\theta(x^{(i)}|z) - \log p(z)\Big) + \log p_\theta(x^{(i)}).$$

After rearranging

$$\log p_\theta(x^{(i)}) - D_{KL}(q(z|x^{(i)})\|p_\theta(z|x^{(i)})) = E_{q(z|x^{(i)})}\Big(\log p_\theta(x^{(i)}|z) + \log p(z) - \log q(z|x^{(i)})\Big).$$

We can rearrange the right hand side, as

$$D_{KL}(q_\phi(z|x^{(i)})\|p(z)) = E_{q(z|x^{(i)})}\Big(\log q(z|x^{(i)}) - p(z)\Big),$$

so

$$\log p_\theta(x^{(i)}) - D_{KL}(q(z|x^{(i)})\|p_\theta(z|x^{(i)})) = E_{q(z|x^{(i)})}\Big(\log p_\theta(x^{(i)}|z)\Big) - D_{KL}(q_\phi(z|x^{(i)})\|p(z)).$$

This last equation is the essence of the variational autoencoder. On the left hand size we have the quantity we want to maximize. Also, we have another term, a Kullback-Leibler divergence which we want to minimize, as we want the q distribution to approximate the likelihood. So by maximizing the left hand side we kill two birds with one stone.

ELBO

The main difference between the VAE and the standard autoencoder lies in the latent space. The main advantage of the VAE is that it learns a smooth latent representation of the input, while the standard autoencoder only learns a function which helps reconstruct the input.

We can see on the left hand side what happens if we focus only on the reconstruction loss. We can separate the distinct classes from each other, so we can recontruction is easier. Unfortunately there are areas in the latent space, which doesn't represent any observed datapoint.

On the other hand if we focus only on the Kullback-Leibler divergence, (so what we want is to pull the latent distribution to the prior), then we describe every datapoint with the same distribution, namely the standard normal. In this case the model thinks that every observation has the same characteristics, so we cant't characterize the original dataset.

But if we optimize the two together, then we get a latent space which maintains the similarity of nearby encodings, so we get clusters again, but globally it is densely packed near the latent space origin.

Now let's look at the right hand side. It is usually called ELBO and it is denoted by

$$\mathcal{L}(\theta, \phi, x^{(i)}) = E_{q_\phi(z|x^{(i)})} \log p_\theta(x^{(i)}|z) - D_{KL}(q_\phi(z|x^{(i)}) \| p(z)).$$

It is also called Variational Lower Bound. It is indeed a lower bound, as the Kullback-Leibler divergence is always non-negative. So instead of maximizing the left hand side, we maximize the ELBO, which is more managable, as we will see.

Usually the q distribution is chosen to be a normal distribution, namely $q(z|x^{(i)}) = \mathcal{N}(\mu_\phi(x^{(i)}), \Sigma_\phi(x^{(i)}))$. It is parametrized by a neural network with parameters ϕ, the μ and Σ are deterministic functions depending on ϕ, they are learned by the network. Usually they choose the Σ to be diagonal.

The reason for this is that we can compute the Kullback-Leibler divergence between two normal distributions in closed form. Let $\mathcal{N}(\mu_0, \Sigma_0)$ and $\mathcal{N}(\mu_1, \Sigma_1)$ be arbitrary normal distributions. Then their KL-divergence is:

$$D_{KL}(\mathcal{N}(\mu_0, \Sigma_0) \| \mathcal{N}(\mu_1, \Sigma_1)) =$$

$$= \frac{1}{2}\Big(\operatorname{tr}(\Sigma_1^{-1}\Sigma_0) + (\mu_1 - \mu_0)^T \Sigma_1^{-1}(\mu_1 - \mu_0) - k + \log\Big(\frac{\det \Sigma_1}{\det \Sigma_0}\Big)\Big),$$

where k is the dimension of the distributions. In our case:

$$D_{KL}(\mathcal{N}(\mu_\phi(x^{(i)}), \Sigma_\phi(x^{(i)})) \| \mathcal{N}(0, I)) =$$

$$= \frac{1}{2}\Big(\operatorname{tr}(\Sigma_\phi(x^{(i)})) + (\mu_\phi(x^{(i)})^T (\mu_\phi(x^{(i)}) - k + \log\det(\Sigma_\phi(x^{(i)}))\Big)$$

The first term is slightly more complicated. We could approximate it by sampling, but people observed that it is enough to take only one sample from the q distribution and for that concrete z value $\log p_\theta(x^{(i)}|z)$ will be a fair approximation of the expected value.

So we have the logarithm of the PDF of a normal distribution. Computing it we have a constant term, which doesn't count, and the squared distance of the $x^{(i)}$ and $f(z, \theta)$ vectors, which is the reconstruction loss.

We would like to find the θ and ϕ parameters, so we want to take the gradient of the ELBO with respect to these. The second term doesn't depend on θ, so

$$\nabla_\theta \mathcal{L}(\theta, \phi, x^{(i)}) = \nabla_\theta E_{q_\phi(z|x^{(i)})} \log p_\theta(x^{(i)}|z).$$

As $q_\phi(z|x^{(i)})$ doesn't depend on θ as well,

$$\nabla_\theta E_{q_\phi(z|x^{(i)})} \log p_\theta(x^{(i)}|z) = E_{q_\phi(z|x^{(i)})}(\nabla_\theta \log p_\theta(x^{(i)}|z)) \simeq$$

$$\simeq \nabla_\theta \log p_\theta(x^{(i)}|z),$$

where the last term is the Monte Carlo estimation of the previous term, and z is a random sample from the q distribution.

Reparametrization trick

Taking the gradient by the ϕ parameters, we will have a problem with the first term, as we cannot take the gradient into the integral. The real problem is that the backpropagation

needs to go through a layer which gives us a z vector from the q distribution by sampling, and this operation is not continuous.

The idea is that we reparametrize the z random variable as a function of $x^{(i)}$, ϕ, and a new random variable ε. Generally if z is a random variable and g is a differentiable and invertible transformation, and $z = g(\varepsilon, \phi, x)$ where the distribution of the ε random variable $r(\varepsilon)$ doesn't depend on x or ϕ, then

$$E_{q_\phi(z|x^{(i)})}(f(z)) = E_{r(\varepsilon)}(f(g(\varepsilon, \phi, x))).$$

In our case the posterior is $\mathcal{N}(\mu_\phi(x^{(i)}), \Sigma_\phi(x^{(i)}))$, let $\varepsilon \sim \mathcal{N}(0, I)$. Then $z = \mu_\phi(x^{(i)} + \Sigma_\phi^{1/2}(x^{(i)}) \cdot \varepsilon$, which is a differentiable, invertible function, so backpropation works and we can do the same thing as before, we can take tha gadient into the expectation.

Testing

After we trained the model and we wish to test it, we don't need the encoder part. The only thing we need to do is sample from the prior and send is through the decoder.