Deep learning

7.4. Variational autoencoders

François Fleuret
https://fleuret.org/dlc/
Coming back to generating a signal, instead of training an autoencoder and modeling the distribution of $Z$, we can try an alternative approach:

**Impose a distribution for $Z$** and then train a decoder $g$ so that $g(Z)$ matches the training data.

---

**Notes**

We have seen in lecture 7.2. “Autoencoders” that autoencoders can, to some extent, model the data distribution by first mapping the data to a smaller dimension latent space, and then fitting a density model.

As seen with the experiments on MNIST, this was not always satisfying because,

- either the latent space is of very small dimension, in which case it makes sense to use a parameterized density model over the latent representation, but the input signal is not modeled properly, or
- the latent space is of higher dimension, in which case it becomes harder to properly model the latent representation.
We consider the following two distributions:

- \( p \) is the distribution on \( \mathcal{X} \times \mathbb{R}^d \) of a pair \((X, Z)\) composed of an encoding state \( Z \sim \mathcal{N}(0, I) \) and the output of the decoder \( g \) on it.

- \( q \) is the distribution on \( \mathcal{X} \times \mathbb{R}^d \) of a pair \((X, Z)\) composed of a sample \( X \) taken from the data distribution and the output of the encoder on it.

**Our goal is that** \( p(X) \) **mimics the data-distribution** \( q(X) \), **that is to find** \( g \) **that maximizes the log-likelihood**

\[
\frac{1}{N} \sum_{n} \log p(x_n) = \mathbb{E}_{q(X)} \left[ \log p(X) \right].
\]

However, while we can sample \( z \) and compute \( g(z) \) for complicated \( g \)s, we cannot compute \( p(x) \) for a given \( x \), and even less compute its derivatives.

---

**Notes**

In what follows, we will use notations such as \( q(X) \) for marginals, and \( q(Z \mid X = x) \) for conditionals.

The quantity \( \mathbb{E}_{q(X)} \) here is the empirical estimation on the training data, since the said training data are i.i.d \( \sim q(X) \).
The Variational Autoencoder proposed by Kingma and Welling (2013) relies on a tractable approximation of this log-likelihood.

Note that their framework involves stochastic encoder $f$, and decoder $g$, whose outputs depend on both their inputs and additional randomness.
Remember that $q(X)$ is the data distribution, and $q(Z \mid X = x)$ is the distribution of the latent encoding $f(x)$.

We want to maximize

$$E_{q(X)} \left[ \log p(X) \right],$$

and it can be shown that

$$\log p(X = x) \geq E_{q(Z \mid X = x)} \left[ \log p(X = x \mid Z) \right] - D_{KL}(q(Z \mid X = x) \parallel p(Z)).$$

"Evidence lower bound" (ELBO)

So it makes sense to maximize

$$E_{q(X,Z)} \left[ \log p(X \mid Z) \right] - E_{q(X)} \left[ D_{KL}(q(Z \mid X) \parallel p(Z)) \right].$$

Notes

We can derive the ELBO using the KL-divergence, which is always positive

$$D_{KL}(a \parallel b) = -E_{a(u)} \left[ \log \frac{b(u)}{a(u)} \right] = -\int_a a(u) \log \frac{b(u)}{a(u)} du \geq -\int_a a(u) \left( \frac{b(u)}{a(u)} - 1 \right) du = 0.$$

$$\log p(X = x)$$

$$= E_{q(z \mid x = x)} [\log p(X = x)]$$

$$= E_{q(z \mid x = x)} \left[ \log p(X = x) \frac{p(z \mid X = x)}{q(z \mid X = x)} \right]$$

$$= E_{q(z \mid x = x)} \left[ \log \frac{p(z)p(X = x \mid z)}{p(z \mid X = x)} \frac{q(z \mid X = x)}{q(z \mid X = x)} \right]$$

$$= E_{q(z \mid x = x)} \left[ \log p(X = x \mid z) + \log \frac{p(z)}{q(z \mid X = x)} + \log \frac{q(z \mid X = x)}{p(z \mid X = x)} \right]$$

$$\geq E_{q(z \mid x = x)} [\log p(X = x \mid z)] - D_{KL} (q(z \mid X = x) \parallel p(z)) \geq 0$$
So the final loss is

$$
\mathcal{L} = \mathbb{E}_{q(X)} \left[ D_{KL}(q(Z \mid X) \parallel p(Z)) \right] - \mathbb{E}_{q(X,Z)} \left[ \log p(X \mid Z) \right].
$$

with

- $q(X)$ is the data distribution
- $p(Z) = \mathcal{N}(0, I)$.

Kingma and Welling propose that both the encoder $f$ and decoder $g$ map to a Gaussian with diagonal covariance. Hence they map to twice the dimension (e.g. $f(x) = (\mu_f(x), \sigma_f(x))$) and

- $q(Z \mid X = x) \sim \mathcal{N}(\mu_f(x), \text{diag}(\sigma_f(x)))$
- $p(X \mid Z = z) \sim \mathcal{N}(\mu_g(z), \text{diag}(\sigma_g(z)))$. 
The first term of $\mathcal{L}$ is the average of

$$D_{KL} \left( q(Z \mid X = x) \mid \mid p(Z) \right) = -\frac{1}{2} \sum_d \left( 1 + 2 \log \sigma_d(x) - \left( \mu_d(x) \right)^2 - \left( \sigma_d(x) \right)^2 \right).$$

over the $x_n$s.
The first term of $\mathcal{L}$ is the average of

$$D_{KL}( q(Z \mid X = x) \parallel p(Z) ) = -\frac{1}{2} \sum_d \left( 1 + 2 \log \sigma_d^f(x) - \left( \mu_d^f(x) \right)^2 - \left( \sigma_d^f(x) \right)^2 \right).$$

over the $x_n$s.

This can be implemented as

```python
param_f = model.encode(input)
mu_f, logvar_f = param_f.split(param_f.size(1)//2, 1)
kl = - 0.5 * (1 + logvar_f - mu_f.pow(2) - logvar_f.exp())
kl_loss = kl.sum() / input.size(0)
```
As Kingma and Welling (2013), we use a constant variance of 1 for the decoder, so the second term of $\mathcal{L}$ becomes the average of

$$-\log p(X = x \mid Z = z) = \frac{1}{2} \sum_d (x_d - \mu^g_d(z))^2 + \text{cst}$$

over the $x_n$, with one $z_n$ sampled for each, i.e.

$$z_n \sim \mathcal{N} \left( \mu^f(x_n), \sigma^f(x_n) \right), \; n = 1, \ldots, N.$$
As Kingma and Welling (2013), we use a constant variance of 1 for the decoder, so the second term of $\mathcal{D}$ becomes the average of

$$-\log p(X = x \mid Z = z) = \frac{1}{2} \sum_d (x_d - \mu^g_d(z))^2 + \text{cst}$$

over the $x_n$, with one $z_n$ sampled for each, i.e.

$$z_n \sim N\left(\mu^f(x_n), \sigma^f(x_n)\right), \quad n = 1, \ldots, N.$$

This can be implemented as

```python
std_f = torch.exp(0.5 * logvar_f)
z = torch.empty_like(mu_f).normal_() * std_f + mu_f
output = model.decode(z)

fit = 0.5 * (output - input).pow(2)
fit_loss = fit.sum() / input.size(0)
```
We had for the standard autoencoder

\[
\begin{align*}
z &= \text{model.encode(input)} \\
\text{output} &= \text{model.decode(z)} \\
\text{loss} &= 0.5 \times (\text{output} - \text{input}).\text{pow}(2).\text{sum()} / \text{input.size(0)}
\end{align*}
\]

and putting everything together we get for the VAE

\[
\begin{align*}
\text{param}_f &= \text{model.encode(input)} \\
\text{mu}_f, \text{logvar}_f &= \text{param}_f.\text{split(param}_f.\text{size(1)} / 2, 1) \\
\text{kl} &= -0.5 \times (1 + \text{logvar}_f - \text{mu}_f.\text{pow}(2) - \text{logvar}_f.\exp()) \\
\text{kl}_\text{loss} &= \text{kl}.\text{sum()} / \text{input.size(0)} \\
\text{std}_f &= \text{torch.exp}(0.5 \times \text{logvar}_f) \\
\text{z} &= \text{torch.empty_like(mu}_f).\text{normal()} \times \text{std}_f + \text{mu}_f \\
\text{output} &= \text{model.decode(z)} \\
\text{fit} &= 0.5 \times (\text{output} - \text{input}).\text{pow}(2) \\
\text{fit}_\text{loss} &= \text{fit}.\text{sum()} / \text{input.size(0)} \\
\text{loss} &= \text{kl}_\text{loss} + \text{fit}_\text{loss}
\end{align*}
\]

During inference we do not sample, and instead use \(\mu^f\) and \(\mu^g\) as prediction.

---

**Notes**

The \texttt{kl_loss} aims at making the distribution in the embedding space close to the normal density.

The fitting loss \texttt{fit_loss} aims at making the reconstructed data point correct in a probabilistic sense: the original data point should be likely under the Gaussian that we get when we come back from the latent space.
Note in particular the **re-parameterization trick**:

```python
z = torch.empty_like(mu_f).normal_() * std_f + mu_f
output = model.decode(z)
```

Implementing the sampling of \( z \) that way allows to compute the gradient w.r.t \( f \)'s parameters without any particular property of `normal_()`.
Notes

The images at the top are original test MNIST samples. The images in the middle are the reconstructed samples with the standard autoencoder as seen in lecture 7.2. "Autoencoders". The images at the bottom are the reconstructed samples obtained with the variational autoencoder.

The results are not as good as with the standard autoencoder, which is not surprising since there is an additional constraint on the distribution in the latent space.
We can look at two latent features to check that they are Normal for the VAE.

**Autoencoder**

**Variational autoencoder**

\[ N(0, 1) \]

---

**Notes**

The first scatter plot on the top-left shows the empirical distribution of two latent dimensions when the encoder is from the standard autoencoder. The plot was generated by choosing at random two dimensions among the 32 ones of the latent space, encode 1000 MNIST samples, and draw a point at the two resulting coordinates. We can see that this is not a normal distribution.

If the same process is done with the variational autoencoder we get the scatter plot at the bottom left, that we can compare with a similar scatter plot obtained by sampling independent normal coordinates, shown at the bottom right.

The variational autoencoder did its job of making the embedded representation follow a Gaussian distribution.
Notes

We can compare the result of sampling data points in the latent space and map them back in the original space with the decoder. To that purpose, we first generate a random Gaussian vector of dimension equal to that of the latent space, \(d = 32\), and then we run the decoder on that random sample, which produces an image in the original space.

The images on the top are obtained with the decoder from the standard autoencoder, and the images on the bottom are for the variational autoencoder. Although these small-scale experiments are far from the state of the art, the latter samples are more realistic.
Making the embedding $\sim \mathcal{N}(0, 1)$, often results in “disentangled” representations.

This effect can be reinforced with a greater weight of the KL term

$$
\mathcal{L} = \beta \mathbb{E}_{q(X)} \left[ D_{KL}(q(Z \mid X) \parallel p(Z)) \right] - \mathbb{E}_{q(X,Z)} \left[ \log p(X \mid Z) \right],
$$

resulting in the $\beta$-VAE proposed by Higgins et al. (2017).
β-VAE

VAE

(a) azimuth

(b) width

(c) leg style

(Higgins et al., 2017)
Figure 1: Manipulating latent variables on celebA: Qualitative results comparing disentangling performance of $\beta$-VAE ($\beta = 250$), VAE (Kingma & Welling, 2014) ($\beta = 1$) and InfoGAN (Chen et al., 2016).

In all figures of latent code traversal each block corresponds to the traversal of a single latent variable while keeping others fixed to either their inferred ($\beta$-VAE, VAE and DC-IGN where applicable) or sampled (InfoGAN) values. Each row represents a different seed image used to infer the latent values in the VAE-based models, or a random sample of the noise variables in InfoGAN. $\beta$-VAE and VAE traversal is over the $[-3, 3]$ range. InfoGAN traversal is over ten dimensional categorical latent variables. Only $\beta$-VAE and InfoGAN learnt to disentangle factors like azimuth (a), emotion (b) and hair style (c), whereas VAE learnt an entangled representation (e.g. azimuth is entangled with emotion, presence of glasses and gender). InfoGAN images adapted from Chen et al. (2016). Reprinted with permission.

Recently a scalable unsupervised approach for disentangled factor learning has been developed, called InfoGAN (Chen et al., 2016). InfoGAN extends the generative adversarial network (GAN) (Goodfellow et al., 2014) framework to additionally maximise the mutual information between a subset of the generating noise variables and the output of a recognition network. It has been reported to be capable of discovering at least a subset of data generative factors and of learning a disentangled representation of these factors. The reliance of InfoGAN on the GAN framework, however, comes at the cost of training instability and reduced sample diversity. Furthermore, InfoGAN requires some a priori knowledge of the data, since its performance is sensitive to the choice of the prior distribution and the number of the regularised noise variables. InfoGAN also lacks a principled inference network (although the recognition network can be used as one). The ability to infer the posterior latent distribution from sensory input is important when using the unsupervised model in transfer learning or zero-shot inference scenarios. Hence, while InfoGAN is an important step in the right direction, we believe that further improvements are necessary to achieve a principled way of using unsupervised learning for developing more human-like learning and reasoning in algorithms as described by Lake et al. (2016).

Finally, there is currently no general method for quantifying the degree of learnt disentanglement. Therefore there is no way to quantitatively compare the degree of disentanglement achieved by different models or when optimising the hyperparameters of a single model.

(Higgins et al., 2017)
References
