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.
We consider 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

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

\[
\frac{1}{N} \sum_n \log p(x_n) = \hat{\mathbb{E}}_{q(X)} \left[ \log p(X) \right].
\]
We consider 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

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

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.
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 $f(x) \sim q(Z \mid X = x)$.

We want to maximize

$$\mathbb{E}_{q(X)} \left[ \log p(X) \right],$$

and it can be shown that

$$\log p(X = x) \geq \mathbb{E}_{q(Z \mid X = x)} \left[ \log p(X = x \mid Z) \right] - \mathbb{D}_{KL}(q(Z \mid X = x) \parallel p(Z)).$$

“Evidence lower bound” (ELBO)
Remember that \( q(X) \) is the data distribution, and \( f(x) \sim q(Z \mid X = x) \).

We want to maximize
\[
\mathbb{E}_{q(X)} \left[ \log p(X) \right],
\]
and it can be shown that
\[
\log p(X = x) \geq \mathbb{E}_{q(Z \mid X = x)} \left[ \log p(X = x \mid Z) \right] - \mathcal{D}_{KL}(q(Z \mid X = x) \parallel p(Z)).
\]

"Evidence lower bound" (ELBO)

So it makes sense to maximize
\[
\mathbb{E}_{q(X,Z)} \left[ \log p(X \mid Z) \right] - \mathbb{E}_{q(X)} \left[ \mathcal{D}_{KL}(q(Z \mid X) \parallel p(Z)) \right].
\]
So the final loss is

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

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

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

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

$$
\mathbb{D}_{KL} \left( \frac{q(Z \mid X = x)}{\mathcal{N}(\mu^f(x), \sigma^f(x))} \parallel \frac{p(Z)}{\mathcal{N}(0, I)} \right) = -\frac{1}{2} \sum_d \left( 1 + 2 \log \sigma^f_d(x) - \left( \mu^f_d(x) \right)^2 - \left( \sigma^f_d(x) \right)^2 \right).
$$

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

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

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

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

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

$$
\mathbb{D}_{KL} \left( q(Z \mid X = x) \mid\mid p(Z) \right) = -\frac{1}{2} \sum_{d} \left[ 1 + 2 \log \sigma_f^d(x) - \left( \mu_f^d(x) \right)^2 - \left( \sigma_f^d(x) \right)^2 \right].
$$

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

$$\mathbb{D}_{\text{KL}} \left( q(Z \mid X = x) \parallel p(Z) \right) = -\frac{1}{2} \sum_d \left( 1 + 2 \log \sigma^f_d(x) - (\mu^f_d(x))^2 - (\sigma^f_d(x))^2 \right).$$

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

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

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 $\mathcal{L}$ becomes the average of

$$-\log p(X = x | Z = z) = \frac{1}{2} \sum_d (x_d - \mu_d^g(z))^2 + \text{cst}$$

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 \( \mathcal{L} \) becomes the average of

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

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 $\mathcal{L}$ becomes the average of

$$-\log p(X = x | Z = z) = \frac{1}{2} \sum_d (x_d - \mu_d^g(z))^2 + \text{cst}$$

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 $\mathcal{L}$ becomes the average of

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

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

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

This can be implemented as

```python
std_f = torch.exp(0.5 * logvar_f)
z = torch.randn_like(mu_f) * 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

```python
z = model.encode(input)
output = model.decode(z)
loss = 0.5 * (output - input).pow(2).sum() / input.size(0)
```

and putting everything together we get for the VAE

```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)
std_f = torch.exp(0.5 * logvar_f)
z = torch.randn_like(mu_f) * std_f + mu_f
output = model.decode(z)
fit = 0.5 * (output - input).pow(2)
fit_loss = fit.sum() / input.size(0)
loss = kl_loss + fit_loss
```

During inference we do not sample, and instead use \( \mu_f \) and \( \mu_g \) as prediction.
We had for the standard autoencoder

```python
z = model.encode(input)
output = model.decode(z)
loss = 0.5 * (output - input).pow(2).sum() / input.size(0)
```

and putting everything together we get for the VAE

```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)

std_f = torch.exp(0.5 * logvar_f)
z = torch.randn_like(mu_f) * std_f + mu_f
output = model.decode(z)

fit = 0.5 * (output - input).pow(2)
fit_loss = fit.sum() / input.size(0)

loss = kl_loss + fit_loss
```
We had for the standard autoencoder

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

and putting everything together we get for the VAE

\[
\begin{align*}
\text{param_f} &= \text{model.encode}(\text{input}) \\
\text{mu_f}, \text{logvar_f} &= \text{param_f}.\text{split(param_f.size(1)//2, 1)} \\
\text{kl} &= -0.5 \times (1 + \text{logvar_f} - \text{mu_f}^2 - \text{logvar_f.exp()}) \\
\text{kl_loss} &= \text{kl.sum() / input.size(0)} \\
\text{std_f} &= \text{torch.exp}(0.5 \times \text{logvar_f}) \\
\text{z} &= \text{torch.randn_like(mu_f)} \times \text{std_f} + \text{mu_f} \\
\text{output} &= \text{model.decode}(z) \\
\text{fit} &= 0.5 \times (\text{output} - \text{input})^2 \\
\text{fit_loss} &= \text{fit.sum() / input.size(0)} \\
\text{loss} &= \text{kl_loss} + \text{fit_loss}
\end{align*}
\]

During inference we do not sample, and instead use \(\mu_f\) and \(\mu_g\) as prediction.
Note in particular the **re-parameterization trick**:

```python
z = torch.randn_like(mu_f) * 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_()`.
Original

Autoencoder reconstruction ($d = 32$)

Variational Autoencoder reconstruction ($d = 32$)
We can look at two latent features to check that they are Normal for the VAE.
We can look at two latent features to check that they are Normal for the VAE.

**Autoencoder**

**Variational autoencoder**

\[ \mathcal{N}(0, 1) \]
Autoencoder sampling ($d = 32$)

Variational Autoencoder sampling ($d = 32$)
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

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

resulting in the $\beta$-VAE proposed by Higgins et al. (2017).
We propose augmenting the original VAE framework with a single hyperparameter $\beta$. For comparison, we introduce a protocol to quantitatively compare the degree of disentanglement learnt by different models. We demonstrate both qualitatively and quantitatively that our $\beta$-VAE outperforms all our baselines on this measure (ICA, PCA, VAE Kingma & Ba (2014), DC-IGN: Kulkarni et al. 2015, InfoGAN Chen et al. 2016). Furthermore, we show that $\beta$-VAE learns an entangled representation (e.g. chair width is entangled with azimuth and leg style (b)). Only $\beta$-VAE, however, learnt about the unlabelled factor of chair leg style (c). InfoGAN and $\beta$-VAE were also able to discover unlabelled factors in the dataset, such as chair width

(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)
The end
References
