Deep learning

7.4. Variational autoencoders

François Fleuret

https://fleuret.org/dlc/

UNIVERSITÉ DE GENÈVE

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

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

We consider two distributions:

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

- q is the distribution on $\mathscr{X} \times \mathbb{R}^d$ 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

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

We consider two distributions:

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

- q is the distribution on $\mathscr{X} \times \mathbb{R}^d$ 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

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

However, while we can sample z and compute $g(z)$ for complicated gs, **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

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

and it can be shown that

$$\log p(X = x) \geq \underbrace{\mathbb{E}_{q(Z \mid X = x)}\Big[\log p(X = x \mid Z)\Big] - \mathbb{D}_{\mathrm{KL}}(q(Z \mid X = x) \,\|\, p(Z))}_{\text{“Evidence lower bound” (ELBO)}}.$$

Remember that $q(X)$ is the data distribution, and $f(x) \sim q(Z \mid X = x)$.

We want to maximize

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

and it can be shown that

$$\log p(X = x) \geq \underbrace{\mathbb{E}_{q(Z \mid X=x)}\Big[\log p(X = x \mid Z)\Big] - \mathbb{D}_{\mathrm{KL}}(q(Z \mid X = x) \,\|\, p(Z))}_{\text{"Evidence lower bound" (ELBO)}}.$$

So it makes sense to maximize

$$\mathbb{E}_{q(X,Z)}\Big[\log p(X \mid Z)\Big] - \mathbb{E}_{q(X)}\Big[\mathbb{D}_{\mathrm{KL}}(q(Z \mid X) \,\|\, p(Z))\Big].$$

So the final loss is

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

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

- $q(Z \mid X = x) \sim \mathcal{N}(\mu^f(x), \operatorname{diag}(\sigma^f(x)))$
- $p(X \mid Z = z) \sim \mathcal{N}(\mu^g(z), \operatorname{diag}(\sigma^g(z)))$.

The first term of $\mathscr{L}$ is the average of

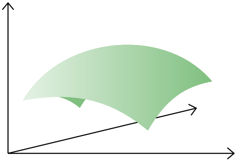
$$\mathbb{D}_{\mathrm{KL}}\Big(\underbrace{q(Z \mid X = x)}_{\mathscr{N}(\mu^f(x),\sigma^f(x))} \,\|\, \underbrace{p(Z)}_{\mathscr{N}(0,I)}\Big) = -\frac{1}{2}\sum_d \left(1 + 2\log\sigma_d^f(x) - \left(\mu_d^f(x)\right)^2 - \left(\sigma_d^f(x)\right)^2\right).$$

over the x_ns.

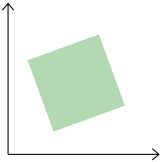
The first term of $\mathscr{L}$ is the average of

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

over the x_ns.



Original space $\mathscr{X}$

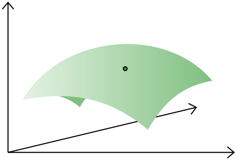


Latent space $\mathscr{F}$

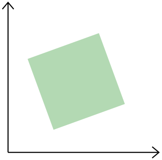
The first term of $\mathscr{L}$ is the average of

$$\mathbb{D}_{\mathrm{KL}}\big(\underbrace{q(Z \mid X = x)}_{\mathscr{N}(\mu^f(x), \sigma^f(x))} \,\|\, \underbrace{p(Z)}_{\mathscr{N}(0,I)}\big) = -\frac{1}{2}\sum_d \left(1 + 2\log\sigma_d^f(x) - \left(\mu_d^f(x)\right)^2 - \left(\sigma_d^f(x)\right)^2\right).$$

over the x_ns.



Original space $\mathscr{X}$

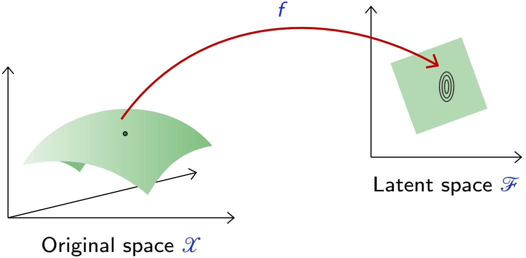


Latent space $\mathscr{F}$

The first term of $\mathscr{L}$ is the average of

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

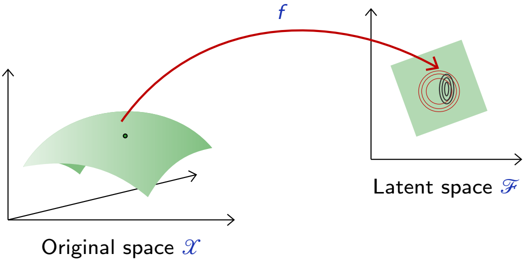
over the x_ns.



The first term of $\mathscr{L}$ is the average of

$$\mathbb{D}_{\mathsf{KL}}\big(\underbrace{q(Z \mid X = x)}_{\mathscr{N}(\mu^f(x),\sigma^f(x))} \,\|\, \underbrace{p(Z)}_{\mathscr{N}(0,I)}\big) = -\frac{1}{2}\sum_d \left(1 + 2\log\sigma_d^f(x) - \left(\mu_d^f(x)\right)^2 - \left(\sigma_d^f(x)\right)^2\right).$$

over the x_ns.



The first term of $\mathscr{L}$ is the average of

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

over the x_ns.

This can be implemented as

```
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 $\mathscr{L}$ 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.

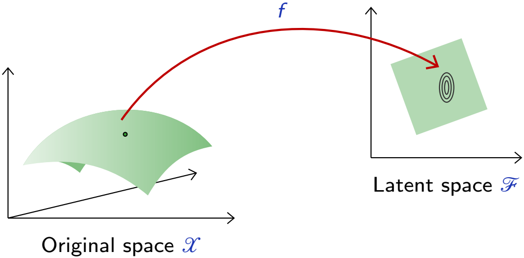
$$z_n \sim \mathscr{N}\left(\mu^f(x_n), \sigma^f(x_n)\right), \; n = 1, \dots, N.$$

As Kingma and Welling (2013), we use a constant variance of 1 for the decoder, so the second term of $\mathscr{L}$ becomes the average of

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

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

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

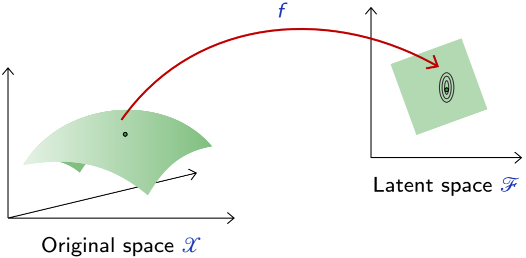


As Kingma and Welling (2013), we use a constant variance of 1 for the decoder, so the second term of $\mathscr{L}$ becomes the average of

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

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

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

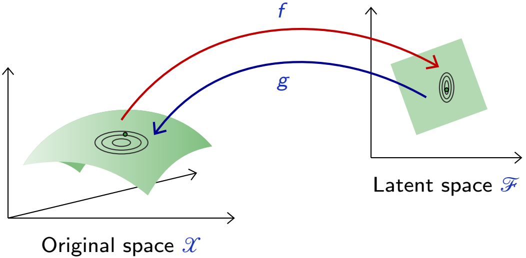


As Kingma and Welling (2013), we use a constant variance of 1 for the decoder, so the second term of $\mathscr{L}$ becomes the average of

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

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

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



As Kingma and Welling (2013), we use a constant variance of 1 for the decoder, so the second term of $\mathscr{L}$ 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.

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

This can be implemented as

```
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

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

We had for the standard autoencoder

```
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

```
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

```
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

```
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
```

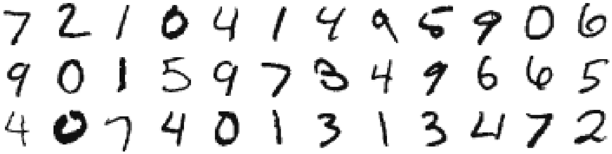
During inference we do not sample, and instead use μ^f and μ^g as prediction.

Note in particular the **re-parameterization trick**:

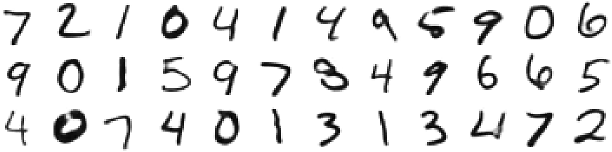
```
z = torch.randn_like(mu_f) * std_f + mu_f
output = model.decode(z)
```

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

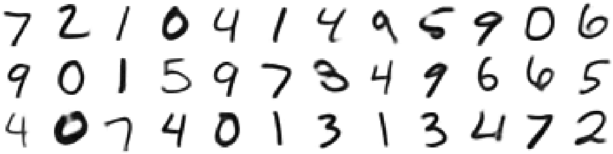
Original



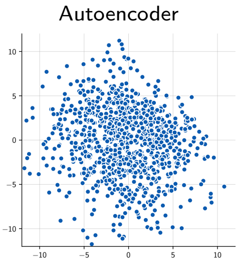
Autoencoder reconstruction ($d = 32$)



Variational Autoencoder reconstruction ($d = 32$)

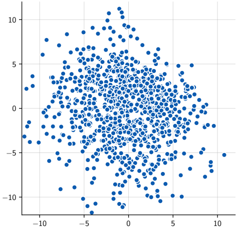


We can look at two latent features to check that they are Normal for the VAE.

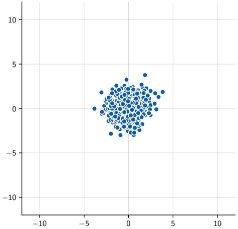


We can look at two latent features to check that they are Normal for the VAE.

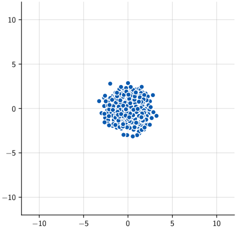
Autoencoder



Variational autoencoder

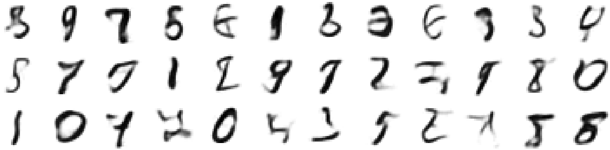


$\mathscr{N}(0,1)$



Autoencoder sampling ($d = 32$)

Variational Autoencoder sampling ($d = 32$)

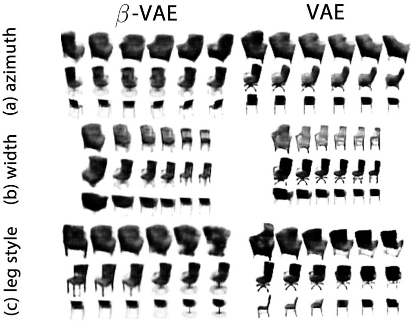


Making the embedding $\sim \mathscr{N}(0,1)$, often results in "disentangled" representations.

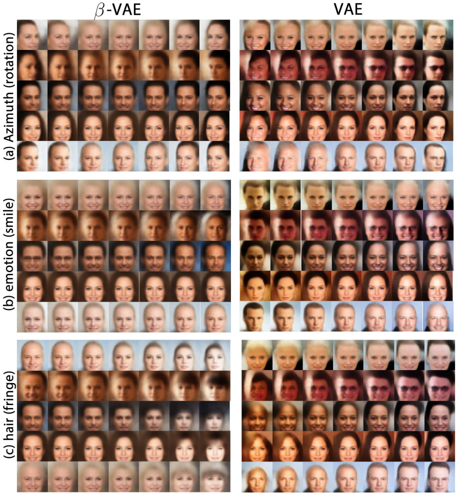
This effect can be reinforced with a greater weight of the KL term

$$\mathscr{L} = \beta\, \mathbb{E}_{q(X)}\Big[\mathbb{D}_{\mathsf{KL}}(q(Z \mid X) \parallel p(Z))\Big] - \mathbb{E}_{q(X,Z)}\Big[\log p(X \mid Z)\Big],$$

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



(Higgins et al., 2017)



(Higgins et al., 2017)

The end

References

I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. **beta-vae: Learning basic visual concepts with a constrained variational framework**. In International Conference on Learning Representations (ICLR), 2017.

D. P. Kingma and M. Welling. **Auto-encoding variational bayes**. CoRR, abs/1312.6114, 2013.