# Multivariate Statistics – Autoencoders

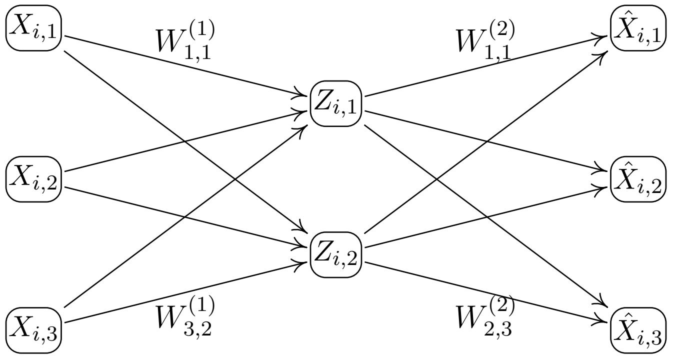
Suppose we have a data matrix $X \in \mathbb{R}^{n\times p}$ and would like to approximate it by some $\hat{X} \in \mathbb{R}^{n\times p}$ by first "compressing" every observation $X_i$ into a hidden state $Z_i \in \mathbb{R}^q$ (with typically $q \ll p$) and then "de-compressing" again to get $\hat{X}_i \in \mathbb{R}^p$ which is hopefully similar to the original $X_i$ for $i \in \{1,\dots,n\}$.

The simplest architecture for this is linear in the sense that for weight matrices $W^{(1)} \in \mathbb{R}^{p\times q}$ and $W^{(2)} \in \mathbb{R}^{q\times p}$,

$$\begin{aligned} Z_{i,\ell} &= \sum_k W^{(1)}_{k,\ell} X_{i,k} && i = 1,\dots,n \text{ and } j = 1,\dots,q \\ \hat{X}_{i,k} &= \sum_\ell W^{(2)}_{\ell,k} Z_{i,\ell} && i = 1,\dots,n \text{ and } k = 1,\dots,p \end{aligned}$$

Or, in matrix notation,

$$\begin{aligned} Z &= XW^{(1)} \\ \hat{X} &= ZW^{(2)} \end{aligned}$$



The goal is now to find $W^{(1)}$ and $W^{(2)}$ as to minimize the Euclidean distance between $X$ and $\hat{X}$,

$$\operatorname{argmin}_{W^{(1)},W^{(2)}} \|X - \hat{X}\|_2^2.$$

**Non-uniqueness.** The problem is clearly not well posed since we can replace, for any solution, with any $c > 0$

$$\begin{aligned} W^{(1)} &\leftarrow c \cdot W^{(1)} \\ W^{(2)} &\leftarrow \frac{1}{c} \cdot W^{(2)} \end{aligned}$$

and the objective $\|X - \hat{X}\|_2^2$ will remain unchanged. We can hence wlog demand that $\mathrm{Cov}(Z) = 1_{q\times 1}$ and that the variables in $Z$ have mean zero (in which case $\mathrm{Cov}(Z) = Z^t Z$). Even then the solution will not be unique in general, as we can for example permute the entries of $Z$ and the corresponding weights to get the same approximation $\hat{X}$.

**A solution.** Let $X = UDV^t$ be the SVD of $X$. Since we have

$$\hat{X} = XW^{(1)}W^{(2)},$$

the approximation error can be written (using orthogonality of $U$ and $V$) as

$$\begin{aligned}\|X - \hat{X}\|_2^2 &= \|UDV^t - UDV^t W^{(1)}W^{(2)}\|_2^2 \\ &= \|D - D(V^t W^{(1)} W^{(2)} V)\|_2^2\end{aligned}$$

Note that

$$\mathrm{rank}(W^{(1)}W^{(2)}) \le q,$$

and

$$\mathrm{argmin}_{K\in\mathbb{R}^{p\times p}:\mathrm{rank}(K)\le q}\|D - DK\|_2^2 = \underbrace{\begin{pmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & & 0 & & \\ & & & & 0 & \\ & & & & & 0 \end{pmatrix}}_{\text{first } q \text{ are nonzero}},$$

always assuming that the singular values in $D$ are monotonically decreasing. This can be achieved by selecting

$$\begin{aligned} W^{(1)} &= \tilde{V} \\ W^{(2)} &= \tilde{V}^t, \end{aligned} \tag{1}$$

where $\tilde{V} \in \mathbb{R}^{p\times q}$ is the matrix $V$ restricted to the first $q$ columns. So the PCA solution is optimizing this auto-encoding problem.

**Enforcing the covariance constraint.** However, we have not yet enforced $\mathrm{Cov}(Z) = 1_{q\times q}$. In order to do so, note that (using mean-centered columns of $X$ as an assumption),

$$\mathrm{Cov}(Z) = \mathrm{Cov}(UDVW^{(1)}) = (W^{(1)})^t V D^t D V W^{(1)},$$

and we can hence choose

$$\begin{aligned} W^{(1)} &= \tilde{V}\tilde{D}^{-1} \\ W^{(2)} &= \tilde{D}\tilde{V}^t, \end{aligned} \tag{2}$$

where $\tilde{D}$ is $D$ restricted to the first $q$ rows and columns. Solution (2) achieves the identical objective $\|X - \hat{X}\|_2$ as solution (1), but additionally ensures that $\mathrm{Cov}(Z) = 1_{q\times q}$.

**Nonlinear autoencoders.** Let in the following $X \in \mathbb{R}^p$ be the original random variable, $Z \in \mathbb{R}^p$ the corresponding latent variable and $\hat{X} \in \mathbb{R}^p$ the transformed data.

In general, we can have any nonlinear 'encoding' function $f$ with

$$Z = f(X)$$

and 'decoding' function $g$ with

$$\hat{X} = g(Z).$$

The samples are called $x_i$, $i = 1, \ldots, n$ and likewise for the latent variables. The basic version of an auto-encoder is trying to minimize

$$\text{argmin}_{f \in \mathcal{F}, g \in \mathcal{G}} \sum_{i=1}^{n} \|x_i - g(f(x_i))\|,$$

where again a constraint on the distribution of the latent samples $z_i$ can be enforced to address the non-uniqueness issue that stems from scaling the latent variables in an arbitrary way.

The function classes $\mathcal{F}$ and $\mathcal{G}$ are often chosen as parameterized by a multi-layer convolutional neural network in case of images.

**Variational Auto-encoders** In a latent variable model, we assume that the latents follow an a-priori distribution

$$Z \sim \Phi,$$

where $\Phi$ is often chosen to be the standard Normal distribution in $q$ dimensions. The distribution

$$X|Z = z$$

is in general unknown and we try to approximate it by choosing a "good" decoder $g$ that maps the latents to the observed space. While there is a full Bayesian interpretation (where $z$ also includes parameters of the model etc., here we would like to choose the 'decoder' $g$ to maximize the likelihood given observations $x = (x_1, \ldots, x_n)$ with $x_i \in \mathbb{R}^p$ for all $i$, as

$$\log p_g(x) = \sum_{i=1}^{n} \log p_g(x_i).$$

The distribution of $X$ can be obtained by marginalizing out the latent variable $Z$,

$$p(x) = \int p(x, z)dz = \int p(x|z)\phi(z)dz,$$

where $\phi$ is the chosen prior distribution for the latent variables, usually a standard normal distribution in $q$ dimensions. Now, the true distribution $p(x|z)$ is unknown in this latent variable model. If we parametrize $p(x|z)$ by $g$, we obtain

$$p_g(x) = \int p_g(x|z)\phi(z)dz.$$

Computing the integral is in general infeasible. The strategy will be to obtain a lower bound on $p_g(x)$ and optimize this lower bound with respect to $g$. The encoder $f$ will determine how tight the bound is and $f$ will be optimized to make the bound as tight as possible.

The variational approximation now uses the 'encoder' $f$ to obtain a close approximation to the posterior $p(z|x)$ in the following sense. We can think of $f$ and $g$ as parameters that determine for example the mean and variance of a chosen class of distributions. Given input $X = x$, we then have a conditional distribution over the latents as

$$Z|X = x \quad \sim \quad Q_{f,x}.$$

Assume the density exist and is denoted by $q_{f,x}$. The encoding function $f$ is now chosen to minimize the distance to the true, unknown posterior $p_g(z|x)$.

To measure the distance between the two distributions, the Kullback-Leibler divergence (KL-divergence) is typically used. For two densities $p_1$ and $p_2$ with respect to measure $\mu$ is given by

$$KL(p_1||p_2) = \int \log \frac{p_1(x)}{p_2(x)} p_1(x) d\mu(x).$$

The KL-divergence is in general nonnegative and strictly positive whenever the distributions are different (and zero if identical), but is also asymmetric in $p_1$ and $p_2$.

First, let $D_{KL}$ be the KL divergence. Then

$$\log p_g(x) = D_{KL}(q_f(z|x)||p_g(z|x)) + elbo(f, g, x),$$

where the evidence lower-bound *elbo* is given by

$$elbo(f, g, x) = E_{Z\sim Q_{f,x}}[-\log q_f(z|x) + \log p_g(x, z)]$$

To see the fomer equality, note that

$$\begin{aligned} D_{KL}(q_f(z|x)||p_g(z|x)) &= E_{Z\sim Q_{f,x}}[\log q_f(Z|x)] - E_{Z\sim Q_{f,x}}[\log p_g(Z|x)] \\ &= E_{Z\sim Q_{f,x}}[\log q_f(Z|x)] - E_{Z\sim Q_{f,x}}[\log p_g(Z, x)] + \log p_g(x) \\ &= -elbo(f, g, x) + \log p_g(x). \end{aligned}$$

Note that the KL-divergence is nonnegative and hence [1]

$$\log p_g(x) \geq elbo(f, g, x)$$

For a fixed $f$, maximizing the *elbo* is hence maximizing a lower-bound on the likelihood since the KL-divergence is nonnegative. The setup is similar to the EM algorithm, where the E-step computes a lower-bound on the likelihood and the M-step is then exploiting this lower bound (by choosing the next update as the maximum of the lower-bound).

The evidence lower-bound $elbo(f, g, x)$ one is trying to maximize can be rewritten as

$$\begin{aligned} elbo(f, g, x) &= E_{Z \sim Q_{f,x}}[\log p_g(x, z) - \log q_f(z|x)] \\ &= E_{Z \sim Q_{f,x}}[\log p_g(x|z)\phi(z) - \log q_f(z|x)] \\ &= E_{Z \sim Q_{f,x}}[\log p_g(x|z) + \log \phi(z) - \log q_f(z|x)] \\ &= E_{Z \sim Q_{f,x}}[\log p_g(x|z)] - D_{KL}(q_f(z|x)||\phi(z))]. \end{aligned}$$

As we would like to maximize the likelihood

$$\log p_g(x) = \log \prod_{i=1}^{n} p_g(x_i) = \sum_{i=1}^{n} \log p_g(x_i),$$

we are summing over all samples by maximizing $\sum_{i=1}^{n} elbo(f, g, x_i)$. The first term

$$\sum_{i=1}^{n} E_{Z \sim Q_{f,x_i}}[\log p_g(x_i|z)]$$

corresponds to the approximation error by checking how well the decoder $g$ approximates the samples. For a deterministic function $f$, can be replaced with standard approximation error

$$\sum_{i=1}^{n} E_{Z \sim Q_{f,x_i}}[\|x_i - Z\|_2^2]$$

[1]The inequality can also be shown directly via Jensens inequality using the fact that log is a concave function. For any concave function $w$ and random variable $X$ (with suitable support) it holds that $w(E(X)) \geq E(w(X))$. Hence, for any $q(z)$ with positive support,

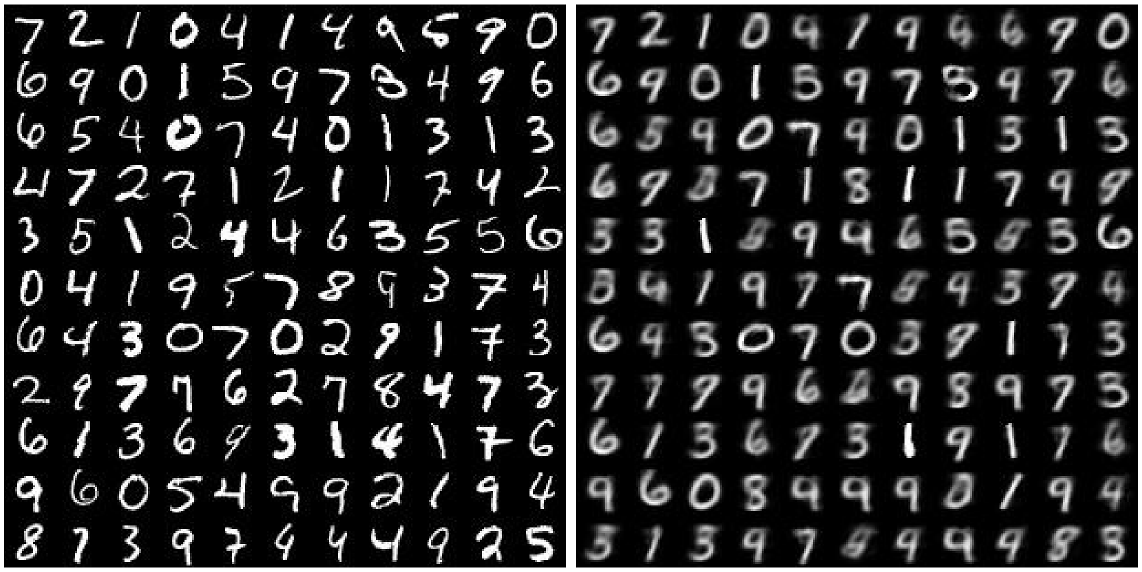
$$\begin{aligned} \log p_g(x) &= \log \int p_g(x, z) dz \\ &= \log \int p_g(x, z) \frac{q(z)}{q(z)} dz \\ &= \log \Big[ E_{Z \sim Q} \frac{p_g(x, Z)}{q(Z)} \Big] \\ &\geq E_{Z \sim Q} \log[p_g(x, Z)] - E_{Z \sim Q}[\log q(Z)] \end{aligned}$$

In the elbo above, the distribution $Q = Q_{f,x}$ depends on both $f$ and $x$.

The latter term $-\sum_{i=1}^{n} D_{KL}(q_f(z|x)||\phi(z))$ is trying to ensure intuitively that the distribution over the latents is as close to the prior $\phi$ as possible.

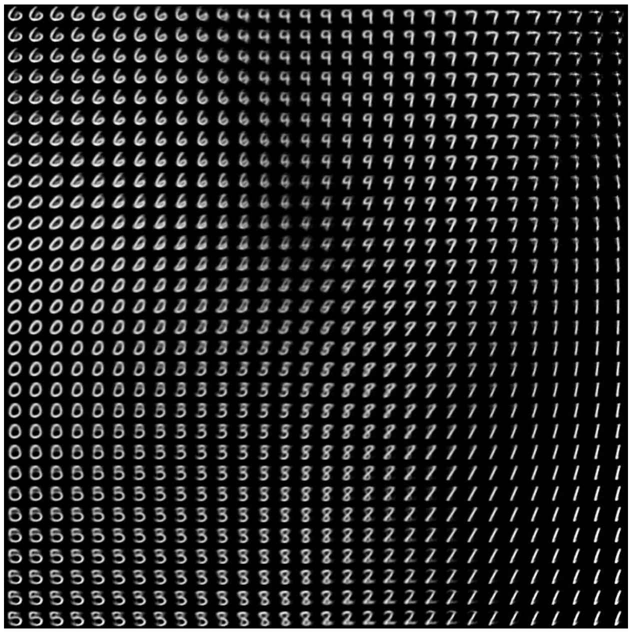
If you are interested in further reading, try for example `https://arxiv.org/pdf/1312.6114.pdf` (original paper on VAE), `https://arxiv.org/pdf/1601.00670.pdf` (overview of variational inference) and `https://arxiv.org/pdf/1606.05908.pdf` (variational infercne for VAEs).

Below is an image showing both original images (left) and reconstructions with a variational auto-encoder (right)



The images are a bit blurry and some contain "mistakes" which can be mimimized by tuning of the parameters.

The next image shows images according to their position on a 2-dimensional latent space $Z$. That is you can get a regular grid in the 2-dimensional latent space and then generate at each grid point $z$ the image $g(z)$ and show the resulting set of images on the rectangular grid.



If you want to try yourself there is a Tensorflow tutorial/code example at
`https://www.tensorflow.org/alpha/tutorials/generative/cvae`.