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

The \textbf{Variational Autoencoder} proposed by Kingma and Welling (2013) relies on a tractable approximation of this log-likelihood.

Note that their framework involves \textit{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
\[
\mathbb{E}_{q(X)} \left[ \log p(X) \right],
\]
and we can show that
\[
-\mathbb{E}_{q(X)} \left[ \log p(X) \right] \leq \mathbb{E}_{q(X)} \left[ D_{KL}(q(Z \mid X) \parallel p(Z)) \right] - \mathbb{E}_{q(X,Z)} \left[ \log p(X \mid Z) \right].
\]
So it makes sense to minimize this latter quantity.

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

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

over the $x_n$s.

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

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

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

\[
z_n \sim \mathcal{N}(\mu^f(x_n), \sigma^f(x_n)), \quad n = 1, \ldots, N.
\]

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

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

and putting everything together we get for the VAE

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

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

Note in particular the re-parameterization trick:

\[
\begin{align*}
    z &= \text{torch.empty_like(mu}_f).\text{normal()} \times \text{std}_f + \text{mu}_f \\
    \text{output} &= \text{model.decode(z)}
\end{align*}
\]

Implementing the sampling of $z$ that way allows to compute the gradient w.r.t $f$’s parameters without any particular property of $\text{normal}()$. 
We can look at two latent features to check that they are Normal for the VAE.
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

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

resulting in the \(\beta\)-VAE proposed by Higgins et al. (2017).
We propose augmenting the original VAE framework with a single hyperparameter $\beta$ with permission.

All models apart from VAE learnt to disentangle the labelled data generative factor, azimuth (a). Only $\beta$ (b). Only achieves state of the art disentangling performance against both the best unsupervised (InfoGAN: (Higgins et al., 2017)) and faces (Paysan et al., 2009) using qualitative evaluation. Finally, to help quantify the differences, we develop a new measure of disentanglement and show that $\beta$ demonstrates both qualitatively and quantitatively that our protocol to quantitatively compare the degree of disentanglement learnt by different models; 3) we $\beta$ achieve $\beta$-VAE and DC-IGN (Kulkarni et al., 2015) transfer learning or zero-shot inference scenarios. Hence, while InfoGAN is an important step in the learning constraints applied to the model. These constraints impose a limit on the capacity of $\beta$ factors. $\beta$ modulation of the data, which is disentangled if the data contains at least some underlying factors of variation $\beta > 1$ (Higgins et al., 2017). With $\beta$-VAE learns an entangled representation (e.g. chair width is entangled with azimuth and leg style (b)). $\beta = 1$ corresponds to the original VAE framework (Kingma & Welling, 2014; Rezende et al., 2014), which brings scalability and training 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 the latent values in the VAE-based models, or a random sample of the noise variables in InfoGAN. $\beta$-VAE (Kingma & Welling, 2014) (Kingma & Welling, 2014) (Kingma & Welling, 2014) (Kingma & Welling, 2014) significantly improve $\beta$-VAE significantly outperforms all our baselines on this measure (ICA, PCA, VAE Kingma & Ba (2014), DC-IGN distribution and the number of the regularised noise variables. InfoGAN also lacks a principled $\beta$. We show that this simple modification allows $\beta$-VAE to significantly improve the degree of disentanglement in learnt latent representations compared to the unmodified VAE $\beta$-VAE approach achieves state-of-the-art disentanglement performance compared to various baselines on a variety of complex datasets.
References
