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 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

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

and we can show that

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

So it makes sense to minimize this latter quantity.

So the final loss is

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

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

$$D_{KL}(\mathcal{N}(\mu_f(x), \sigma_f(x)) \parallel \mathcal{N}(0, I)) = -\frac{1}{2} \sum_d \left( 1 + 2 \log \sigma_d^f(x) - \left( \mu_d^f(x) \right)^2 - \left( \sigma_d^f(x) \right)^2 \right).$$

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

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

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

$$z_n \sim \mathcal{N}\left(\mu_f(x_n), \sigma_f(x_n)\right), \quad n = 1, \ldots, N.$$ 

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

```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.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)

loss = kl_loss + fit_loss
```

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.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_()`.
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

$$
\mathcal{L} = \beta E_{q(\mathbf{X})}[D_{KL}(q(Z | X) \parallel p(Z))] - E_{q(\mathbf{X}, Z)}[\log p(X | Z)],
$$

resulting in the $\beta$-VAE proposed by Higgins et al. (2017).
Our main contributions are the following: 1) we propose a \( \beta \)-VAE approach to develop more human-like learning and reasoning in algorithms; 2) we devise a protocol to quantitatively compare the degree of disentanglement learnt by different models; 3) we demonstrate both qualitatively and quantitatively that our \( \beta \)-VAE approach achieves state-of-the-art disentangling performance against both the best unsupervised \( \beta \)-VAE (InfoGAN: Chen et al., 2016) framework to additionally maximise the mutual information between a representation of these factors. The reliance of InfoGAN on the GAN framework, however, comes with the risk of learning entangled representations (e.g. chair width is entangled with azimuth and leg style (b)).

\( \beta \)-VAE and VAE traversal is over the \([-1, 1]\) range. VAE always learns an entangled representation (e.g. chair width is entangled with azimuth and leg style (b)). During training of the \( \beta \)-VAE, a new unsupervised approach for learning disentangled representations of independent visual data generative factors, we propose a new \( \beta \)-VAE framework (Kingma & Welling, 2014; Rezende et al., 2014), which brings scalability and training efficiency.

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 (\( \beta = 1 \)) or the model is pushed to learn a more efficient latent representation \( \beta = 5 \). VAE traversal is over the \([-3, 3]\) range. \( \beta \)-VAE with \( \beta = 250 \) were also able to discover unlabelled factors in the dataset, such as chair width and style (a), emotion (b) and hair (c). The \( \beta \)-VAE significantly outperforms all our baselines on this measure (ICA, PCA, \( \beta \)-VAE Kingma & Ba (2014), DC-IGN Kulkarni et al., 2015), prompting the development of more elaborate semi-supervised \( \beta \)-VAE-based approaches for disentangled factor learning on a number of benchmark datasets, such as CelebA (Liu et al., 2015), chairs (Aubry et al., 2016) and faces (Paysan et al., 2009) using qualitative evaluation. Finally, to help quantify the degree of disentanglement achieved by different models or when optimising the hyperparameters of a single model. (Higgins et al., 2017)
References
