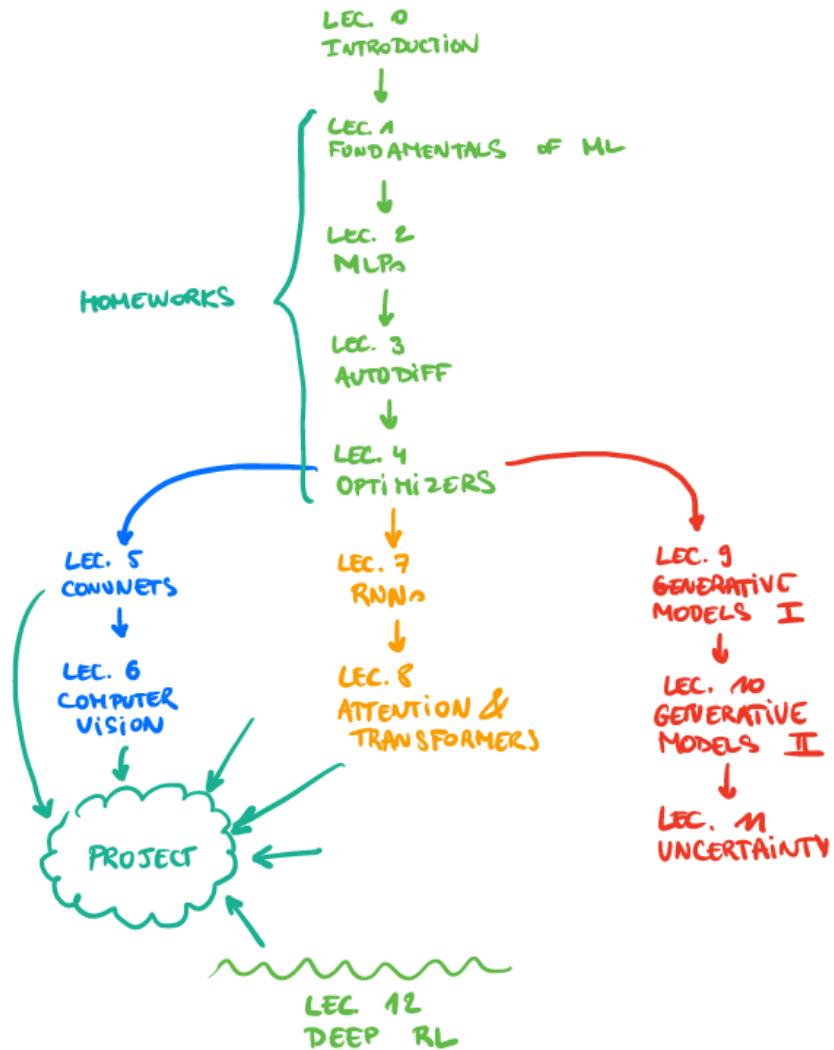


Deep Learning

Lecture 9: Auto-encoders and variational auto-encoders

Prof. Gilles Louppe
g.louppe@uliege.be





Today

Learn a model of the data.

- Generative models
- Auto-encoders
- Variational inference
- Variational auto-encoders

Generative models

Deep unsupervised learning

Deep unsupervised learning is about capturing rich patterns in raw data with deep networks in a **label-free** way.

- Generative models: recreate raw data distribution.
- Self-supervised learning: solve "puzzle" tasks that require semantic understanding.

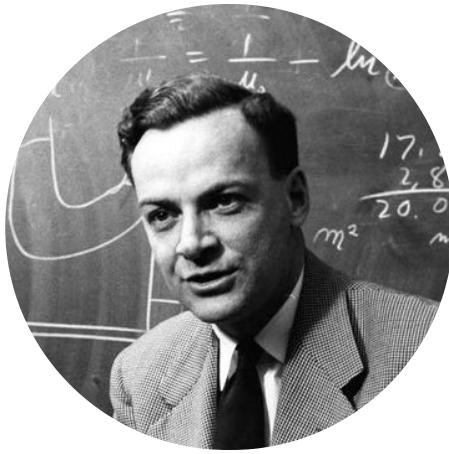
Generative models

A **generative model** is a probabilistic model p that can be used as a simulator of the data. Its purpose is to generate synthetic but realistic high-dimensional data

$$\mathbf{x} \sim p(\mathbf{x}; \theta),$$

that is as close as possible from the true but unknown data distribution $p(\mathbf{x})$, but for which we have empirical samples.

Why do we care?



What I cannot create, I do not understand.

Richard Feynman



"The brain has about 10^{14} synapses and we only live for about 10^9 seconds. So we have a lot more parameters than data. This motivates the idea that we must do a lot of unsupervised learning since the perceptual input (including proprioception) is the only place we can get 10^5 dimensions of constraint per second."

Geoffrey Hinton, 2014.



■ "Pure" Reinforcement Learning (cherry)

- ▶ The machine predicts a scalar reward given once in a while.
- ▶ **A few bits for some samples**

■ Supervised Learning (icing)

- ▶ The machine predicts a category or a few numbers for each input
- ▶ Predicting human-supplied data
- ▶ **10→10,000 bits per sample**

■ Unsupervised/Predictive Learning (cake)

- ▶ The machine predicts any part of its input for any observed part.
- ▶ Predicts future frames in videos
- ▶ **Millions of bits per sample**

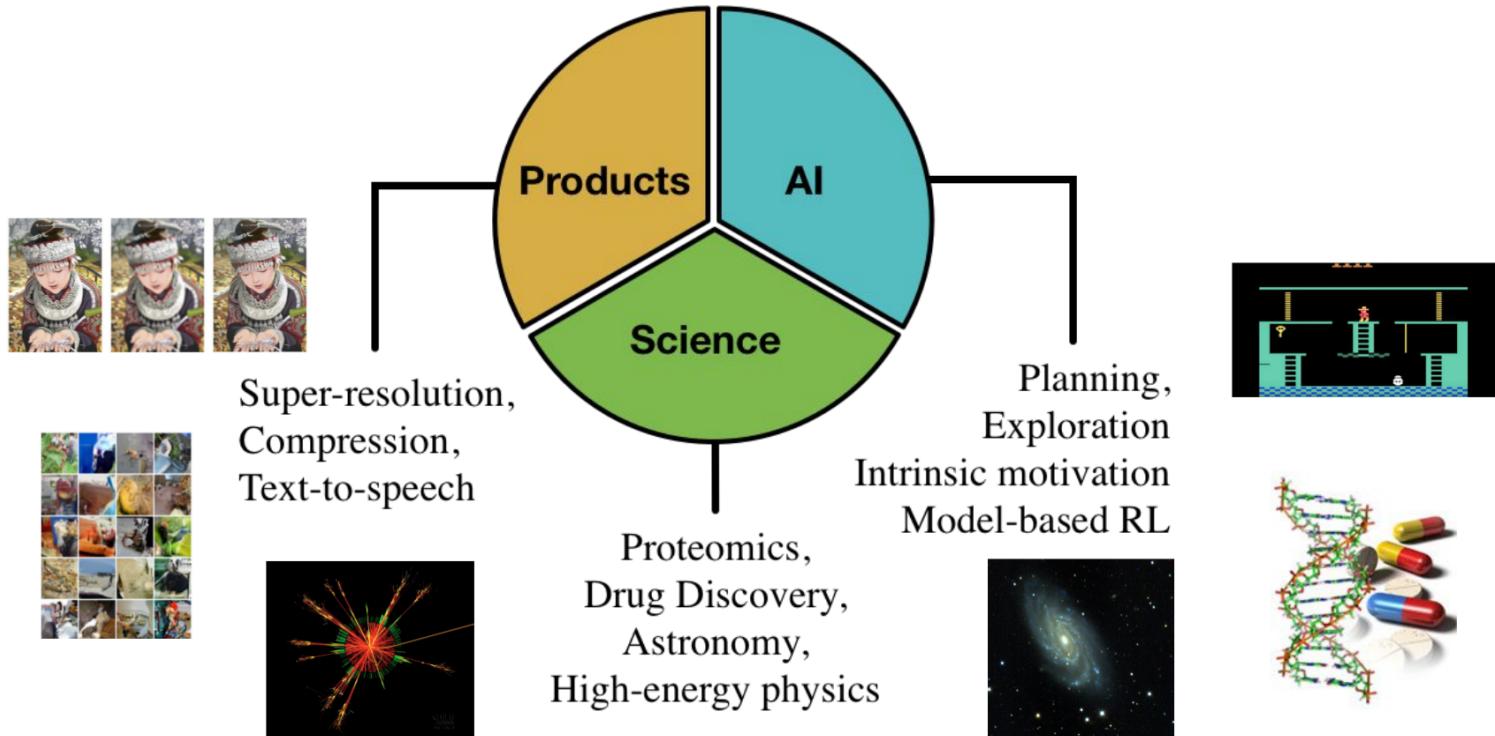
■ (Yes, I know, this picture is slightly offensive to RL folks. But I'll make it up)



"We need tremendous amount of information to build machines that have common sense and generalize."

Yann LeCun, 2016.

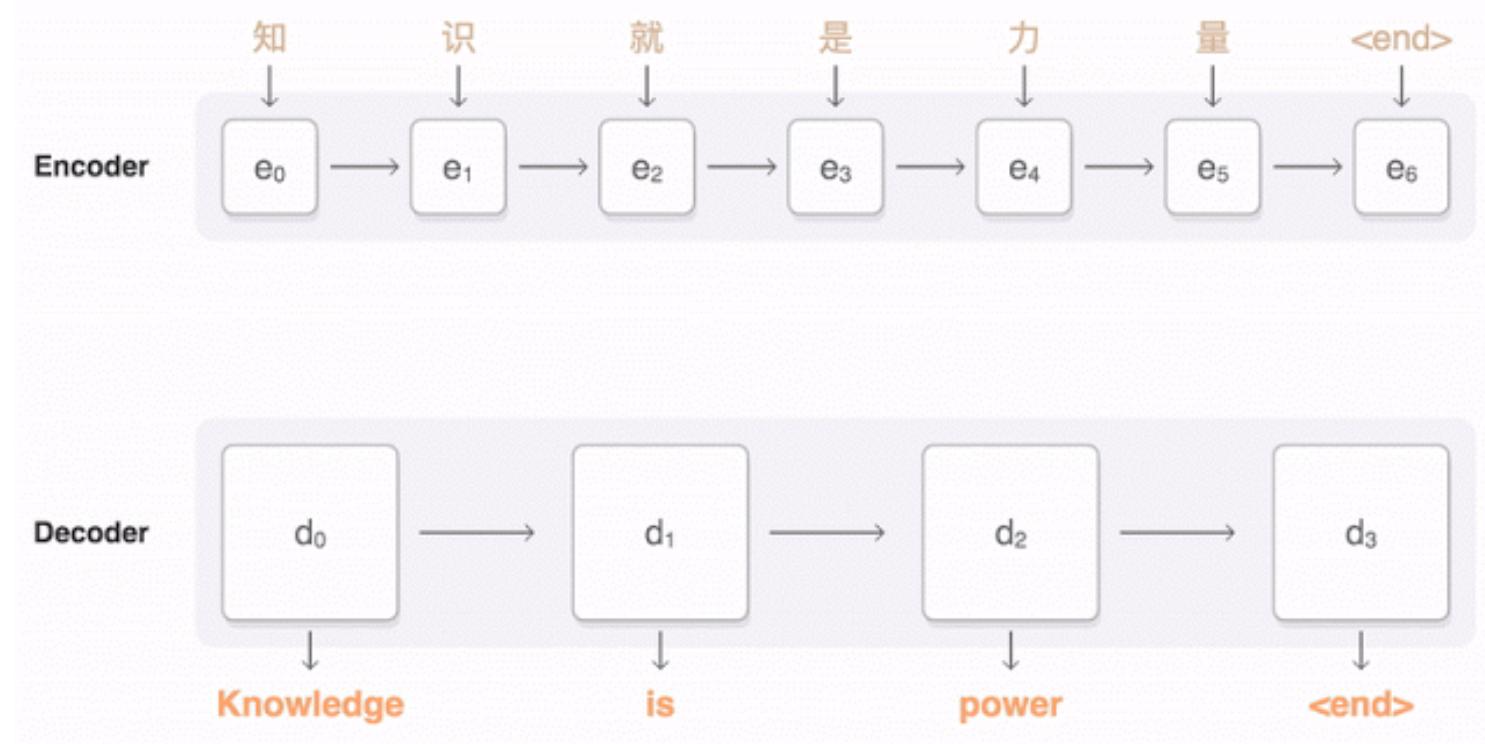
Immediate applications



Generative models have a role in many important problems

Machine translation

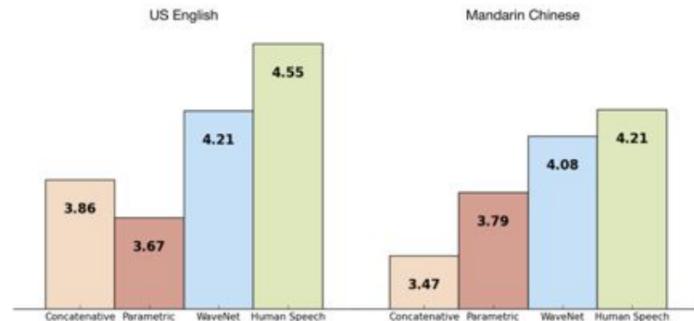
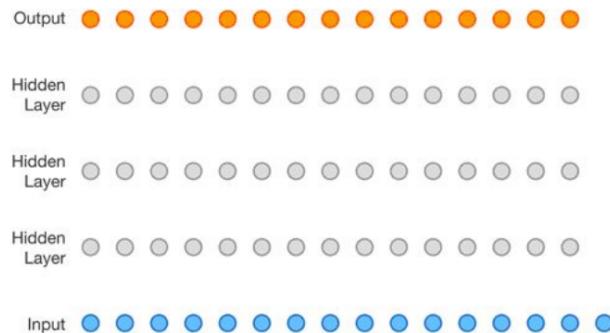
Attention-based sequence-to-sequence synthesis.



(Wu et al, 2016)

Text-to-speech synthesis

Generating audio conditioned on text.



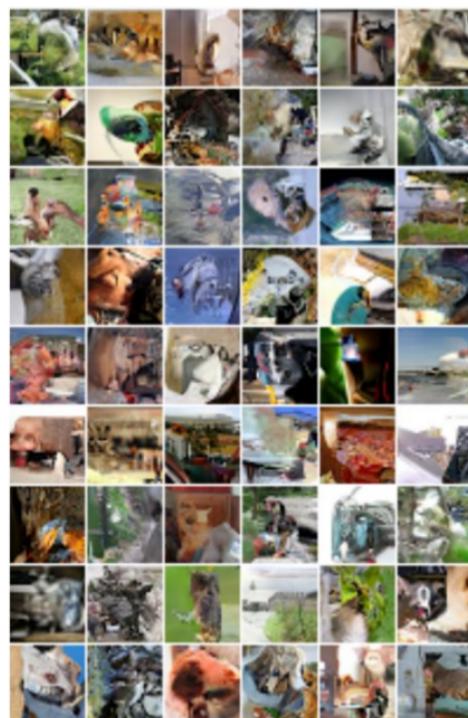
(Oord et al, 2016)

Image and content generation

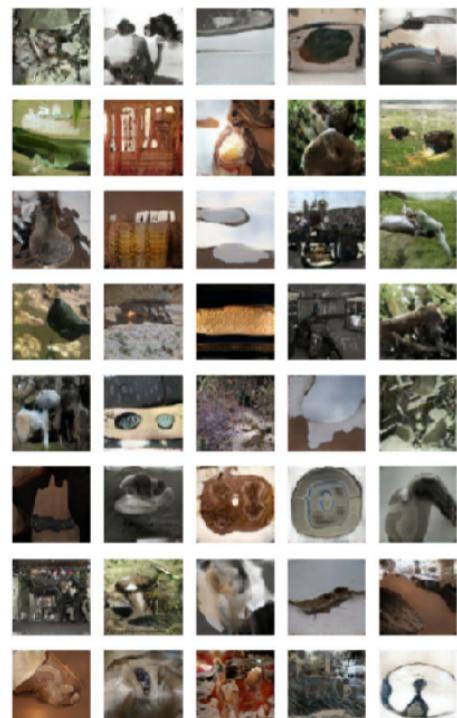
Generating images and video content.



DRAW



Pixel RNN



ALI

(Gregor et al, 2015; Oord et al, 2016; Dumoulin et al, 2016)

Communication and compression

Hierarchical compression of images and other data.

Original images



Compression rate: 0.2bits/dimension

JPEG



JPEG-2000



RVAE v1



RVAE v2

(Gregor et al, 2016)

Image super-resolution

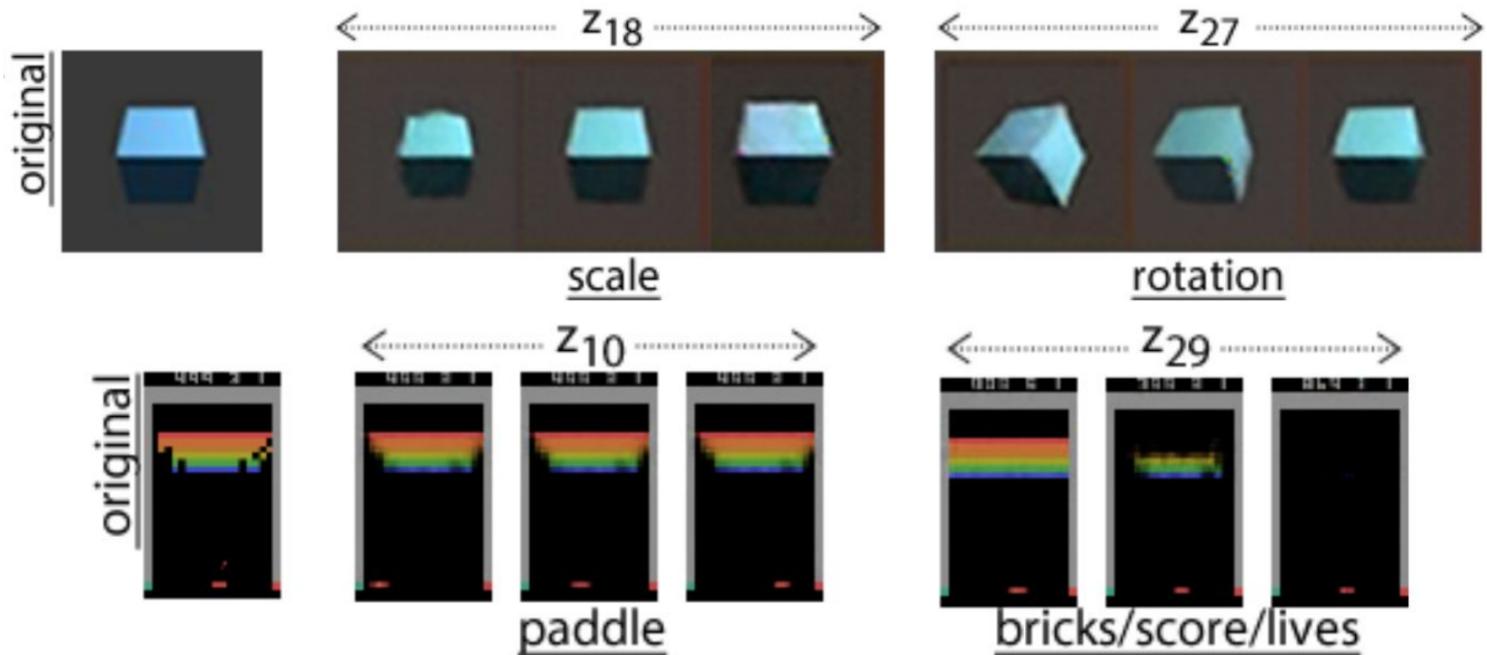
Photo-realistic single image super-resolution.



(Ledig et al, 2016)

Visual concept learning

Understanding the factors of variation and invariances.



(Higgins et al, 2017)

Future simulation

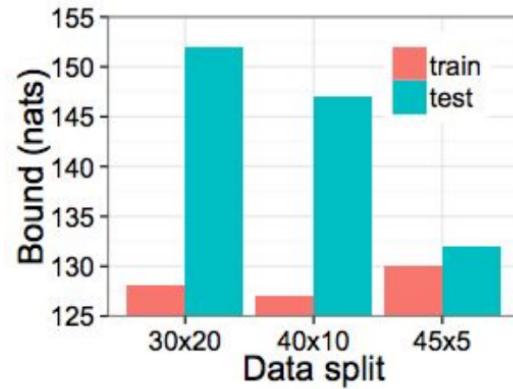
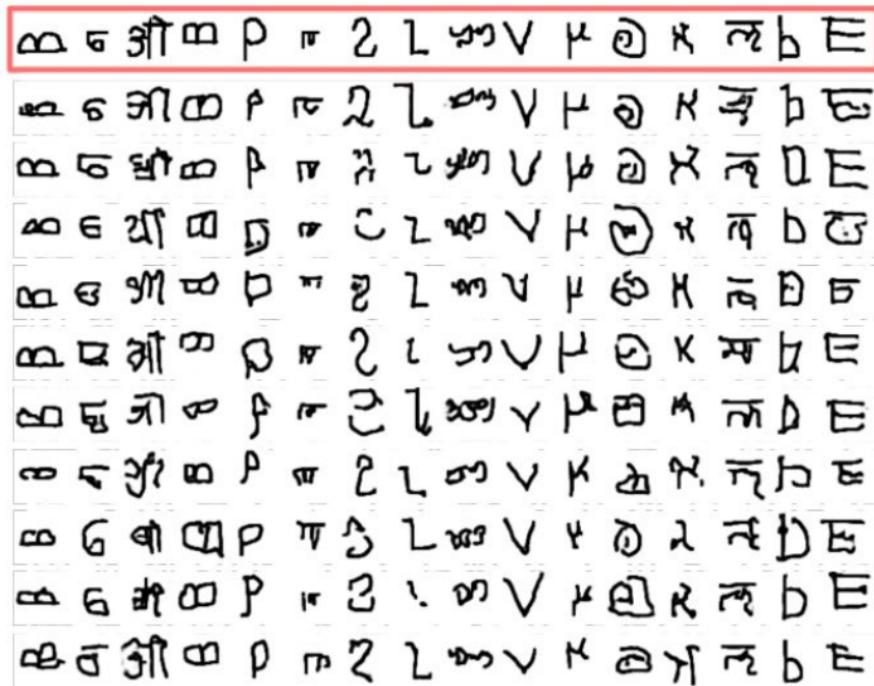
Simulate future trajectories of environments based on actions for planning.



(Finn et al, 2016)

One-shot generalization

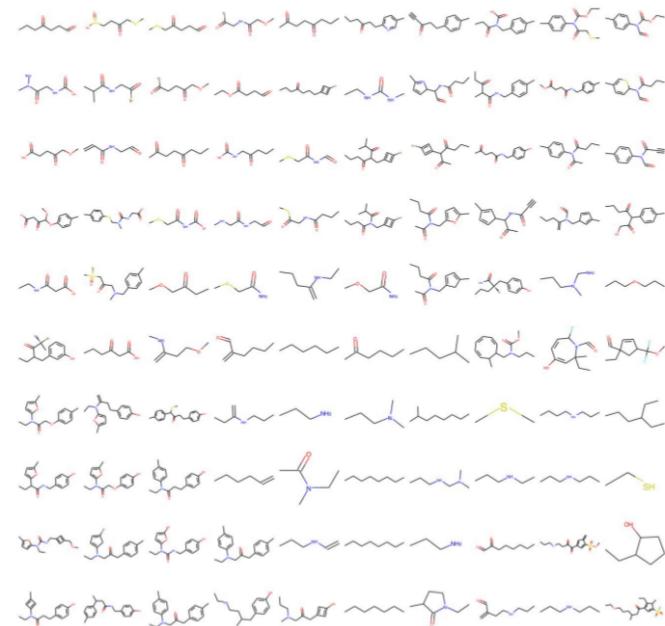
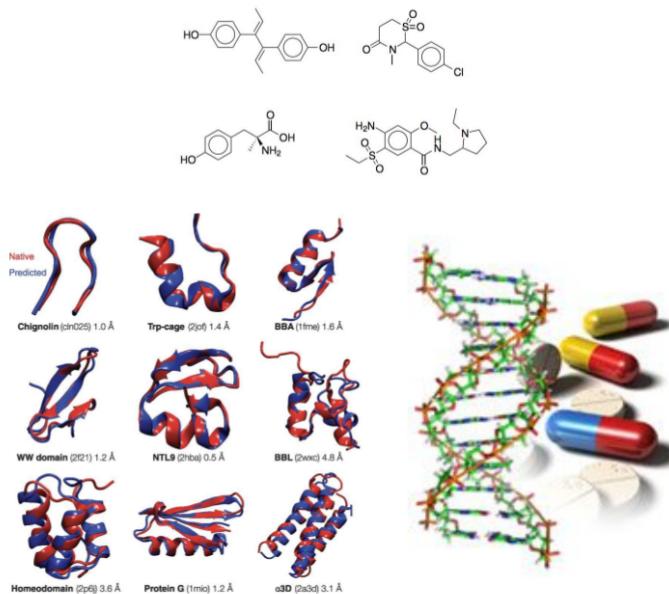
Rapid generalization of novel concepts.



(Gregor et al, 2016)

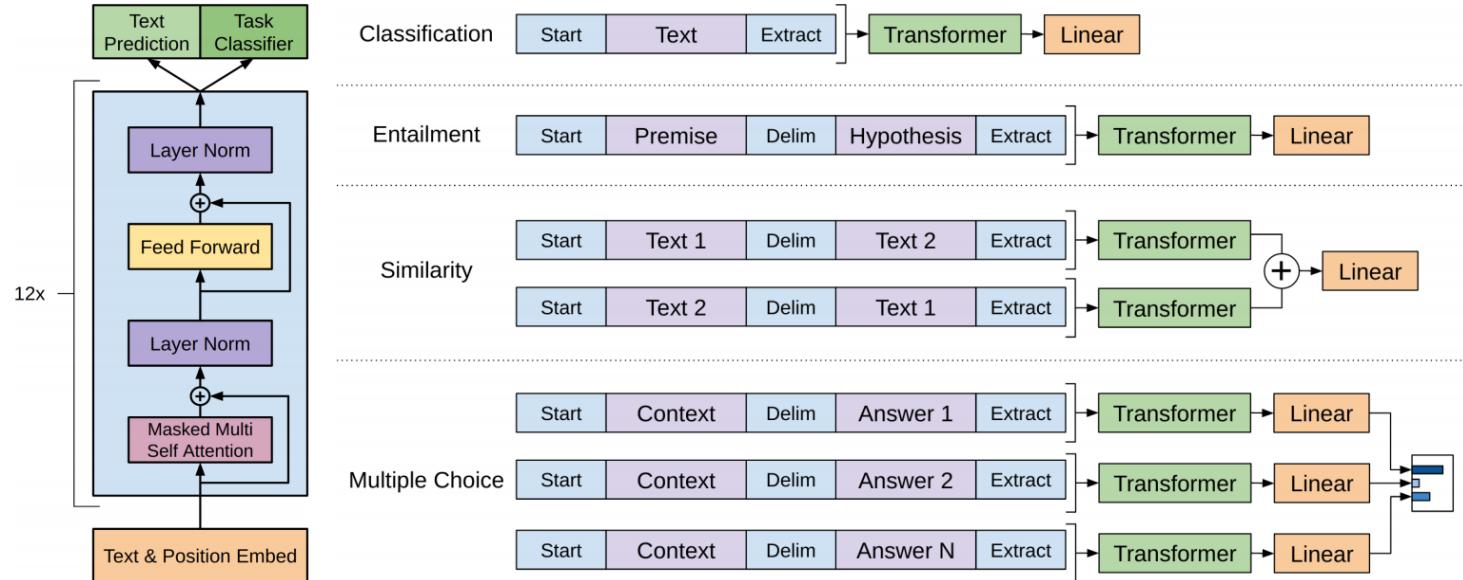
Drug design and response prediction

Generative models for proposing candidate molecules and for improving prediction through semi-supervised learning.



(Gomez-Bombarelli et al, 2016)

Improving downstream tasks with un(self)supervised pre-training

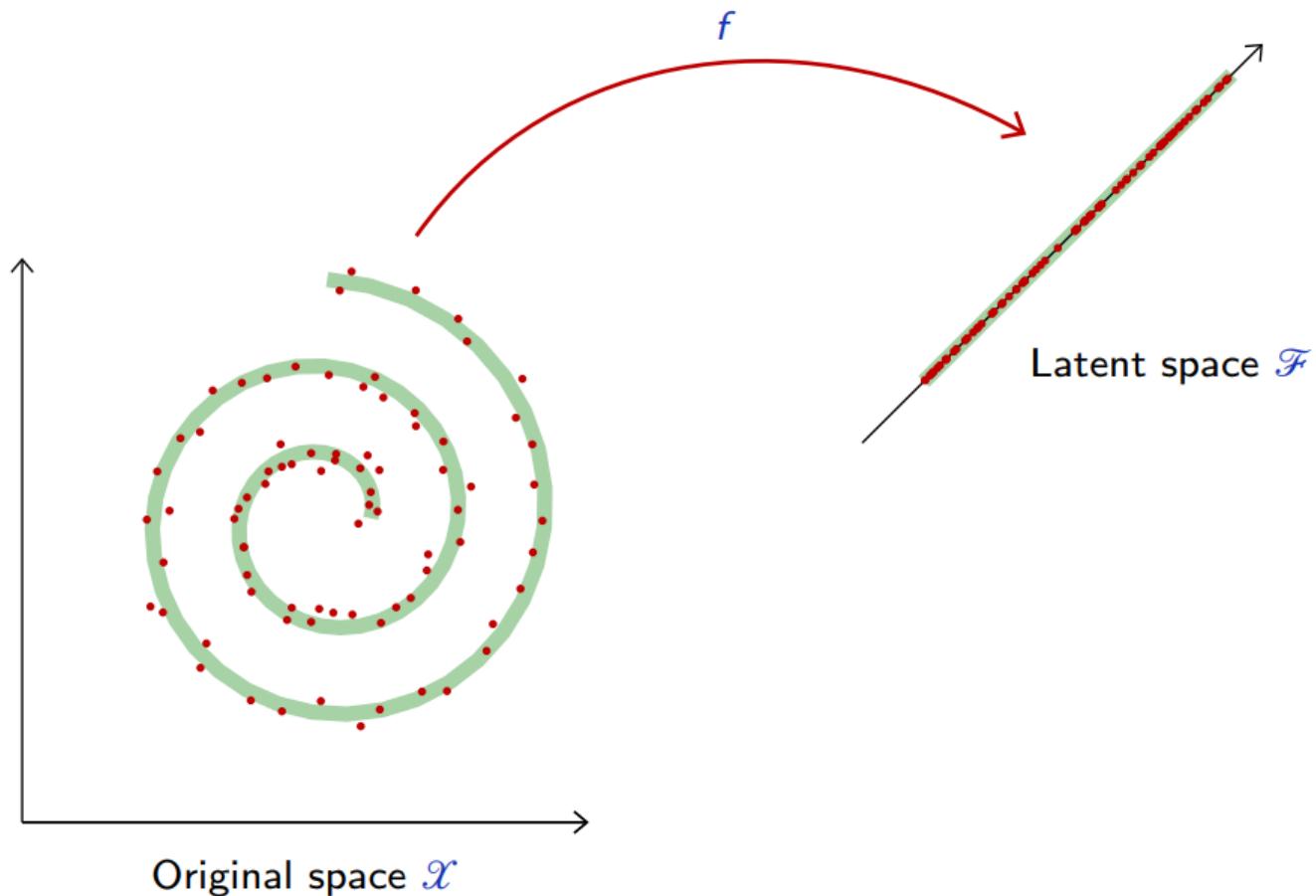


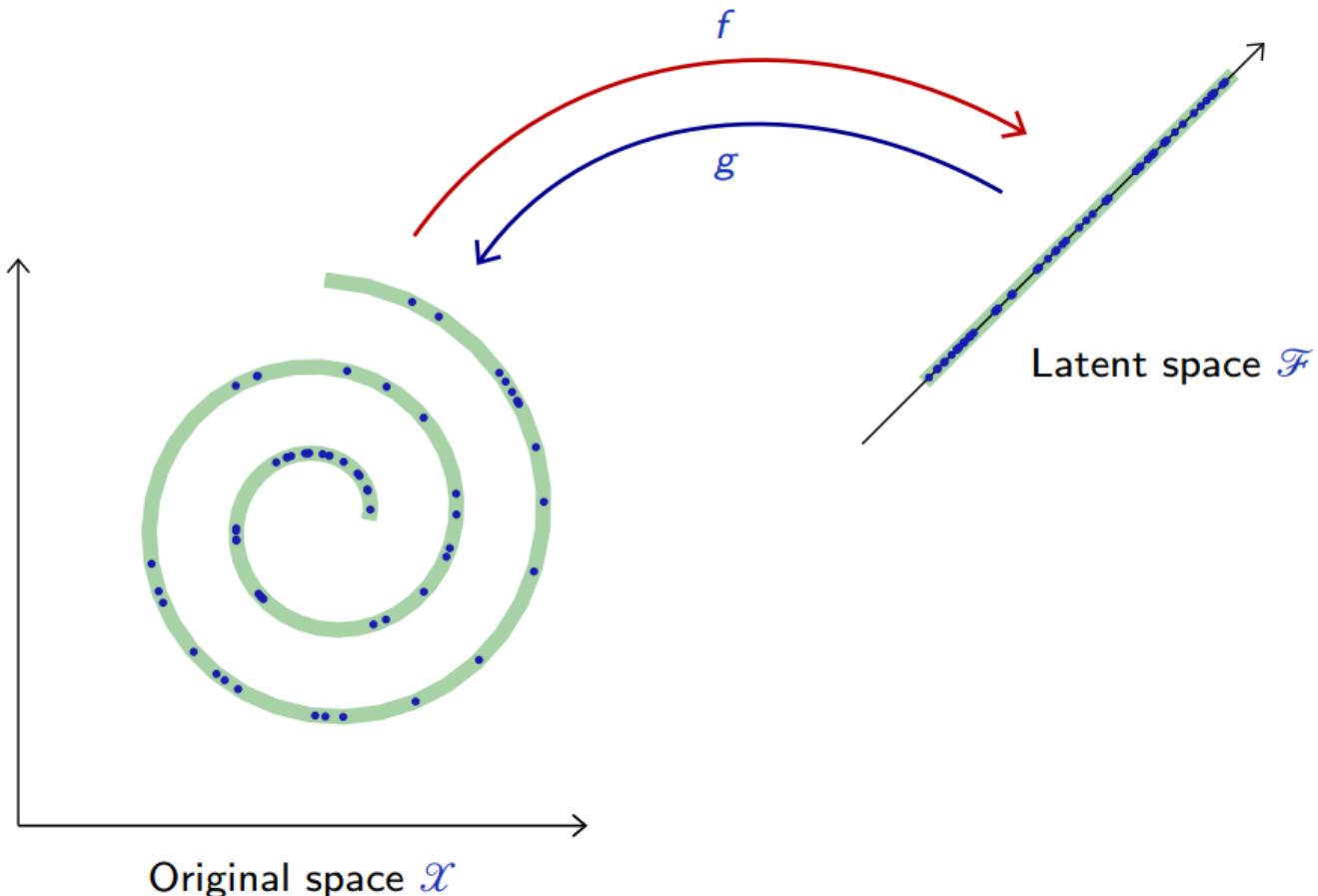
(Radford et al, 2018)

Auto-encoders

Many applications such as image synthesis, denoising, super-resolution, speech synthesis or compression, require to **go beyond** classification and regression and model explicitly a high-dimensional signal.

This modeling consists of finding "*meaningful degrees of freedom*", or "*factors of variations*", that describe the signal and are of lesser dimension.



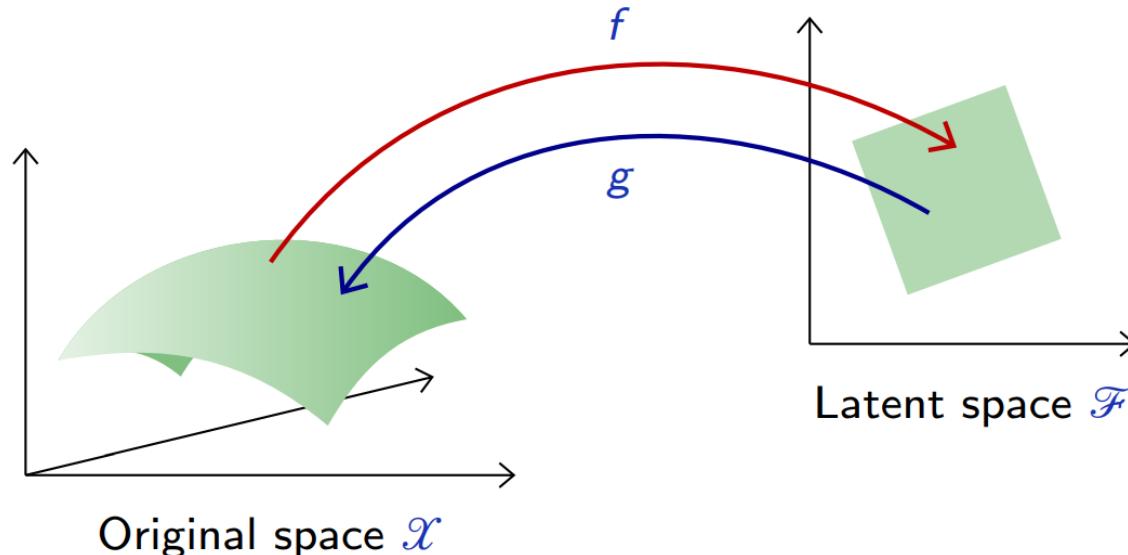


Auto-encoders

An auto-encoder is a composite function made of

- an **encoder** f from the original space \mathcal{X} to a latent space \mathcal{Z} ,
- a **decoder** g to map back to \mathcal{X} ,

such that $g \circ f$ is close to the identity on the data.



Let $p(\mathbf{x})$ be the data distribution over \mathcal{X} . A good auto-encoder could be characterized with the reconstruction loss

$$\mathbb{E}_{\mathbf{x} \sim p(\mathbf{x})} [||\mathbf{x} - g \circ f(\mathbf{x})||^2] \approx 0.$$

Given two parameterized mappings $f(\cdot; \theta_f)$ and $g(\cdot; \theta_g)$, training consists of minimizing an empirical estimate of that loss,

$$\theta = \arg \min_{\theta_f, \theta_g} \frac{1}{N} \sum_{i=1}^N ||\mathbf{x}_i - g(f(\mathbf{x}_i, \theta_f), \theta_g)||^2.$$

For example, when the auto-encoder is linear,

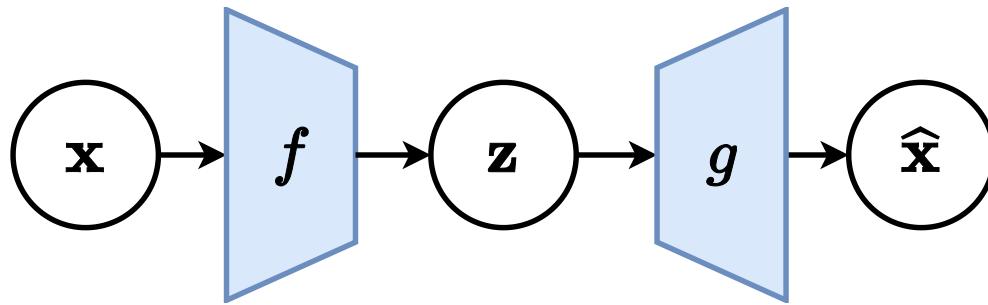
$$\begin{aligned}f &: \mathbf{z} = \mathbf{U}^T \mathbf{x} \\g &: \hat{\mathbf{x}} = \mathbf{U} \mathbf{z},\end{aligned}$$

with $\mathbf{U} \in \mathbb{R}^{p \times d}$, the reconstruction error reduces to

$$\mathbb{E}_{\mathbf{x} \sim p(\mathbf{x})} [||\mathbf{x} - \mathbf{U}\mathbf{U}^T \mathbf{x}||^2].$$

In this case, an optimal solution is given by PCA.

Deep auto-encoders



Better results can be achieved with more sophisticated classes of mappings than linear projections, in particular by designing f and g as deep neural networks.

For instance,

- by combining a multi-layer perceptron encoder $f : \mathbb{R}^p \rightarrow \mathbb{R}^d$ with a multi-layer perceptron decoder $g : \mathbb{R}^d \rightarrow \mathbb{R}^p$.
- by combining a convolutional network encoder $f : \mathbb{R}^{w \times h \times c} \rightarrow \mathbb{R}^d$ with a decoder $g : \mathbb{R}^d \rightarrow \mathbb{R}^{w \times h \times c}$ composed of the reciprocal transposed convolutional layers.

$\textcolor{blue}{X}$ (original samples)

7 2 1 0 4 1 4 9 5 9 0 6
9 0 1 5 9 7 3 4 9 6 6 5
4 0 7 4 0 1 3 1 3 4 7 2

$g \circ f(\textcolor{blue}{X})$ (CNN, $d = 2$)

7 2 1 0 9 1 9 9 6 9 0 6
9 0 1 5 9 7 5 9 9 6 6 5
9 0 7 9 0 1 5 1 3 6 7 2

$g \circ f(\textcolor{blue}{X})$ (PCA, $d = 2$)

9 3 1 0 9 1 9 9 0 9 0 0
9 0 1 8 9 9 8 9 9 0 9 0
9 0 9 9 0 1 8 1 3 0 9 0

$\textcolor{blue}{X}$ (original samples)

7 2 1 0 4 1 4 9 5 9 0 6
9 0 1 5 9 7 3 4 9 6 6 5
4 0 7 4 0 1 3 1 3 4 7 2

$g \circ f(\textcolor{blue}{X})$ (CNN, $d = 8$)

7 2 1 0 4 1 4 9 5 9 0 6
9 0 1 5 9 7 3 4 9 6 6 5
4 0 7 4 0 1 3 1 3 4 7 2

$g \circ f(\textcolor{blue}{X})$ (PCA, $d = 8$)

7 3 1 0 4 1 3 9 0 7 0 0
9 0 1 0 9 7 3 4 7 6 0 5
4 0 7 4 0 1 3 1 3 0 7 0

$\textcolor{blue}{X}$ (original samples)

7 2 1 0 4 1 4 9 5 9 0 6
9 0 1 5 9 7 3 4 9 6 6 5
4 0 7 4 0 1 3 1 3 4 7 2

$g \circ f(\textcolor{blue}{X})$ (CNN, $d = 32$)

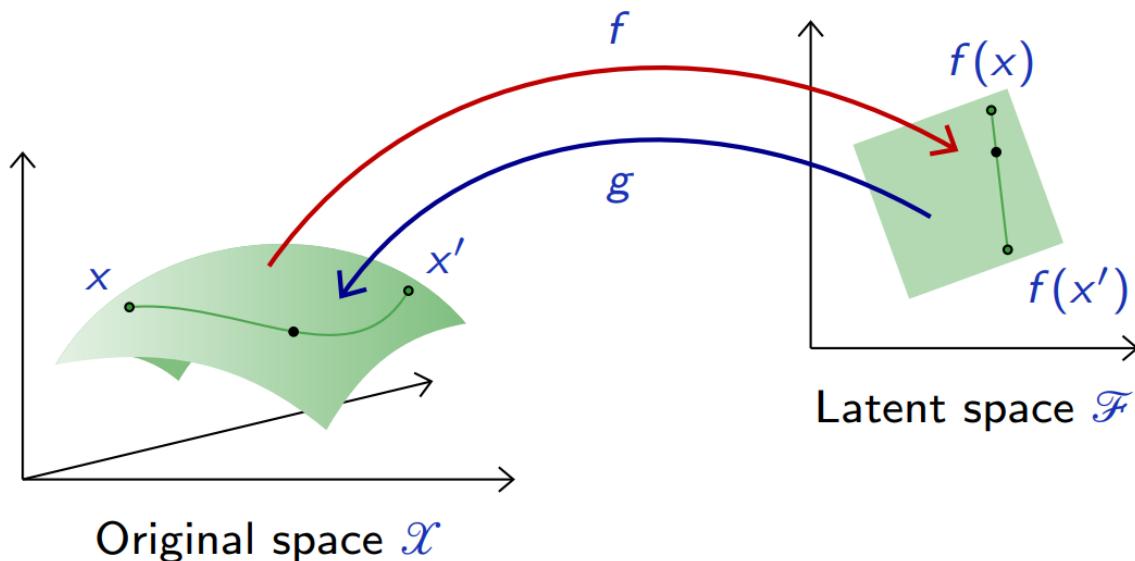
7 2 1 0 4 1 4 9 5 9 0 6
9 0 1 5 9 7 3 4 9 6 6 5
4 0 7 4 0 1 3 1 3 4 7 2

$g \circ f(\textcolor{blue}{X})$ (PCA, $d = 32$)

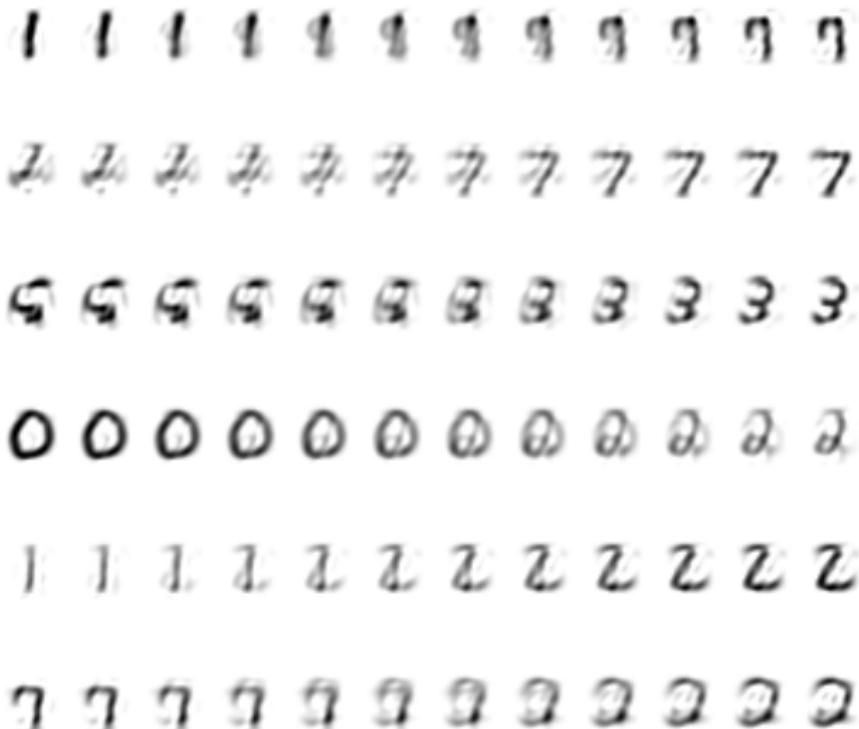
7 2 1 0 4 1 4 9 5 9 0 6
9 0 1 5 9 7 3 4 9 6 6 5
4 0 7 4 0 1 3 1 3 4 7 2

Interpolation

To get an intuition of the learned latent representation, we can pick two samples \mathbf{x} and \mathbf{x}' at random and interpolate samples along the line in the latent space.



PCA interpolation ($d = 32$)



Autoencoder interpolation ($d = 32$)

0 0 0 0 0 0 9 9 9 9 9 9

7 7 7 7 7 7 5 5 5 5 5 5

4 4 4 4 4 4 2 2 2 2 2 2

7 7 7 7 7 7 7 7 7 7 7 7

2 2 2 2 2 4 4 4 4 4 4 4

4 4 4 4 4 9 7 7 7 7 7 7

Denoising auto-encoders

Besides dimension reduction, auto-encoders can capture dependencies between signal components to restore degraded or noisy signals.

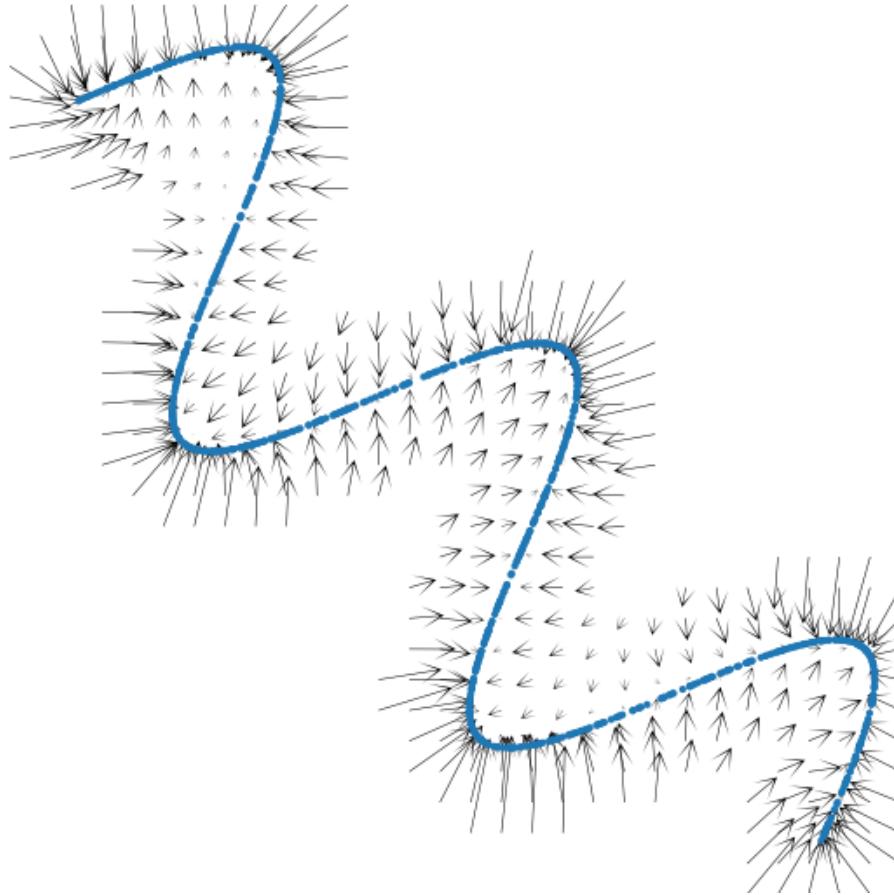
In this case, the composition

$$h = g \circ f : \mathcal{X} \rightarrow \mathcal{X}$$

is referred to as a **denoising** auto-encoder.

The goal is to optimize h such that a perturbation $\tilde{\mathbf{x}}$ of the signal \mathbf{x} is restored to \mathbf{x} , hence

$$h(\tilde{\mathbf{x}}) \approx \mathbf{x}.$$



Original

7 2 1 0 4 1 4 9 5 9 0 6
9 0 1 5 9 7 8 4 9 6 6 5
4 0 7 4 0 1 3 1 3 4 7 2

Corrupted ($p = 0.5$)

7 2 1 0 4 1 4 9 5 9 0 6
9 0 1 5 9 7 8 4 9 6 6 5
4 0 7 4 0 1 3 1 3 4 7 2

Reconstructed

7 2 1 0 4 1 4 9 5 9 0 6
9 0 1 5 9 7 8 4 9 6 6 5
4 0 7 4 0 1 3 1 3 4 7 2

Original

7 2 1 0 4 1 4 9 5 9 0 6
9 0 1 5 9 7 3 4 9 6 6 5
4 0 7 4 0 1 3 1 3 4 7 2

Corrupted ($p = 0.9$)

7 2 1 0 4 1 4 9 5 9 0 6
9 0 1 5 9 7 3 4 9 6 6 5
4 0 7 4 0 1 3 1 3 4 7 2

Reconstructed

7 2 1 0 4 1 4 3 6 9 0 6
9 0 1 5 9 7 3 4 9 6 6 5
4 0 7 4 0 1 3 1 3 4 7 2

Original

7 2 1 0 4 1 4 9 5 9 0 6
9 0 1 5 9 7 3 4 9 6 6 5
4 0 7 4 0 1 3 1 3 4 7 2

Corrupted ($\sigma = 4$)

7 2 1 0 4 1 4 9 5 9 0 6
9 0 1 5 9 7 3 4 9 6 6 5
4 0 7 4 0 1 3 1 3 4 7 2

Reconstructed

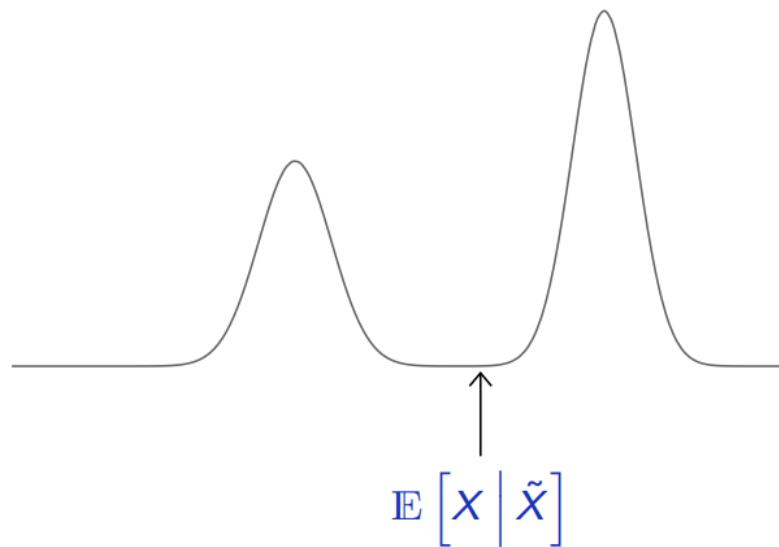
7 2 1 0 4 1 4 9 5 9 0 6
9 0 1 5 9 7 3 4 9 6 6 5
4 0 7 4 0 1 3 1 3 4 7 2

A fundamental weakness of denoising auto-encoders is that the posterior $p(\mathbf{x}|\tilde{\mathbf{x}})$ is possibly multi-modal.

If we train an auto-encoder with the quadratic loss, then the best reconstruction is

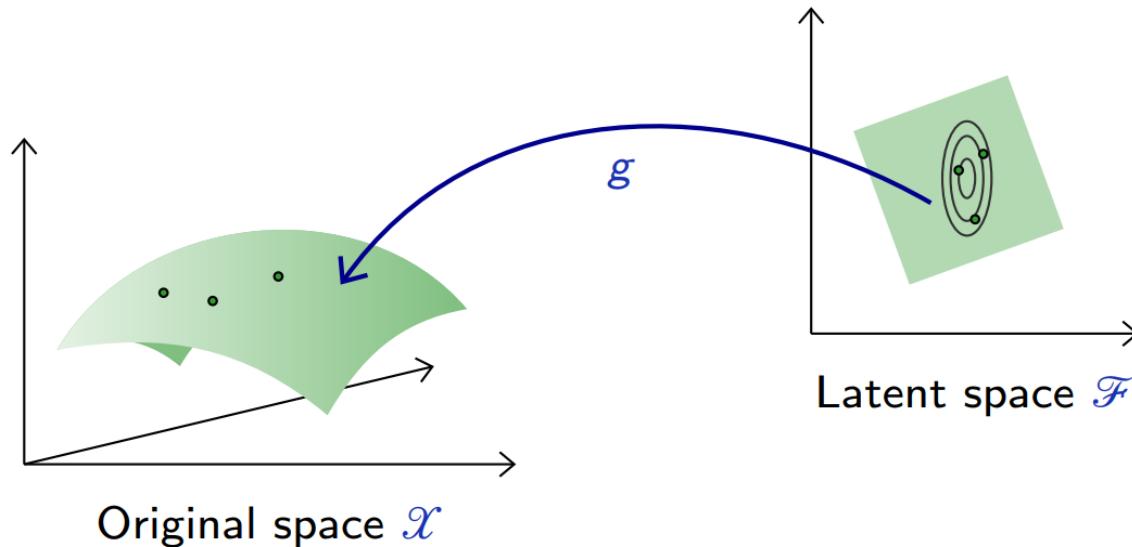
$$h(\tilde{\mathbf{x}}) = \mathbb{E}[\mathbf{x}|\tilde{\mathbf{x}}],$$

which may be very unlikely under $p(\mathbf{x}|\tilde{\mathbf{x}})$.



Sampling from an AE's latent space

The generative capability of the decoder g in an auto-encoder can be assessed by introducing a (simple) density model q over the latent space \mathcal{Z} , sample there, and map the samples into the data space \mathcal{X} with g .

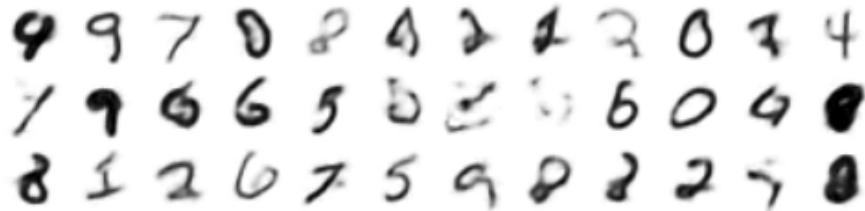


For instance, a factored Gaussian model with diagonal covariance matrix,

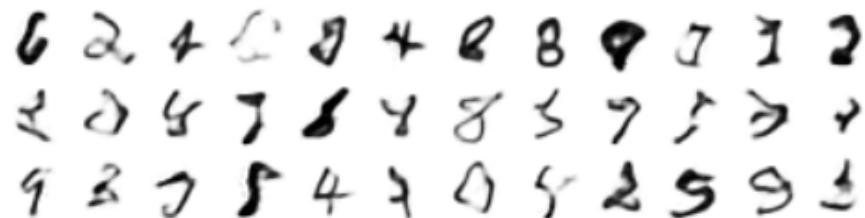
$$q(\mathbf{z}) = \mathcal{N}(\hat{\mu}, \hat{\Sigma}),$$

where both $\hat{\mu}$ and $\hat{\Sigma}$ are estimated on training data.

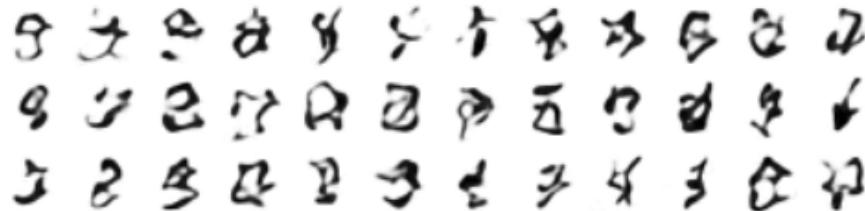
Autoencoder sampling ($d = 8$)



Autoencoder sampling ($d = 16$)



Autoencoder sampling ($d = 32$)

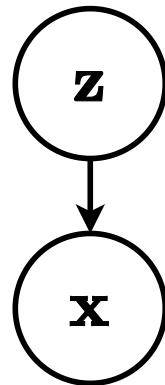


These results are not satisfactory because the density model on the latent space is **too simple and inadequate**.

Building a good model in latent space amounts to our original problem of modeling an empirical distribution, although it may now be in a lower dimension space.

Variational inference

Latent variable model



Consider for now a **prescribed latent variable model** that relates a set of observable variables $\mathbf{x} \in \mathcal{X}$ to a set of unobserved variables $\mathbf{z} \in \mathcal{Z}$.

The probabilistic model defines a joint probability distribution $p(\mathbf{x}, \mathbf{z})$, which decomposes as

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}|\mathbf{z})p(\mathbf{z}).$$

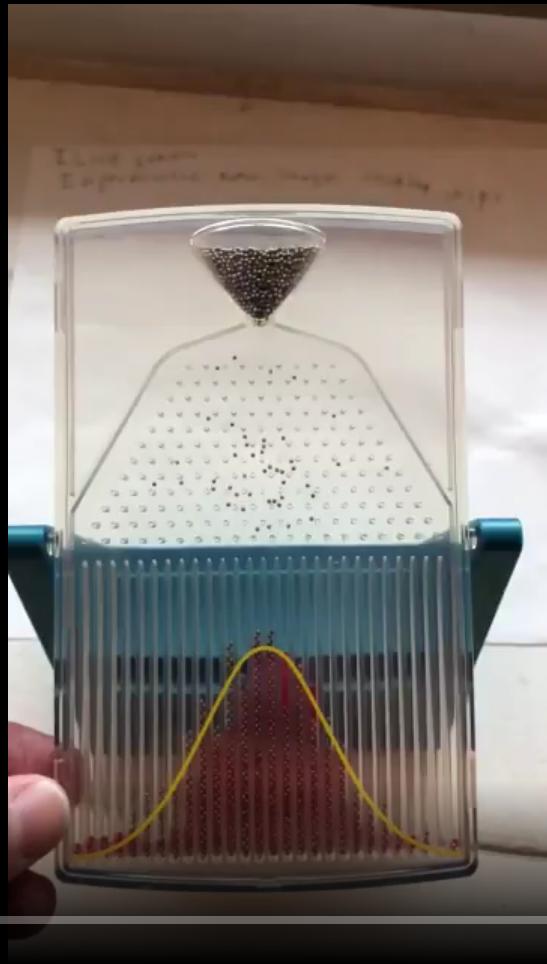
If we interpret \mathbf{z} as causal factors for the high-dimension representations \mathbf{x} , then sampling from $p(\mathbf{x}|\mathbf{z})$ can be interpreted as **a stochastic generating process** from \mathcal{Z} to \mathcal{X} .

For a given model $p(\mathbf{x}, \mathbf{z})$, inference consists in computing the posterior

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{x})}.$$

For most interesting cases, this is usually intractable since it requires evaluating the evidence

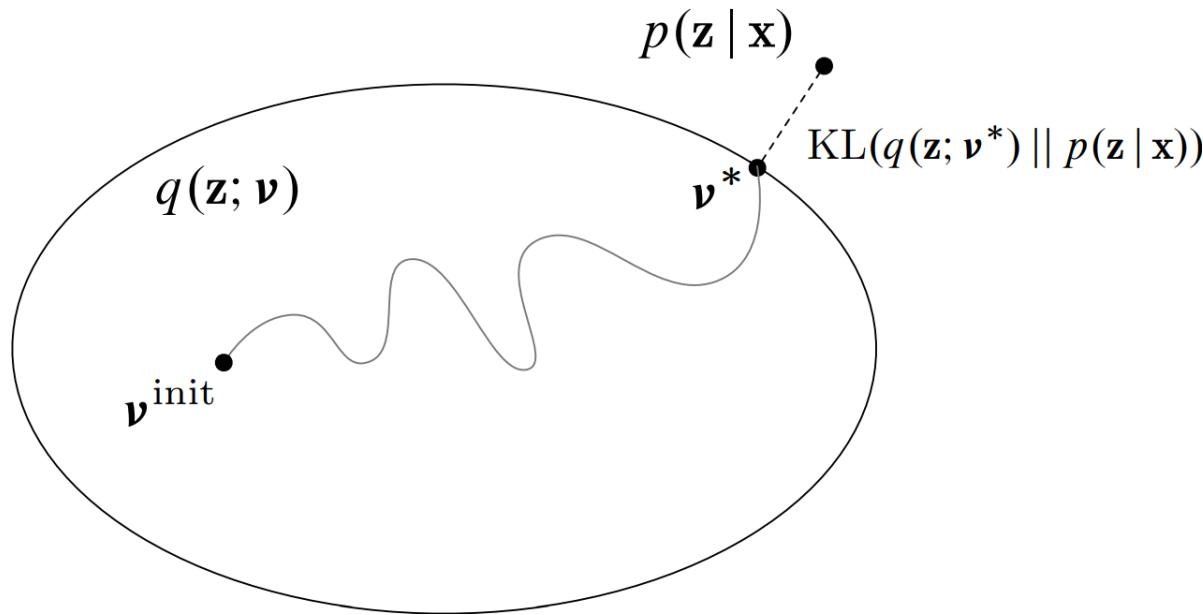
$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}.$$



II 0:00 / 0:45

🔇 🔍 ⏮

Variational inference



Variational inference turns posterior inference into an optimization problem.

- Consider a family of distributions $q(\mathbf{z}|\mathbf{x}; \nu)$ that approximate the posterior $p(\mathbf{z}|\mathbf{x})$, where the variational parameters ν index the family of distributions.
- The parameters ν are fit to minimize the KL divergence between $p(\mathbf{z}|\mathbf{x})$ and the approximation $q(\mathbf{z}|\mathbf{x}; \nu)$.

Formally, we want to minimize

$$\begin{aligned} KL(q(\mathbf{z}|\mathbf{x}; \nu) || p(\mathbf{z}|\mathbf{x})) &= \mathbb{E}_{q(\mathbf{z}|\mathbf{x}; \nu)} \left[\log \frac{q(\mathbf{z}|\mathbf{x}; \nu)}{p(\mathbf{z}|\mathbf{x})} \right] \\ &= \mathbb{E}_{q(\mathbf{z}|\mathbf{x}; \nu)} [\log q(\mathbf{z}|\mathbf{x}; \nu) - \log p(\mathbf{x}, \mathbf{z})] + \log p(\mathbf{x}). \end{aligned}$$

For the same reason as before, the KL divergence cannot be directly minimized because of the $\log p(\mathbf{x})$ term.

However, we can write

$$KL(q(\mathbf{z}|\mathbf{x}; \nu) || p(\mathbf{z}|\mathbf{x})) = \log p(\mathbf{x}) - \underbrace{\mathbb{E}_{q(\mathbf{z}|\mathbf{x}; \nu)} [\log p(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z}|\mathbf{x}; \nu)]}_{\text{ELBO}(\mathbf{x}; \nu)}$$

where $\text{ELBO}(\mathbf{x}; \nu)$ is called the **evidence lower bound objective**.

- Since $\log p(\mathbf{x})$ does not depend on ν , it can be considered as a constant, and minimizing the KL divergence is equivalent to maximizing the evidence lower bound, while being computationally tractable.
- Given a dataset $\mathbf{d} = \{\mathbf{x}_i | i = 1, \dots, N\}$, the final objective is the sum $\sum_{\{\mathbf{x}_i \in \mathbf{d}\}} \text{ELBO}(\mathbf{x}_i; \nu)$.

Remark that

$$\begin{aligned}\text{ELBO}(\mathbf{x}; \nu) &= \mathbb{E}_{q(\mathbf{z}|\mathbf{x}; \nu)} [\log p(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z}|\mathbf{x}; \nu)] \\ &= \mathbb{E}_{q(\mathbf{z}|\mathbf{x}; \nu)} [\log p(\mathbf{x}|\mathbf{z})p(\mathbf{z}) - \log q(\mathbf{z}|\mathbf{x}; \nu)] \\ &= \mathbb{E}_{q(\mathbf{z}|\mathbf{x}; \nu)} [\log p(\mathbf{x}|\mathbf{z})] - KL(q(\mathbf{z}|\mathbf{x}; \nu) || p(\mathbf{z}))\end{aligned}$$

Therefore, maximizing the ELBO:

- encourages distributions to place their mass on configurations of latent variables that explain the observed data (first term);
- encourages distributions close to the prior (second term).

Optimization

We want

$$\begin{aligned}\nu^* &= \arg \max_{\nu} \text{ELBO}(\mathbf{x}; \nu) \\ &= \arg \max_{\nu} \mathbb{E}_{q(\mathbf{z}|\mathbf{x}; \nu)} [\log p(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z}|\mathbf{x}; \nu)] .\end{aligned}$$

We can proceed by gradient ascent, provided we can evaluate $\nabla_{\nu} \text{ELBO}(\mathbf{x}; \nu)$.

In general, this gradient is difficult to compute because the expectation is unknown and the parameters ν are parameters of the distribution $q(\mathbf{z}|\mathbf{x}; \nu)$ we integrate over.

Variational auto-encoders

So far we assumed a prescribed probabilistic model motivated by domain knowledge. We will now directly learn a stochastic generating process with a neural network.

Variational auto-encoders

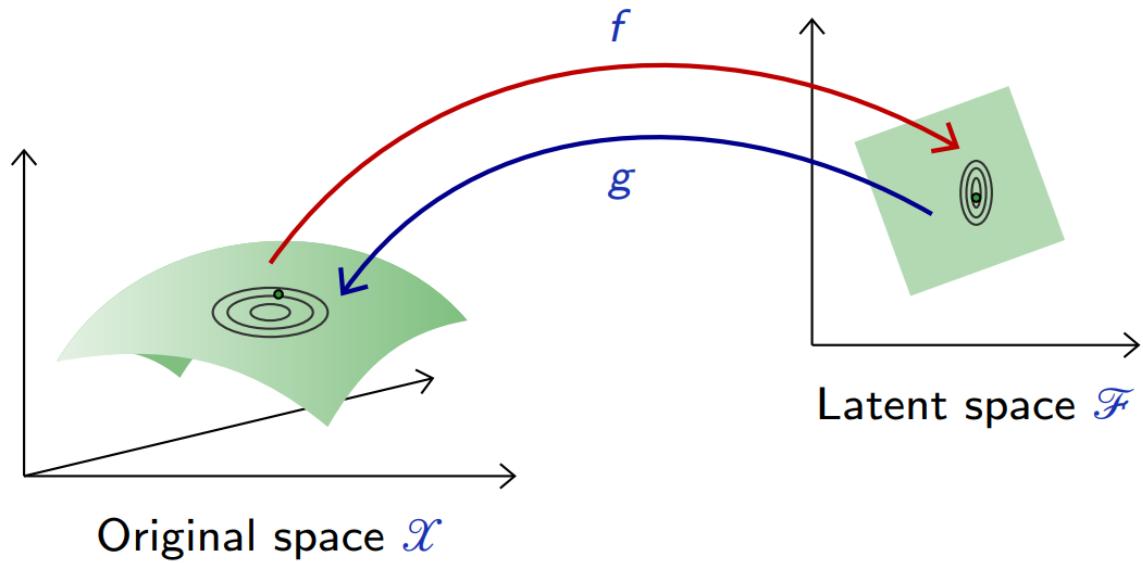
A variational auto-encoder is a deep latent variable model where:

- The likelihood $p(\mathbf{x}|\mathbf{z}; \theta)$ is parameterized with a **generative network** NN_θ (or decoder) that takes as input \mathbf{z} and outputs parameters $\phi = \text{NN}_\theta(\mathbf{z})$ to the data distribution. E.g.,

$$\begin{aligned}\mu, \sigma &= \text{NN}_\theta(\mathbf{z}) \\ p(\mathbf{x}|\mathbf{z}; \theta) &= \mathcal{N}(\mathbf{x}; \mu, \sigma^2 \mathbf{I})\end{aligned}$$

- The approximate posterior $q(\mathbf{z}|\mathbf{x}; \varphi)$ is parameterized with an **inference network** NN_φ (or encoder) that takes as input \mathbf{x} and outputs parameters $\nu = \text{NN}_\varphi(\mathbf{x})$ to the approximate posterior. E.g.,

$$\begin{aligned}\mu, \sigma &= \text{NN}_\varphi(\mathbf{x}) \\ q(\mathbf{z}|\mathbf{x}; \varphi) &= \mathcal{N}(\mathbf{z}; \mu, \sigma^2 \mathbf{I})\end{aligned}$$



As before, we can use variational inference, but to jointly optimize the generative and the inference networks parameters θ and φ .

We want

$$\begin{aligned}\theta^*, \varphi^* &= \arg \max_{\theta, \varphi} \text{ELBO}(\mathbf{x}; \theta, \varphi) \\ &= \arg \max_{\theta, \varphi} \mathbb{E}_{q(\mathbf{z}|\mathbf{x}; \varphi)} [\log p(\mathbf{x}, \mathbf{z}; \theta) - \log q(\mathbf{z}|\mathbf{x}; \varphi)] \\ &= \arg \max_{\theta, \varphi} \mathbb{E}_{q(\mathbf{z}|\mathbf{x}; \varphi)} [\log p(\mathbf{x}|\mathbf{z}; \theta)] - KL(q(\mathbf{z}|\mathbf{x}; \varphi) || p(\mathbf{z})).\end{aligned}$$

- Given some generative network θ , we want to put the mass of the latent variables, by adjusting φ , such that they explain the observed data, while remaining close to the prior.
- Given some inference network φ , we want to put the mass of the observed variables, by adjusting θ , such that they are well explained by the latent variables.

Unbiased gradients of the ELBO with respect to the generative model parameters θ are simple to obtain:

$$\begin{aligned}\nabla_{\theta} \text{ELBO}(\mathbf{x}; \theta, \varphi) &= \nabla_{\theta} \mathbb{E}_{q(\mathbf{z}|\mathbf{x}; \varphi)} [\log p(\mathbf{x}, \mathbf{z}; \theta) - \log q(\mathbf{z}|\mathbf{x}; \varphi)] \\ &= \mathbb{E}_{q(\mathbf{z}|\mathbf{x}; \varphi)} [\nabla_{\theta} (\log p(\mathbf{x}, \mathbf{z}; \theta) - \log q(\mathbf{z}|\mathbf{x}; \varphi))] \\ &= \mathbb{E}_{q(\mathbf{z}|\mathbf{x}; \varphi)} [\nabla_{\theta} \log p(\mathbf{x}, \mathbf{z}; \theta)],\end{aligned}$$

which can be estimated with Monte Carlo integration.

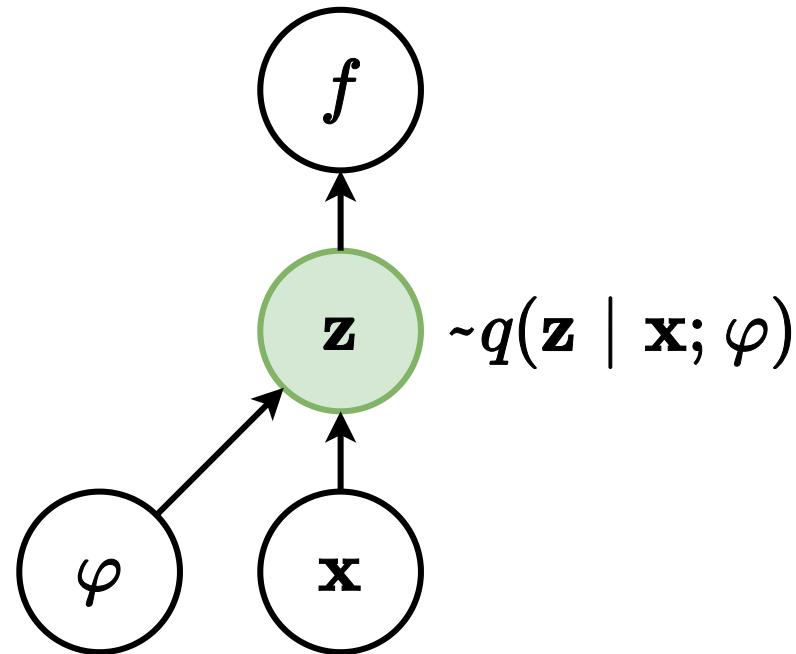
However, gradients with respect to the inference model parameters φ are more difficult to obtain:

$$\begin{aligned}\nabla_{\varphi} \text{ELBO}(\mathbf{x}; \theta, \varphi) &= \nabla_{\varphi} \mathbb{E}_{q(\mathbf{z}|\mathbf{x}; \varphi)} [\log p(\mathbf{x}, \mathbf{z}; \theta) - \log q(\mathbf{z}|\mathbf{x}; \varphi)] \\ &\neq \mathbb{E}_{q(\mathbf{z}|\mathbf{x}; \varphi)} [\nabla_{\varphi} (\log p(\mathbf{x}, \mathbf{z}; \theta) - \log q(\mathbf{z}|\mathbf{x}; \varphi))]\end{aligned}$$

Let us abbreviate

$$\begin{aligned}\text{ELBO}(\mathbf{x}; \theta, \varphi) &= \mathbb{E}_{q(\mathbf{z}|\mathbf{x}; \varphi)} [\log p(\mathbf{x}, \mathbf{z}; \theta) - \log q(\mathbf{z}|\mathbf{x}; \varphi)] \\ &= \mathbb{E}_{q(\mathbf{z}|\mathbf{x}; \varphi)} [f(\mathbf{x}, \mathbf{z}; \varphi)].\end{aligned}$$

We have



We cannot backpropagate through the stochastic node \mathbf{z} to compute $\nabla_\varphi f$!

Reparameterization trick

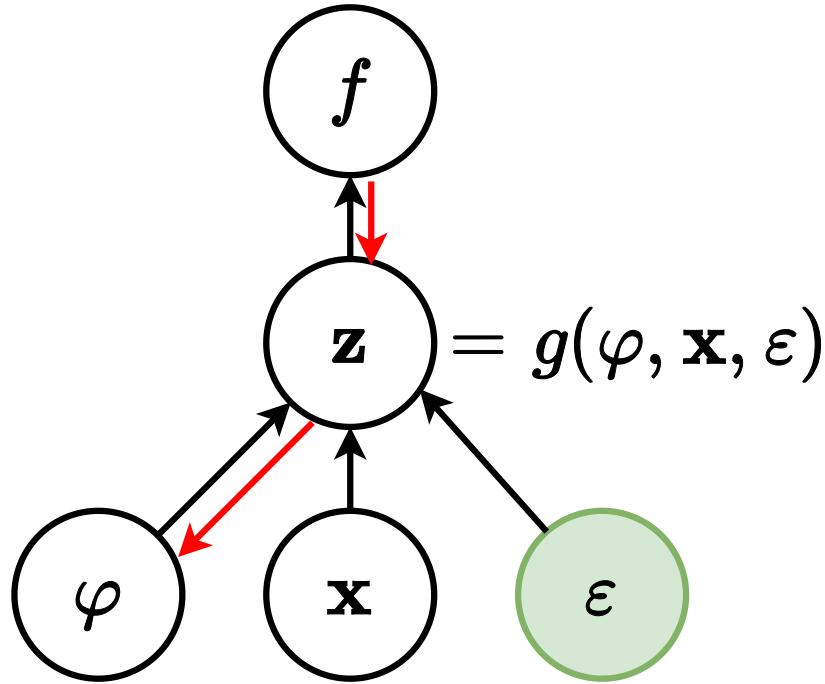
The **reparameterization trick** consists in re-expressing the variable

$$\mathbf{z} \sim q(\mathbf{z}|\mathbf{x}; \varphi)$$

as some differentiable and invertible transformation of another random variable ϵ given \mathbf{x} and φ ,

$$\mathbf{z} = g(\varphi, \mathbf{x}, \epsilon),$$

such that the distribution of ϵ is independent of \mathbf{x} or φ .



For example, if $q(\mathbf{z}|\mathbf{x}; \varphi) = \mathcal{N}(\mathbf{z}; \mu(\mathbf{x}; \varphi), \sigma^2(\mathbf{x}; \varphi))$, where $\mu(\mathbf{x}; \varphi)$ and $\sigma^2(\mathbf{x}; \varphi)$ are the outputs of the inference network NN_φ , then a common reparameterization is:

$$p(\epsilon) = \mathcal{N}(\epsilon; \mathbf{0}, \mathbf{I})$$

$$\mathbf{z} = \mu(\mathbf{x}; \varphi) + \sigma(\mathbf{x}; \varphi) \odot \epsilon$$

Given such a change of variable, the ELBO can be rewritten as:

$$\begin{aligned}\text{ELBO}(\mathbf{x}; \theta, \varphi) &= \mathbb{E}_{q(\mathbf{z}|\mathbf{x}; \varphi)} [f(\mathbf{x}, \mathbf{z}; \varphi)] \\ &= \mathbb{E}_{p(\epsilon)} [f(\mathbf{x}, g(\varphi, \mathbf{x}, \epsilon); \varphi)]\end{aligned}$$

Therefore,

$$\begin{aligned}\nabla_{\varphi} \text{ELBO}(\mathbf{x}; \theta, \varphi) &= \nabla_{\varphi} \mathbb{E}_{p(\epsilon)} [f(\mathbf{x}, g(\varphi, \mathbf{x}, \epsilon); \varphi)] \\ &= \mathbb{E}_{p(\epsilon)} [\nabla_{\varphi} f(\mathbf{x}, g(\varphi, \mathbf{x}, \epsilon); \varphi)],\end{aligned}$$

which we can now estimate with Monte Carlo integration.

The last required ingredient is the evaluation of the likelihood $q(\mathbf{z}|\mathbf{x}; \varphi)$ given the change of variable g . As long as g is invertible, we have:

$$\log q(\mathbf{z}|\mathbf{x}; \varphi) = \log p(\epsilon) - \log \left| \det \left(\frac{\partial \mathbf{z}}{\partial \epsilon} \right) \right|.$$

Example

Consider the following setup:

- Generative model:

$$\mathbf{z} \in \mathbb{R}^d$$

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$$

$$p(\mathbf{x}|\mathbf{z}; \theta) = \mathcal{N}(\mathbf{x}; \mu(\mathbf{z}; \theta), \sigma^2(\mathbf{z}; \theta)\mathbf{I})$$

$$\mu(\mathbf{z}; \theta) = \mathbf{W}_2^T \mathbf{h} + \mathbf{b}_2$$

$$\log \sigma^2(\mathbf{z}; \theta) = \mathbf{W}_3^T \mathbf{h} + \mathbf{b}_3$$

$$\mathbf{h} = \text{ReLU}(\mathbf{W}_1^T \mathbf{z} + \mathbf{b}_1)$$

$$\theta = \{\mathbf{W}_1, \mathbf{b}_1, \mathbf{W}_2, \mathbf{b}_2, \mathbf{W}_3, \mathbf{b}_3\}$$

- Inference model:

$$q(\mathbf{z}|\mathbf{x}; \varphi) = \mathcal{N}(\mathbf{z}; \mu(\mathbf{x}; \varphi), \sigma^2(\mathbf{x}; \varphi)\mathbf{I})$$

$$p(\epsilon) = \mathcal{N}(\epsilon; \mathbf{0}, \mathbf{I})$$

$$\mathbf{z} = \mu(\mathbf{x}; \varphi) + \sigma(\mathbf{x}; \varphi) \odot \epsilon$$

$$\mu(\mathbf{x}; \varphi) = \mathbf{W}_5^T \mathbf{h} + \mathbf{b}_5$$

$$\log \sigma^2(\mathbf{x}; \varphi) = \mathbf{W}_6^T \mathbf{h} + \mathbf{b}_6$$

$$\mathbf{h} = \text{ReLU}(\mathbf{W}_4^T \mathbf{x} + \mathbf{b}_4)$$

$$\varphi = \{\mathbf{W}_4, \mathbf{b}_4, \mathbf{W}_5, \mathbf{b}_5, \mathbf{W}_6, \mathbf{b}_6\}$$

Note that there is no restriction on the generative and inference network architectures. They could as well be arbitrarily complex convolutional networks.

Plugging everything together, the objective can be expressed as:

$$\begin{aligned}\text{ELBO}(\mathbf{x}; \theta, \varphi) &= \mathbb{E}_{q(\mathbf{z}|\mathbf{x}; \varphi)} [\log p(\mathbf{x}, \mathbf{z}; \theta) - \log q(\mathbf{z}|\mathbf{x}; \varphi)] \\ &= \mathbb{E}_{q(\mathbf{z}|\mathbf{x}; \varphi)} [\log p(\mathbf{x}|\mathbf{z}; \theta)] - KL(q(\mathbf{z}|\mathbf{x}; \varphi) || p(\mathbf{z})) \\ &= \mathbb{E}_{p(\epsilon)} [\log p(\mathbf{x}|\mathbf{z} = g(\varphi, \mathbf{x}, \epsilon); \theta)] - KL(q(\mathbf{z}|\mathbf{x}; \varphi) || p(\mathbf{z}))\end{aligned}$$

where the KL divergence can be expressed analytically as

$$KL(q(\mathbf{z}|\mathbf{x}; \varphi) || p(\mathbf{z})) = \frac{1}{2} \sum_{j=1}^d (1 + \log(\sigma_j^2(\mathbf{x}; \varphi)) - \mu_j^2(\mathbf{x}; \varphi) - \sigma_j^2(\mathbf{x}; \varphi)),$$

which allows to evaluate its derivative without approximation.

Consider as data **d** the MNIST digit dataset:

The image shows a 10x10 grid of handwritten digits, likely from the MNIST dataset. The digits are arranged in rows, starting with 0s at the top and ending with 9s at the bottom. Each digit is a black outline on a white background. The digits are somewhat stylized and vary slightly in orientation and thickness.

0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	9	9

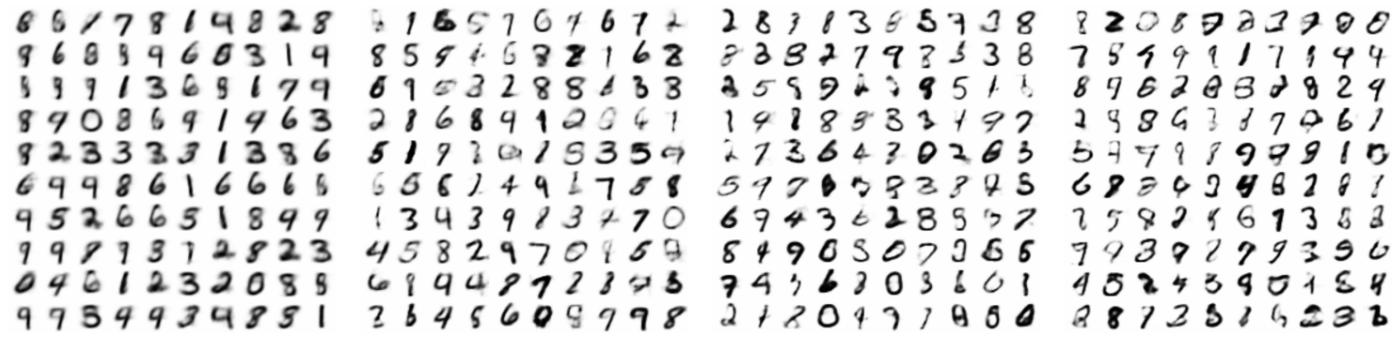
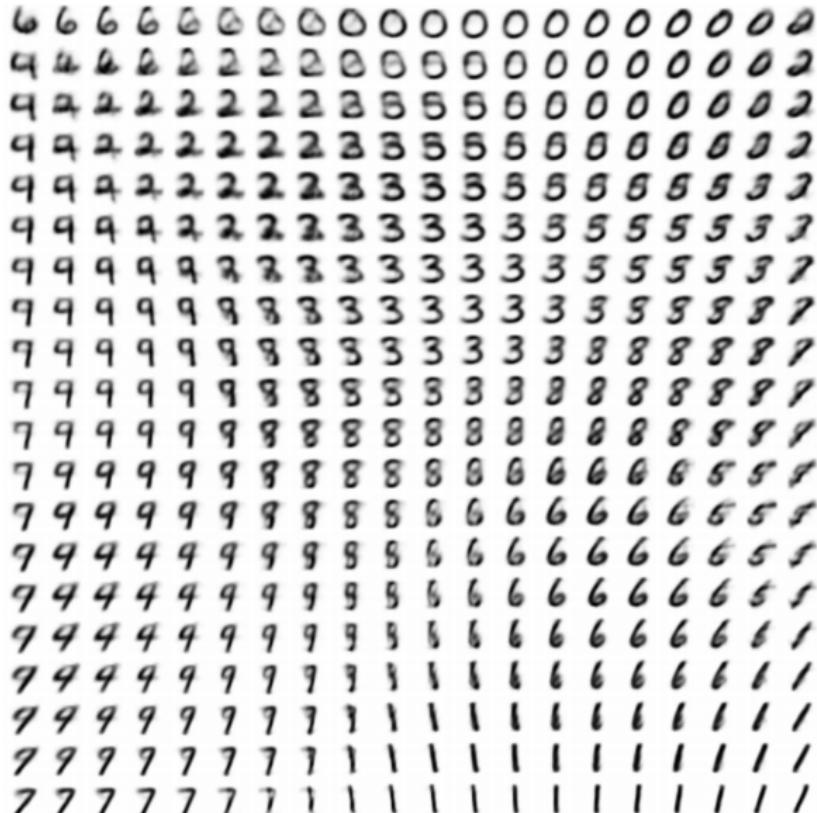


Figure 5: Random samples from learned generative models of MNIST for different dimensionalities of latent space.

(Kingma and Welling, 2013)



(a) Learned Frey Face manifold



(b) Learned MNIST manifold

Figure 4: Visualisations of learned data manifold for generative models with two-dimensional latent space, learned with AEVB. Since the prior of the latent space is Gaussian, linearly spaced coordinates on the unit square were transformed through the inverse CDF of the Gaussian to produce values of the latent variables \mathbf{z} . For each of these values \mathbf{z} , we plotted the corresponding generative $p_{\theta}(\mathbf{x}|\mathbf{z})$ with the learned parameters θ .

(Kingma and Welling, 2013)

Applications



Face manifold from conv/deconv variational aut...



Later bekijk...



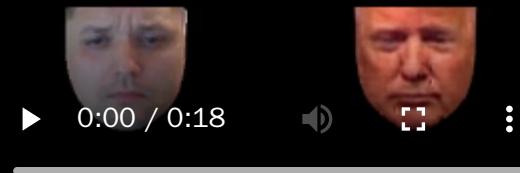
Delen



Random walks in latent space. (Alex Radford, 2015)

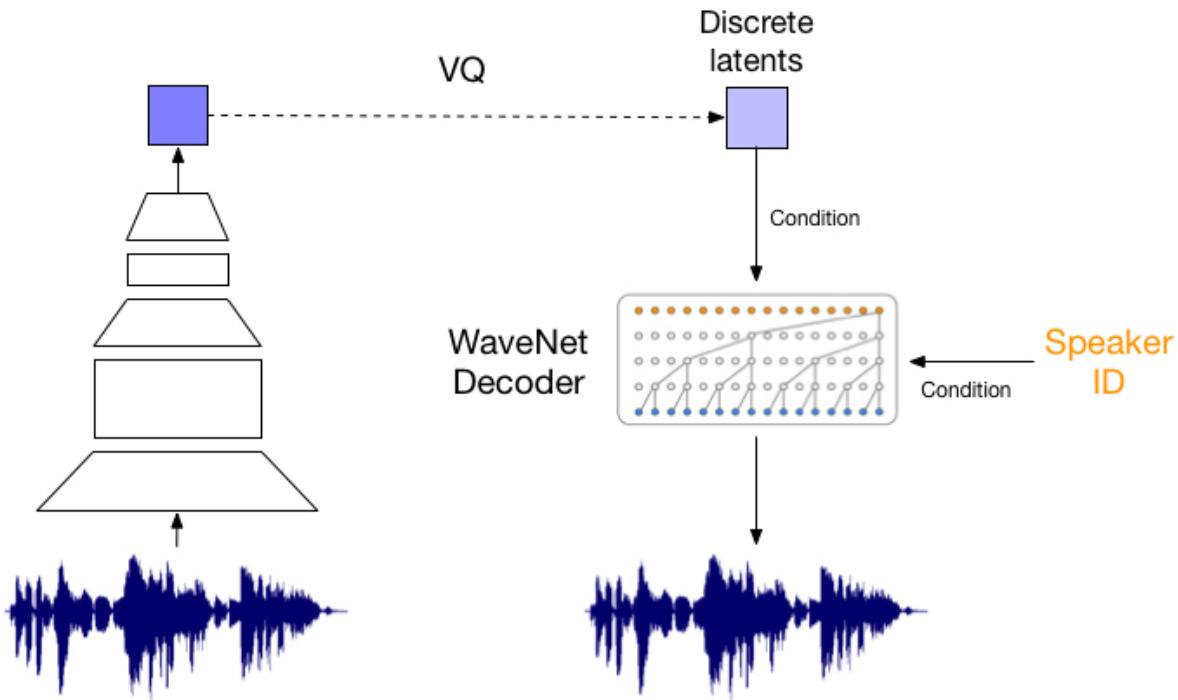


Random walks in latent space. (Vahdat and Kautz, 2020)



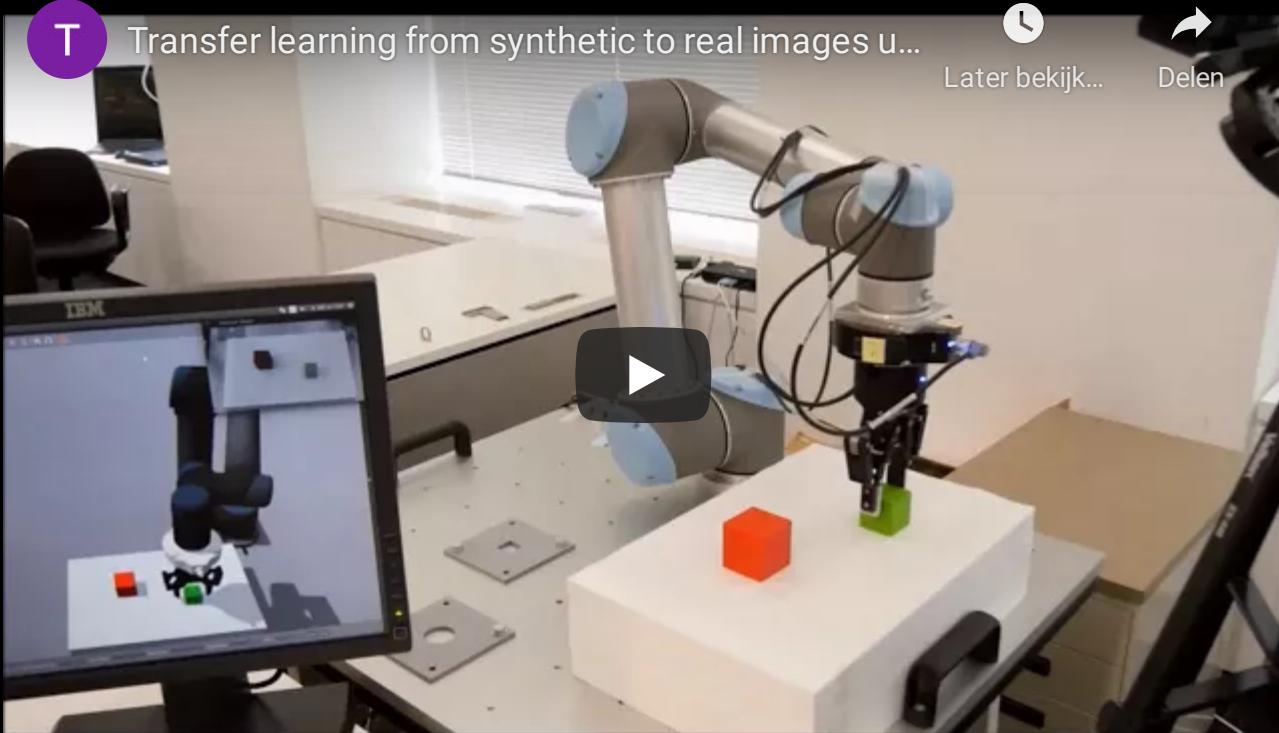
Impersonation by encoding-decoding an unknown face.

(Kamil Czarnogórski, 2016)



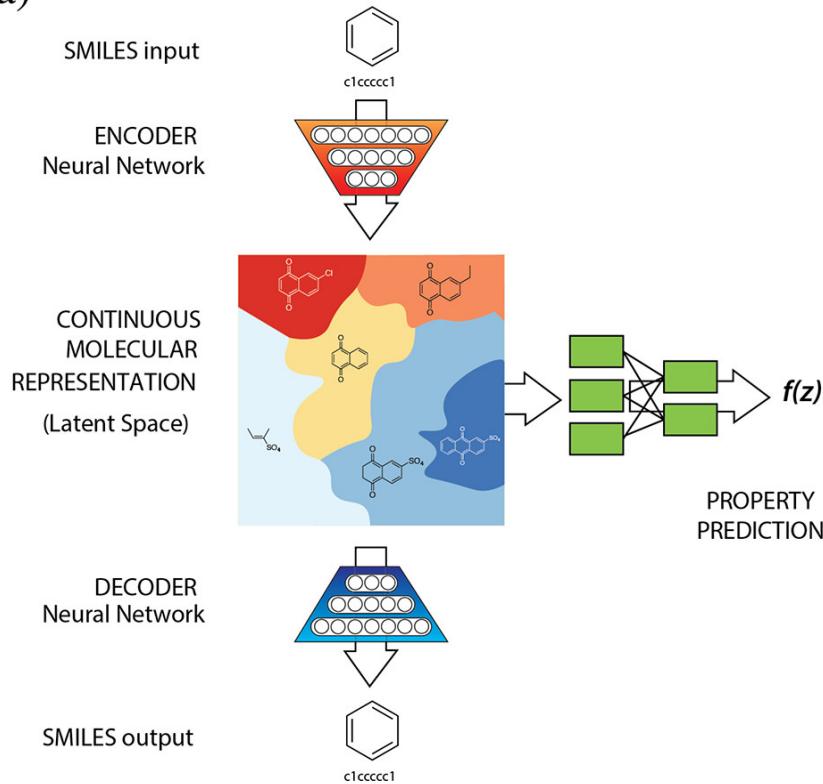
Voice style transfer [[demo](#)]

(van den Oord et al, 2017)

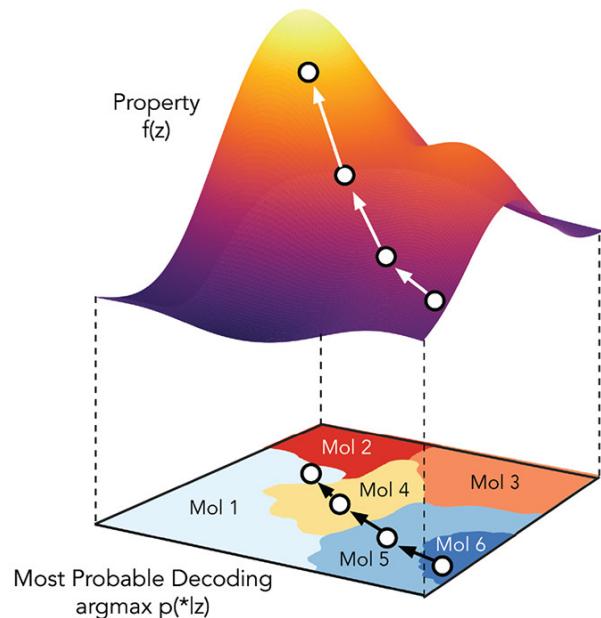


(Inoue et al, 2017)

(a)



(b)



Design of new molecules with desired chemical properties.
(Gomez-Bombarelli et al, 2016)

The end.

References

- Mohamed and Rezende, "[Tutorial on Deep Generative Models](#)", UAI 2017.
- Blei et al, "[Variational inference: Foundations and modern methods](#)", 2016.
- Kingma and Welling, "[Auto-Encoding Variational Bayes](#)", 2013.