A brief introduction to (variational) autoencoders

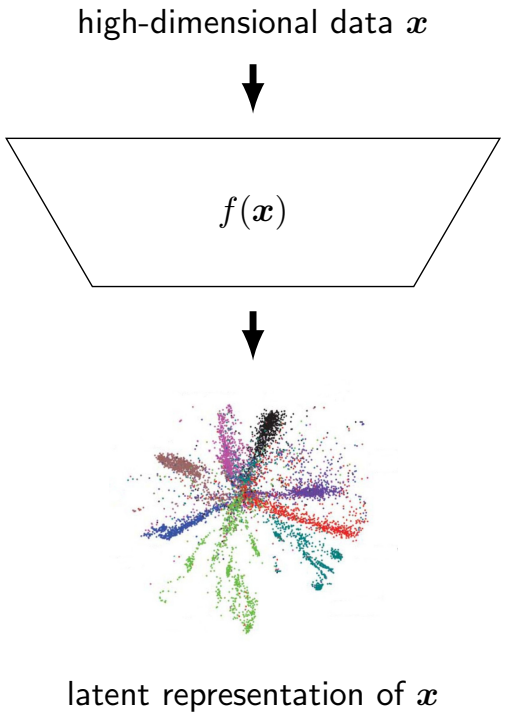
Alexej Klushyn

argmax.ai
Volkswagen Group Machine Learning Research Lab

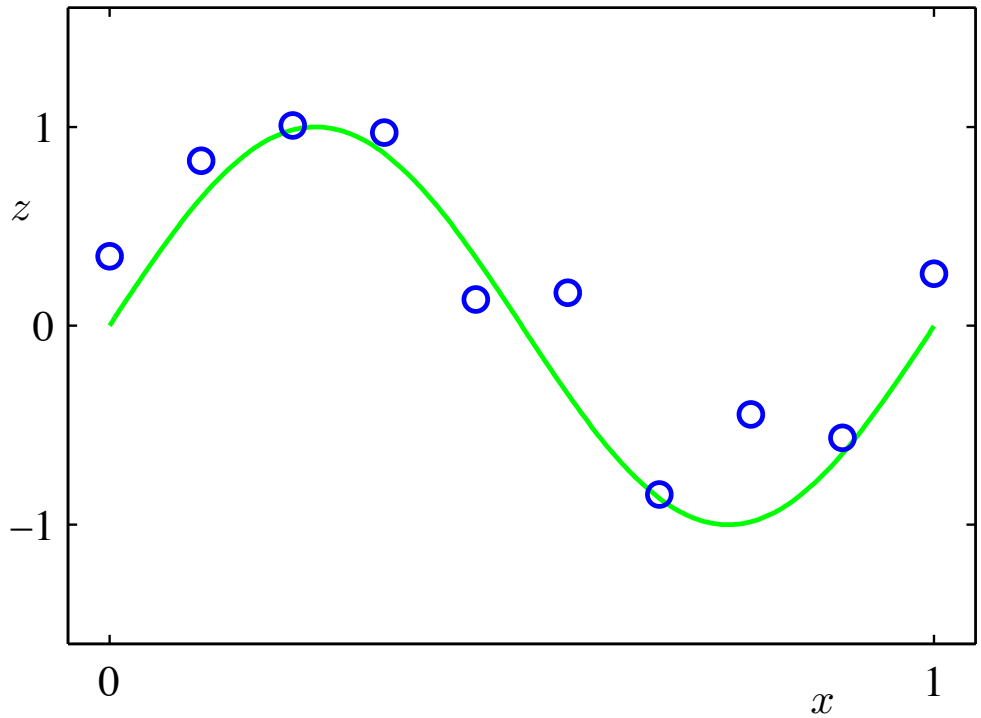
Can we learn a latent representation of our data?

- Dimensionality reduction methods map high-dimensional data to a low-dimensional latent space.
- Often this mapping is linear or restricted in some other way.

Can we learn an arbitrary complex mapping $f(\boldsymbol{x})$?



A noisy real-valued function

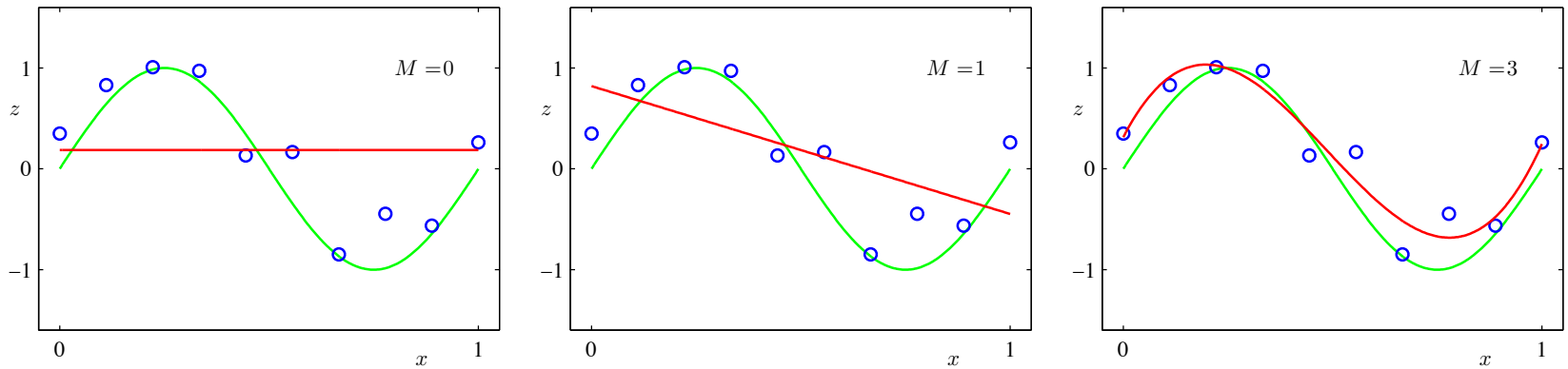


inputs: $\boldsymbol{x} = (x_1, \ldots, x_N)^{\mathsf{T}}$

targets: $\boldsymbol{z} = (z_1, \ldots, z_N)^{\mathsf{T}}, \quad z_i = h(x_i) + \epsilon = \sin(2\pi x_i) + \epsilon$

These figures are from C. Bishop: Pattern Recognition and Machine Learning

Model: 0th, 1st, and 3rd order polynomial



0th order: $y(\boldsymbol{x}, \boldsymbol{w}) = w_0$

1st order: $y(\boldsymbol{x}, \boldsymbol{w}) = w_0 + w_1 x$

3rd order: $y(\boldsymbol{x}, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2 + w_3 x^3$

Problem definition

We have input vectors $\boldsymbol{x}$ and associated output vectors $\boldsymbol{z}$. We want to describe the underlying functional relation.

What about the following model?

$$y(\boldsymbol{x}, \boldsymbol{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\boldsymbol{x}) = \boldsymbol{w}^{\mathsf{T}} \boldsymbol{\phi}(\boldsymbol{x})$$

where

$\boldsymbol{\phi}$	basis function	—	many choices, can be nonlinear
w_0	bias	—	equivalent to defining $\phi_0 \equiv 1$

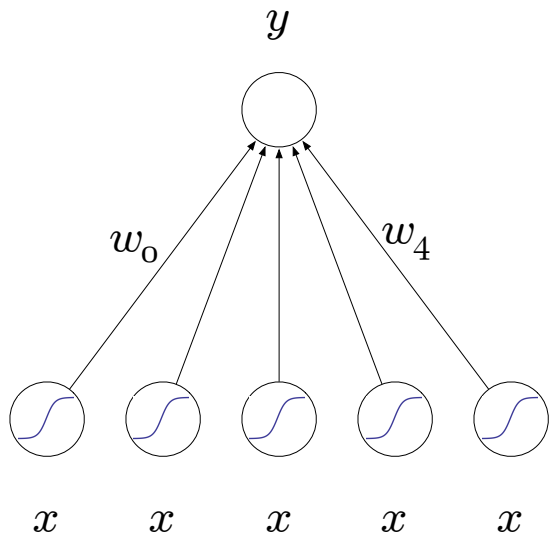
Every continuous function in the function space can be represented as a linear combination* of basis functions!

*Nothing new if you know Taylor expansion, Fourier transform, ...

Towards nonlinear systems

How do we find the optimal basis function ϕ?

The above system could be graphically represented like this

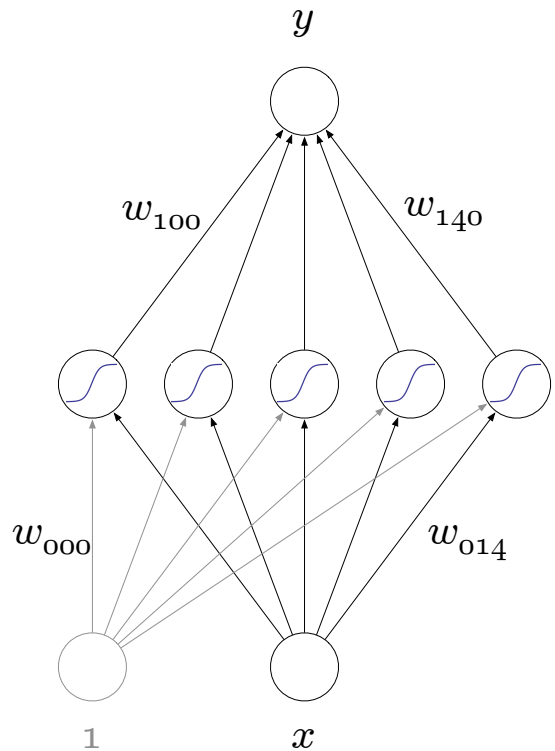


where the arrows represent the weights $\boldsymbol{w}$ and the circles the basis functions.

Why don't we let the system find the optimal ϕ?

The multi-layered perceptron = neural network

Idea: we can extend the system by an additional layer, so that each basis function is itself a nonlinear function of a linear combination of the inputs:



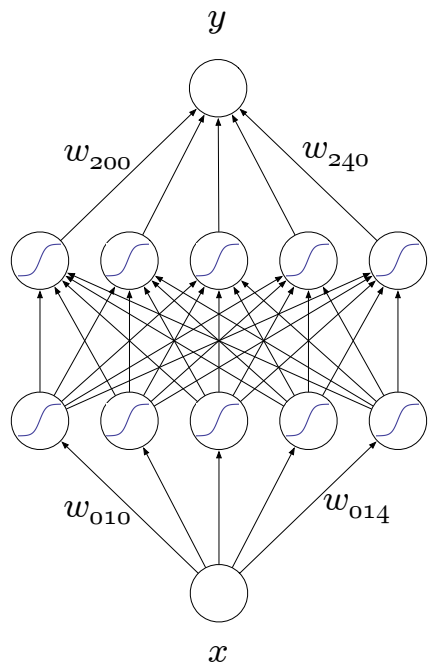
(for simplicity, the constant "1" is usually and from now on not depicted. But you always need it!)

We have generalised to $y(\boldsymbol{x}, \boldsymbol{w}_0, \boldsymbol{w}_1) = \boldsymbol{w}_1^\mathsf{T} \boldsymbol{h}(\boldsymbol{w}_0^T \boldsymbol{x})$.

Note: in the neural network model, circles represent differentiable, nonlinear *activation functions* $\boldsymbol{h}$ (*ReLU*, *sigmoid*, *tanh*, ...).

The deep neural network

We can continue adding more hidden layers



and get a deep neural network: $y(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{w}_2^{\mathsf{T}} \boldsymbol{h}\big(\boldsymbol{w}_1^{\mathsf{T}} \boldsymbol{h}(\boldsymbol{w}_0^T \boldsymbol{x})\big)$.

This reduces the problem to finding the optimal $\boldsymbol{w}$!

How do we find $\boldsymbol{w}$?

Usually we minimise the mean squared error (MSE):

$$\mathcal{L}(\boldsymbol{w}) = \sum_{n=1}^{N} \big(z_n - y(x_n, \boldsymbol{w})\big)^2$$

There is one difference w.r.t. linear regression: $\mathcal{L}(\boldsymbol{w})$ is no longer convex!

How can this be minimised? The minimum is located where its gradient is 0. So one typically minimises by using the gradient:

$$\boldsymbol{w}_{\text{new}} = \boldsymbol{w}_{\text{old}} - \alpha \nabla \mathcal{L}(\boldsymbol{w})$$

How do we compute the gradient $\nabla \mathcal{L}$? Backpropagation does this by using the chain rule:

$$f(x) = u(v(x)) \quad \rightarrow \quad f'(x) = u'(v(x)) \cdot v'(x).$$

Algorithm for backprop ("on-line" aka "stochastic" learning)

backpropagation algorithm:

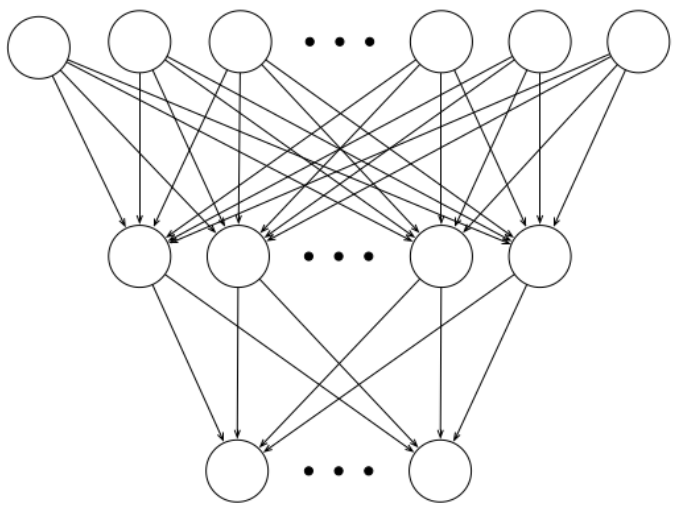
initialise the weights
repeat
for each training sample $(\boldsymbol{x}, \boldsymbol{z})$ do
begin
compute $\boldsymbol{o} = y(\boldsymbol{w}, \boldsymbol{x})$ (forward pass)
calculate residual $\delta_{kj} = \boldsymbol{z} - \boldsymbol{o}$ at the output units
for all k:
propagate δ_{kj} back one layer by $\delta_{k-1,i} = \sum_j \delta_{kj} w_{k-1,i,j}$
update the weights using $\partial \mathcal{L} / \partial w_{kij} = \delta_{kj}\, \phi'(\cdot)\, x_i$
end
(this is called one epoch)
until stopping criterion satisfied

Unsupervised learning with neural networks

Problem: often we have unlabelled data

1. we have many data $\boldsymbol{x}$...
2. but no the related output $\boldsymbol{z}$.

$\boldsymbol{x}$

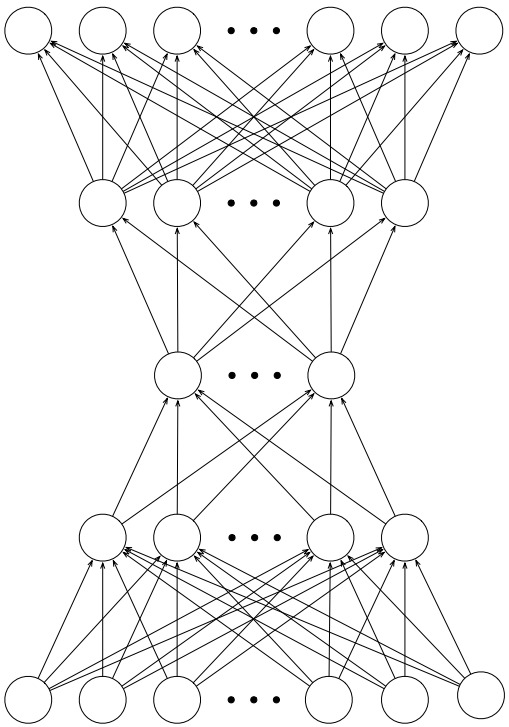


?

Autoencoder networks

Idea: find compact representation of inputs (unsupervised!) by

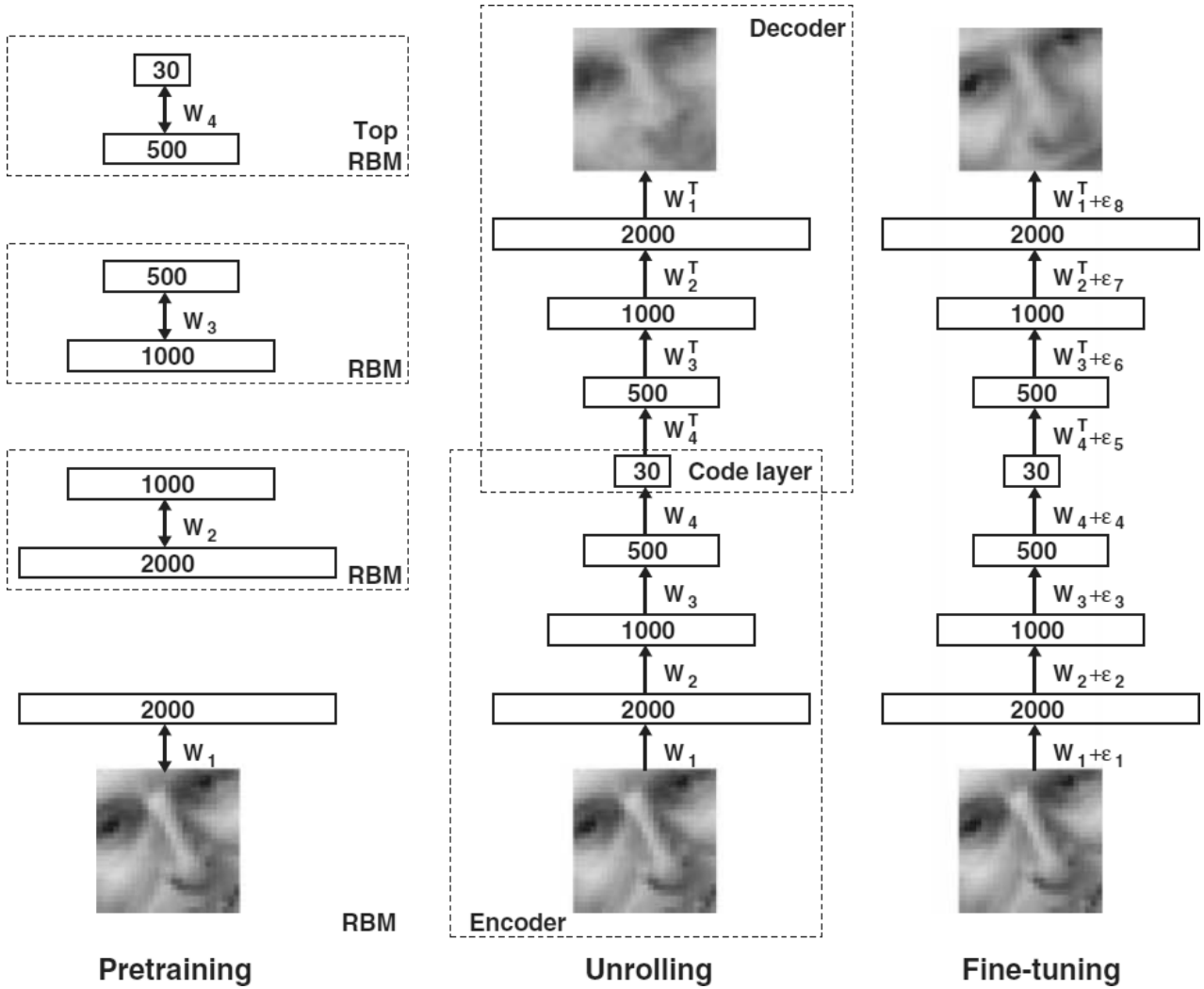
1. using a second neural network as decoder;
2. letting the neural network compute $\boldsymbol{y}(\boldsymbol{x}) = \boldsymbol{x}$.



- $\mathcal{L}_{\mathrm{AE}}(\boldsymbol{w}) = \|\boldsymbol{x} - \boldsymbol{y}(\boldsymbol{x}, \boldsymbol{w})\|^2$
- middle layer ("latents") usually has fewer neurons
- latent representation $\boldsymbol{z}$ = compact code for $\boldsymbol{x}$

These networks make a compact representation of data (*dimensionality reduction*).

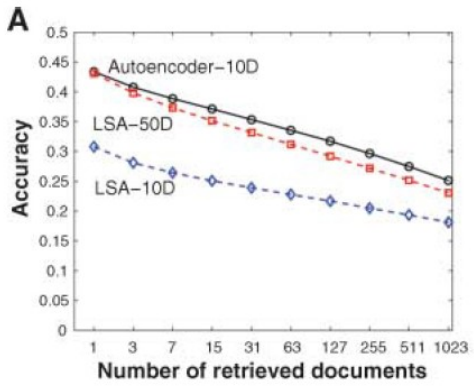
Autoencoder example: face compression

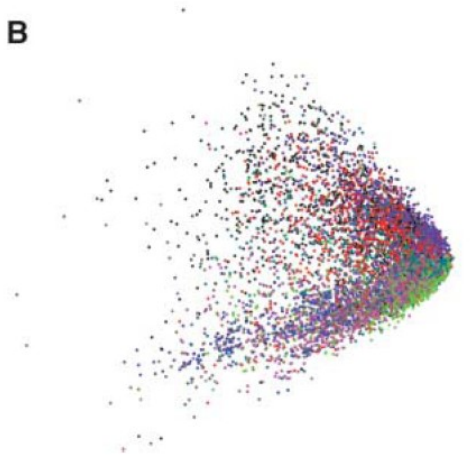


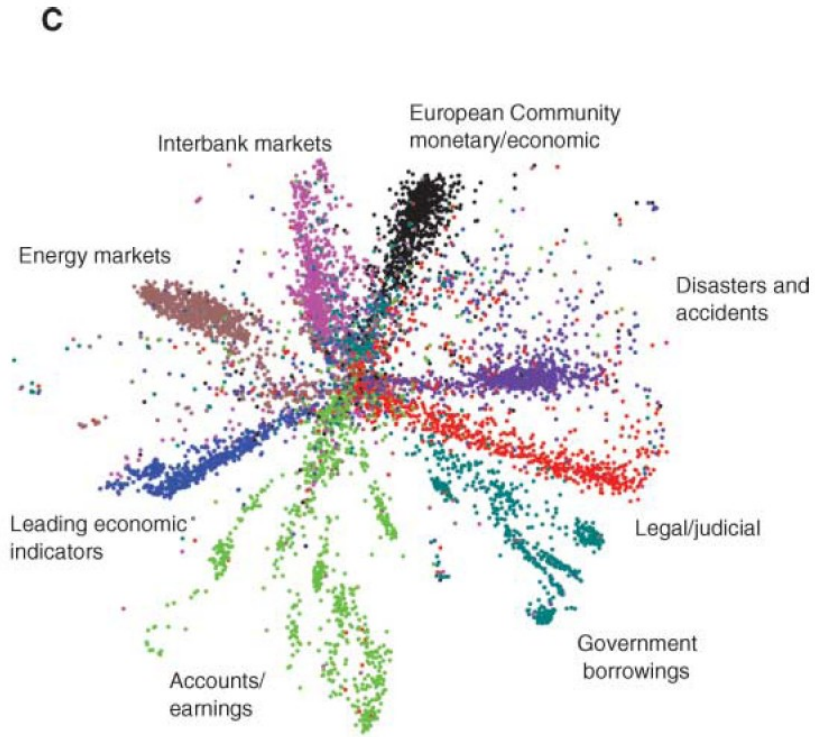
Hinton, G. E. and Salakhutdinov, R. R. Reducing the Dimensionality of Data with Neural Networks. *Science*, 2006.

Autoencoder example: text categorisation

Fig. 4. (**A**) The fraction of retrieved documents in the same class as the query when a query document from the test set is used to retrieve other test set documents, averaged over all 402,207 possible queries. (**B**) The codes produced by two-dimensional LSA. (**C**) The codes produced by a 2000-500-250-125-2 autoencoder.



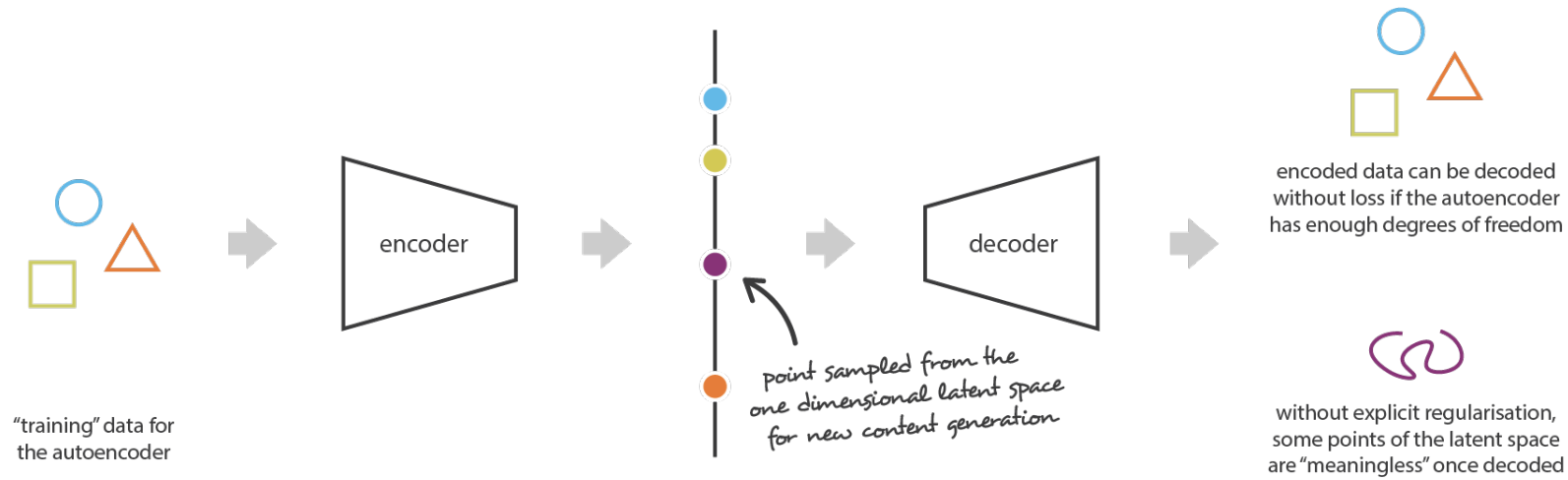




Hinton, G. E. and Salakhutdinov, R. R. Reducing the Dimensionality of Data with Neural Networks. *Science*, 2006.

Limitations of autoencoders

An autoencoder cannot *generalise* from training data such that it is capable of generating new data!



The reason is we don't learn the underlying distribution of our data $p(\boldsymbol{x})$.

Joseph Rocca's blog post *Understanding Variational Autoencoders*

The probabilistic perspective is crucial

Why is the underlying distribution of our data interesting?

- Knowing $p(\boldsymbol{x})$ enables the generation of realistic objects, movements, etc.
- If our model learns, e.g., the syntax of a language, it can generate meaningful text.
- In case of sequential data, learning the physics/dynamics of a system allows the model to predict the future.

Generating photorealistic faces

Karras, T. et al. A Style-Based Generator Architecture for Generative Adversarial Networks. *CVPR*, 2019.

Generating coherent paragraphs of text

SYSTEM PROMPT (HUMAN-WRITTEN)	*Miley Cyrus was caught shoplifting from Abercrombie and Fitch on Hollywood Boulevard today.*
MODEL COMPLETION (MACHINE-WRITTEN, SECOND TRY)	The 19-year-old singer was caught on camera being escorted out of the store by security guards. The singer was wearing a black hoodie with the label 'Blurred Lines' on the front and 'Fashion Police' on the back. Scroll down for video Shoplifting: Miley Cyrus was caught shoplifting from Abercrombie and Fitch on Hollywood Boulevard today (pictured) The singer was wearing a black hoodie with the label 'Blurred Lines' on the front and 'Fashion Police' on the back The singer was also wearing a pair of black-rimmed glasses, a black jacket, black jeans and black sandals. She was carrying a pair of black and white striped gloves and a small black bag.

Radford, A. et al. Language Models are Unsupervised Multitask Learners. 2019.

Beyond visualisation: taking advantage of the latent representation

- Latent variables $\boldsymbol{z}$ are often interpreted as the *quintessential information*.
- What if we could utilise $\boldsymbol{z}$ to learn the underlying distribution of our data $p(\boldsymbol{x})$?
- Let's use the definition of latent-variable models as starting point:

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}, \boldsymbol{z}) \, \mathrm{d}\boldsymbol{z} = \int p(\boldsymbol{x}|\boldsymbol{z}) \, p(\boldsymbol{z}) \, \mathrm{d}\boldsymbol{z} = \mathbb{E}_{p(\boldsymbol{z})}[p(\boldsymbol{x}|\boldsymbol{z})].$$

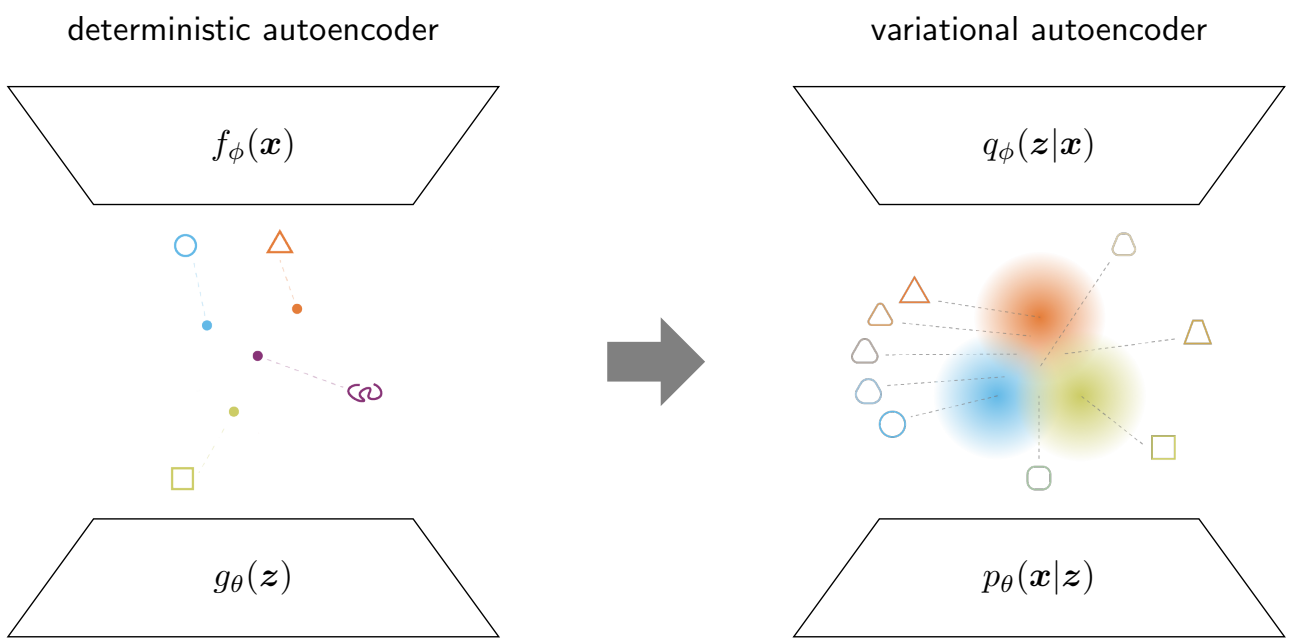
How do we find/learn $\mathbb{E}_{p(\boldsymbol{z})}[p(\boldsymbol{x}|\boldsymbol{z})]$?*

*The prior $p(\boldsymbol{z})$ is usually defined in advance (e.g. as standard normal distribution).

Variational autoencoder—getting an intuition

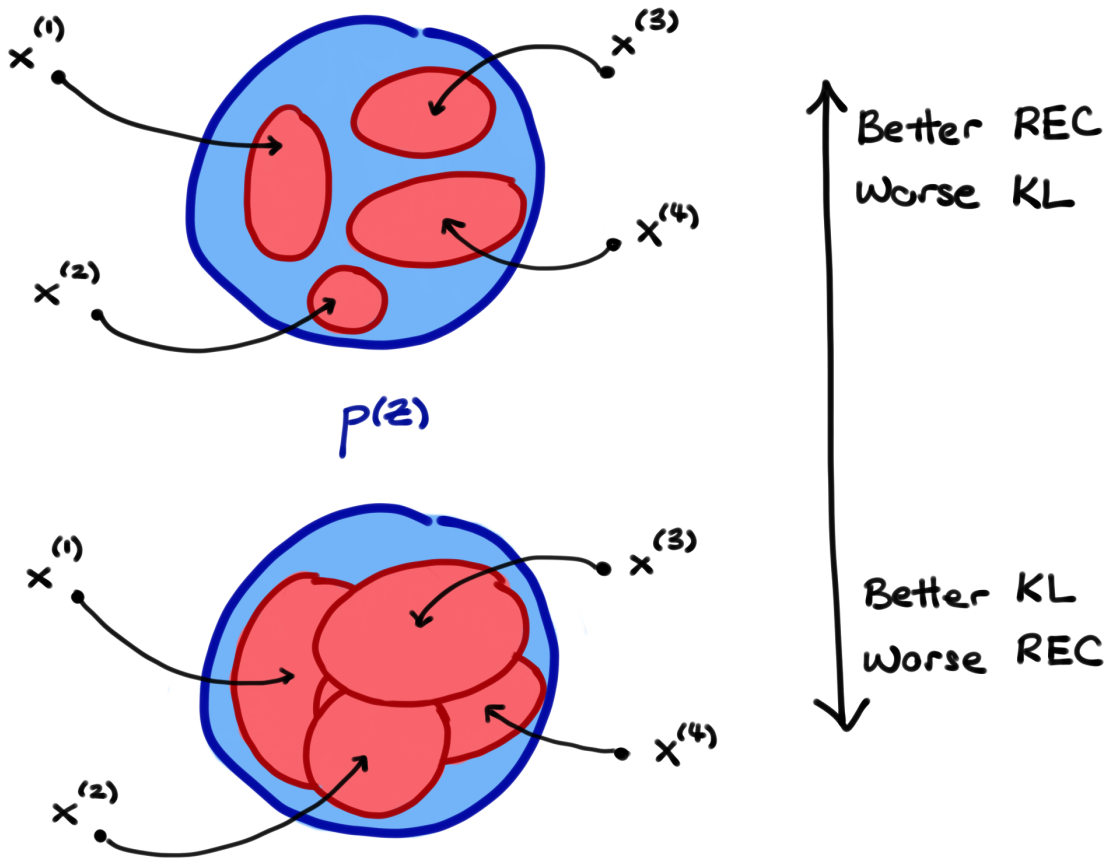
To learn $\mathbb{E}_{p(\boldsymbol{z})}[p(\boldsymbol{x}|\boldsymbol{z})]$, we introduce a variational distribution $q(\boldsymbol{z}|\boldsymbol{x})$.
The resulting objective function is known as the evidence lower bound:

$$\log p(\boldsymbol{x}) \geq \underbrace{\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})]}_{\text{reconstruction}} - \underbrace{\mathrm{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \,||\, p(\boldsymbol{z}))}_{\text{regularisation}} \equiv ELBO(\phi, \theta)$$



Note: ϕ and θ denote the parameters of the functions/distributions learned by the neural networks.

KL divergence as regulariser



Rui Shu's blog post *Density Estimation: Variational Autoencoders*

Deriving the evidence lower bound

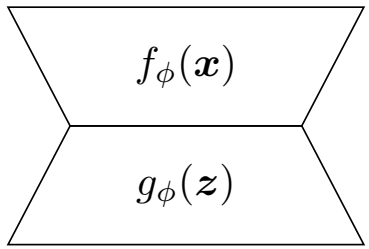
$$
\begin{aligned}
\log p(\boldsymbol{x}) &= \log \int p_\theta(\boldsymbol{x}|\boldsymbol{z})\, p(\boldsymbol{z})\, \mathrm{d}\boldsymbol{z} \\
&= \log \int \frac{p_\theta(\boldsymbol{x}|\boldsymbol{z})\, p(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\, q_\phi(\boldsymbol{z}|\boldsymbol{x})\, \mathrm{d}\boldsymbol{z} \\
&= \log \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\frac{p_\theta(\boldsymbol{x}|\boldsymbol{z})\, p(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right] \\
&\overset{1}{\geq} \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_\theta(\boldsymbol{x}|\boldsymbol{z})\, p(\boldsymbol{z})}{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\right] \\
&= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] - \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{q_\phi(\boldsymbol{z}|\boldsymbol{x})}{p(\boldsymbol{z})}\right] \\
&= \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] - \mathrm{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \,||\, p(\boldsymbol{z})) \\
&\left(\overset{2}{\propto} -\mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\frac{1}{2}\underbrace{\|\boldsymbol{x} - \boldsymbol{\mu}_\theta(\boldsymbol{z})\|^2}_{\substack{\text{reconstruction} \\ \text{error}}}\right] - \mathrm{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \,||\, p(\boldsymbol{z})) \right)
\end{aligned}
$$

[1] Jensen's inequality

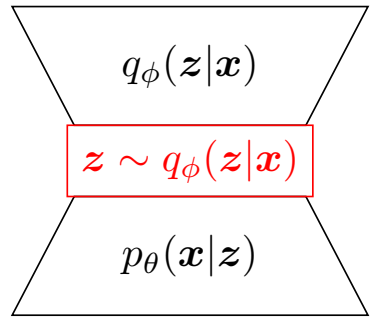
[2] If $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ is a diagonal Gaussian with $\Sigma = \mathbb{1}$.

Deterministic vs. variational autoencoder—Gaussian case*

deterministic autoencoder



variational autoencoder



$$\nabla_{\phi,\theta}\, \mathcal{L}_{\text{AE}} = \nabla_{\phi,\theta}\, \|\boldsymbol{x} - g_\theta(f_\phi(\boldsymbol{x}))\|^2$$

$$\begin{aligned}\nabla_{\phi,\theta}\, \mathcal{L}_{\text{VAE}} &= -\nabla_{\phi,\theta}\, ELBO \\ &= \nabla_{\phi,\theta} \left[\mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{x})} \left[\frac{1}{2}\|\boldsymbol{x} - \boldsymbol{\mu}_\theta(\boldsymbol{z})\|^2\right] + \text{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \,||\, p(\boldsymbol{z}))\right]\end{aligned}$$

The ELBO rewards good reconstruction (first term), and comes with an inbuilt regulariser (the KL) that prevents collapsing to the deterministic autoencoder.

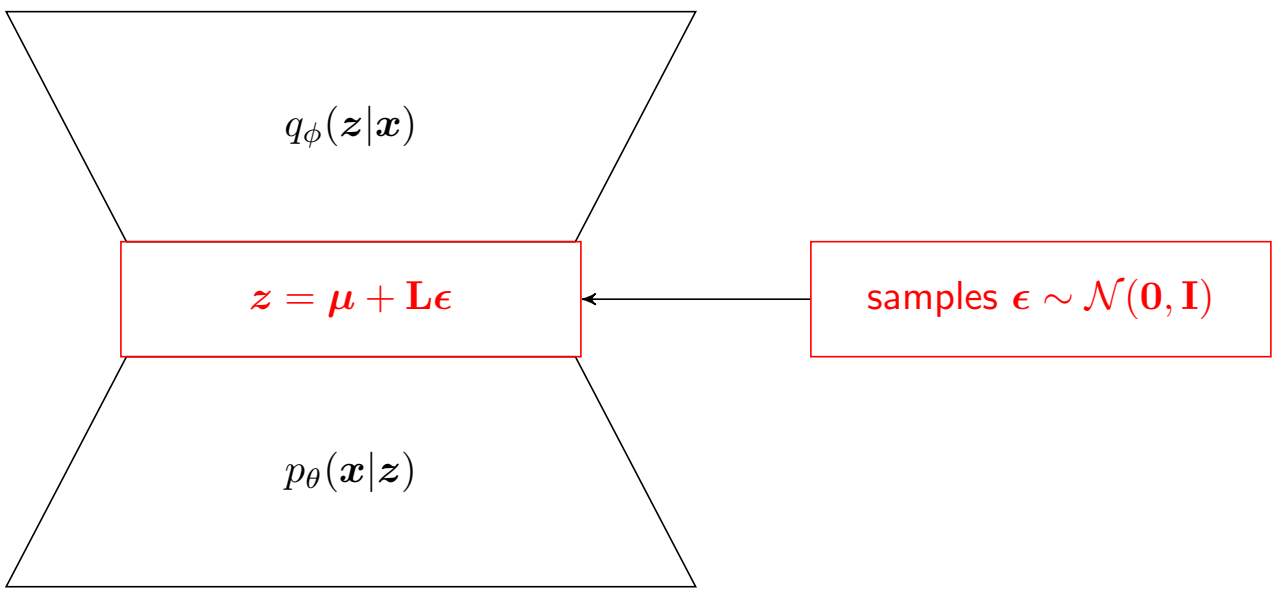
*In the VAE setting, $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ and $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ are typically specified to be diagonal Gaussian distributions.

One important detail: backpropagation through random sampling?

Solution: reparametrisation

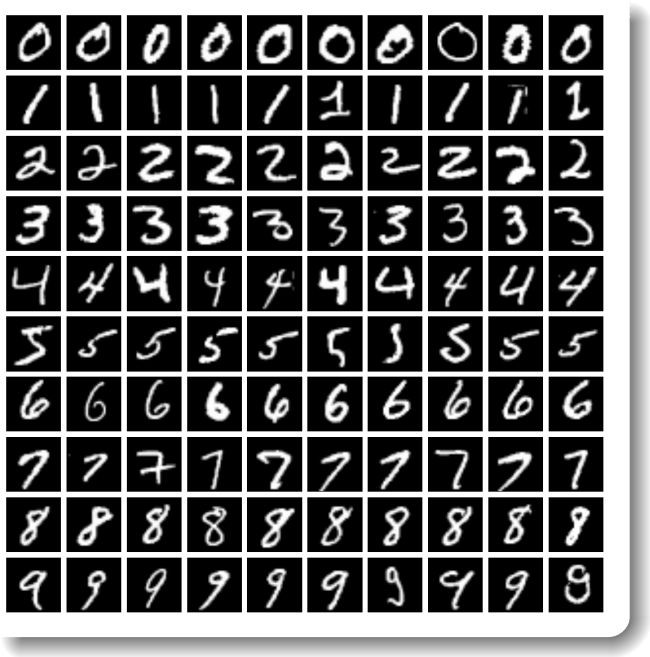
$$\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \quad \rightsquigarrow \quad \boldsymbol{z} = \boldsymbol{\mu} + \mathbf{L}\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \quad \boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^T$$

This allows taking partial derivatives w. r. t. $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$.



The popular MNIST dataset

Consider the following data:

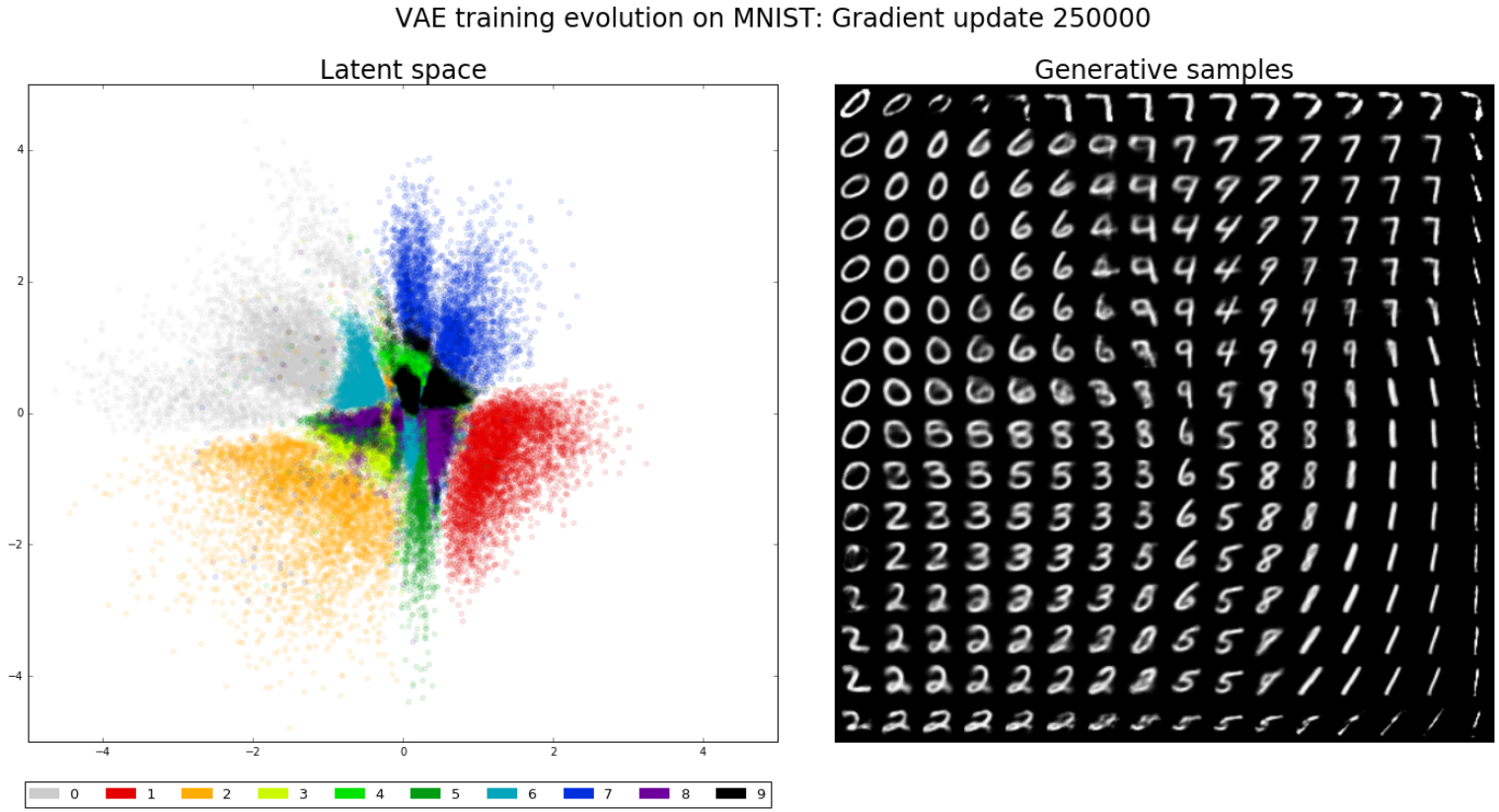


MNIST handwritten digits
`http://deeplearning.net/data/mnist/`

What might be the underlying distribution $p(\boldsymbol{x})$?

Our standard set of distributions is not expressive enough: 

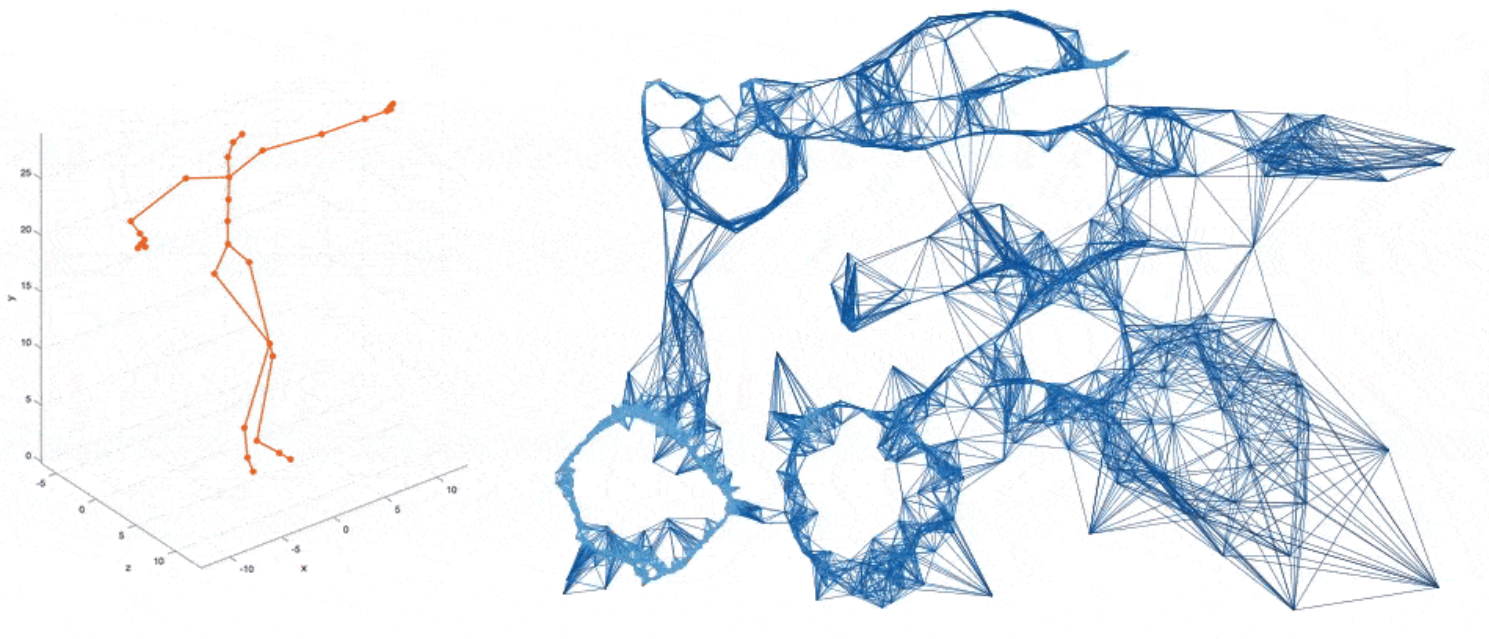
Learning process of a variational autoencoder



If this figure is not animated, you might want to try a different pdf reader.

For visualisation, the model only uses a two-dimensional latent space. A higher dimension would allow the model to achieve better results.

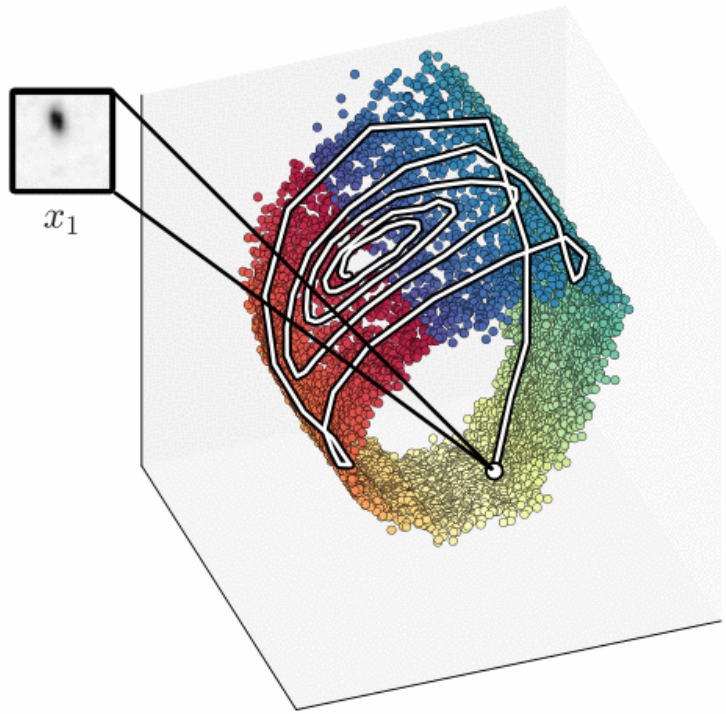
Some results from our lab: human motion



If this figure is not animated, you might want to try a different pdf reader.

Klushyn, A. et al. Learning Hierarchical Priors in VAEs. *NeurIPS*, 2019.

Some results from our lab: pendulum (sequential data)



If this figure is not animated, you might want to try a different pdf reader.

Karl, M. et al. Deep Variational Bayes Filters: Unsupervised Learning of State Space Models from Raw Data. *ICLR*, 2017.

What we learned

- Neural networks are function approximators.
- Two neural networks can be combined to an autoencoder for learning a compact/latent representation of our data in an unsupervised fashion.
- The variational autoencoder—a probabilistic method—can learn the underlying distribution of our data by means of latent variables.