University of Minho
School of Engineering

ISLab

Aprendizagem Profunda

Generative models

AP@ MEI/1º ano – 2º Semestre

Victor Alves

Part VII

Exemplos/ figuras adaptados de F. Chollet, “Deep Learning with Python” e Andrew Ng (deeplearning.ai)

Generative algorithms

- The examples we have seen of DL so far have all been of supervised problems: classification/ regression - mapping between a set of input variables and one or more output variables
- There are, however, applications of DL models to other types of tasks, including unsupervised models, reinforcement learning, semi-supervised, etc.
- In this session, we will discuss various architectures and examples of applying DL models to the generation of new data of various types, including sequences (e.g. text, music, etc.) and images
- The models used are called generative (deep generative models)

Generative algorithms

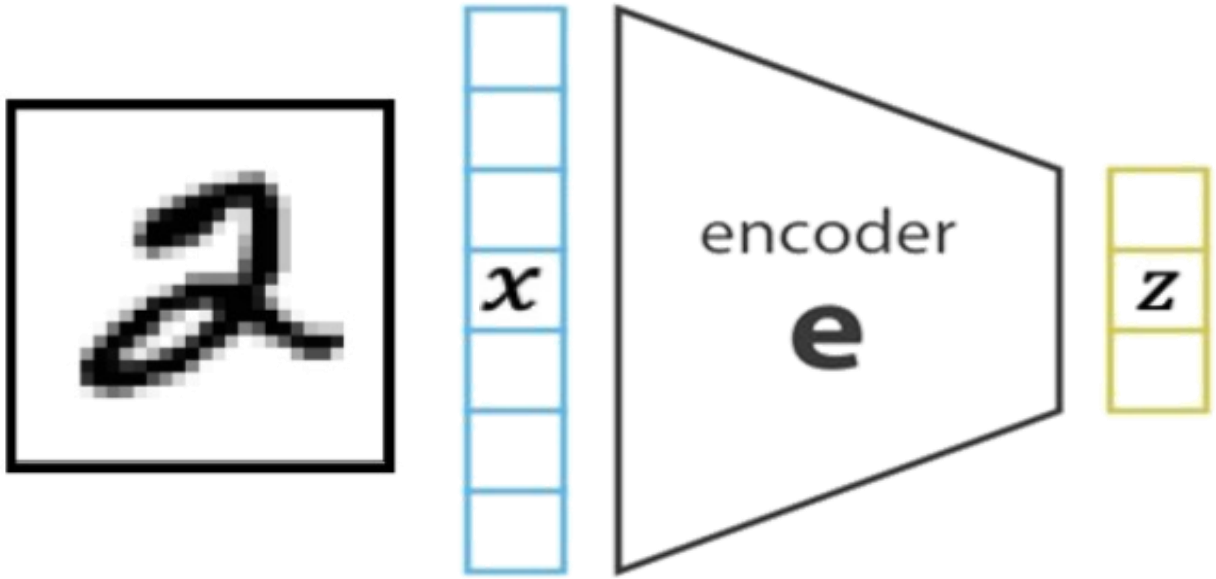
- The idea of generative models is to be trained with training examples representing data where we want to learn the "original distribution", i.e. the characteristics of the examples we want to generate
- The goal will be to create models that can generate other data, following the characteristics/distributions of the original data.
- There are different types of architectures that can be used, depending on the type of original data and the goals to be achieved - these include, among others, models such as RNNs/ LSTMs, **Autoencoders**, **Variational autoencoders**, **Generative Adversarial Networks (GANs), Diffusion Models**

Generative algorithms - applications

- One of the main applications of these algorithms has been in the production of artistic creations in an automated way: images, music, etc.
- Other applications include:
  - Text generation (e.g. automatic response to written messages)
  - Generation of new molecules with certain characteristics (e.g. drugs)
  - Network security: generation of traffic similar to cyber-attacks
  - Image generation for enhancement of biometric systems against attacks/fraud;
  - Detection of biases and outliers in the training dataset.
  - Syntetic data

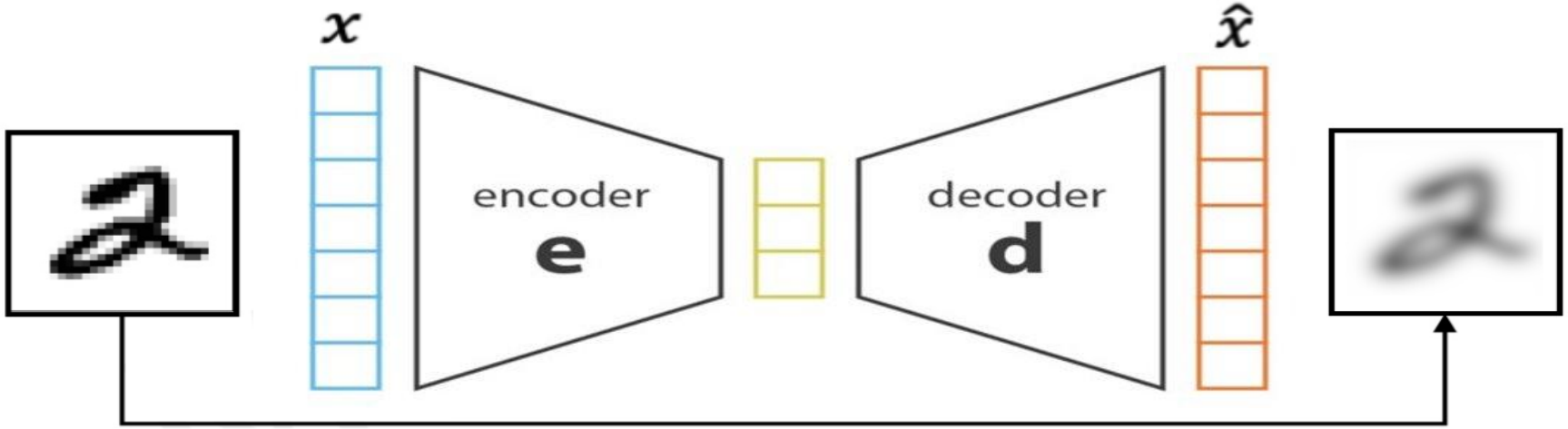
Autoencoders

- Autoencoders are based on the idea of representing any type of input data **x** as lower dimensional representations (e.g. vector of real numbers) - latent space **z**
- Similar concept to what is used in *embeddings* and the use of algebraic methods (e.g. PCA, PLS, LDA)



Autoencoders

- In the case of autoencoders, the idea is to learn this mapping by creating models that can reproduce the input data in the outputs, with the latent space being represented in the intermediate layers that are of lower dimensionality

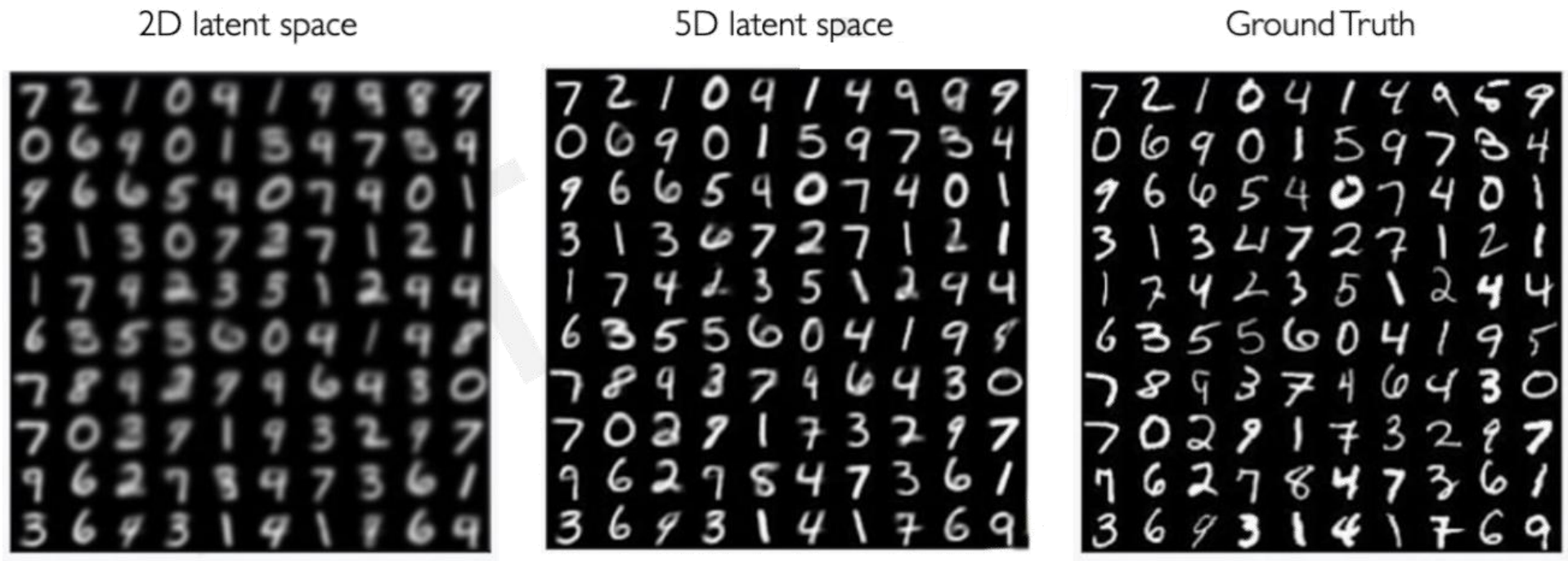


Supervised or Unsupervised model?

It's an unsupervised learning method, trained using supervised learning methods, referred to as **self-supervised**.

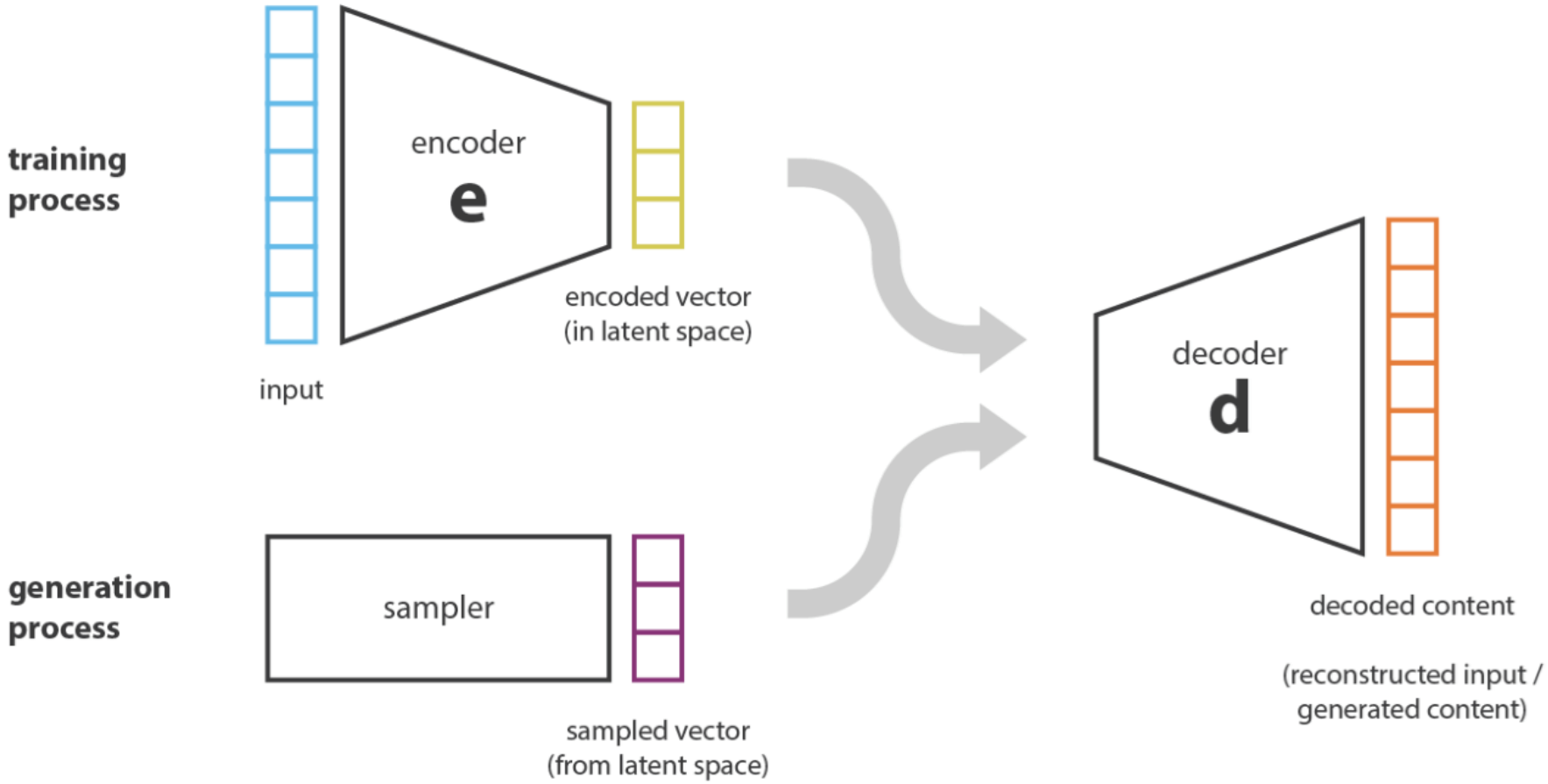
Autoencoders

- Autoencoders are a form of compression.
- *Smaller latent spaces* create l*arger bottlenecks*, forcing *greater compression*.
- The reconstruction loss forces the latent space to capture as much information about the data as possible.



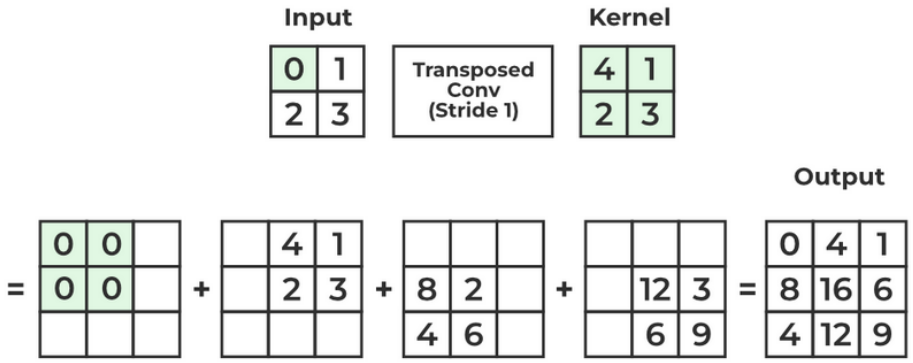
Autoencoders

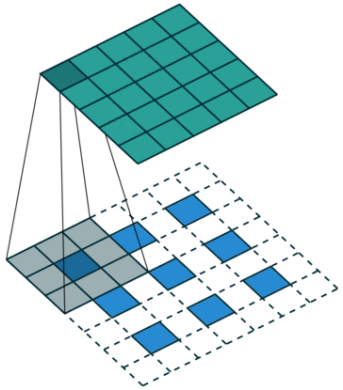
- Note that, once trained, we can use the Decoder part to, given a real vector, obtain a new dataset similar to the inputs (images, sequences, ...)



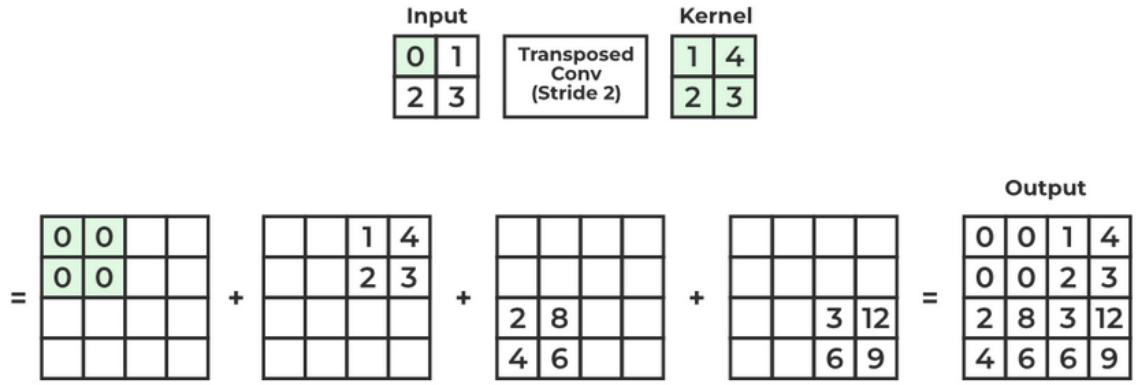
2D Transposed Convolution Operation

- Transpose convolution, stride = 1



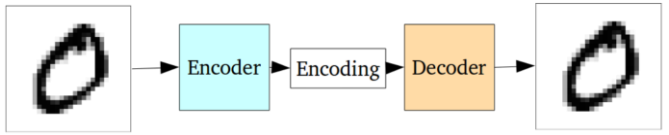


- Transpose convolution, stride = 2

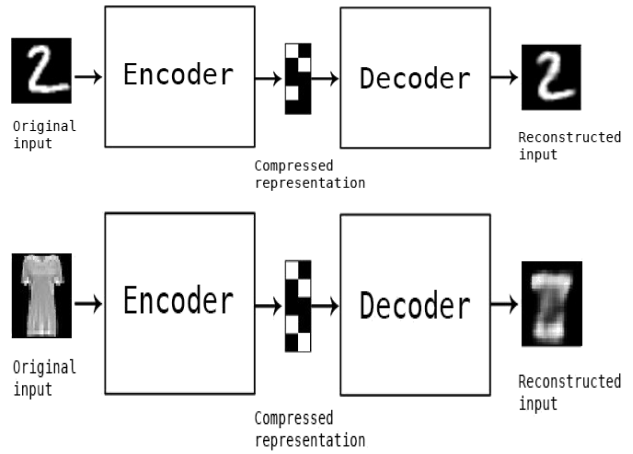


Laboratory Classes

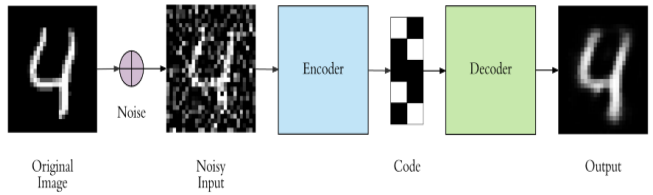
- Autoencoder with MNIST dataset
  - With MLP: 1_pyt_AE_MLP_treino_MNIST.ipynb
  - With CNN: 2_pyt_AE_CONV_treino_MNIST.ipynb



- Autoencoder with MNIST dataset to detect anomalies
  - With MLP: 3_pyt_AE_MLP_anomaly_MNIST.ipynb
  - With CNN: 4_pyt_AE_CONV_anomaly_MNIST.ipynb

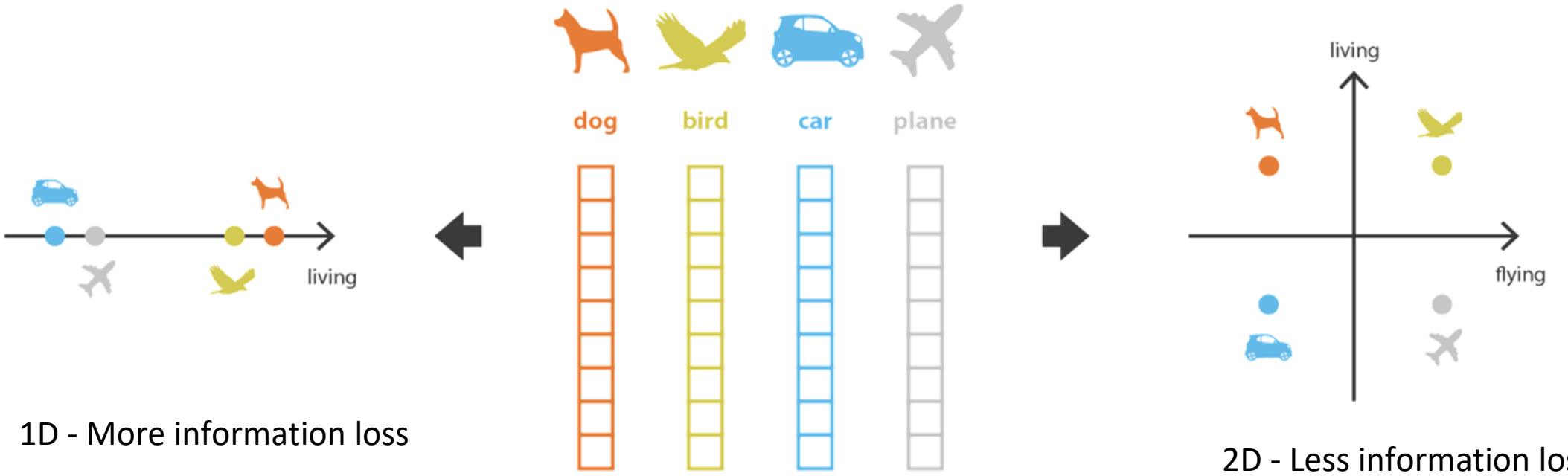


- Autoencoder with MNIST dataset to apply denoise
  - With MLP: 5_pyt_MLP_denoise_MNIST.ipynb
  - With CNN: 6_pyt_AE_CONV_denoise_MNIST.ipynb



Autoencoders

- However, the original autoencoders do not create structured (regularised) spaces that allow generating new interesting examples suffering from overfitting to the training data

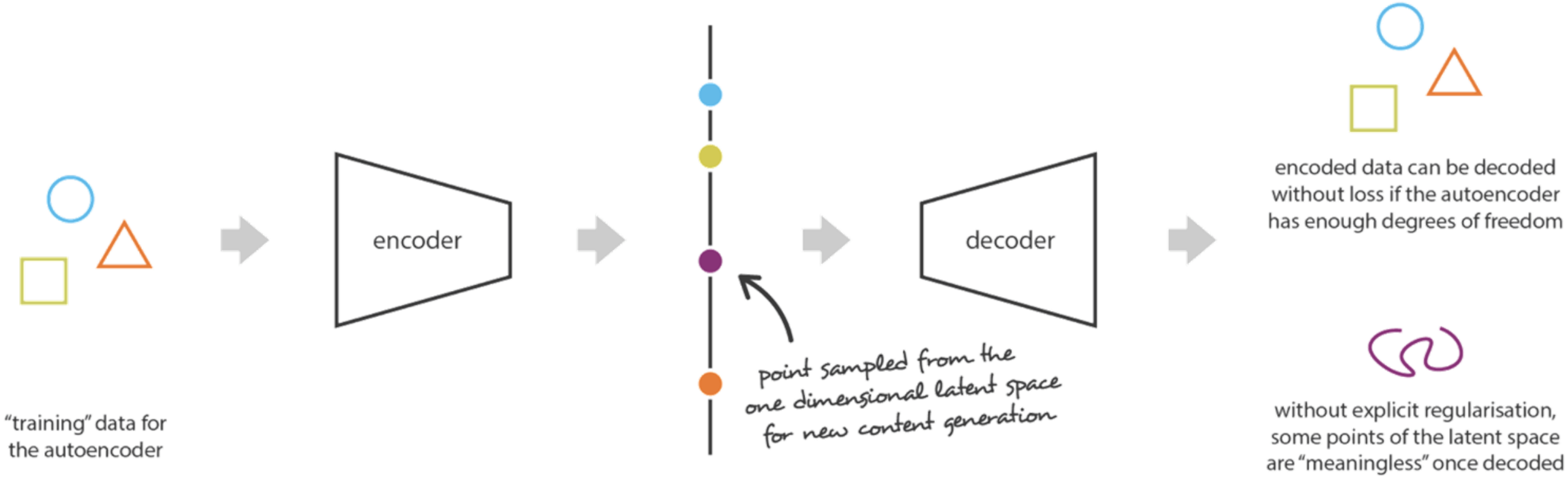


1D - More information loss

2D - Less information loss

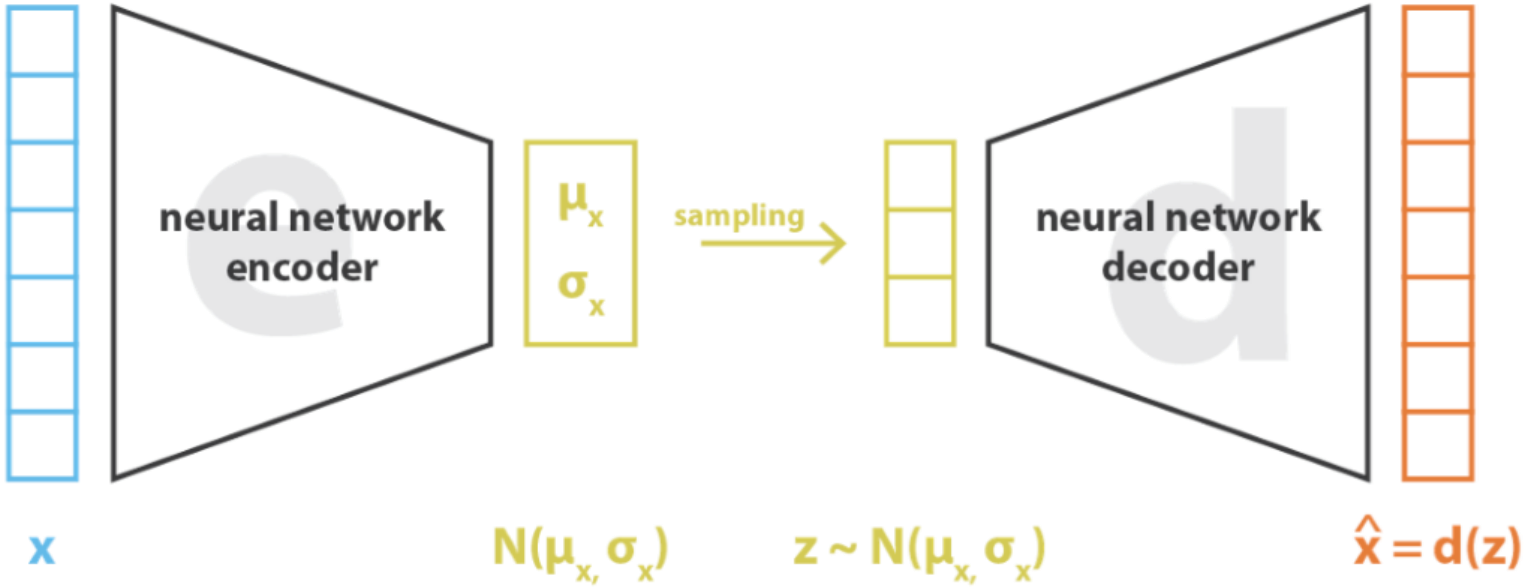
Autoencoders

- Overfitting makes generation of new examples more difficult – new data may not follow the same features as the original



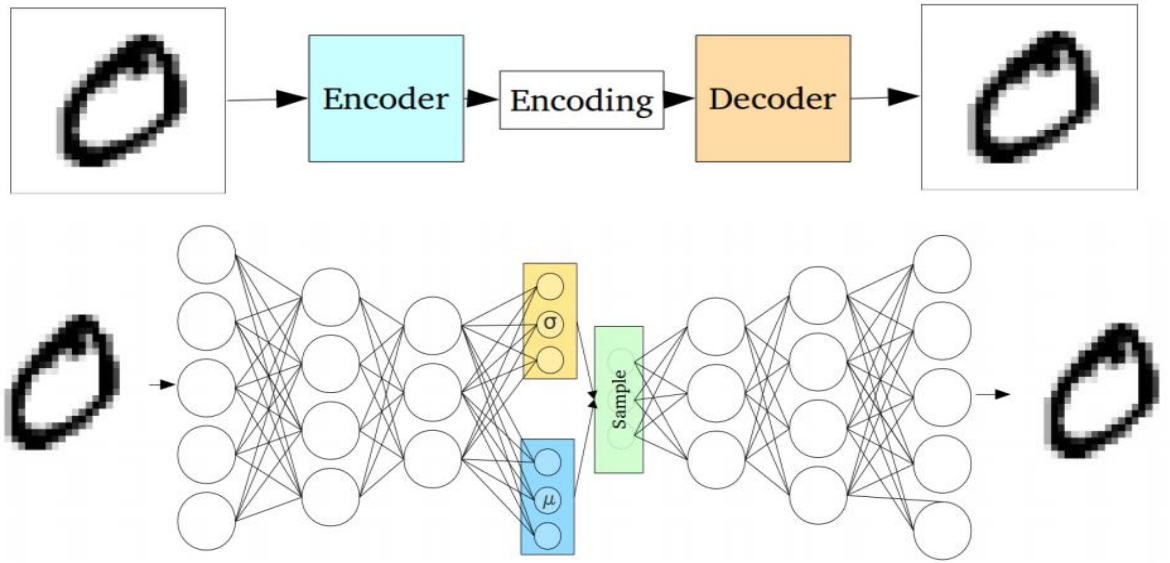
Variational Autoencoders

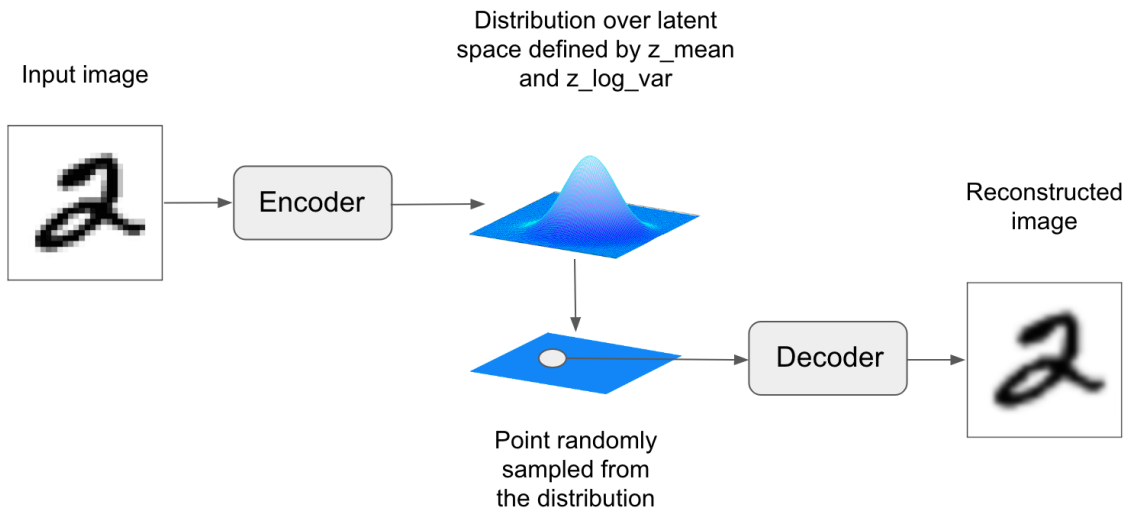
- To address the problem, constraints can be imposed on the latent representation space that is learned in order to have more organised and useful representations that can translate to relevant example generation
- **Variational Autoencoders (VAE)** - the latent representation follows a statistical distribution and the encoder maps an input example to the parameters of this distribution, minimising overfitting.



Variational Autoencoders: training

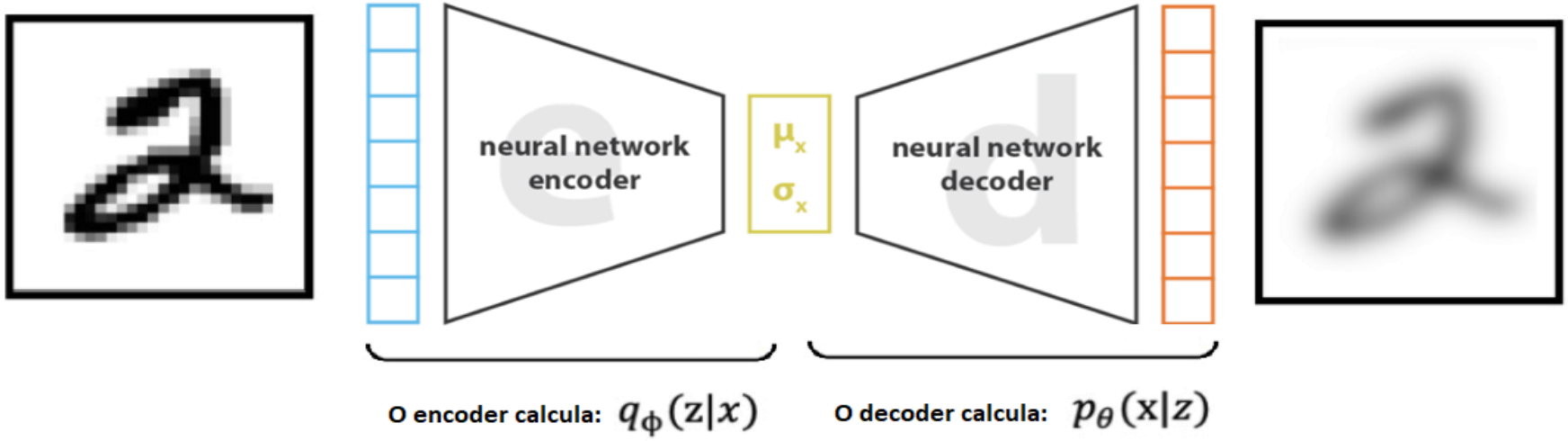
- Training a VAE model goes through the following steps:
  1. The inputs are encoded (by the encoder) as a distribution in latent space
  2. A point is sampled from this distribution
  3. This point is decoded (by the decoder); reconstruction error is calculated
  4. Reconstruction error is back-propagated





Variational Autoencoders

- **Reconstruction** - measures the distance between output and input (as in the original autoencoder)
- **Regularisation** - seeks to introduce regularity into the latent space, reducing overfitting

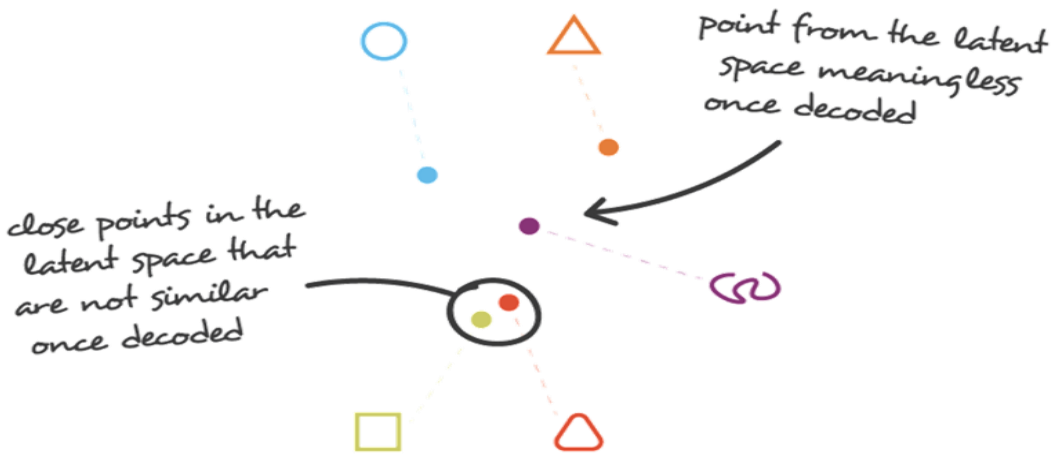


$$\mathcal{L}(\phi, \theta, x) = (\text{reconstruction loss}) + (\text{regularization term})$$

Reconstruction loss - binary_cross_entropy
Regularization – vem de KLD (Kulback-Leibler divergence) = -0.5 * torch.sum(1 + log_var - mu.pow(2) - log_var.exp())

Variational Autoencoders: “regular” space

- Regularisation allows two essential properties to be obtained in latent space:
  - **Continuity** - two nearby points in the latent space must correspond to similar examples after decoding
  - **Completeness** - for a given distribution, a sampled point in the latent space must return a valid example



irregular latent space

regular latent space

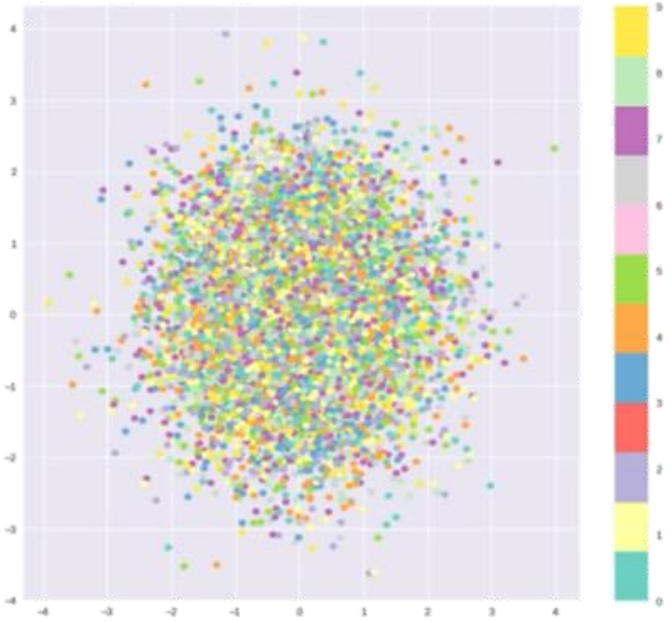
Variational Autoencoders:“regular” space

- KL (Kulback-Leibler) divergence (calculates the difference between the distributions)

$$D\left(q_\phi(z|x) \parallel p(z)\right)$$

inferred distribution for the latent variables

prior distribution imposed on the latent variables



Most commonly used prior distribution:

Standard Normal Distribution

$$p(z) = \mathcal{N}(\mu = 0, \sigma^2 = 1)$$

Encourage the encoder to distribute the encodings around the centre of the latent space;

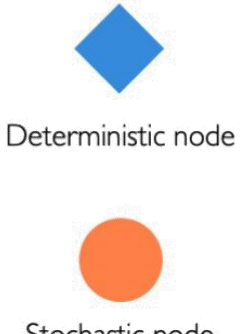
Penalise the network when it tries to join points in specific regions in an attempt to memorise the data.

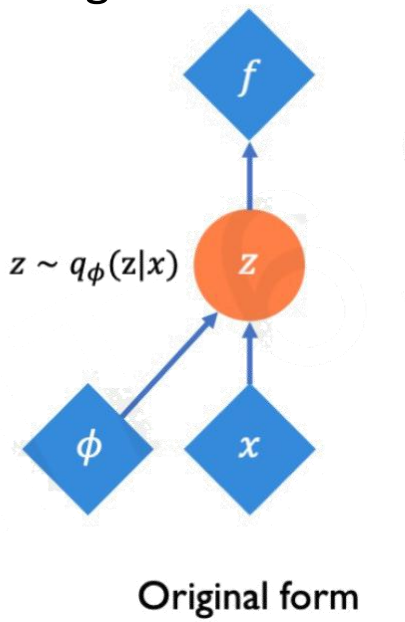
Variational Autoencoders: “regular” space

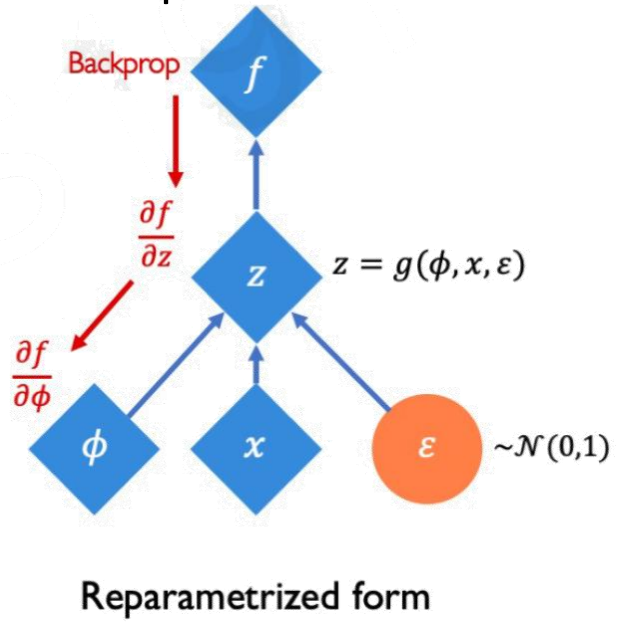
- **Encoder module**: processes the input data to calculate two representation parameters: mean and logarithm of the variance (for each dimension of the latent state)
- **Decoder module**: generates new examples from a vector in the latent state (as with the autoencoder)
- To allow back-propagation during training, randomness is removed from sampling the latent space, generating a parameter ε, and calculating variables **z** of the latent space as:

$$z = \mu + \sigma \odot \varepsilon$$

```
def sampling(self, mu, log_var):
    #param mu: mean from the encoder's latent space
    #param log_var: log variance from the encoder's latent space
    std = torch.exp(0.5*log_var) # standard deviation
    eps = torch.randn_like(std) # `randn_like` as we need the same size
    sample = mu + (eps * std) # sampling
    #sample = eps.mul(std).add_(mu) # alternativa
    return sample
```

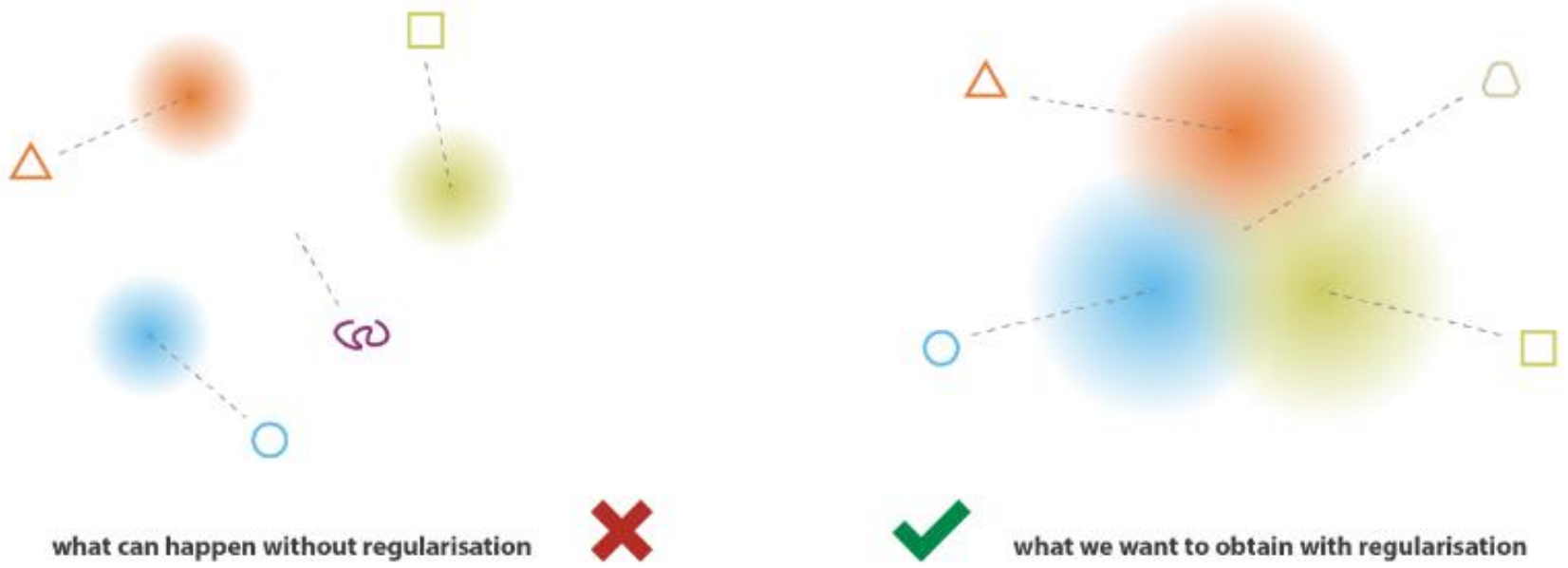






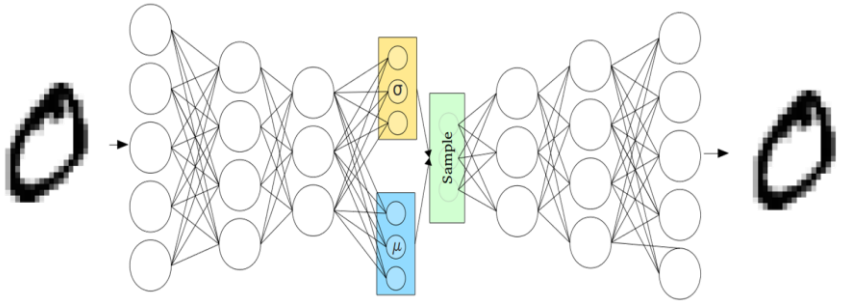
Variational Autoencoders

- Ideally, codings should be as close as possible to each other but distinct, thus allowing smooth interpolation in the construction of new samples



Laboratory Classes

- Variational Autoencoder with MNIST dataset
  - With MLP: 7_pyt_VAE_MLP_treino_MNIST.ipynb
  - With CNN: 8_pyt_VAE_CONV_treino_MNIST.ipynb



- Autoencoder MNIST dataset with clustering
  - 9_pyt_AE_MLP_clustering_MNIST.ipynb

- Variational Autoencoder MNIST dataset with clustering
  - 10_pyt_VAE_MLP_clustering_MNIST.ipynb