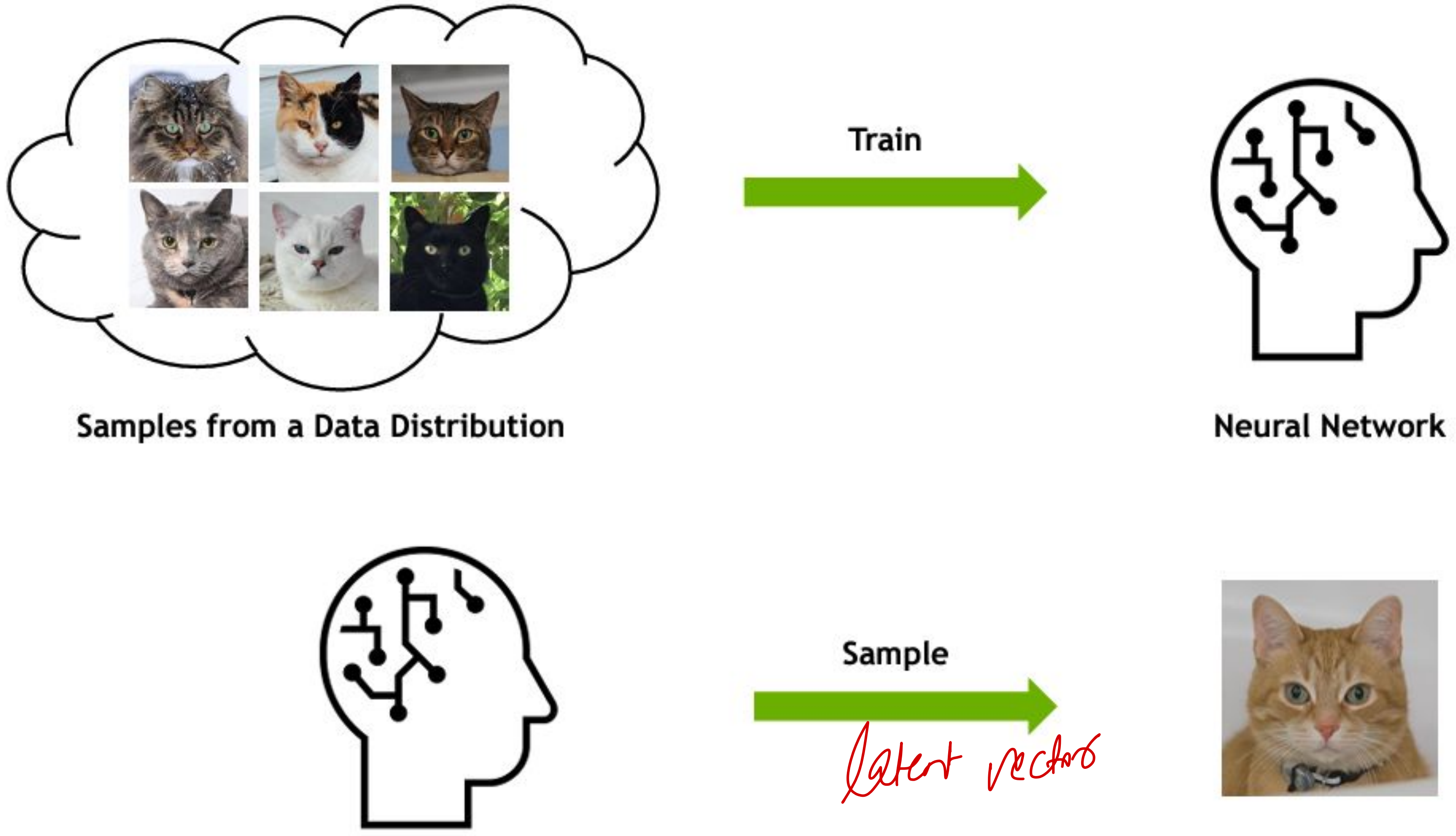
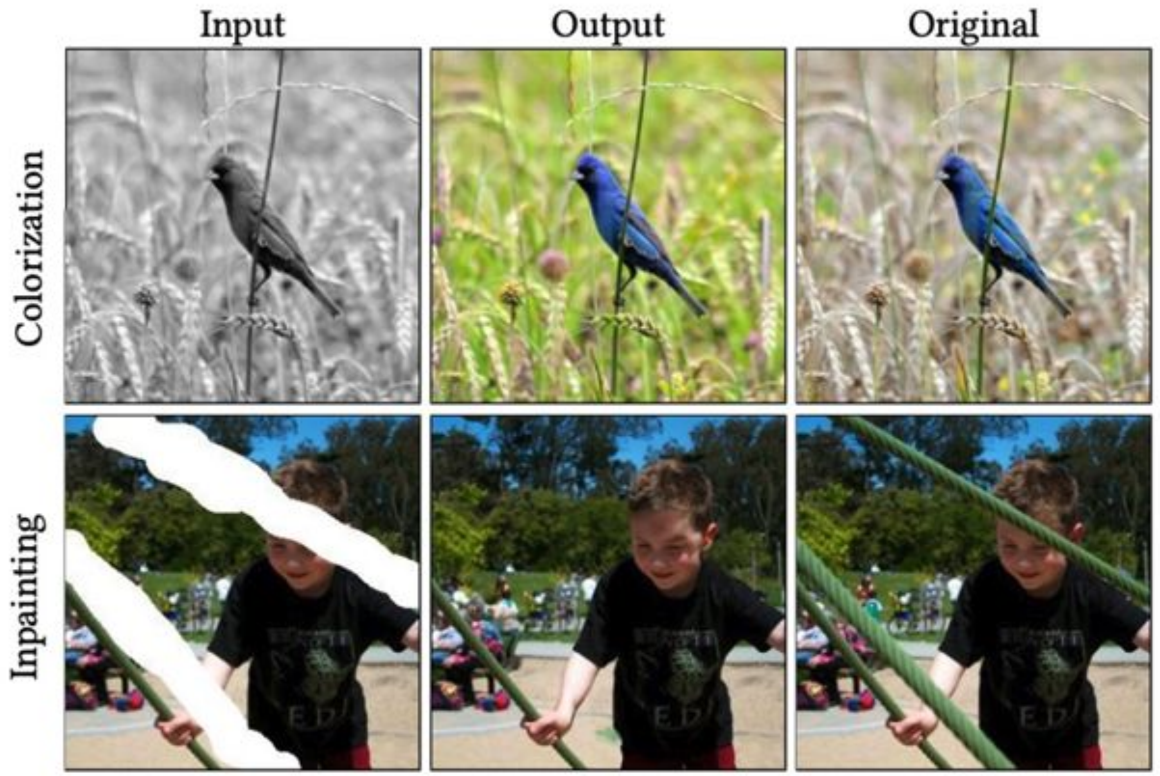
Deep Generative Modeling via VAEs

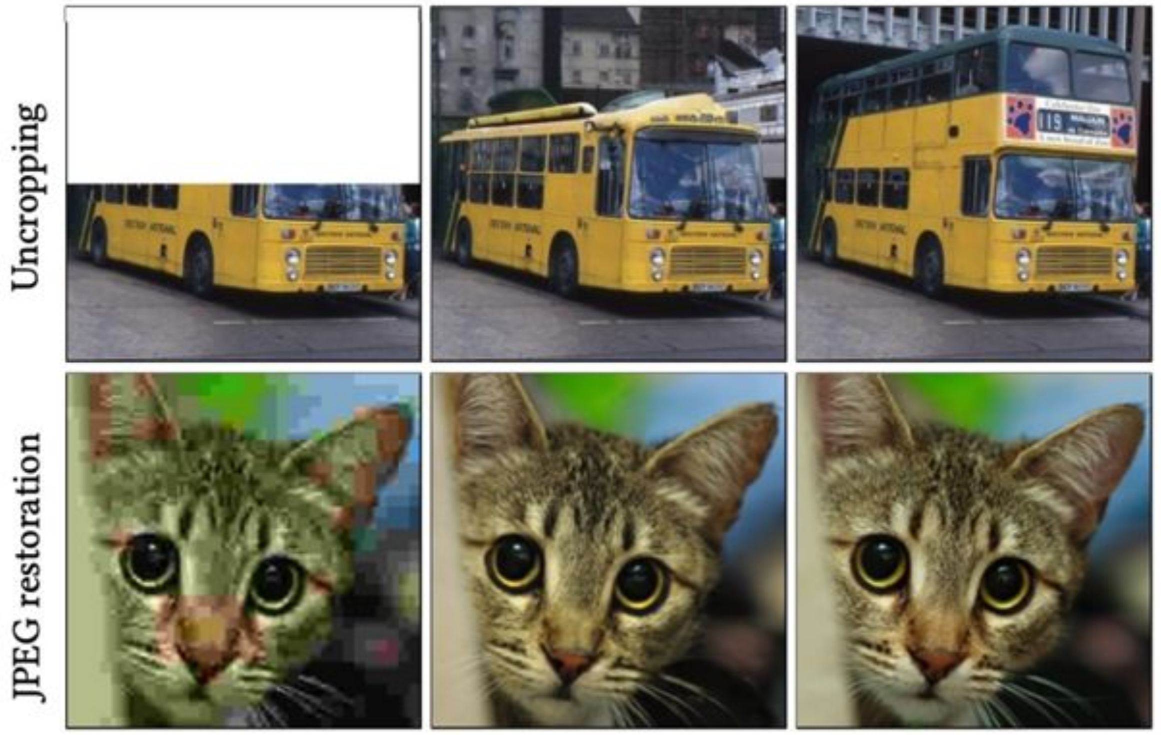
Deep Learning (CS60010)

Deep Generative Models -- *learning to generate data*

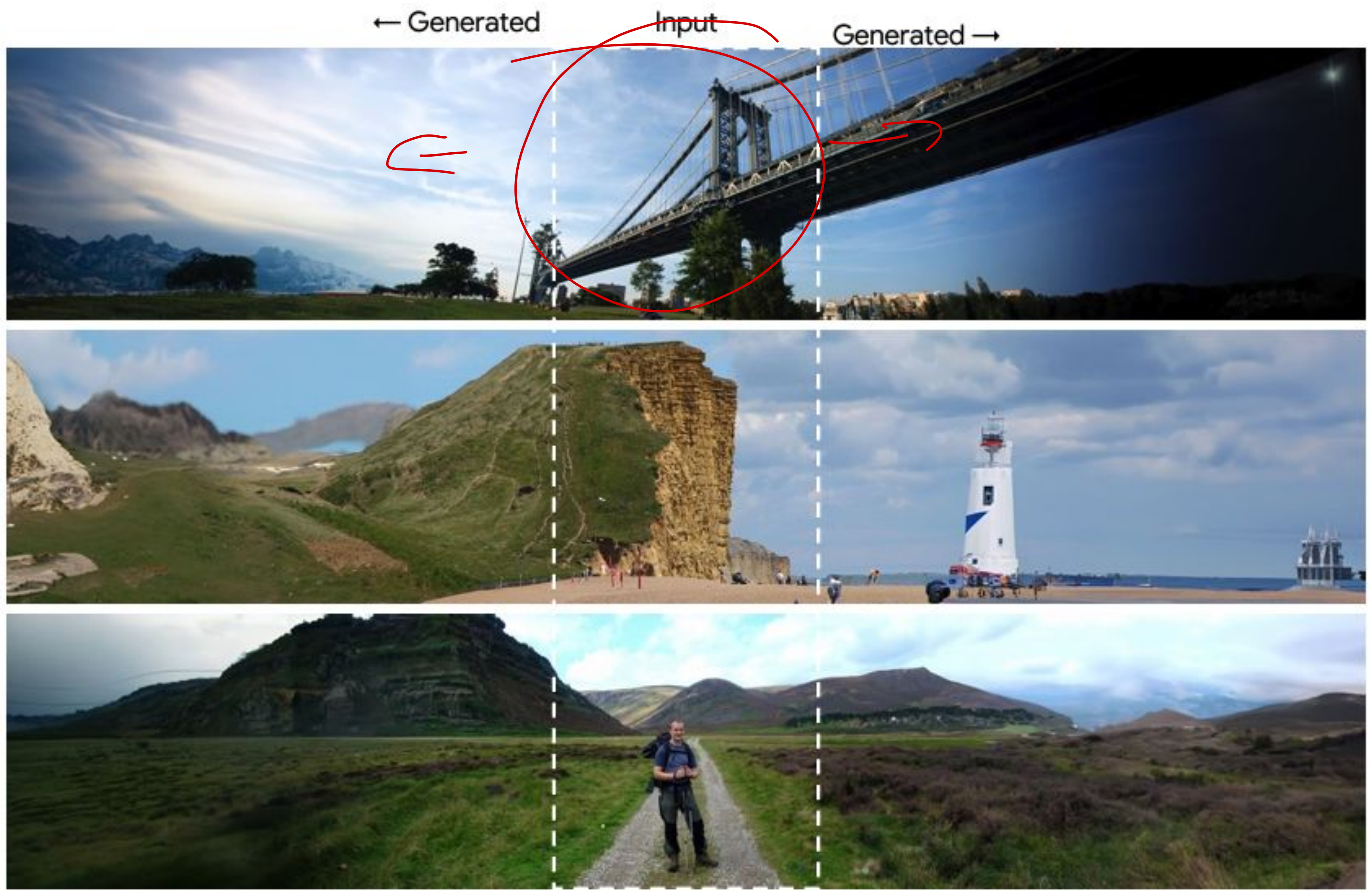


Applications: Colorization, Inpainting, Restoration





Applications: Outfilling



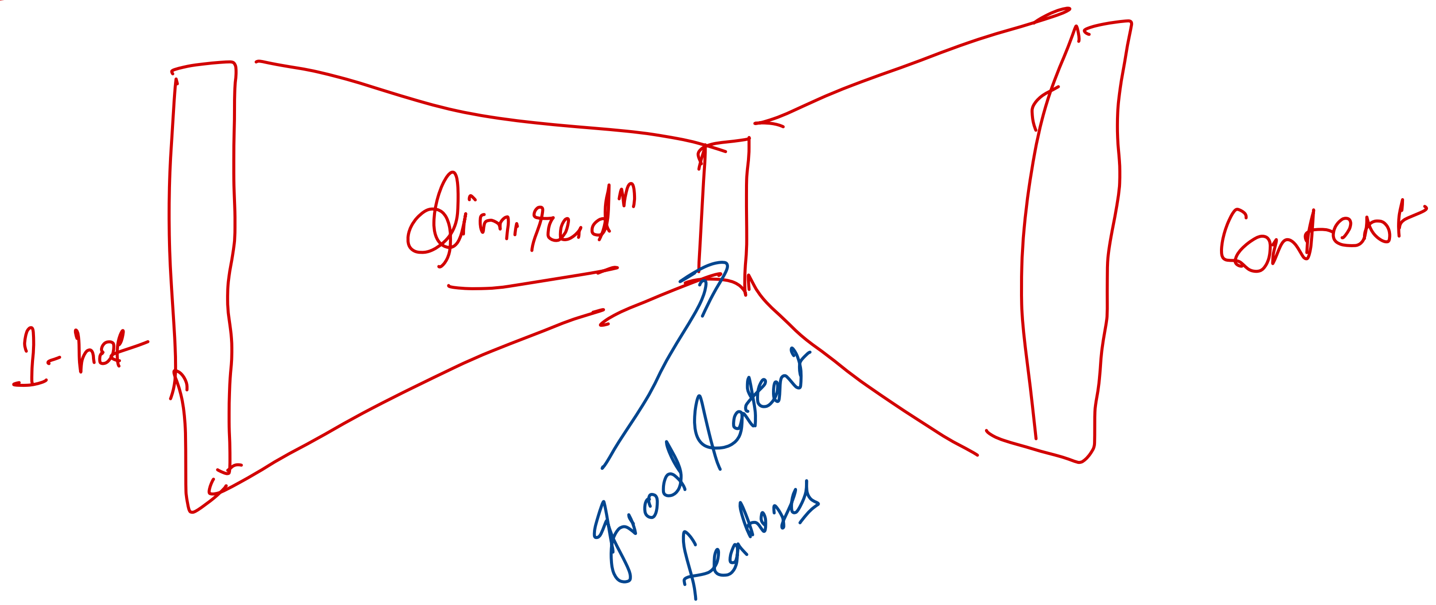
Variational Autoencoders (VAEs)

- VAE is an *autoencoder* whose encodings distribution is regularised during the training in order to ensure that its latent space has good properties allowing us to *generate some new data*.

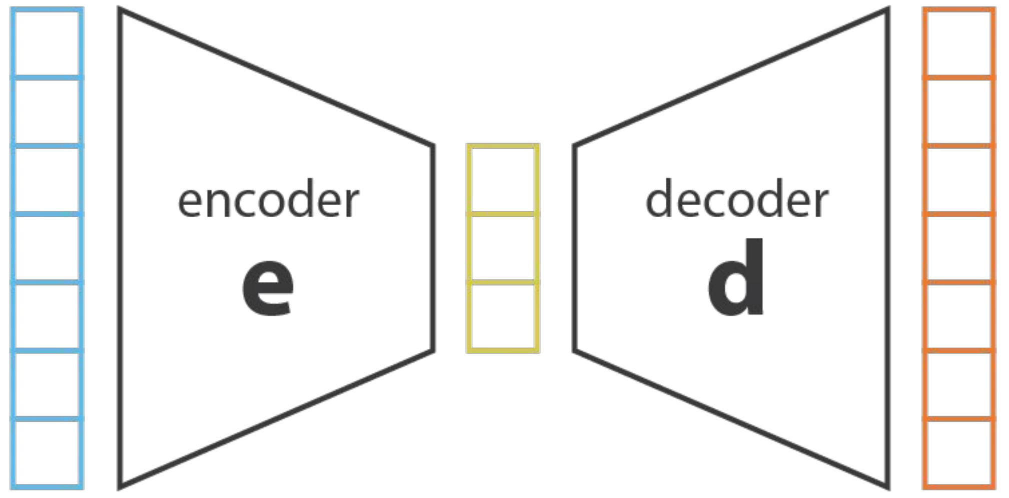
- *What is an autoencoder?*

What is an autoencoder?

- Basic Idea: Representation learning through dimensionality reduction
- Has an encoder and a decoder
- let's call **encoder** the process that produce the "latent features" representation from the "raw features" representation
- **decoder** is the reverse process



What is an autoencoder?



x | $e(x)$ | $\boxed{d(e(x))}$

initial data
in space $\mathbb{R}^n$

encoded data
in latent space $\mathbb{R}^m$ (with $m<n$)

encoded-decoded data
back in the initial space $\mathbb{R}^n$

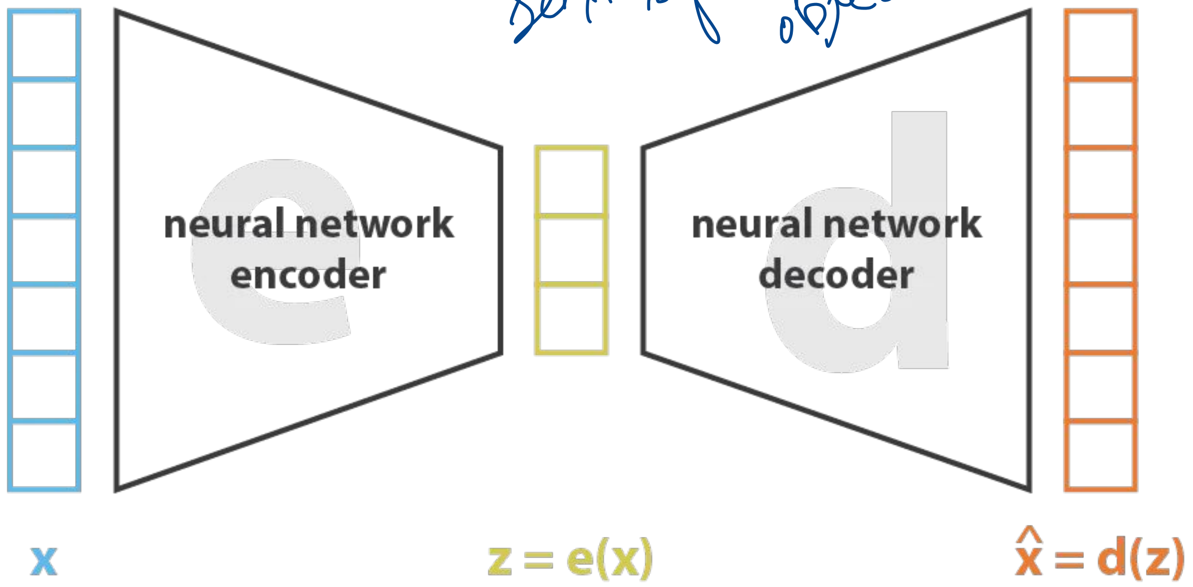
$x = d(e(x))$ ➡ **lossless encoding**
no information is lost when reducing the number of dimensions

$x \neq d(e(x))$ ➡ **lossy encoding**
some information is lost when reducing the number of dimensions and can't be recovered later

Autoencoder objective

- For a given set of possible encoders and decoders, we are looking for the pair that **keeps the maximum of information when encoding**.
- **So, it has the minimum of reconstruction error when decoding**.

Self-supervised objective

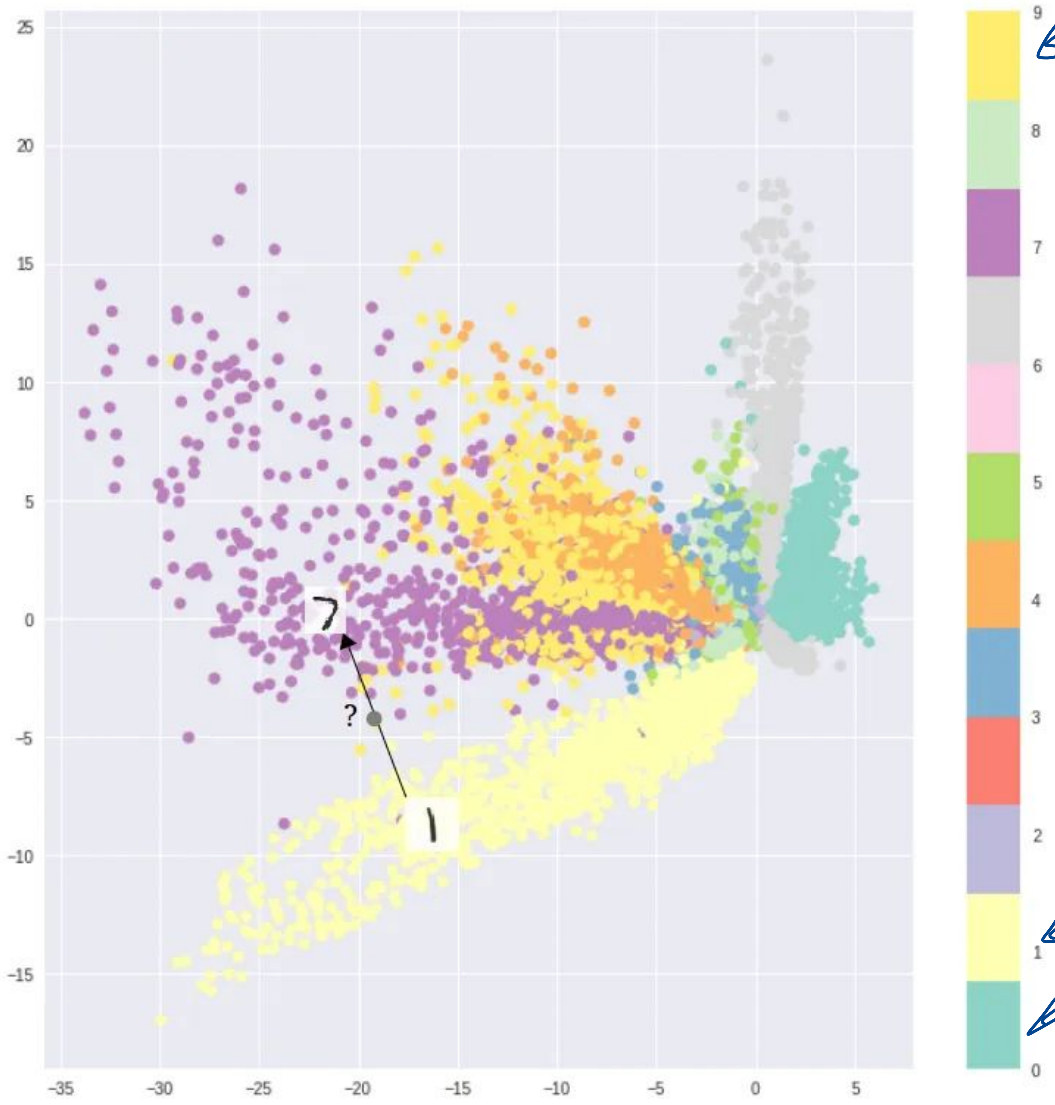


The overall autoencoder architecture creates a bottleneck for data that ensures only the main information can go through and be reconstructed.

Helps in representation learning!

$$\text{loss} = \| x - \hat{x} \|^2 = \| x - d(z) \|^2 = \| x - d(e(x)) \|^2$$

But can autoencoders generate new content?



Latent space for a trained autoencoder on MNIST dataset

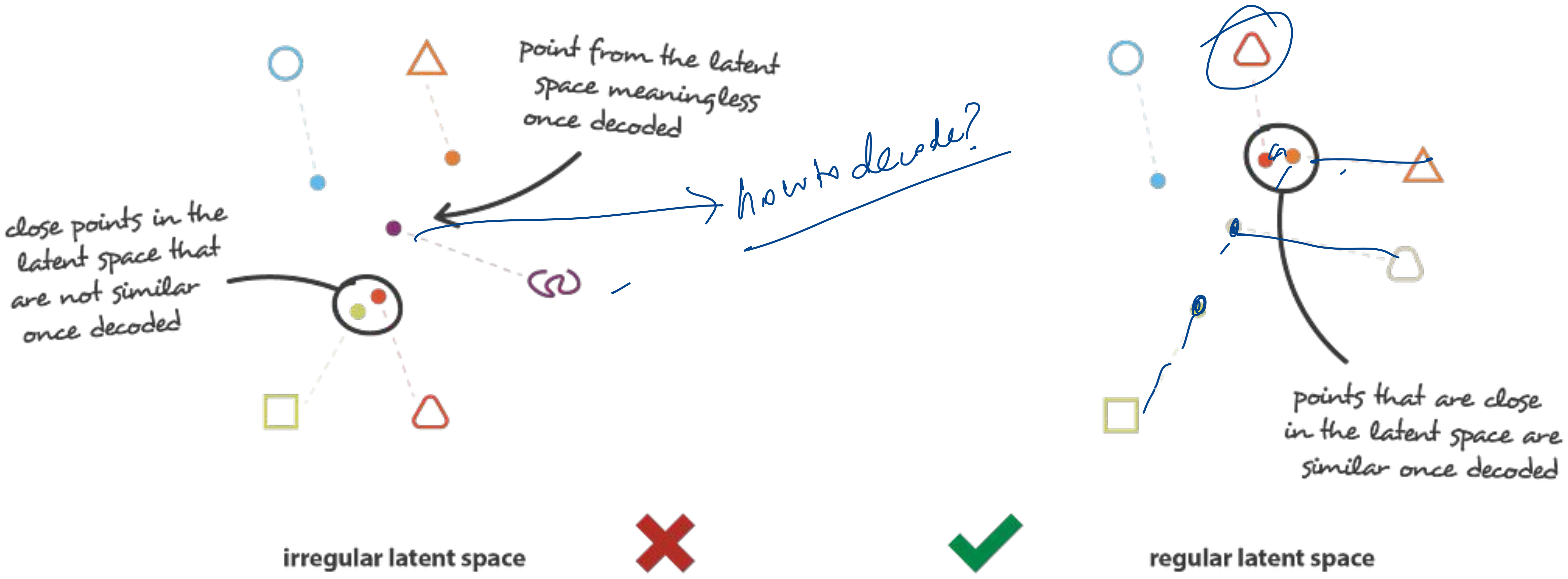
Observation: formation of distinct clusters.

This makes sense, as distinct encodings for each image type makes it far easier for the decoder to decode them. This is fine if you're just *replicating* the same images.

But why would this not work for generating new content?

If the space has discontinuities (eg. gaps between clusters) and you sample/generate a variation from there, the decoder will simply generate an unrealistic output, because the decoder has no idea how to deal with that region of the latent space.

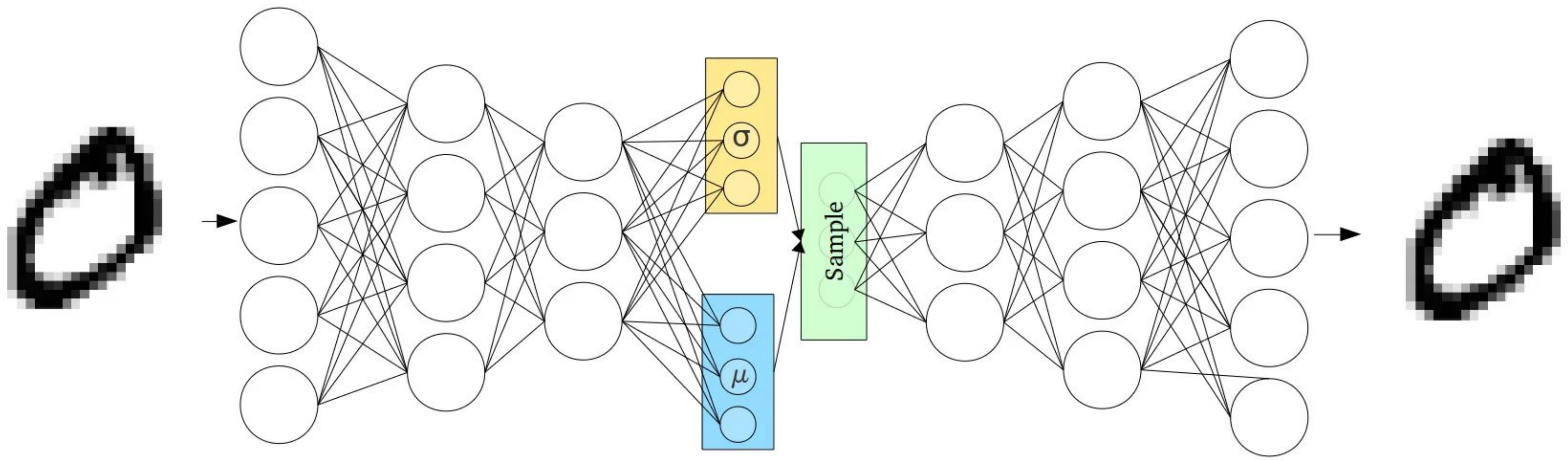
Desired properties of latent space: continuity and completeness



point from the latent space meaningless once decoded
close points in the latent space that are not similar once decoded
how to decode?
points that are close in the latent space are similar once decoded
irregular latent space
regular latent space

How to make the latent space continuous?

Variational Autoencoders

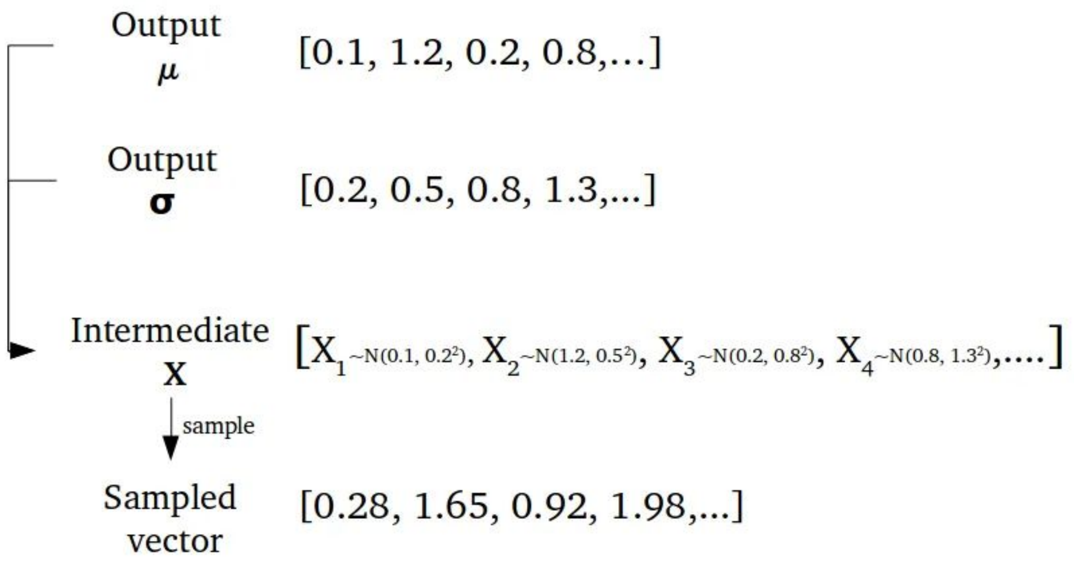


their latent spaces are, *by design*, continuous, allowing easy random sampling and interpolation.

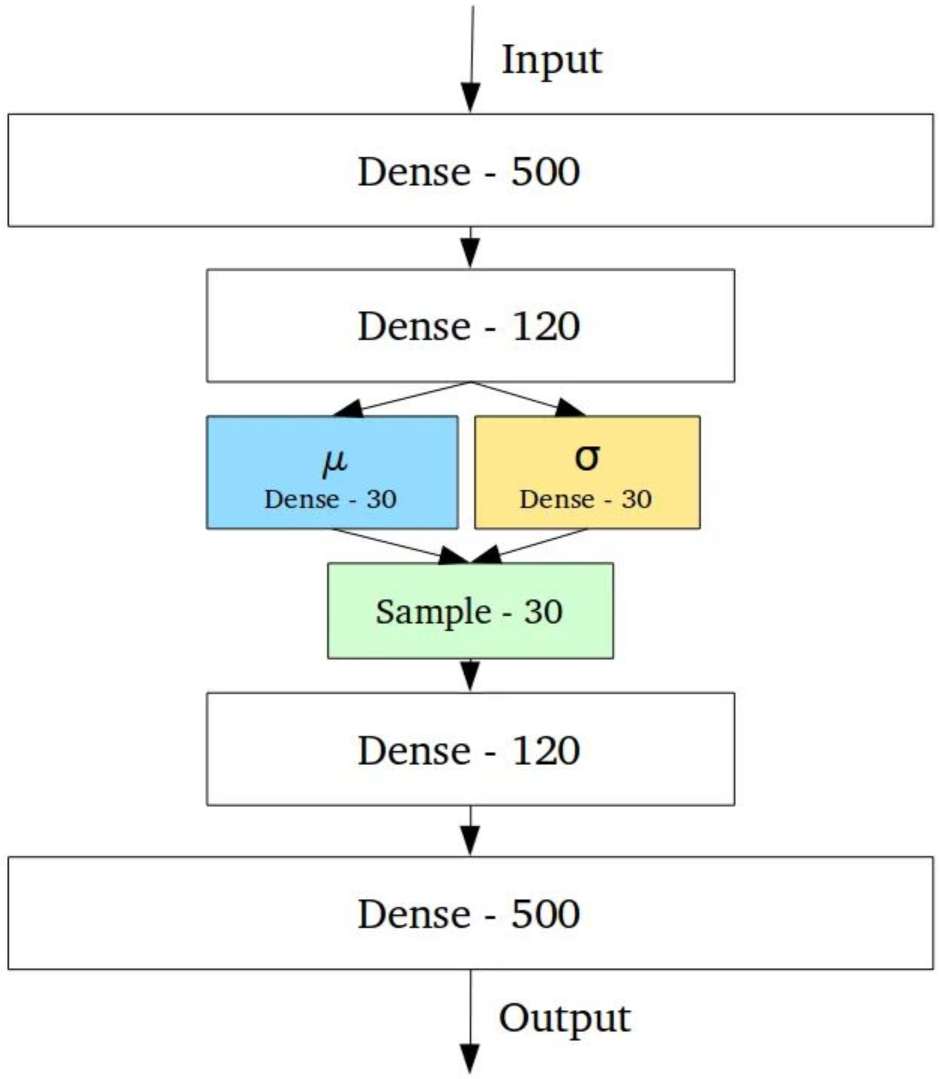
VAEs: Towards continuous latent space

Their latent spaces are, *by design,* continuous, allowing easy random sampling and interpolation.

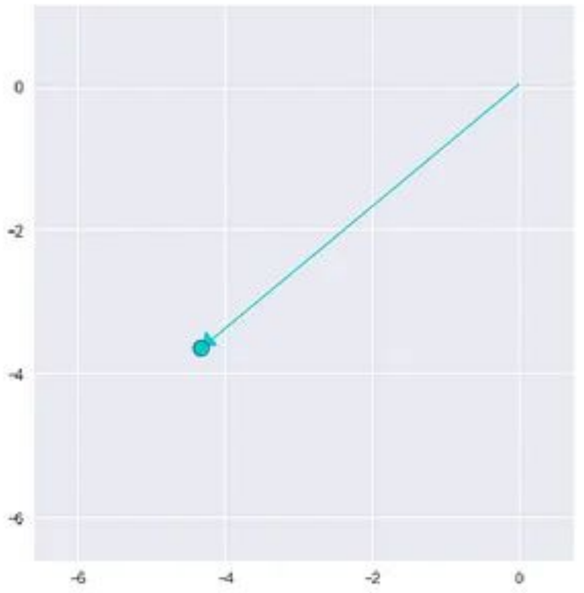
It achieves this by making its encoder not output an encoding vector of size n, rather, outputting two vectors of size n: a vector of means, **μ**, and another vector of standard deviations, **σ**.



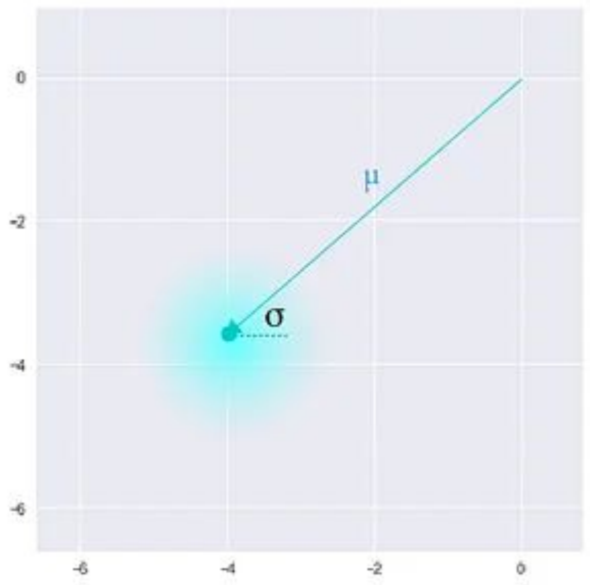
Output μ
[0.1, 1.2, 0.2, 0.8,…]
Output $\boldsymbol{\sigma}$
[0.2, 0.5, 0.8, 1.3,...]
Intermediate X
$[X_{1 \sim N(0.1,\,0.2^2)}, X_{2 \sim N(1.2,\,0.5^2)}, X_{3 \sim N(0.2,\,0.8^2)}, X_{4 \sim N(0.8,\,1.3^2)}, \ldots.]$
sample
Sampled vector
[0.28, 1.65, 0.92, 1.98,...]



How does it help?



Standard Autoencoder
(direct encoding coordinates)



Variational Autoencoder
(μ and σ initialize a probability distribution)

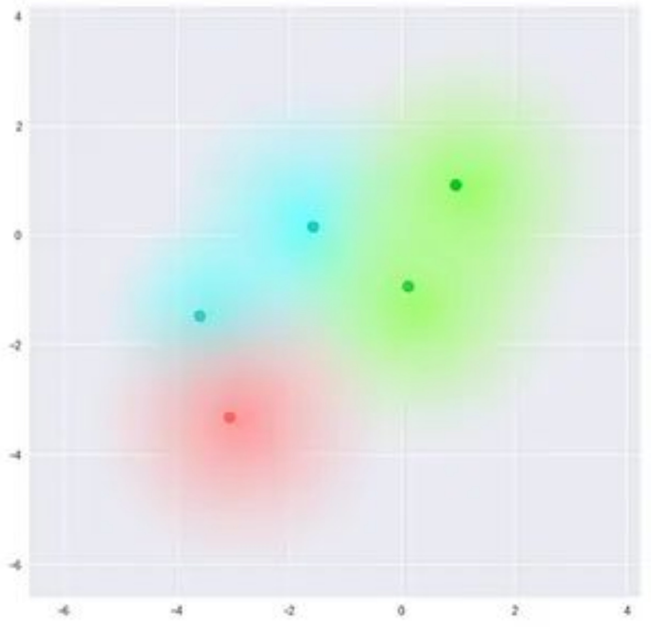
Intuitively, the mean vector controls where the encoding of an input should be centered around, while the standard deviation controls the “area”, how much from the mean the encoding can vary.

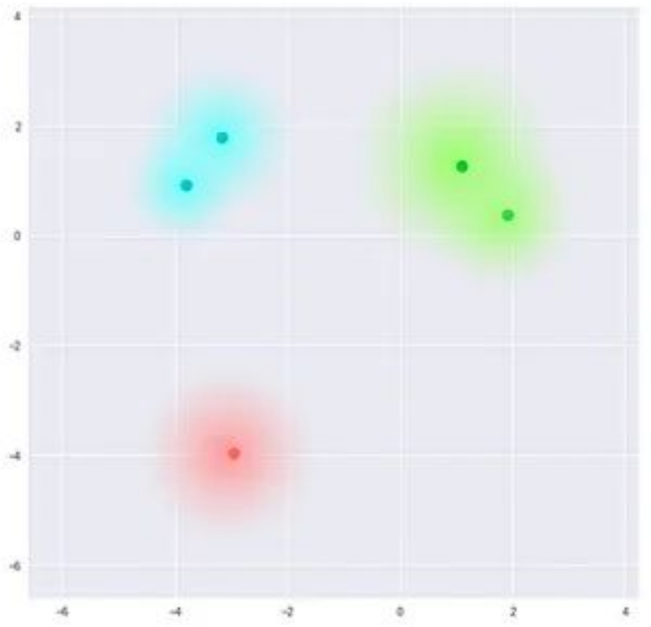
As encodings are generated at random from anywhere inside the “circle” (the distribution), the decoder learns that not only is a single point in latent space referring to a sample of that class, but all nearby points refer to the same as well.

This allows the decoder to not just decode single, specific encodings in the latent space (leaving the decodable latent space discontinuous), but ones that slightly vary too, as the decoder is exposed to a range of variations of the encoding of the same input during training.

Ok, but is it just local smoothing?

- Ideally, we want overlap between samples that are not very similar too, in order to interpolate *between* classes.
- However, since there are *no limits* on what values vectors **μ** and **σ** can take on, the encoder can learn to generate very different **μ** for different classes, clustering them apart,
- and minimize **σ**, making sure the encodings themselves don't vary much for the same sample (that is, less uncertainty for the decoder).
- This allows the decoder to efficiently reconstruct the *training* data.


What we require


What we may inadvertently end up with

How can we achieve global smoothness?

- What we ideally want are encodings, *all* of which are as close as possible to each other while still being distinct, allowing smooth interpolation, and enabling the construction of *new* samples.
- In order to force this, we introduce the Kullback–Leibler divergence into the loss function.
- For VAEs, the KL loss is equivalent to the *sum* of all the KL divergences between the *component* $X_i \sim N(\mu_i, \sigma_i^2)$ in **X**, and the standard normal. It's minimized when $\mu_i = 0$, $\sigma_i = 1$.
- Intuitively, this loss encourages the encoder to distribute all encodings (for all types of inputs, eg. all MNIST numbers), evenly around the center of the latent space. If it tries to "cheat" by clustering them apart into specific regions, away from the origin, it will be penalized.