Autoencoders, anomaly detection and generative models

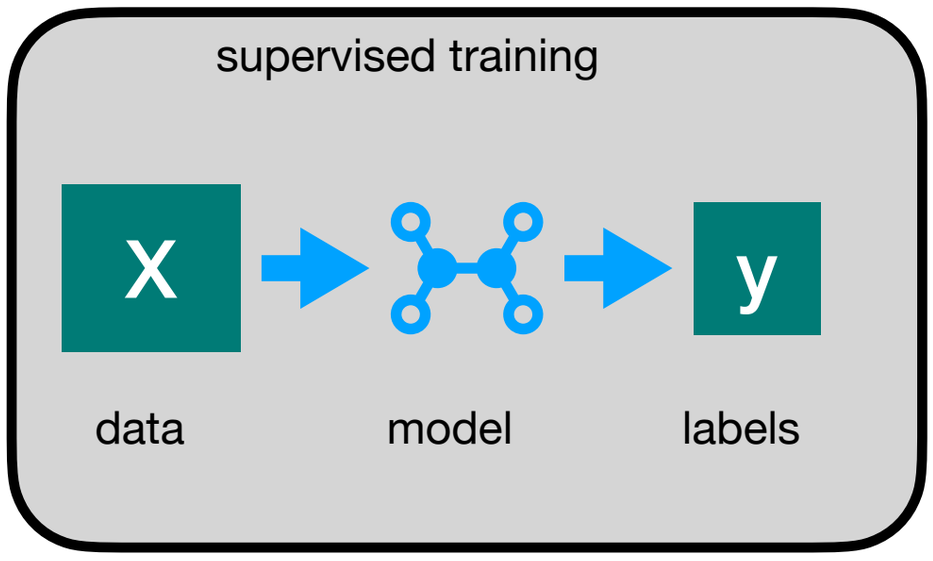
Inar Timiryasov (NBI)
inar.timiryasov@nbi.ku.dk

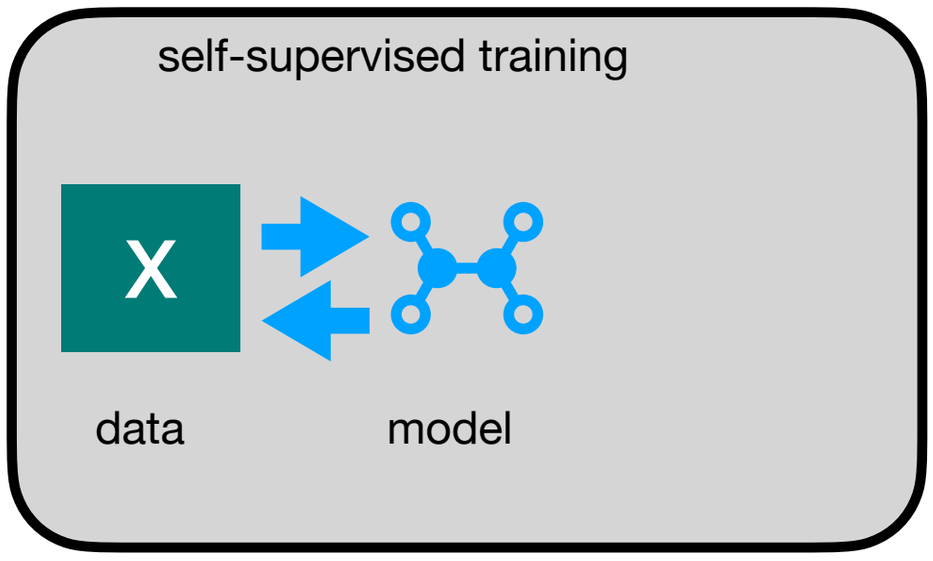
May 13, 2024

Supervised and self-supervised learning

- Supervised learning: we have data and labels. The model is trained to predict the labels.

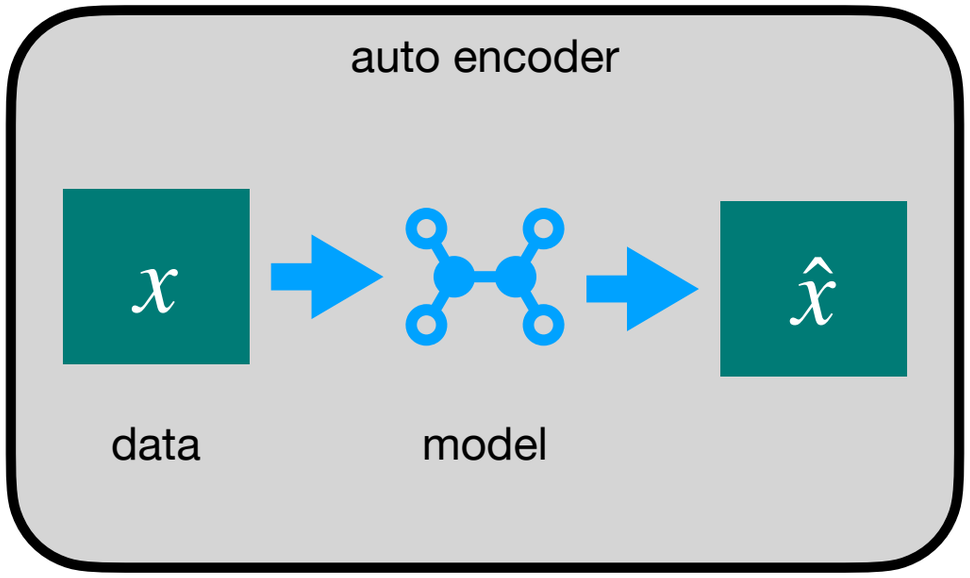
- Can we do without the labels?
  Self-supervised learning.





Autoencoders

- Encode data into a *latent representation z*
- Decode from the latent representation
- Train the model to match inputs with outputs
- How to avoid trivial copying? Introduce a constraint:
  - Restrict the dimensionality of z
  - Corrupt input
  - Require scarcity of z



Autoencoder: single hidden layer

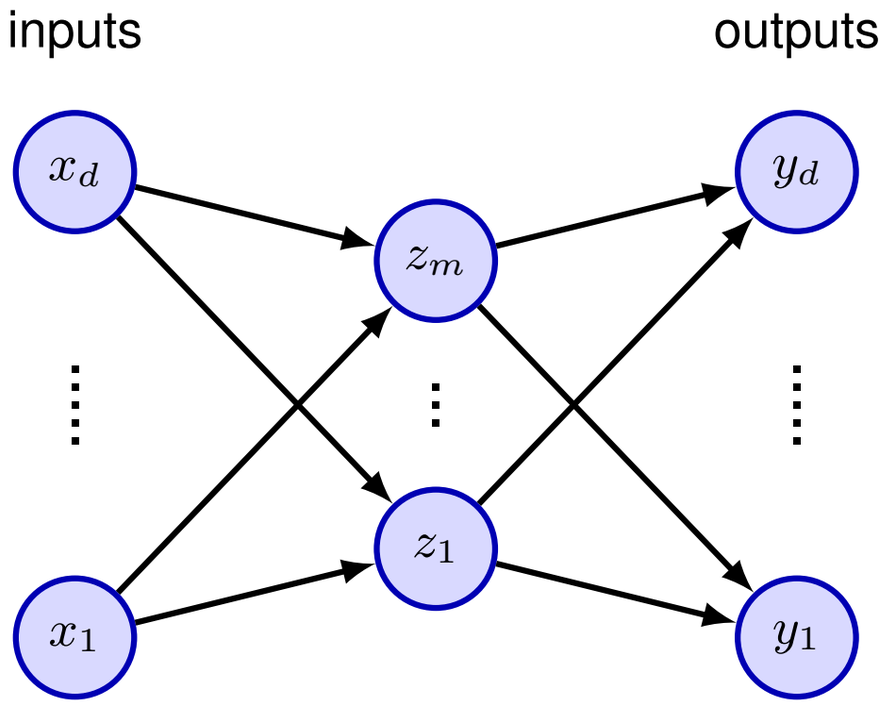


Image source: Bishop Bishop
https://www.bishopbook.com/

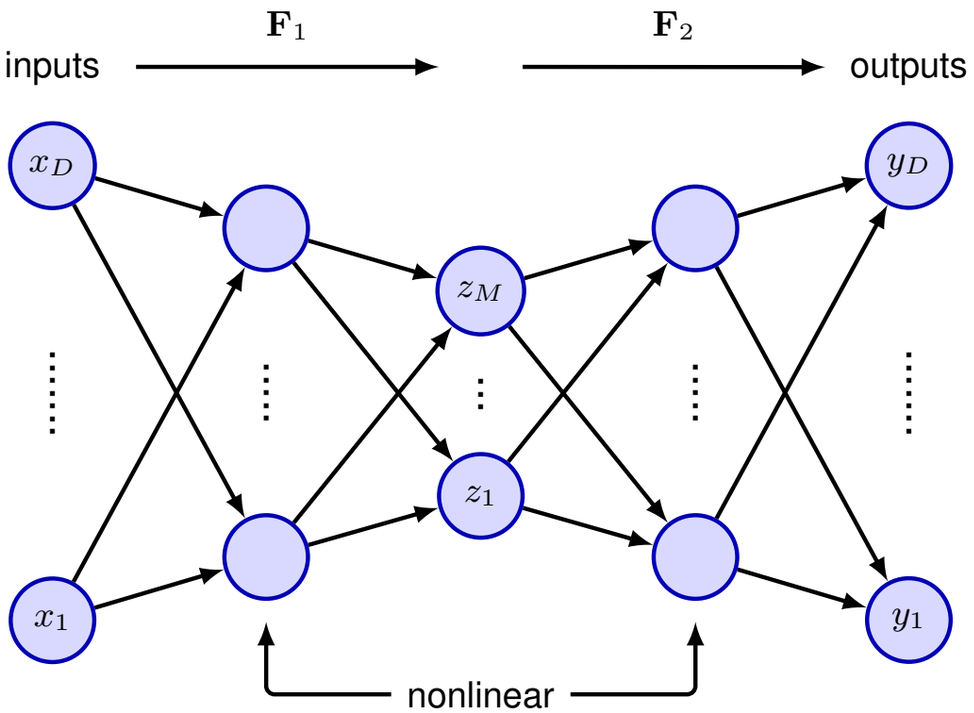
- Loss function: MSE

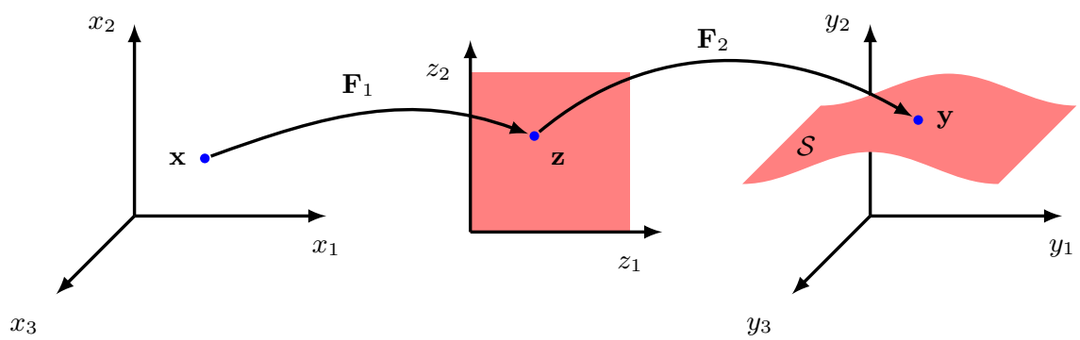
$$\mathscr{L} = \frac{1}{2} \sum_n \|y(x_n) - x_n\|^2$$

- With a single hidden layer the model effectively performs principal component analysis (PCA) (with and without non-linearity)

- Better use SGD based PCA

Deep Autoencoders





- Loss function: MSE

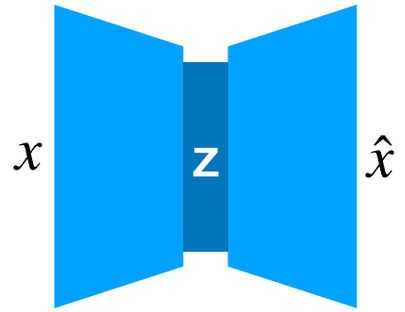
$$\mathscr{L} = \frac{1}{2} \sum_n \|y(x_n) - x_n\|^2$$

- This model is much more powerful

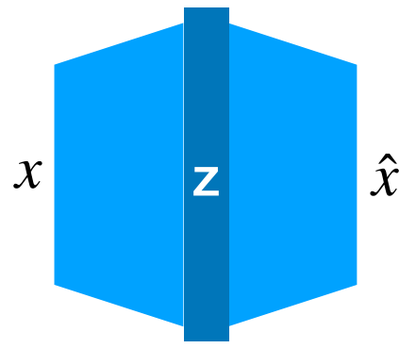
- It maps inputs into the latent space in a non-linear way

Different types of Autoencoders

- **Restricted dimensionality** of the latent space (bottleneck)

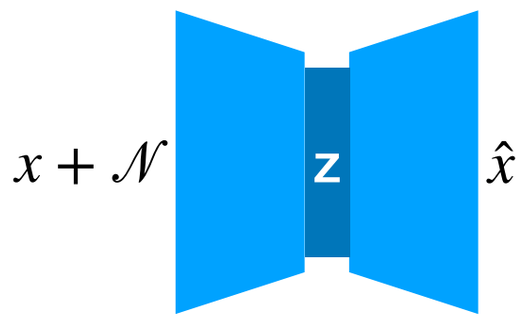


- **Space autoencoders**: sparcity of the latent space
  $\mathscr{L} = \frac{1}{2}\sum_n \|y(x_n) - x_n\|^2 + \lambda \sum k\,|z_k|$, where k is the index in the latent space.
  Latent space has larger dimensionality than the input.
  This is useful for interpretability (currently actively used to understand LLMs like GPT)



- **Denoising autoencoders**
  Input is corrupted by noise

- **Masked autoencoders**
  Part of input is hidden

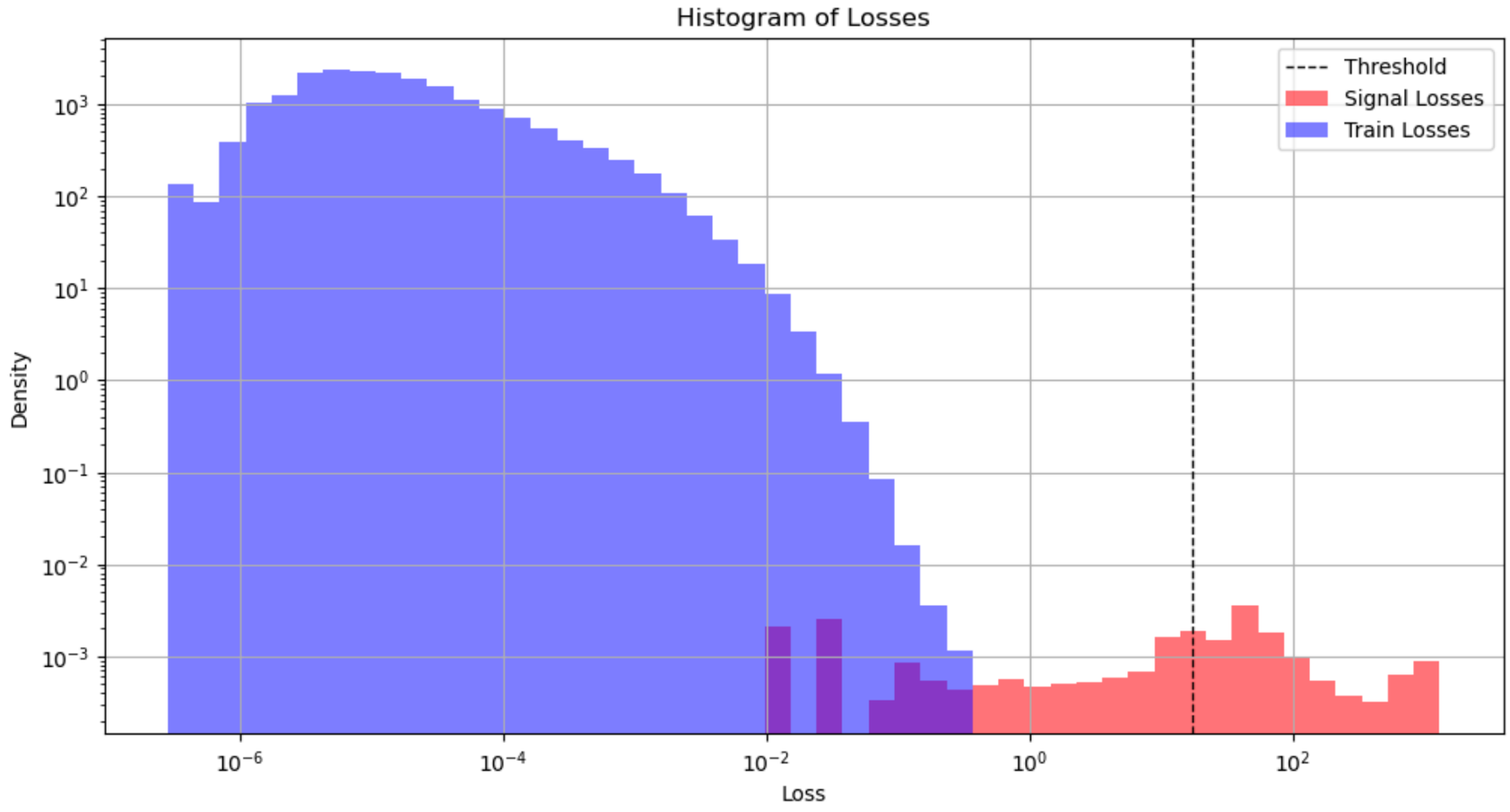


Deep Autoencoders for Anomaly Detection

- The model learns to compress and decompress the data
- To do this efficiently it needs to learn a representation of the data
- If a sample s is out of distribution, the model will struggle to reconstruct it. As a result the loss function for this sample will be large: $\mathscr{L}_s = \frac{1}{2}\|y(x_s) - x_s\|^2$

Deep Autoencoders for Anomaly Detection

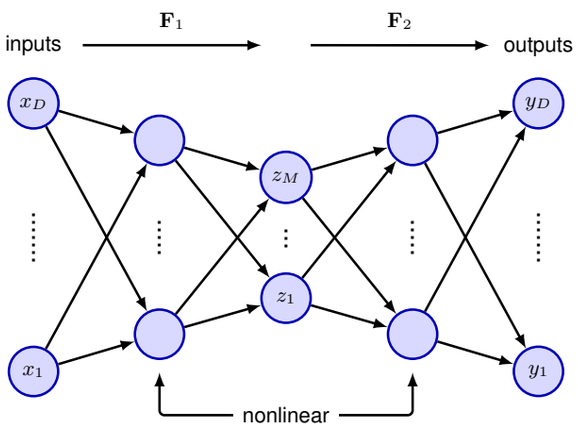
- Per-sample loss $\mathscr{L}_s = \frac{1}{2}\|y(x_s) - x_s\|^2$ can be used to identify anomalous examples



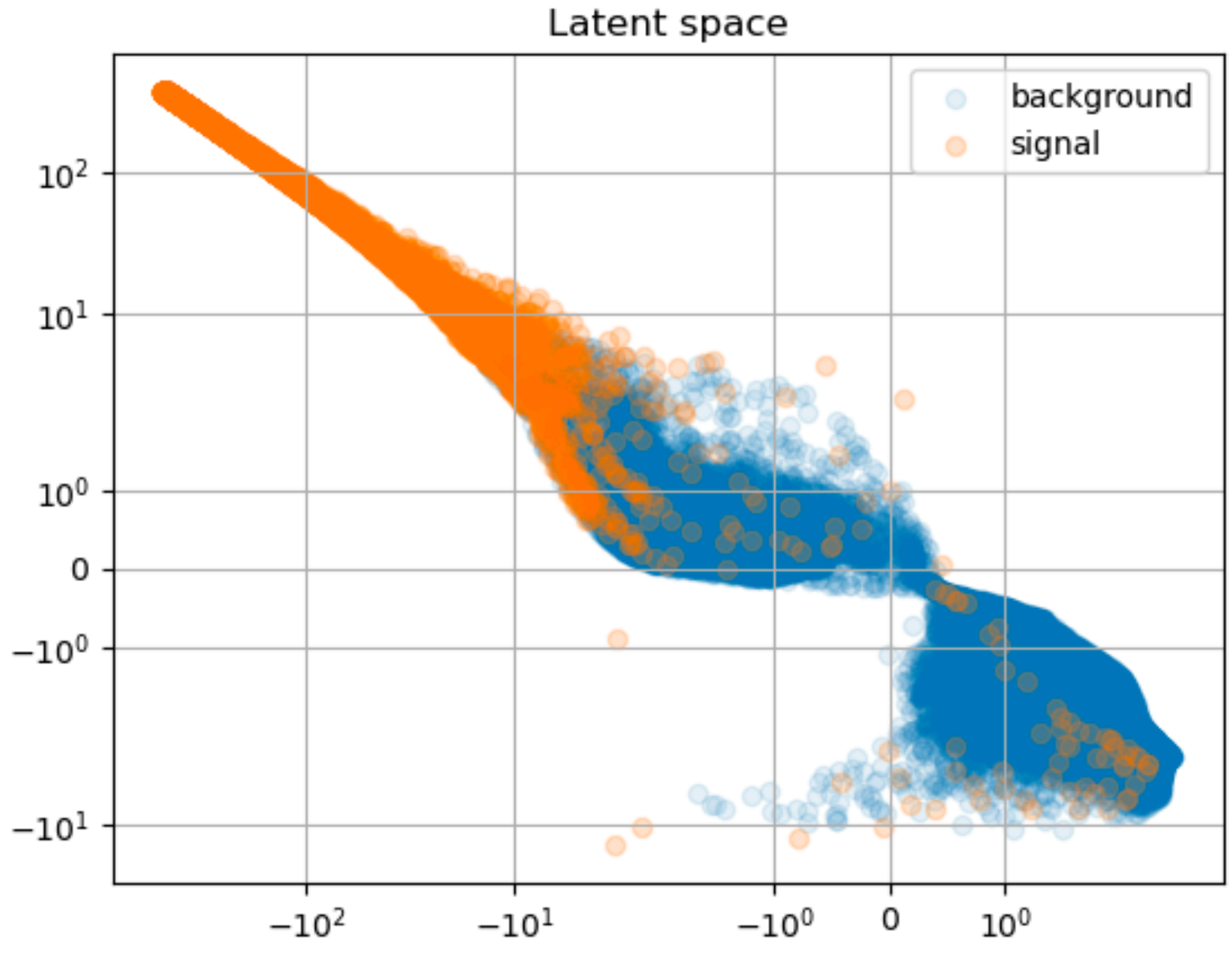
Example from particle physics: quark jets in Future Circular Collided. 59 input features, 16 dim latent space

Latent Space of Deep Autoencoders

- The inputs are mapped into the latent space z
- How does it look?

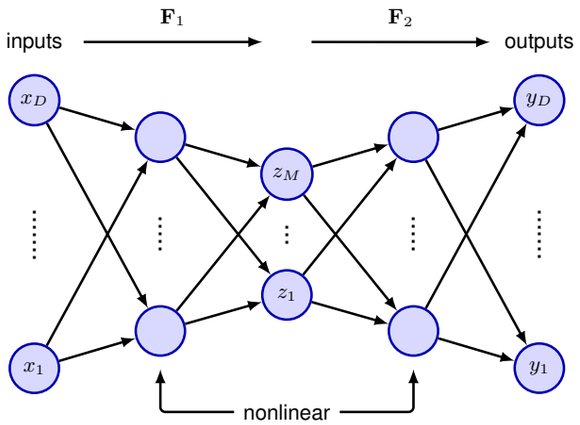


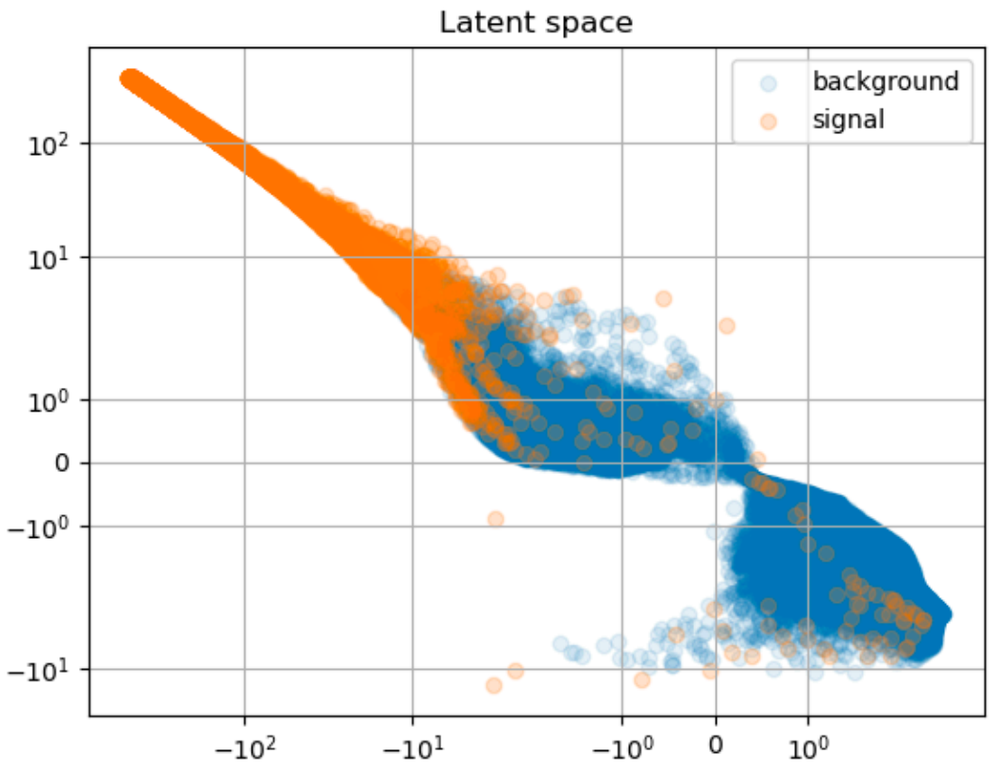
Example from
particle physics:
quark jets in
Future Circular Collided.
8 input features,
2 dim latent space



Latent Space of Deep Autoencoders

- The inputs are mapped into the latent space z
- Decoder maps the latent space into the original space
- Would be nice to use it to ***generate*** the data
- But the latent space is pretty ugly





Variational Auto-Encoders

Variational AutoEncoders

An auto-encoder (AE) is a method (typically based on neural networks) to learn efficient data codings in an unsupervised manner (hence the “auto”). The idea is “old” (80ies) and closely related to (the basis of) Generative Networks.

However, the latent space of AutoEncoders are “complex”, which means that you can not simply choose a “random number” from this space, and expect it to represent something “realistic” when decoded:

Thus, due to non-regularized latent space AE, the decoder can not be used to generate valid input data from vectors sampled from the latent space.

Variational AutoEncoders

An auto-encoder (AE) is a method (typically based on neural networks) to learn efficient data codings in an unsupervised manner (hence the "auto"). The idea is "old" (80ies) and closely related to (the basis of) Generative Networks.

> Here is one natural strategy for generating images. Build an autoencoder. Now generate random codes, and feed them into the decoder. It's worth trying this to reassure yourself that it really doesn't work. It doesn't work for two reasons. First, the codes that come out of a decoder have a complicated distribution, and generating codes from that distribution is difficult because we don't know it. Notice that choosing one code from the codes produced by a training dataset isn't good enough—the decoder will produce something very close to a training image, which isn't what we're trying to achieve. Second, the decoder has been trained to decode the training codes *only*. The training procedure doesn't force it to produce sensible outputs for codes that are *near* training codes, and most decoders in fact don't do so.

[David Forsyth, 19.3.1, why AEs are not VAEs]

Variational AutoEncoders

An auto-encoder (AE) is a method (typically based on neural networks) to learn efficient data codings in an unsupervised manner (hence the "auto"). The idea is "old" (80ies) and closely related to (the basis of) Generative Networks.

However, the latent space of AutoEncoders are "complex", which means that you can not simply choose a "random number" from this space, and expect it to represent something "realistic" when decoded:

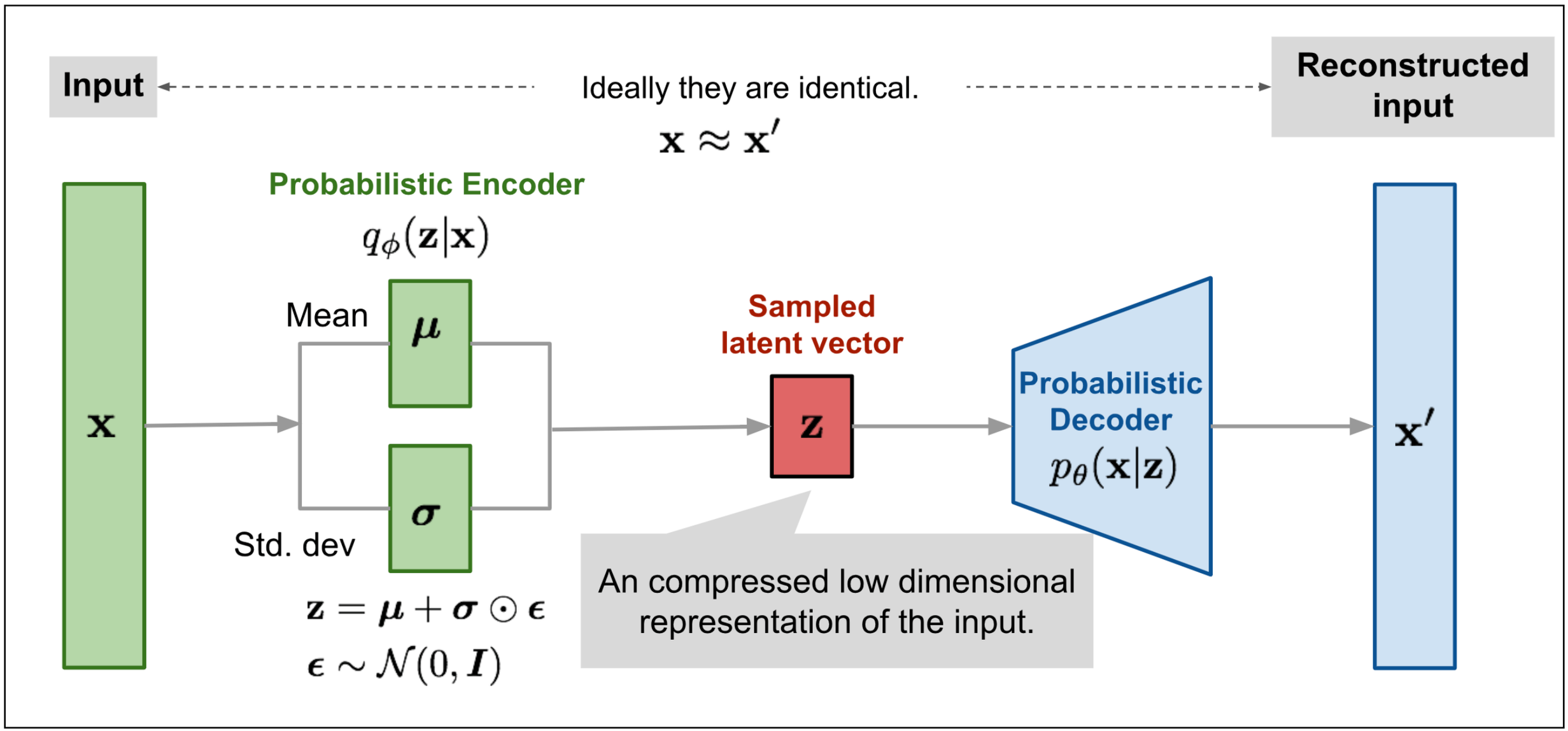
Thus, due to non-regularized latent space AE, the decoder can not be used to generate valid input data from vectors sampled from the latent space.

This requires a special type of AE, **so-called variational autoencoder (VAE)**.
Here, the encoder outputs parameters of a pre-defined distribution (multi-dim Gaussian) in the latent space for every input.

The constraint imposed by the VAE ensures that the latent space is **regularised**. This in turn allows one to take a value from the latent space and produce a realistic output.

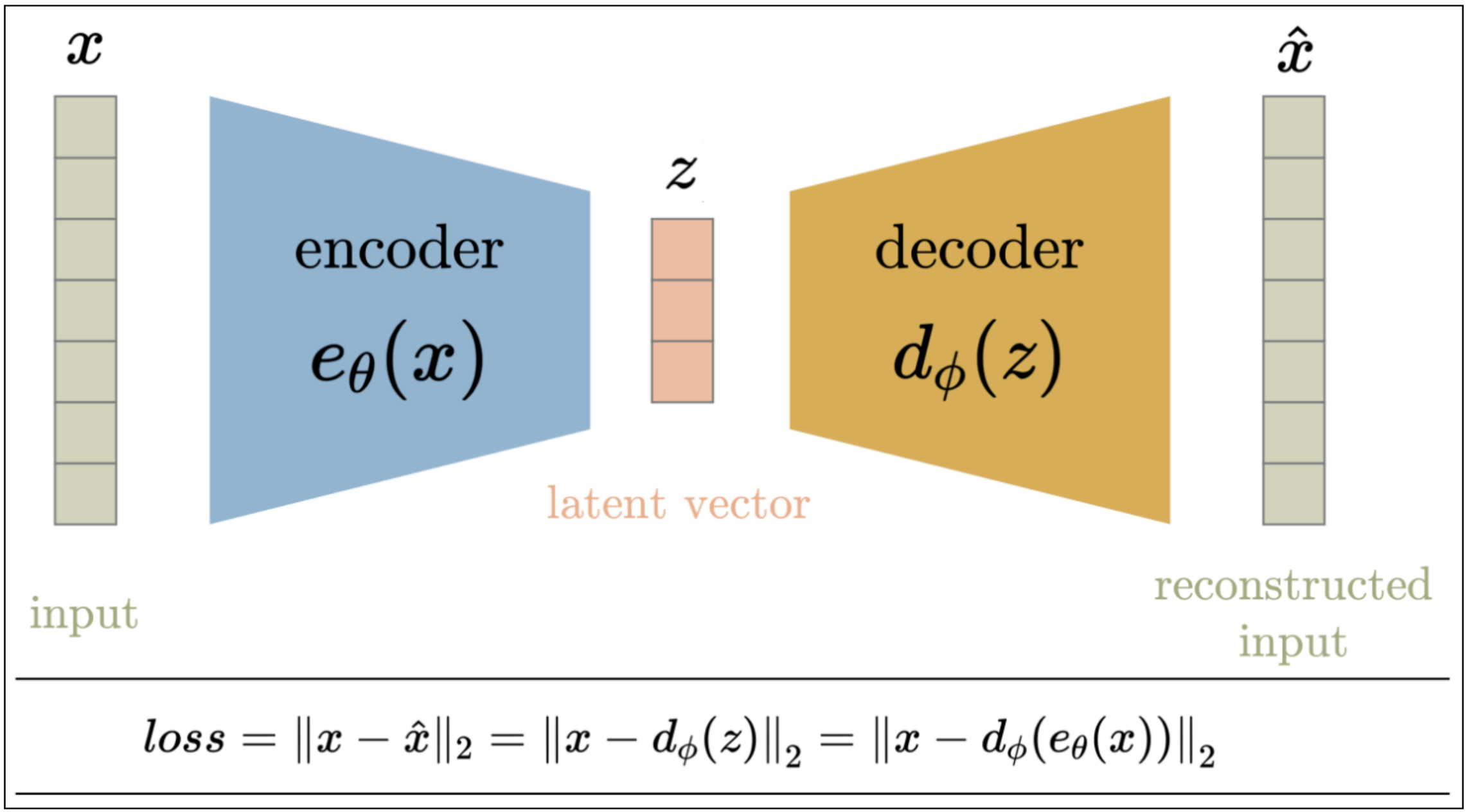
Variational AutoEncoders

A variational autoencoder thus uses a Gaussian-like latent space distribution. It is probabilistic in nature - it produces random cases close (i.e. ε away from) to the original.



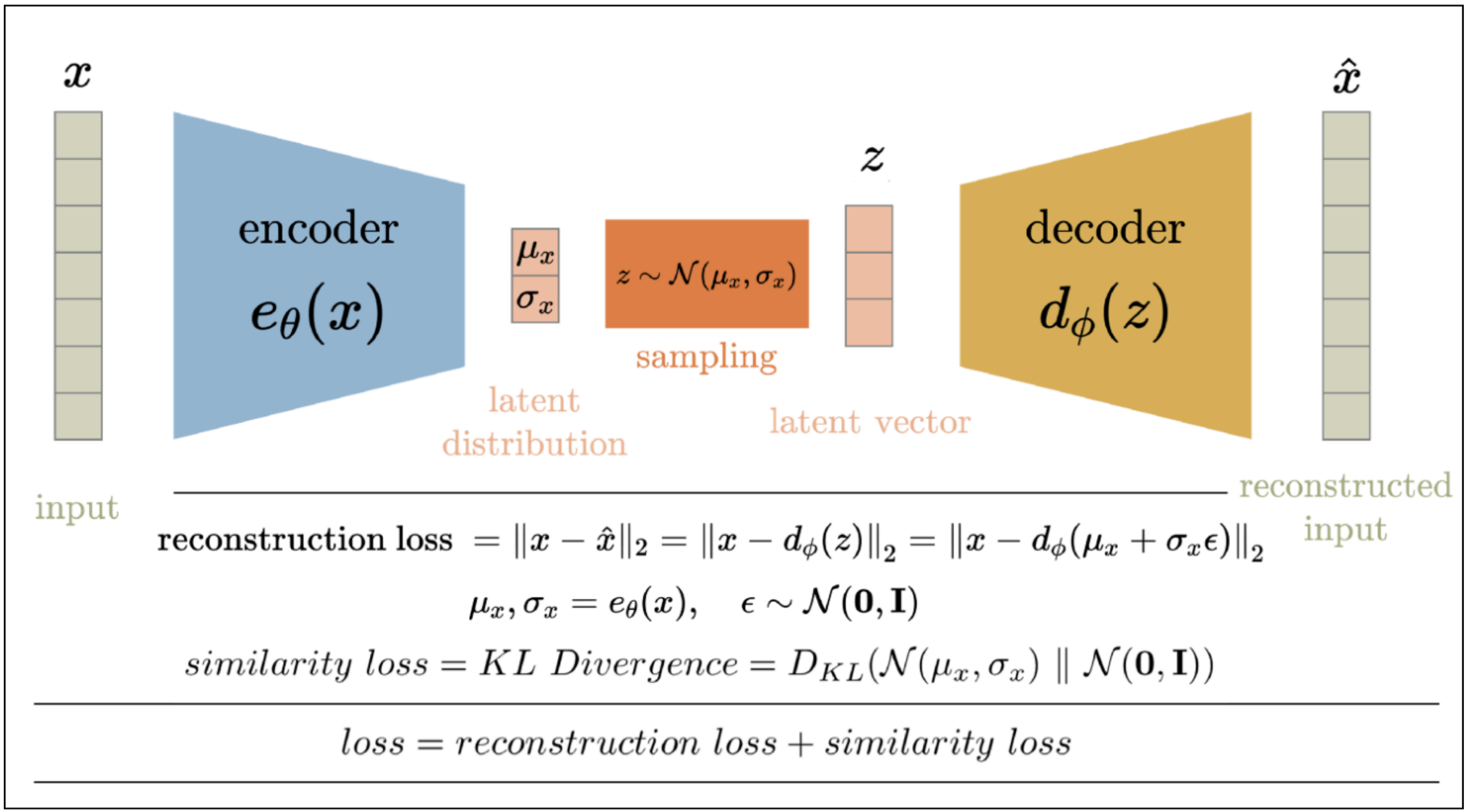
Variational AutoEncoders

A VAE thus uses a Gaussian-like latent space distribution. It is probabilistic in nature - it produces random cases close (i.e. ε away from) to the original. This is achieved by a "smart" loss function with the Kullback–Leibler (KL) divergence.



Variational AutoEncoders

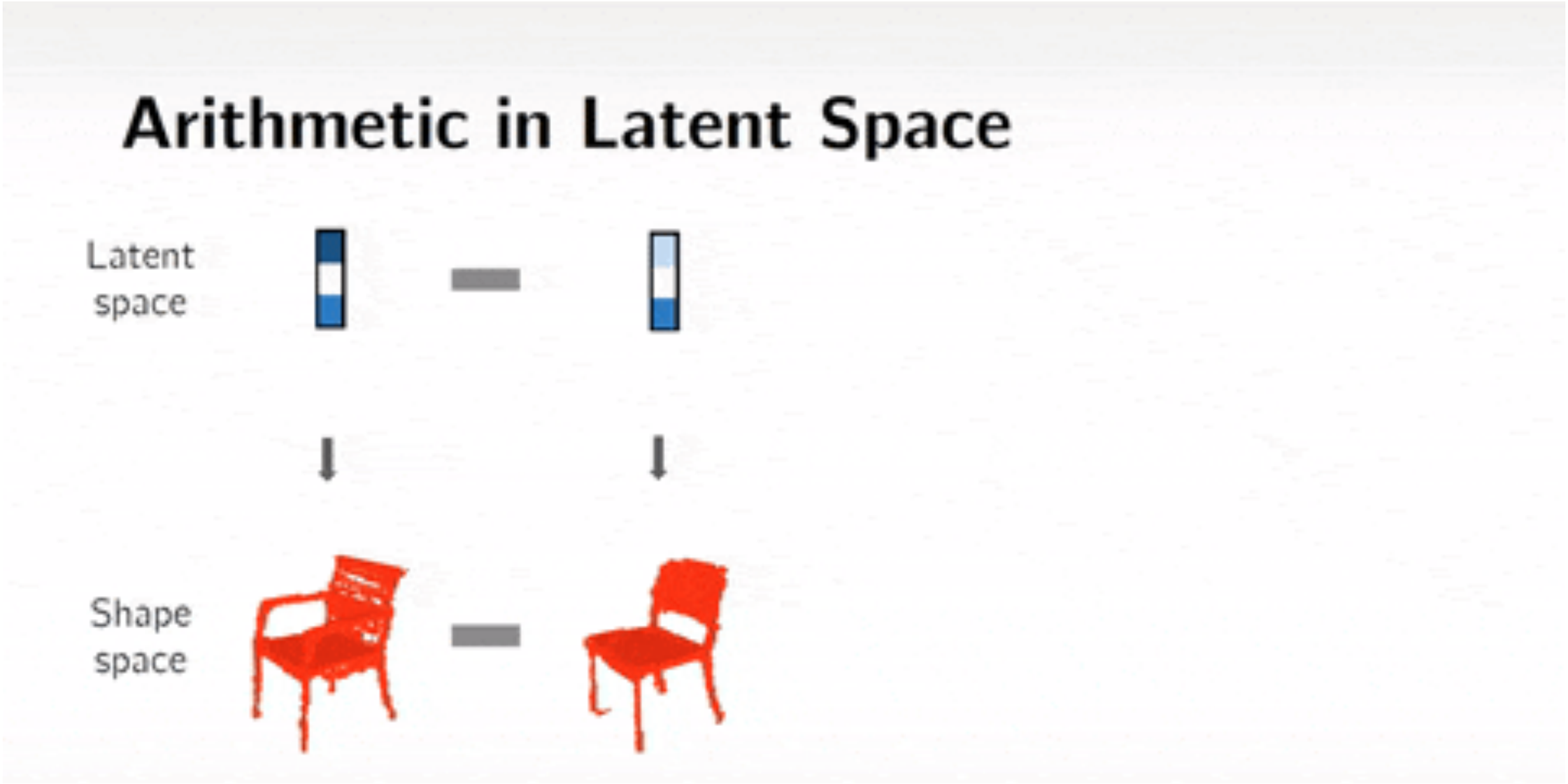
A VAE thus uses a Gaussian-like latent space distribution. It is probabilistic in nature - it produces random cases close (i.e. ε away from) to the original. This is achieved by a "smart" loss function with the Kullback–Leibler (KL) divergence.



Latent space illustration

The below animation shows how VAE latent spaces are a simplified representation of the more complex objects, containing the main features of these.

For this reason, one can do arithmetics (typically interpolate) between the inputs:



Latent space illustration

The below animation shows how VAE latent spaces are a simplified representation of the more complex objects, containing the main features of these.

For this reason, one can do arithmetics (typically interpolate) between the inputs:

Bonus:

Generative Adversarial Networks

Generative Adversarial Networks

Invented (partly) by Ian Goodfellow in 2014, Generative Adversarial Networks (GANs) is a method for learning how to produce new (simulated) datasets from existing data.

The basic idea is, that **two networks "compete" against each other**:

- **Generative Network**: Produces new data trying to make it match the original.
- **Adversarial (Discriminatory) Network**: Tries to classify original and new data.

Typically, the generator is a de-convolutional NN, while the discriminating (adversarial) is convolutional NN.

The concept is related to (Variational) Auto-Encoders.

"The coolest idea in machine learning in the last twenty years"

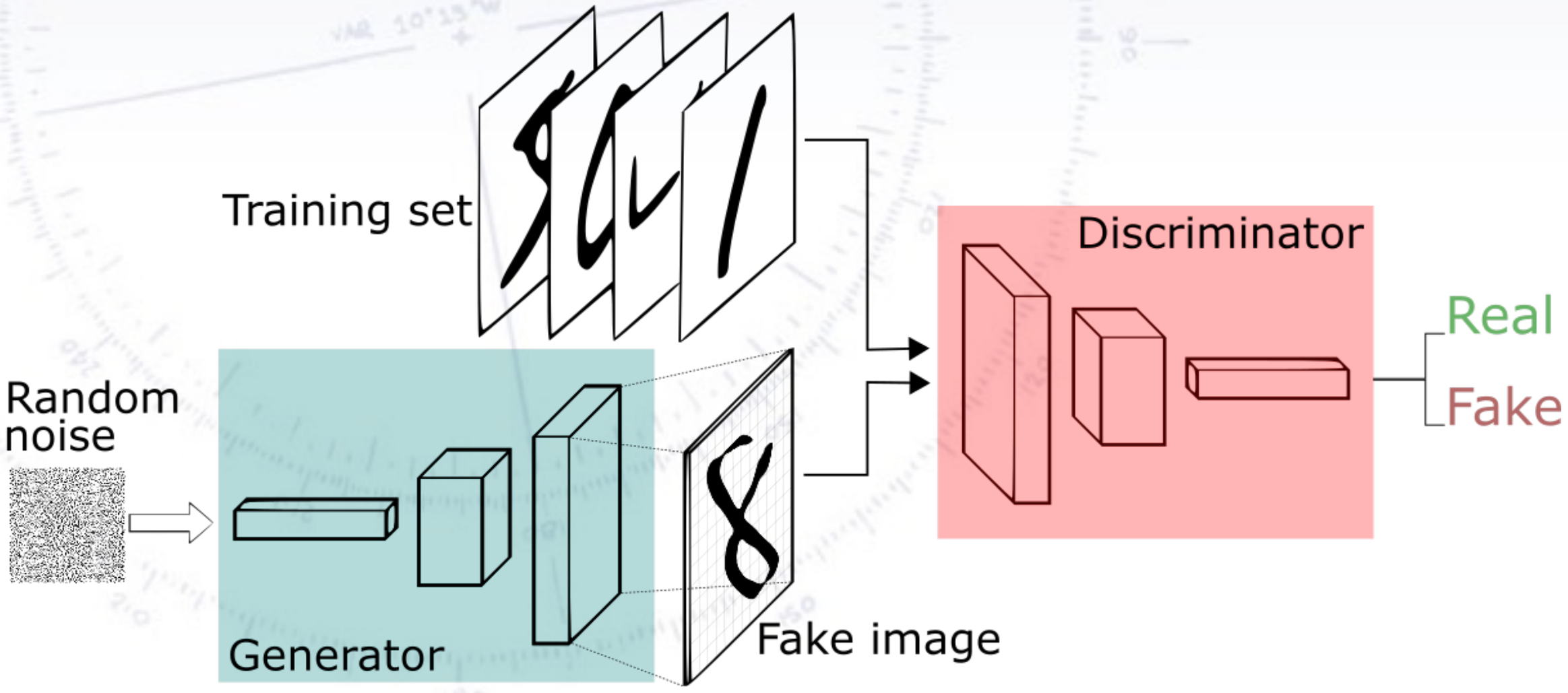
[Yann LeCun, French computer scientist]

nowadays: Diffusion models!

GAN drawing

Imagine that you want to write numbers that looks like hand writing.

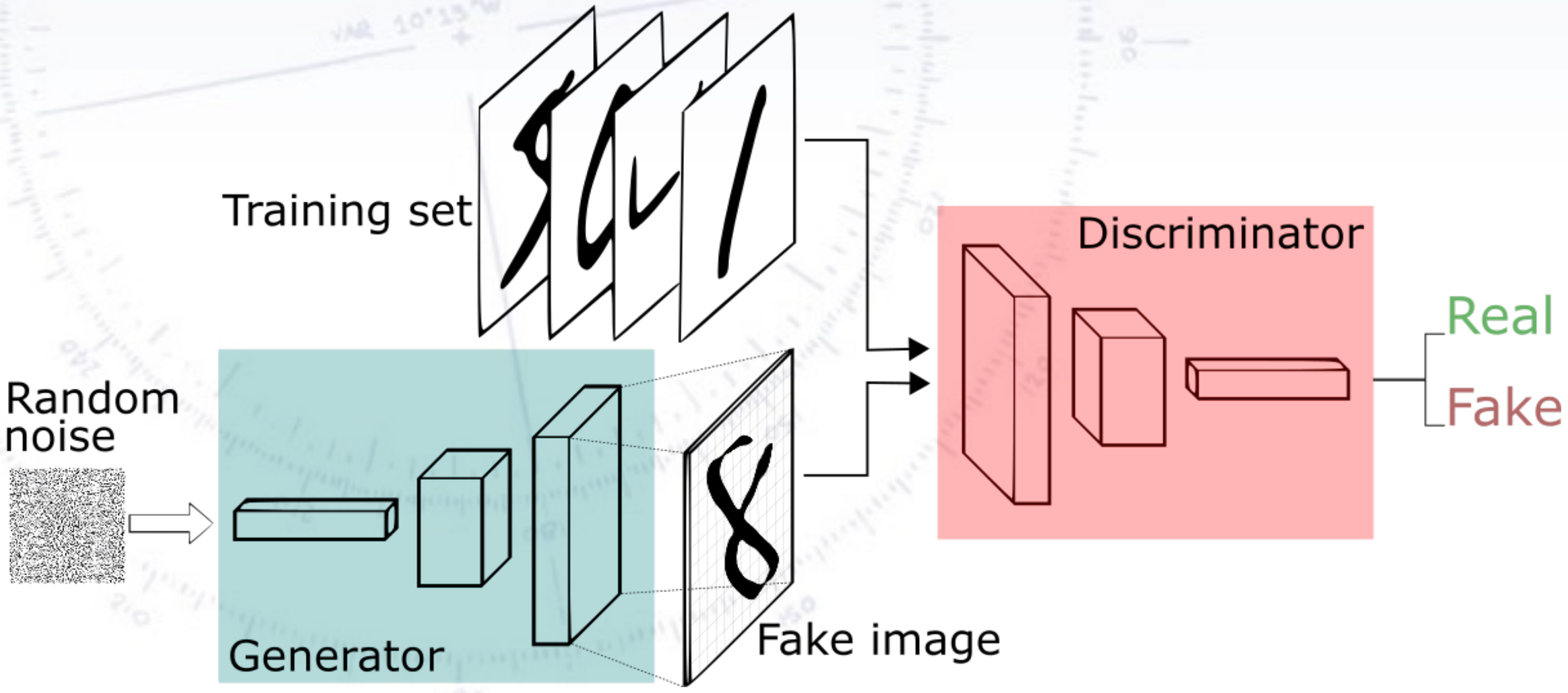
Given a large training set, you can ask you GAN to produce numbers. At first it will do poorly, but as it is “punished” by the discriminator, it improves, and at the end it might be able to produce numbers of **equal quality to real data**:



GAN drawing

The discriminator/adversarial can also be seen as an addition to loss function, penalising (with λ) an ability to see differences between real and fake:

$$\text{Loss} = \text{Loss} + \lambda \cdot L_{\text{Adversarial}}$$



GANs producing face images

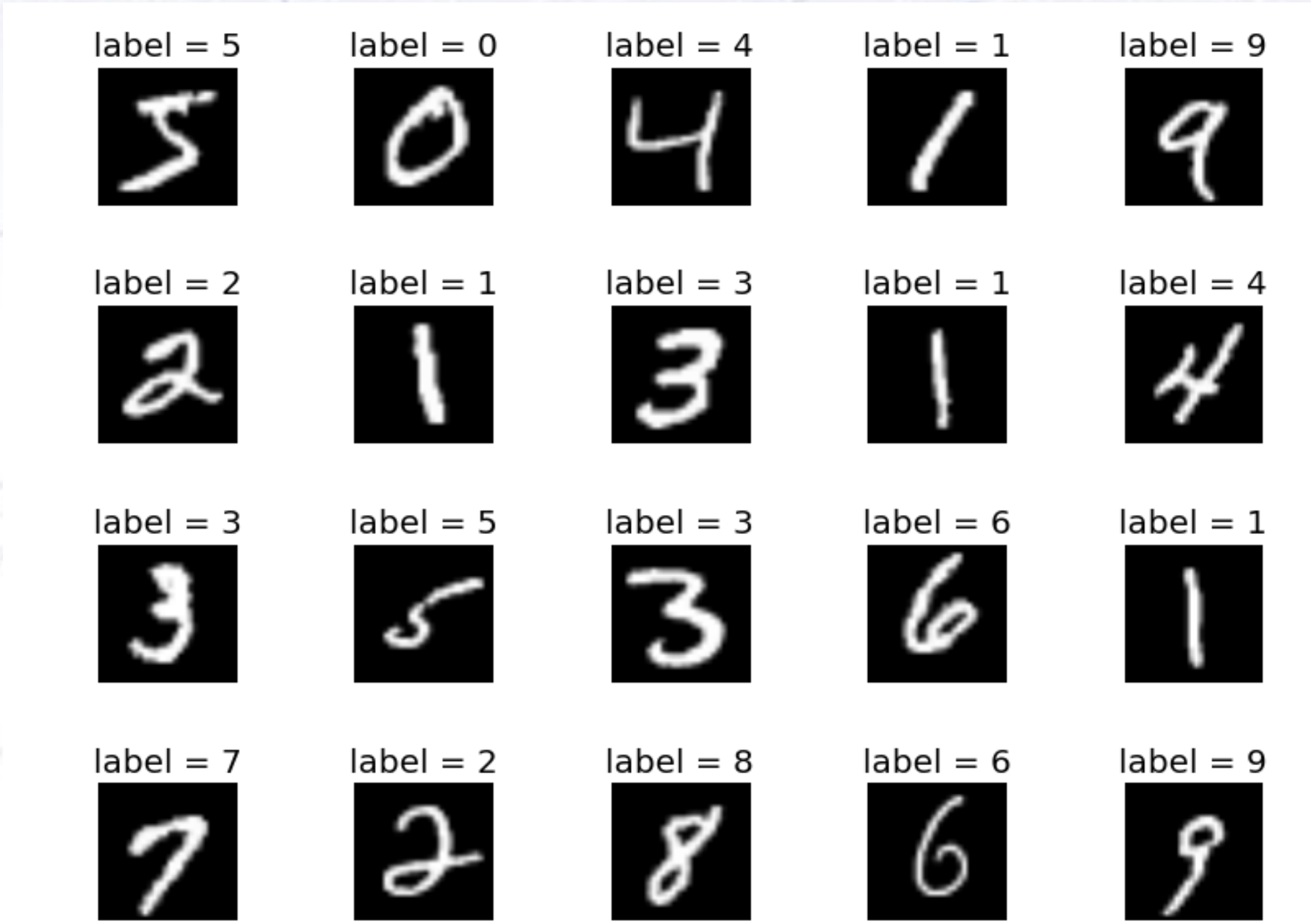
In 2017, Nvidia published the result of their "AI" GANs for producing celebrity faces. There is of course a lot of training data… here are the results:

Evolution in facial GANs

There is quiet a fast evolution in GANs, and their ability to produce realistic results….

MNist data: Handwritten numbers

A “famous case” has been hand written numbers. The data consists of 28x28 gray scale images of numbers. While that spans a large space, the latent space is probably (surely!) much smaller, as far from all combinations of pixels and intensities are present.



MNist data: Handwritten numbers

A “famous case” has been hand written numbers. The data consist[illegible]28 gray scale images of numbers. While that spans a large space, the la[illegible]bly (surely!) much smaller, as far from all combinati[illegible]are present.

With GANs, you can produce handwritten Letters again – sort of!