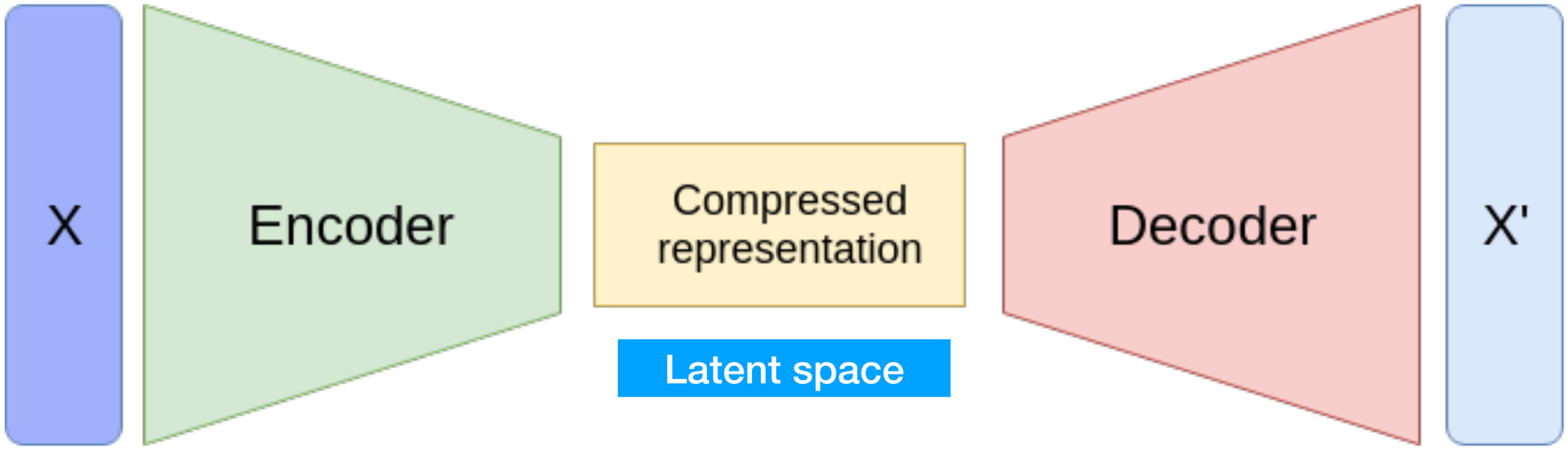
Lecture 8: Autoencoders, Unsupervised Learning and Anomaly Detection

Program for Today

Date	Topic	Tutorial
Sep 17	Intro & class description	Linear Algebra in a nutshell + prob and stat
Sep 24	Basic of machine learning + Dense NN	*Basic jupyter + DNN on mnist (give jet dnn as homework)*
Oct 1	Convolustional NN	*Convolutional NNs with MNIST*
Oct 8	Training in practice: regularization, optimization, etc	*Practical methodology*
Oct 15	*Google tutorial*	
Oct 22	Recurrent NN	*Hands-on exercise*
Oct 29	Graph NNs	*Tutorial on Graph NNs*
Nov 5	Unsupervised learning and anomaly detection	*Autoencoders with MNIST*
Nov 12	Generative models: GANs, VAEs, etc	Normalizing flows
Nov 19	Transformers	
Nov 26	Network compression (pruning, quantization, Knoweldge Distillation)	
Dec 3	Tutorial on hls4ml/qkeras	
Dec 10	Quantum Machine Learning	
Dec 17	Q/A Prior to exam	

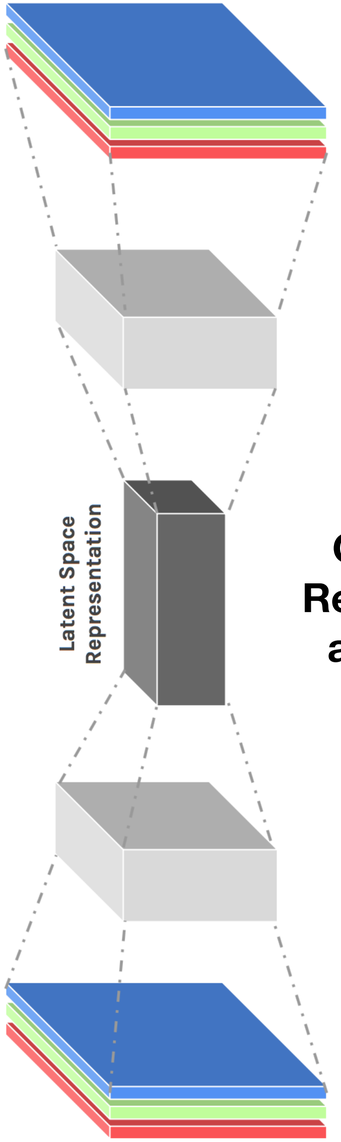
Autoencoders

- *Autoencoders are networks trained to copy its input to its output*
  - *They consist of an encoder* $z = q(x)$ *and a decoder* $x' = p(z) = p(q(x))$
  - *They have a typical "bottleneck" structure (under complete autoencoders): they go from* $\mathbb{R}^n \rightarrow \mathbb{R}^k$ *and from* $\mathbb{R}^k \rightarrow \mathbb{R}^n$ *with k<n.* $\mathbb{R}^k$ *is the latent space*

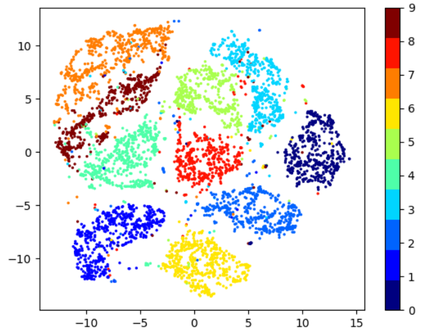


Importance of Bottleneck

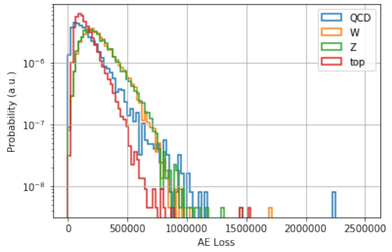
- *Ideally, one would learn* $p(q(x)) = q^{-1}(q(x)) = x$*. But that would not be very useful*
- *Instead, we aim at learning* $x' = p(q(x))$ *as close as possible to x but not identical (k<n implies that some information is lost)*
- *Because the transformation can't be perfect, the network needs to prioritize the relevant aspects and discard the rest.This is what trigger learning from data as opposed to force the decoder to just mirror the encoder.*
- *This make autoencoders useful for*
  - *Cata compression*
  - *Clustering*
  - *Anomaly Detection*



Compressed Representation, also good for clustering

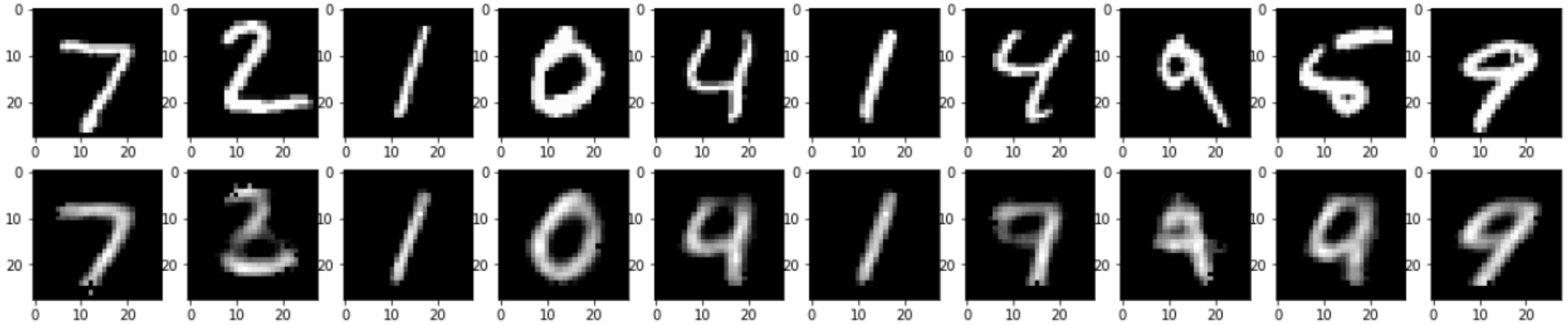


Distance to input for Anomaly Detection



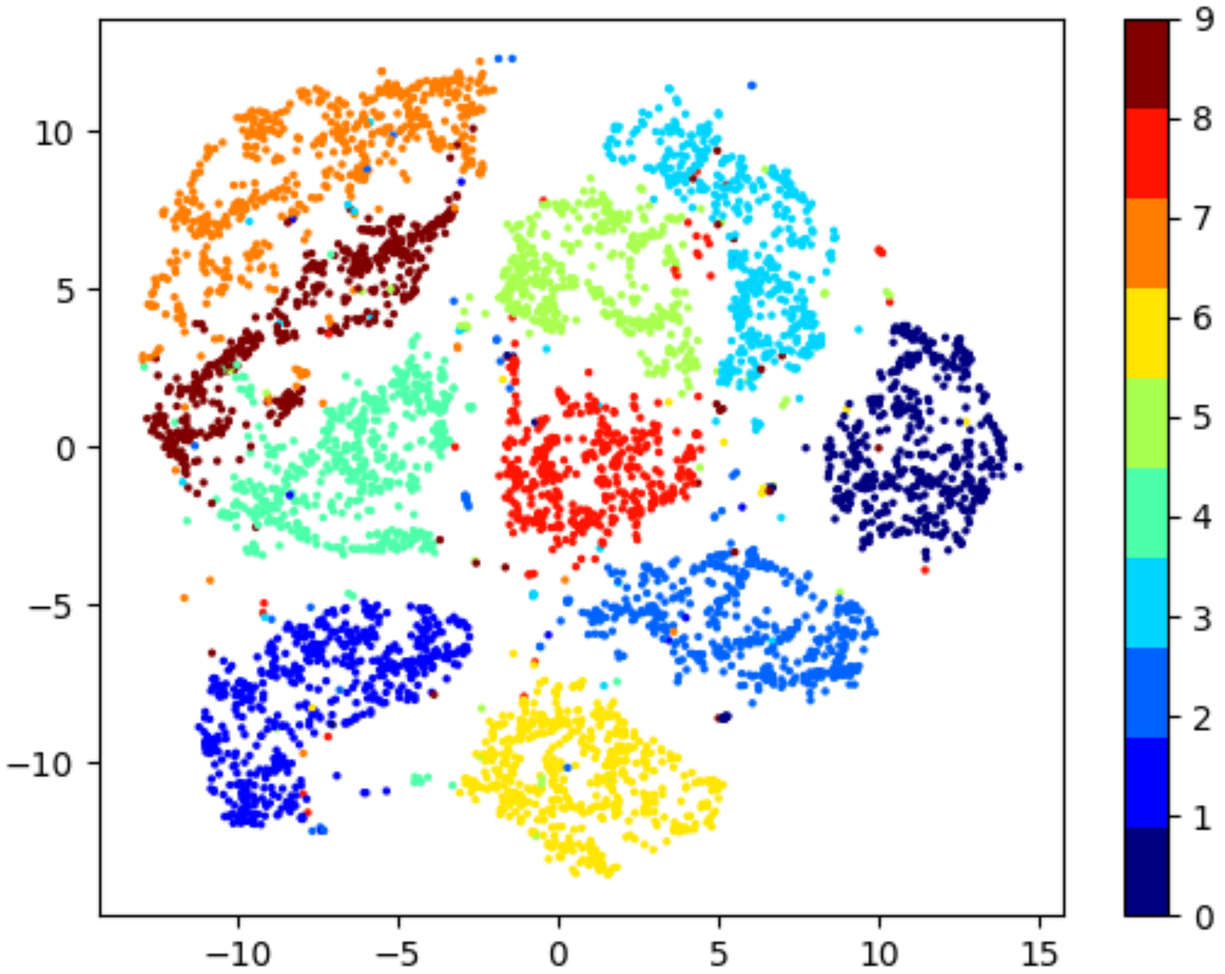
Compression

- *Autoencoders can be seen as compression algorithms*
  - *The n inputs are reduced to k quantities by the encoder*
  - *Through the decoder, the input can be reconstructed from the k quantities*
- *As a compression algorithm, an auto encoder allows to save (n-k)/n of the space normally occupied by the input dataset*



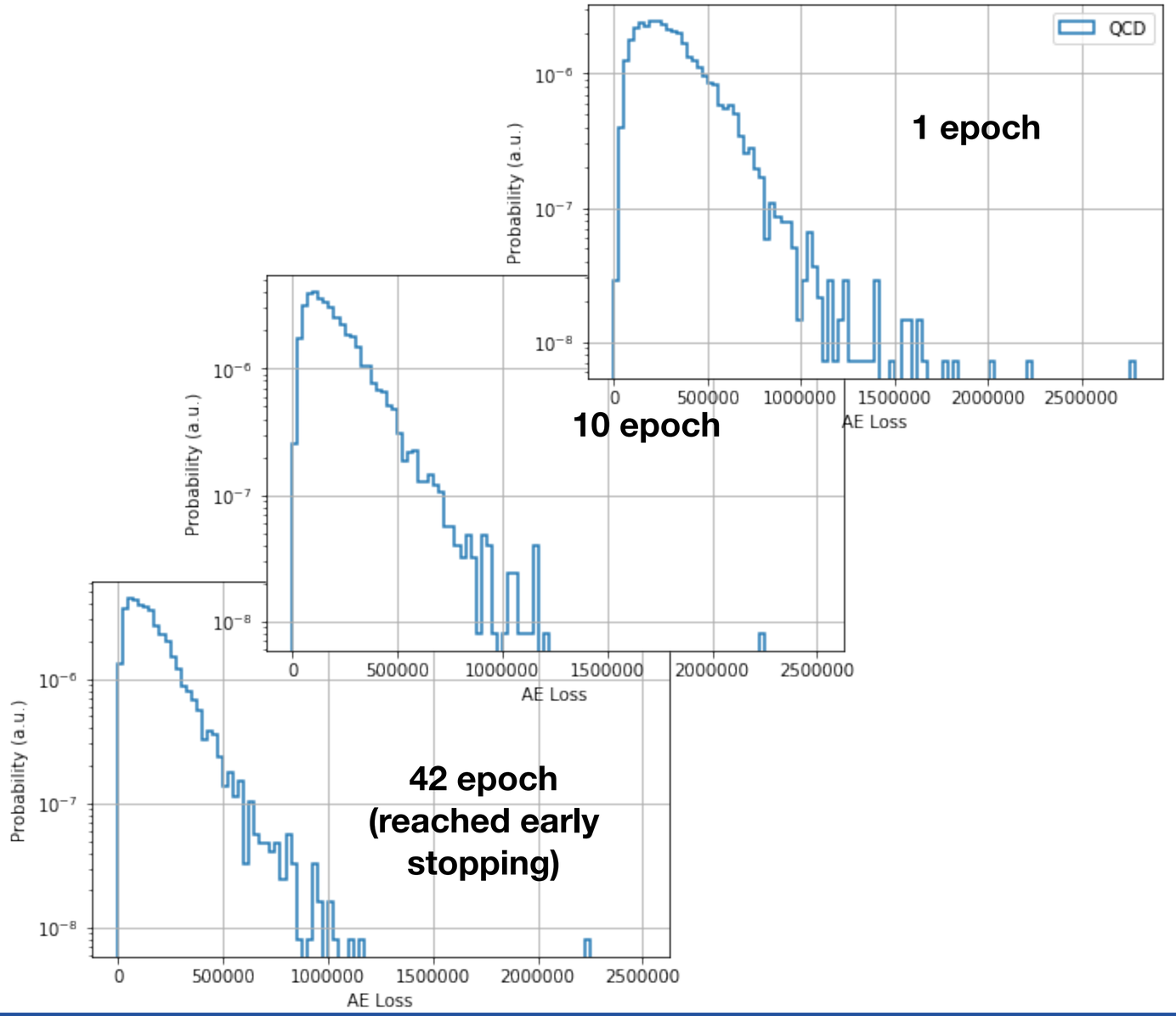
Clustering

- *The auto encoder can be used as a clustering algorithm*
  - *Alike inputs tend to populate the same region of the latent space*
  - *Different inputs tend to be far away*



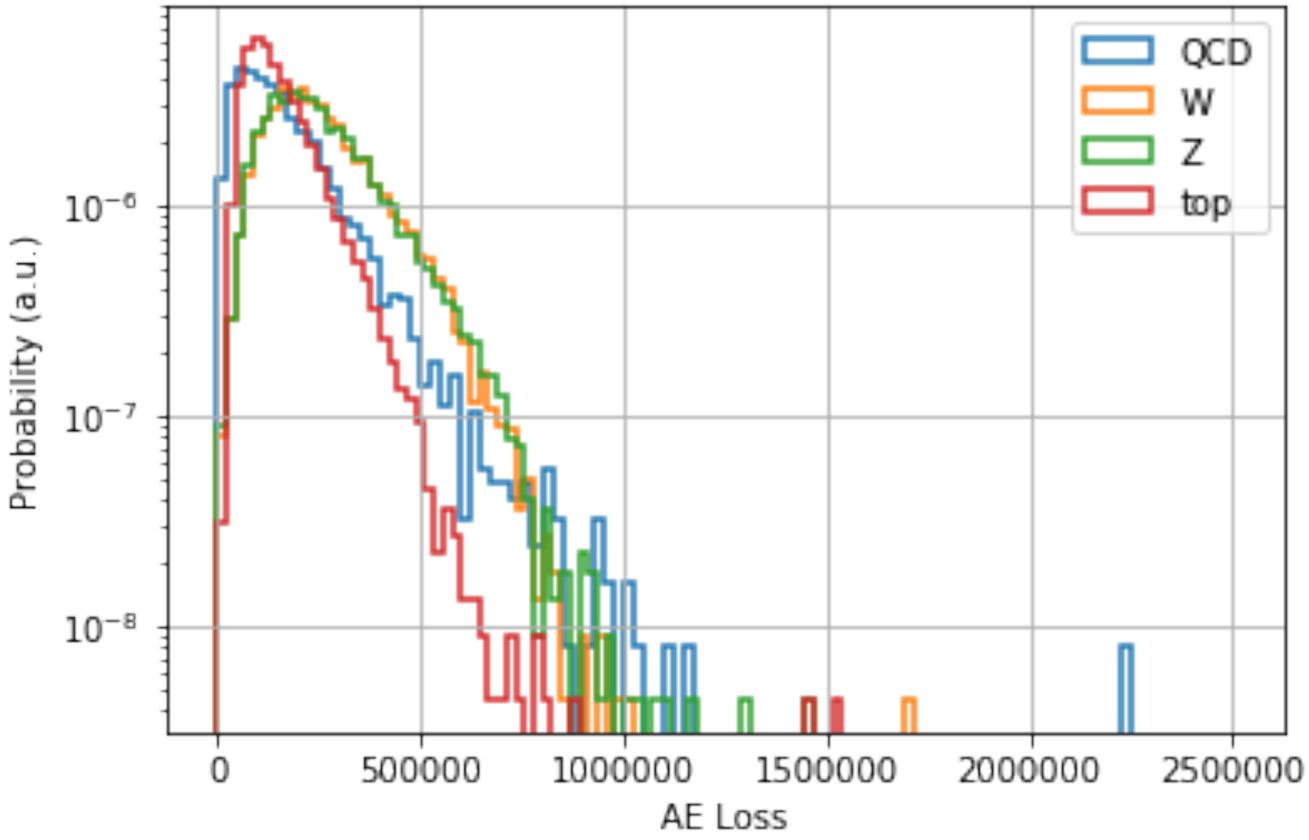
Anomaly Detection

- *AEs are training minimizing the distance between the inputs and the corresponding outputs*
- *The loss function represents some distance metric between the two*
  - *e.g., MSE loss*
- *A minimal distance guarantees that the latent representation + decoder is enough to reconstruct the input information*



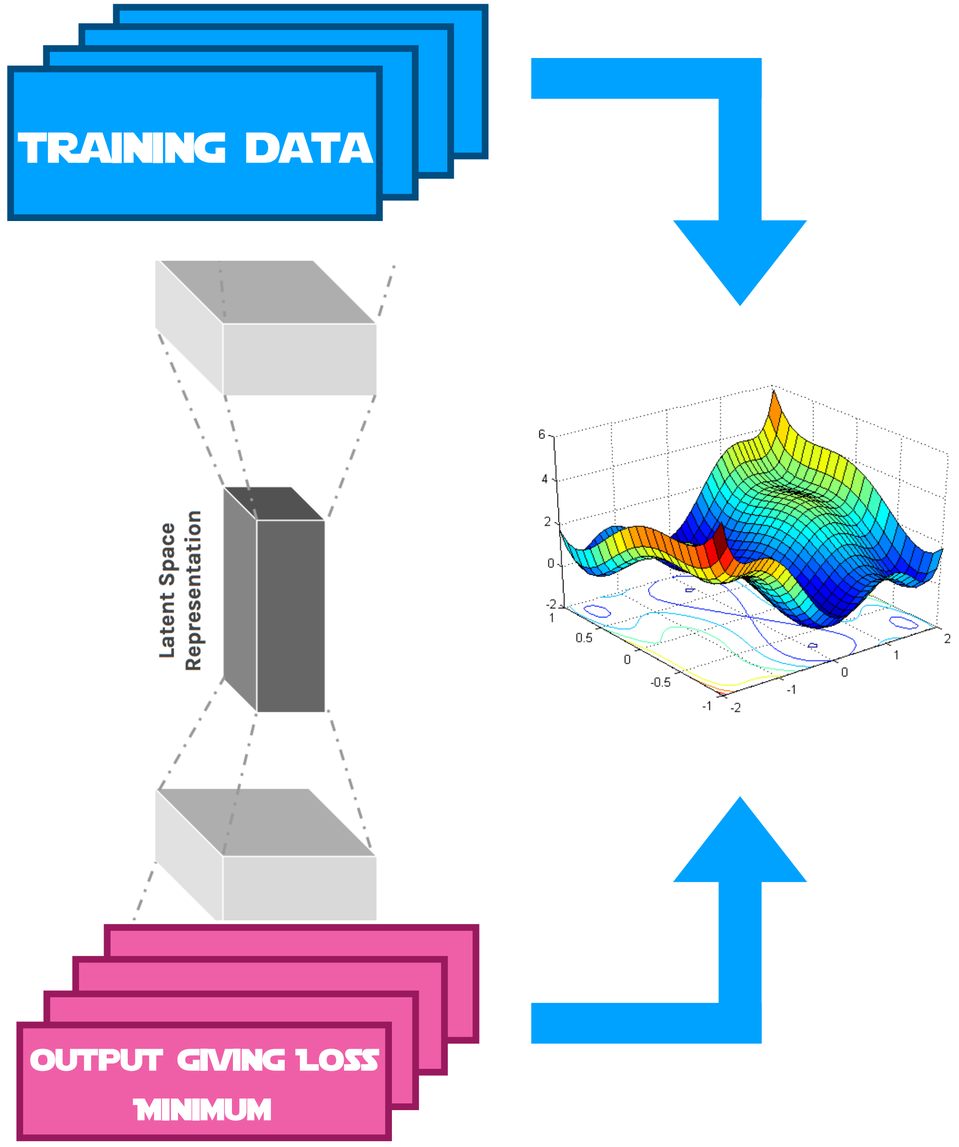
Anomaly Detection

- *Once trained, an autoencoder can reproduce new inputs of the same kind of the training dataset*
  - *The distance between the input and the output will be small*
- *If presented an event of some new kind (anomaly), the encoding-decoding will tend to fail*
  - *In this circumstance, the loss (=distance between input and output) will be bigger*



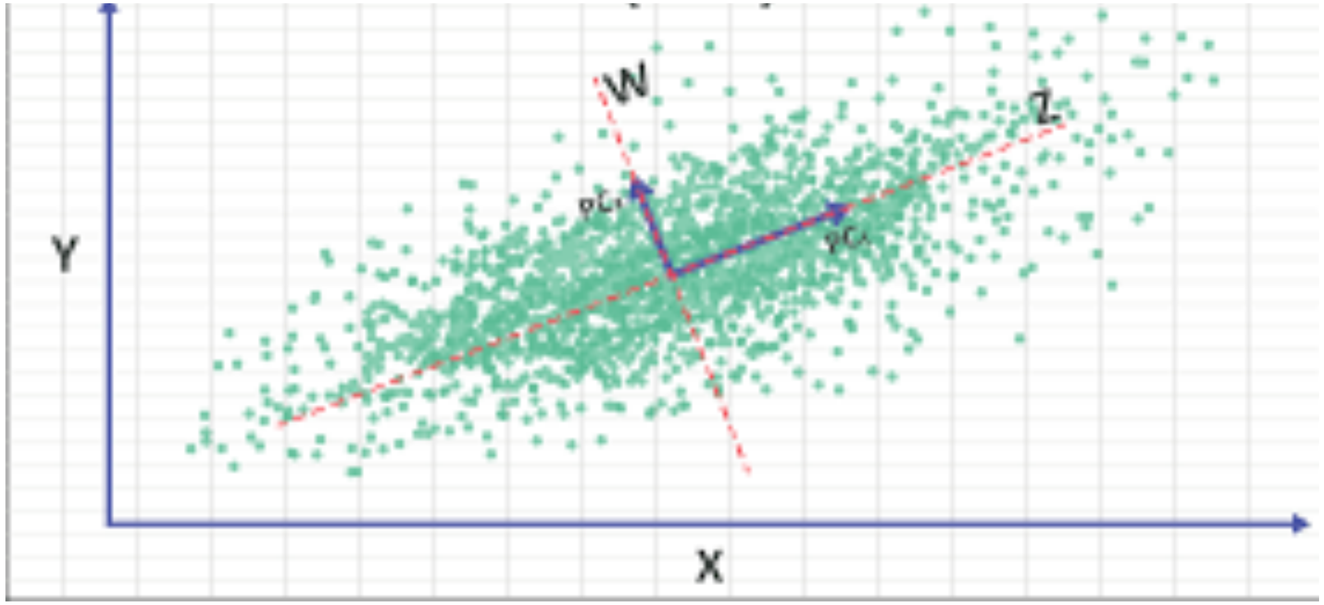
Training and Autoencoder

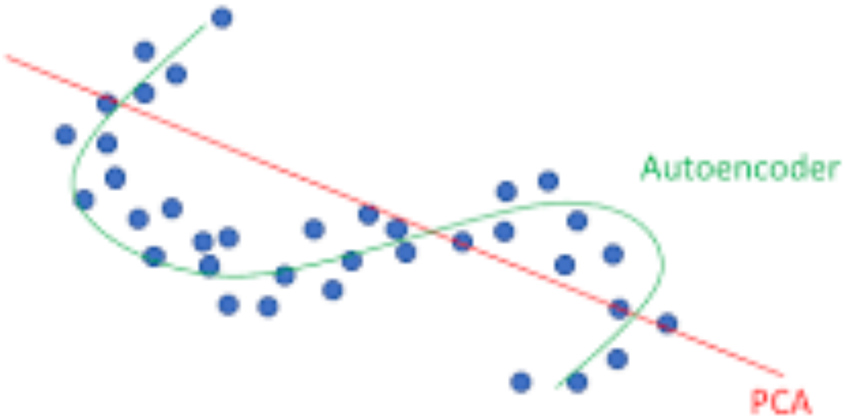
- *Ingredients:*
  - *A training dataset* x
  - *A model returning an output* x' *with the same dimensionality as* x
  - *A loss function penalizing* x' *if far away from* x
    - *For example,* x *and* x' *could be images and the loss the pixel-by-pixel MSE*
  - *No ground truth y: this is an UNSUPERVISED training*



Autoencoder vs PCA

- *PCA is a linear process*
  - *Diagonalize the covariance matrix and take the first k eigenvectors*
- *Autoencoders generalize this procedure*
  - *A linear autoencoder with MSE performs a PCA*
  - *A non-linear autoencoder can learn more from the data (more effective compression, etc.)*





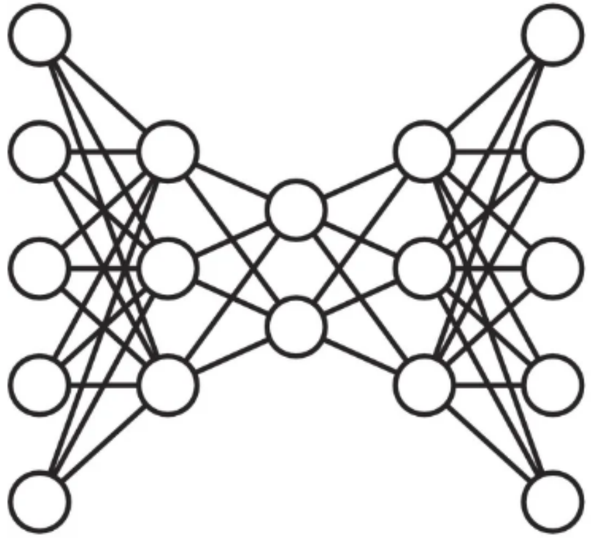
Do We Need a Bottleneck?

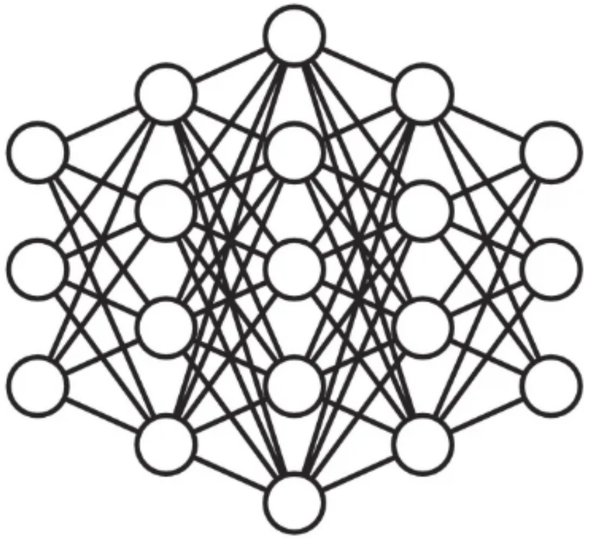
- *Autoencoders can be undercomplete (inner dimensionality = input) or over complete (inner dimensionality > input)*

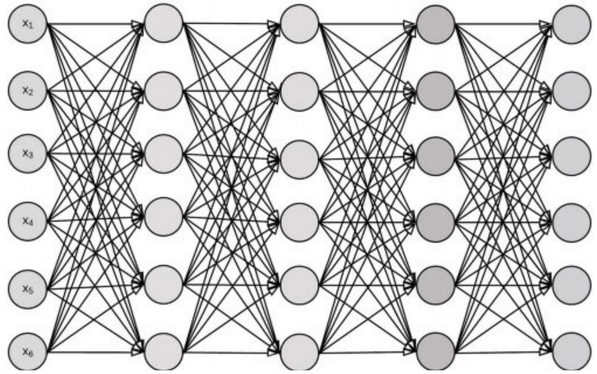
- *One can still force the autoencoder towards a lossy compression using regularization: add to the loss a term that limits the model capacity*

$$L(x, p(q(x))) + \Omega(p(q(x)))$$

- *This can be achieved in various ways*
  - *Robustness to noise (denoising)*
  - *Sparsity*
  - *Generative properties*

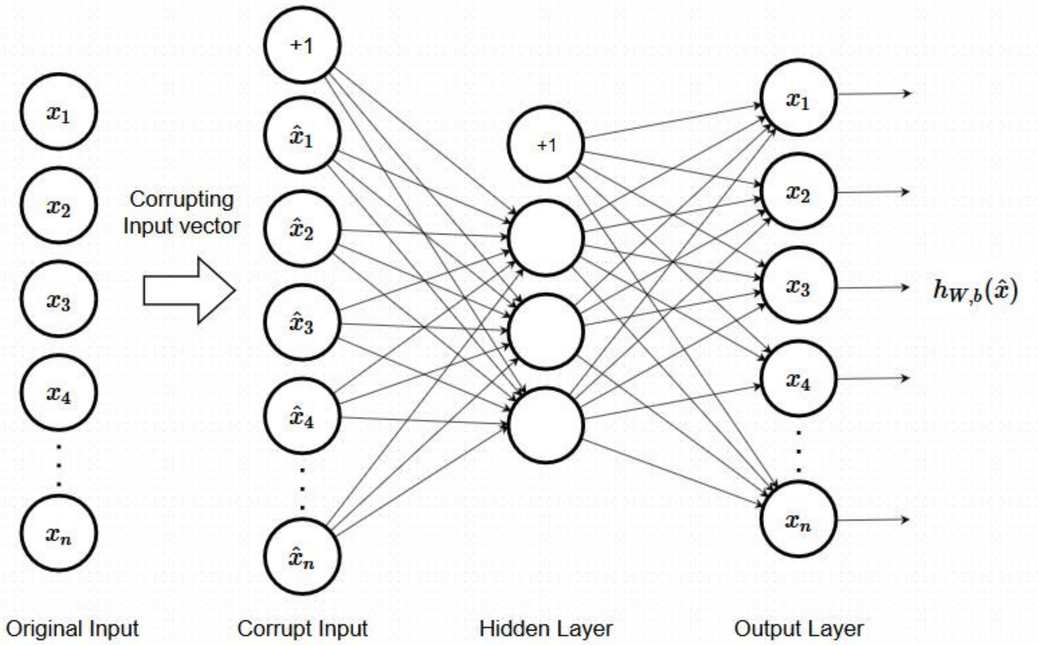


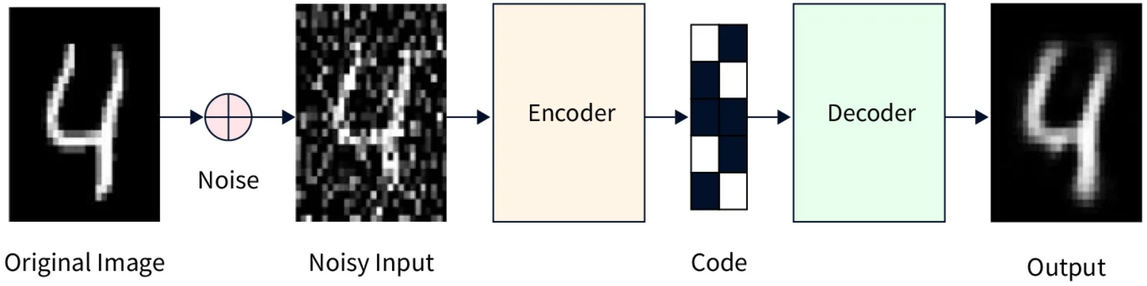




Denoising Autoencoders

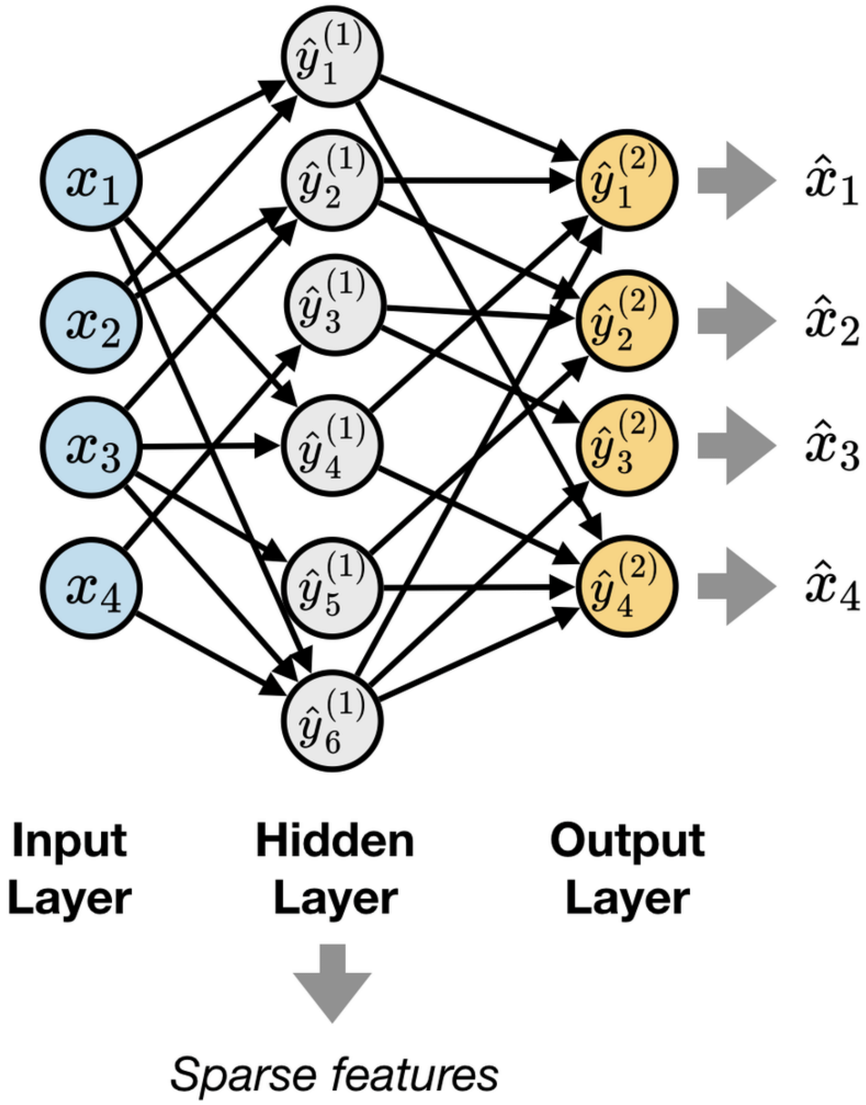
- A denoising AE is an AE trained to be robust against a preturbation of the data
- This is done minimizing the loss $L(x, p(q(\tilde{x})))$, where $\tilde{x}$ is obtained adding some noise to x
- Because of the noise added
  - The network is regularized so that it is prevented from learning the identity
  - As a result of that, the network is forced to learn the distribution of the data





Sparse Autoencoders

- *The simplest regularizer is a term that penalizes the training every time a the $z = q(x)$ quantities are pulled away from 0*
  - *This is similar to the $L_p = ||w||_p$ norm terms we discussed already, but now acting on the output of the latent-space nodes*
  - *Any other metric can be used to the same purpose*
- *Even in absence of a bottle neck, a sparse regularization prevents the autoencoder from learning exactly $g = f^{-1}$*



The Bayesian Interpretation

- *Imagine to have a dataset with some observable variables* x *and some other latent variables* z

- *One can write the likelihood as*

Since we don't know the actual latent quantities z that determines the observed x, z are estimated from data using the encoder $z = q(x)$

$$-\log p(x) = -\sum_z \log p(x,z) = -\sum_z \log[p(x\,|\,z)\pi(z)] = -\sum_z \log p(x\,|\,z) - \sum_z \log \pi(z)$$

- *Let's take as prior* $\pi(z) = \frac{\lambda}{2}\prod_i e^{-\lambda|z_i|}$ *and then obtain*

The sparsity loss term

$$-\log p(x) = \left[-\sum_h \log p(x\,|\,z)\right] + \left[\lambda \sum_i |h_i|\right]$$

The usual loss term

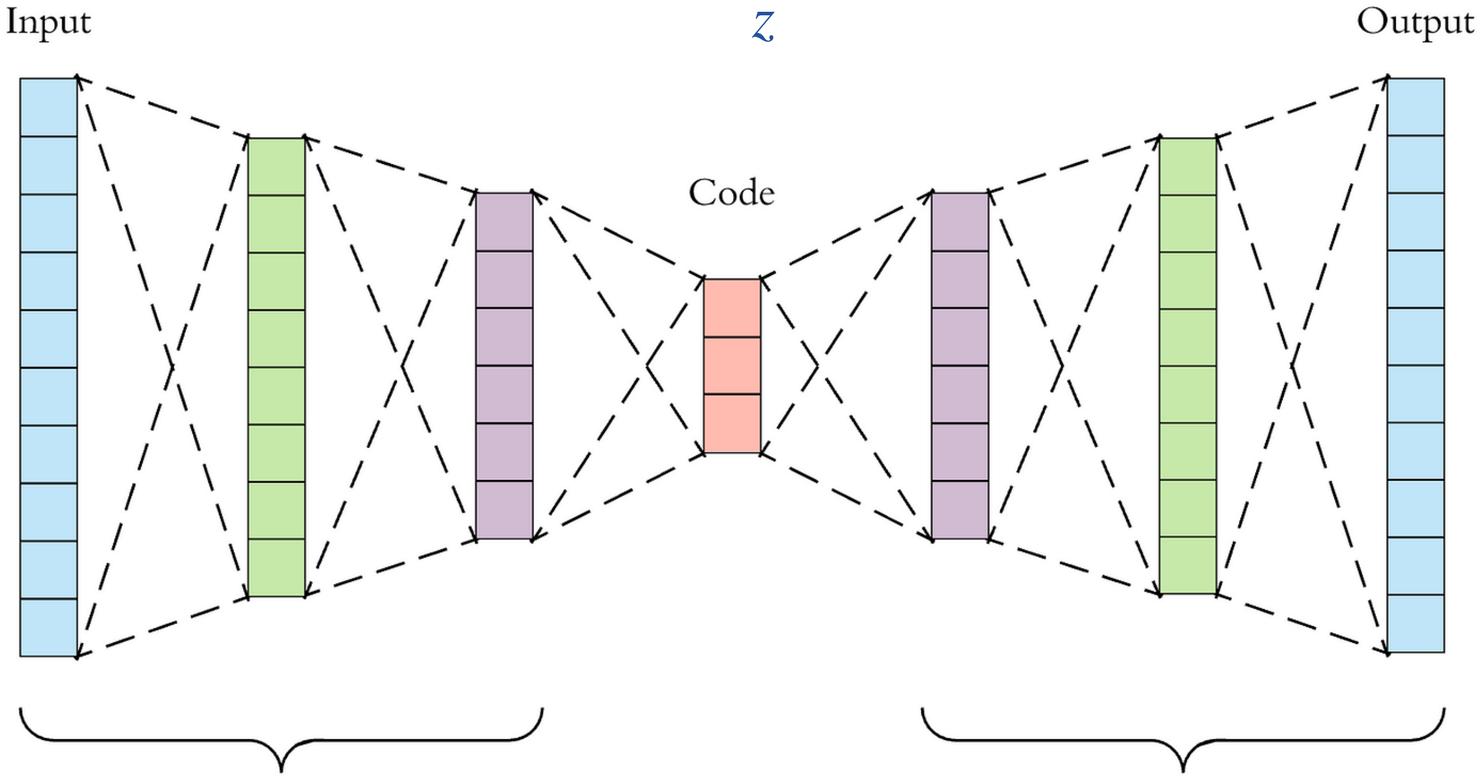
hyperparameter

Plain AEs

There are some observables x depending on some latent (unknown) z

$$L(x, z) = -\sum_{z} \log p(x \mid z)$$

The loss is trying to approximate the conditional probability of x over h



We don't know the pdf of z, so we use the encoder to obtain a point estimate of h

$$p(z \mid x)$$

$$p(x \mid z)$$

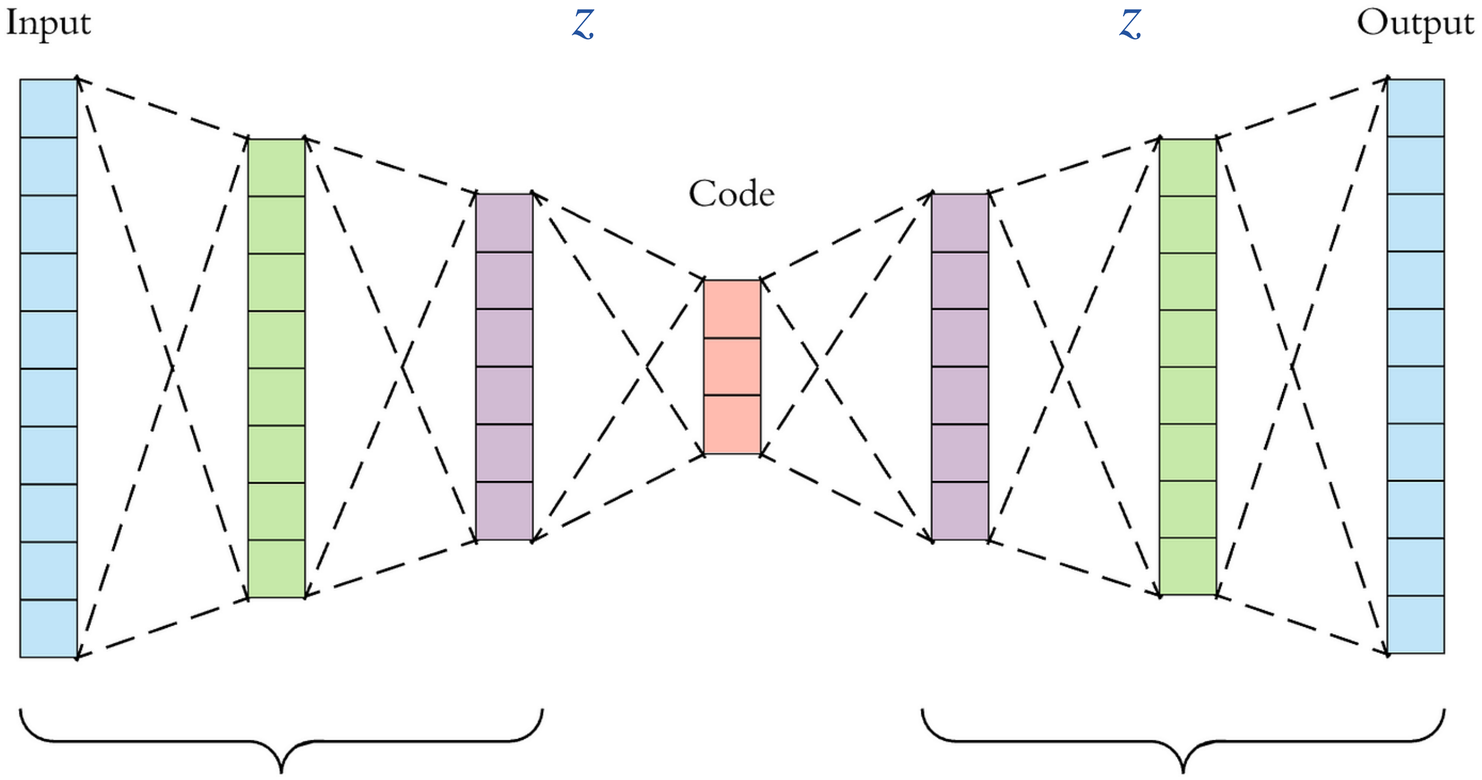
We use the decoder to compute the conditional probability

Sparse AEs as Generative Models

The loss is still trying to approximate the probability of x

$$-\log p(x) = -\sum_{z} \log p(x \mid z) - \sum_{z} \log \pi(z)$$

The regularization term takes into account the distribution of z (the choice of z is in a certain sense the arbitrary choice of a prior)



We don't know the pdf of z, so we use the encoder to obtain a point estimate of h

$$p(z \mid x)$$

$$p(x \mid z)$$

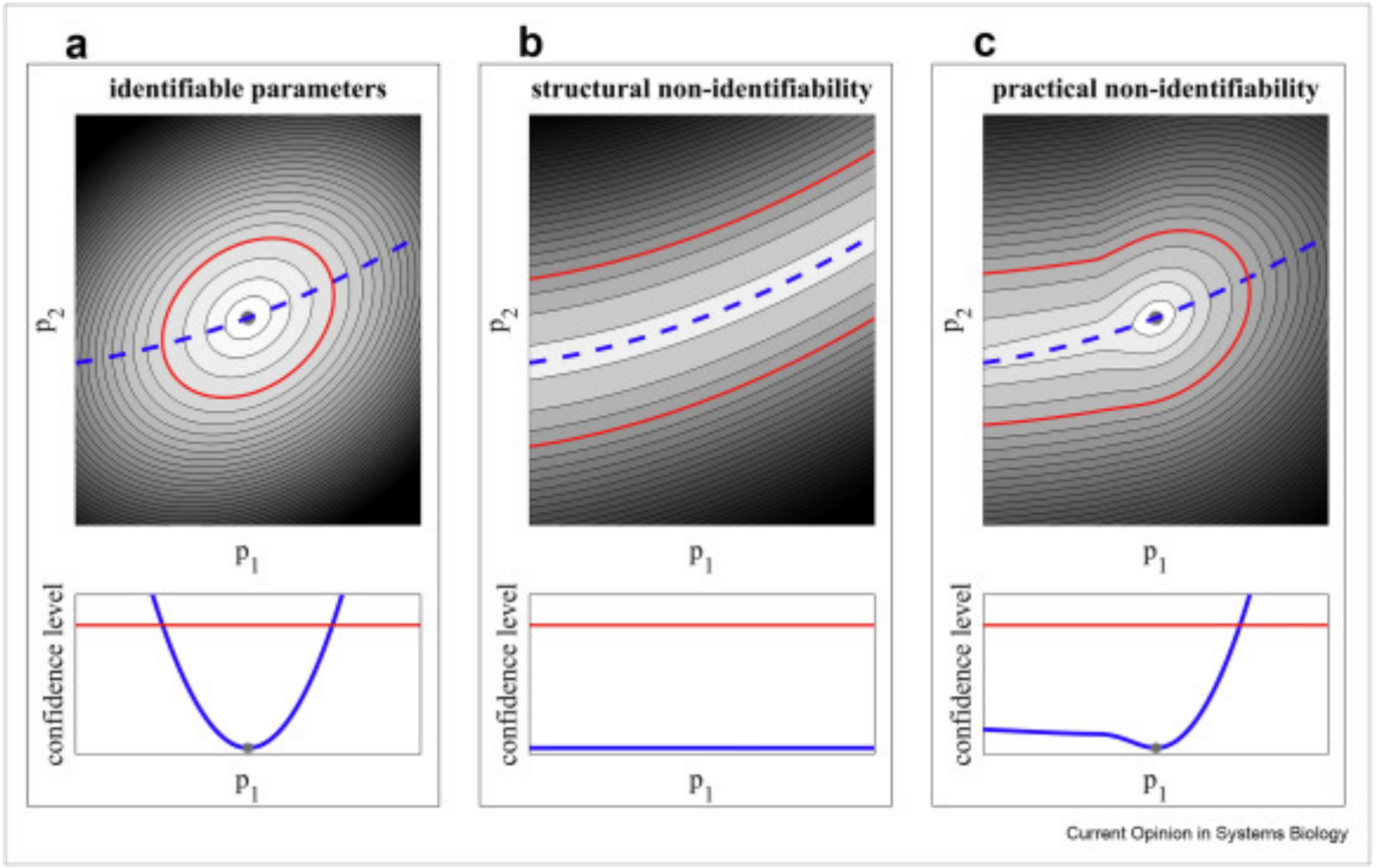
We use the decoder to compute the conditional probability

Sparse AEs as Generative Models

- *With this interpretation*

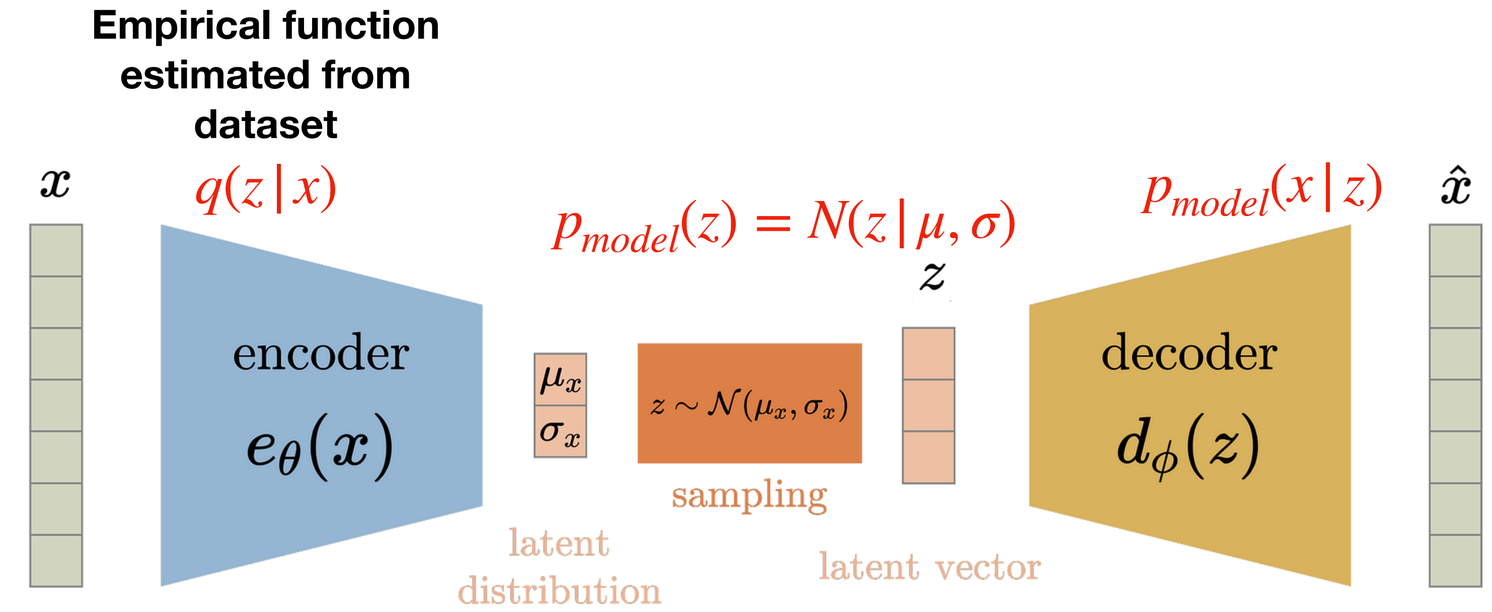
$$-\log p(x) = -\sum_{z} \log p(x \mid z) + \lambda \sum_{i} |z_i|$$

  - *The regularization term comes from the modeling of the distribution of the latent variables*
  - *The training corresponds to maximizing the likelihood, of the given data, over the model parameters and the latent variables*
    - *In the training we are profiling the dependence on z in the joint probability distribution, replacing the pdf with its maximum*
  - *In this perspective, training an autoencoder corresponds to approximately training a generative model. We will see next how this generalizes to account for the distribution of z (variational autoencoders)*



Variational Autoencoders

- *The VAE is a generalization of a sparse AE, in which one takes into account the full pdf in the latent space, rather than approximating it with its best-point value*
- *This is done*
  - *Assuming that the latent quantities are Gaussian distributed*
  - *The training forces the approximate latent distribution p(z|x) to be as close as possible to a Gaussian N(z)*
    - *The Kullback-Leibler divergence is used as a metric of distance*
- *This procedure is reflected in the architecture (modified wrt AE: the latent state nodes are the means and sigmas of the Gaussians) and the loss function*



$$\text{reconstruction loss} = \|x - \hat{x}\|_2 = \|x - d_\phi(z)\|_2 = \|x - d_\phi(\mu_x + \sigma_x \epsilon)\|_2$$

$$\mu_x, \sigma_x = e_\theta(x), \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$similarity\ loss = KL\ Divergence = D_{KL}(\mathcal{N}(\mu_x, \sigma_x) \parallel \mathcal{N}(\mathbf{0}, \mathbf{I}))$$

$$loss = reconstruction\ loss + similarity\ loss$$

Shannon Entropy and Kullback-Leibler Divergence

- Shannon Entropy: $H[p(x)] = -\mathbb{E}[log(p(x))] = -\int dx p(x) \log p(x)$
  - When the log is in base 2, Shannon Entropy has units of bits
- Kullaback-Leibler divergence $D_{KL}(p(x), q(x))$: number of bits required to transform q(x) into p(x)

$$D_{KL}(p(x), q(x)) = \int dx\ p(x) \log \frac{p(x)}{q(x)}$$

- From the definition

$$D_{KL}(p(x), q(x)) = H_p(q) - H_p(p)$$

- which explains why $D_{KL}(p(x), q(x))$ is the relative Shannon entropy between p(x) and q(x)

Shannon Entropy and Kullback-Leibler Divergence

- *The Kullaback-Leibler divergence is not a distance:*
  - $D_{KL}(p(x), q(x)) \neq D_{KL}(q(x), p(x))$
- *Jensen-Shannon divergence is a distance metric derived from the KL divergence*
  - $D_{JS}(p(x), q(x)) = \frac{D_{KL}(p(x), q(x)) + D_{KL}(q(x), p(x))}{2}$

ELBO and VAE loss

- Normally, we would train the VAE minimizing $-\log p_{model}(x)$, or maximizing $\log p_{model}(x)$
- Instead, we maximize a function $\mathscr{L}(q) \leq \log p_{model}(x)$ (q being the encoder function $q(z|x)$)
- In particular we choose

$$\mathscr{L}(q) = \mathbb{E}_{z\sim q(z|x)} \log p_{model}(z,x) + \mathscr{H}(q(z|x)) = \mathbb{E}_{z\sim q(z|x)}[\log p_{model}(z|x)p_{model}(x)] - \mathbb{E}_{z\sim q(z|x)} \log q(z|x)$$

$$= \mathbb{E}_{z\sim q(z|x)}[\log p_{model}(x)] + \mathbb{E}_{z\sim q(z|x)} \log \frac{p_{model}(z|x)}{q(z|x)} = \log p(x) - \int dz q(z|x) \frac{q(z|x)}{p_{model}(z|x)} =$$

$$\log p(x) - D_{KL}(q(z|x), p_{model}(z|x)) \leq \log p(x)$$

- Since D_{KL} is positive defined, the last inequality holds

ELBO and VAE loss

- To understanding what this ELBO is doing, we should write the loss differently

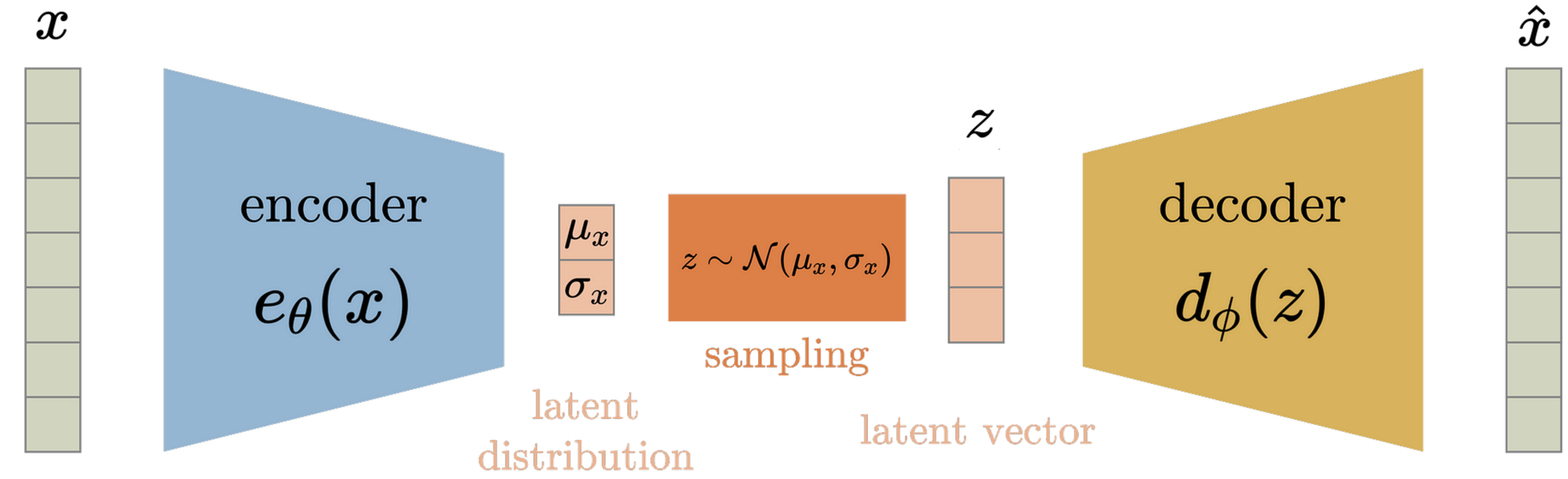
$$\mathscr{L}(q) = \mathbb{E}_{z\sim q(z|x)} \log p_{model}(z,x) + \mathscr{H}(q(z\,|\,x)) = \mathbb{E}_{z\sim q(z|x)}[\log p_{model}(x\,|\,z)p_{model}(z)] - \mathbb{E}_{z\sim q(z|x)} \log q(z\,|\,x)$$

$$= \mathbb{E}_{z\sim q(z|x)} \log p_{model}(x\,|\,z) + \mathbb{E}_{z\sim q(z|x)} \log p_{model}(z) - \mathbb{E}_{z\sim q(z|x)} \log q(z\,|\,x)$$

$$= \mathbb{E}_{z\sim q(z|x)} \log p_{model}(x\,|\,z) + \mathbb{E}_{z\sim q(z|x)} \log \frac{p_{model}(z)}{q(z\,|\,x)} = \mathbb{E}_{z\sim q(z|x)} \log p_{model}(x\,|\,z) - D_{KL}(q(z\,|\,x), p_{model}(z))$$

- This looks like the sparse autoencoder loss, where now the second term is forcing the approximate prior $q(z\,|\,x)$ to be as close as possible to some specific prior choice $p_{model}(z)$
- Usually, one uses a Gaussian for $p_{model}(z)$, so that the KL divergence is differentiable
- Usually, one puts a hyperparameter β in front of the second term. This is called β–VAE

The VAE loss decrypted



$$\text{reconstruction loss} = \|x - \hat{x}\|_2 = \|x - d_\phi(z)\|_2 = \|x - d_\phi(\mu_x + \sigma_x \epsilon)\|_2$$

$$\mu_x, \sigma_x = e_\theta(x), \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$similarity\ loss = KL\ Divergence = D_{KL}(\mathcal{N}(\mu_x, \sigma_x) \parallel \mathcal{N}(\mathbf{0}, \mathbf{I}))$$

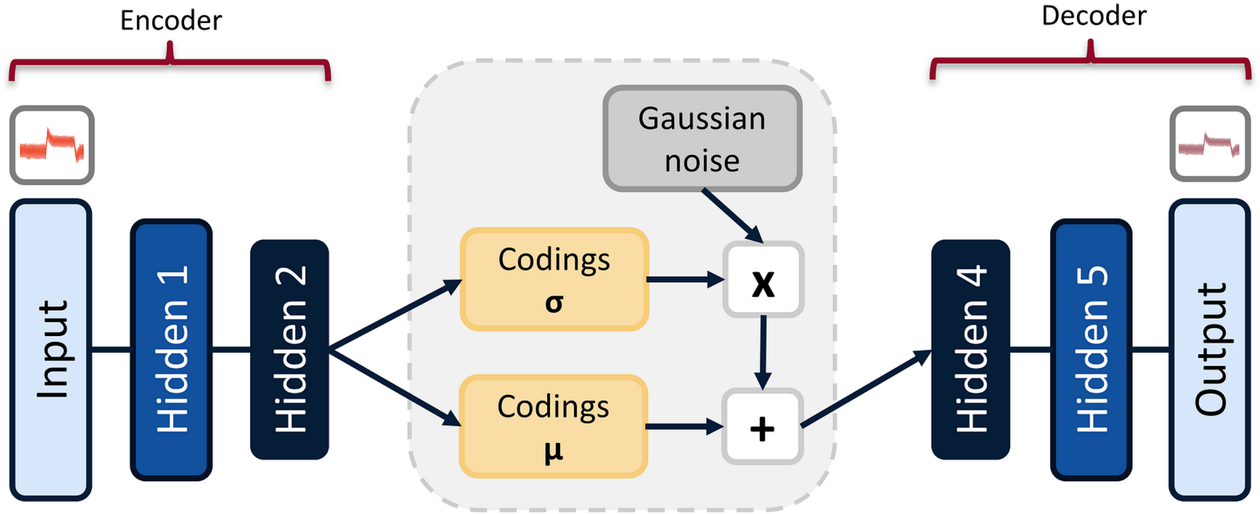
$$loss = reconstruction\ loss + similarity\ loss$$

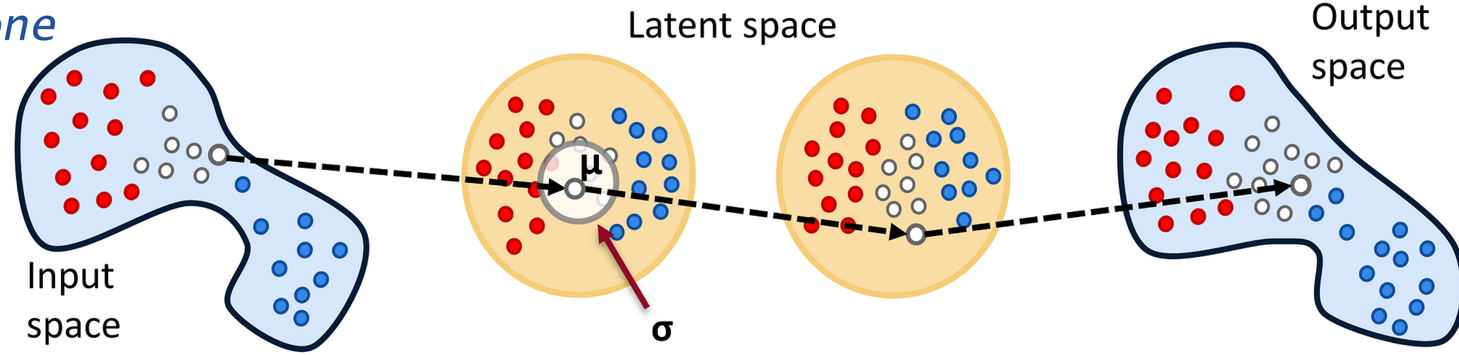
What the VAE loss does

- *The reconstruction terms ensures that the decoded data are close to the input (as for the plain AE)*
  - *We have a metric of distance (the loss) that can be used for Anomaly Detection*
- *The similarity term ensures that the convergence of the training prefers a solution in which the latent variables are distributed Gaussian*
  - *We can then sample from a Gaussian and obtain new examples, similar to the input data*
    - *The VAE is a generative model*

VAEs for Anomaly Detection

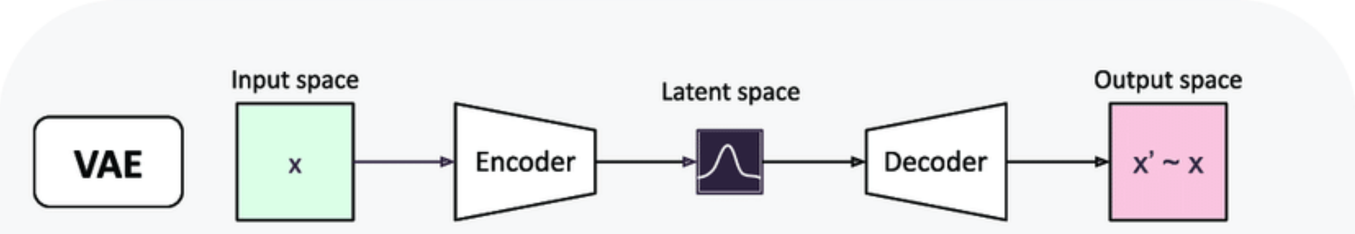
- *The idea of anomaly detection for (V)AEs is that*
  - *The network learns to decode standard data with high fidelity*
  - *The decoding fails when data not belonging to the original dataset (outliers/anomalies) are processed*
  - *A large loss indicates an anomaly, so it can be used as an anomaly score*
- *Given the rich structure of the VAE, one has other options*
  - *For instance, the Gaussian in the latent space might be pulled away from 0, so that a large* $|\mu| = \sqrt{\sum_i \mu_i^2}$ *can be used as anomaly score*

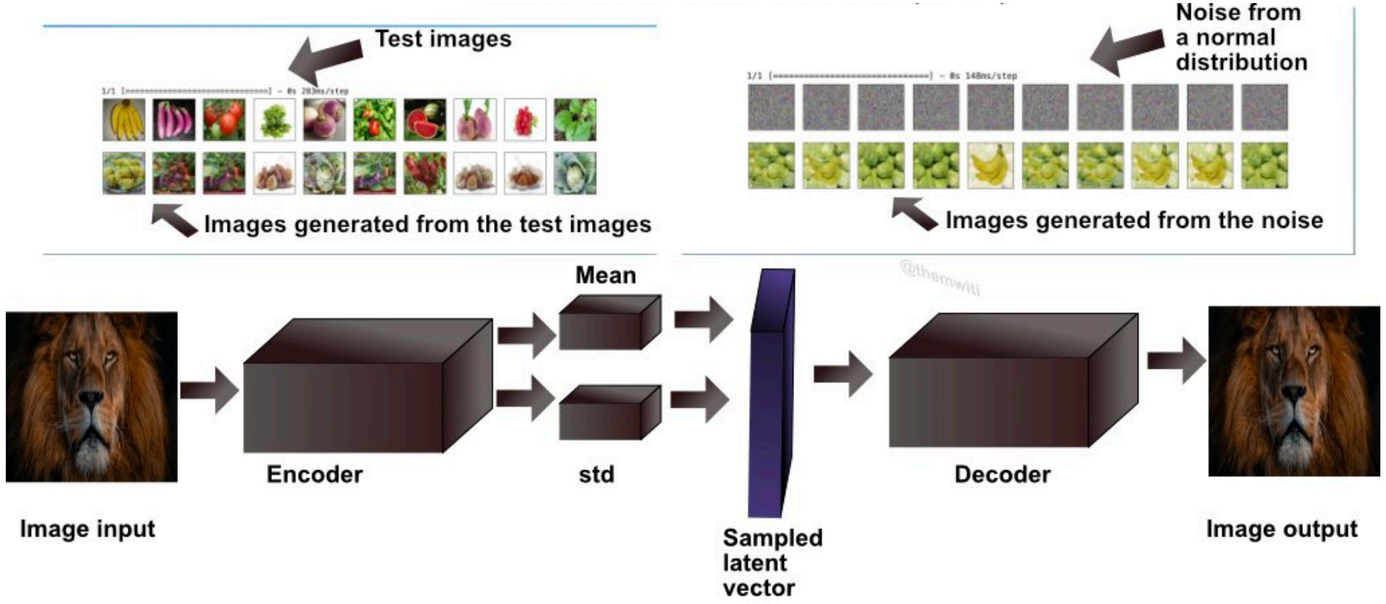




VAEs as Generative Models

- *In inference, the VAE returns an output starting from a randomly sampled Gaussian vector*
  - *The inference is not deterministic: for a given input one can have various outputs*
- *One can then use the decoder of a VAE as a generative algorithm and creates new examples*





VAEs as Generative Models

- *Passing all information through the bottle neck is complicated*
  - *This typically results in blurred images for convolutional VAE*
- *This is why VAEs to be used for generation are usually designed to be over complete*
- *Also, one plays with β to enhance the generating capability over the reconstructing capability*
  - *Which makes them great for generation, but bad for anomaly detection*
- *We will see next week that there are other options for generative models*