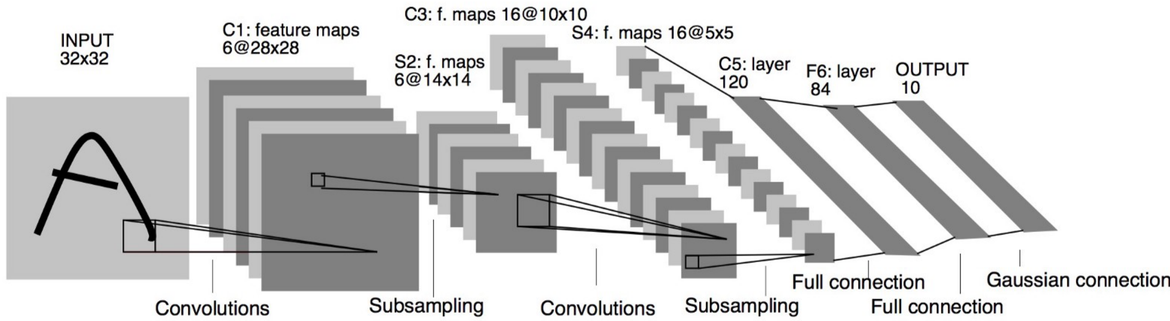
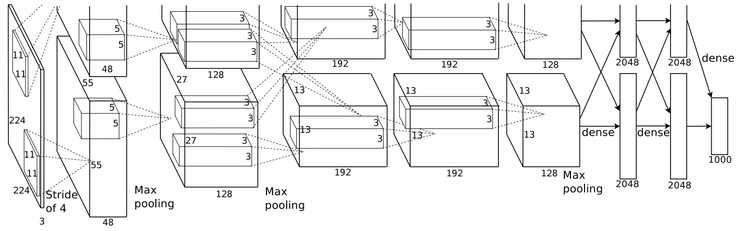
Lecture 5: Autoencoders, Unsupervised Learning and Anomaly Detection

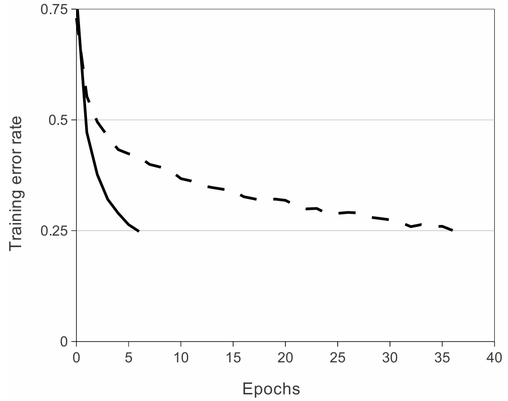
Maurizio Pierini (CERN), Thea Aarrestad (ETH)

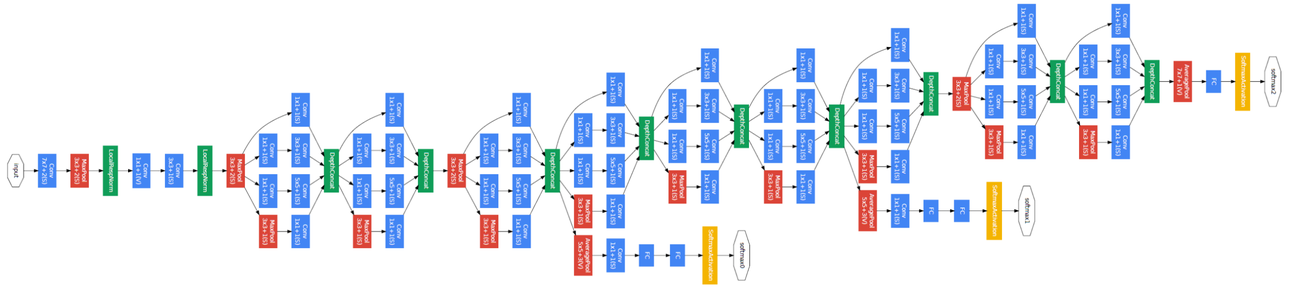
A history of ConvNNs

- *Neocognitron (1980): translation-invariant image processing with NNs*
- *LeNet (1989): considered the very first ConvNN, designer for digit recognition (ZIP codes)*
- *AlexNet (2012): the first big ConvNN (60M parameters, 650K nodes), setting the stage of the art: trained on GPUs, using ReLU and Dropout*
- *GoogleNet (2014): built on AlexNet, introduced an inception model to reduce e the number of parameters*



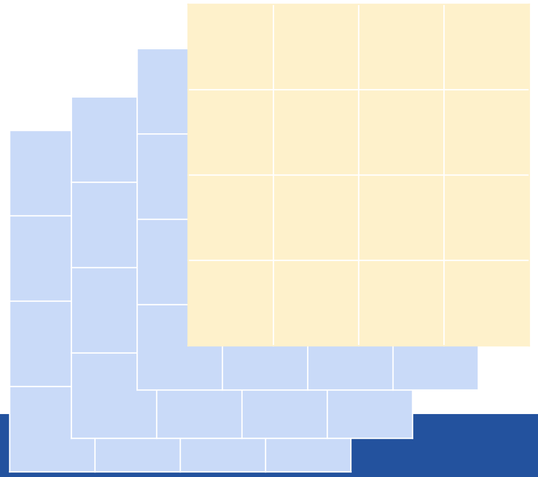


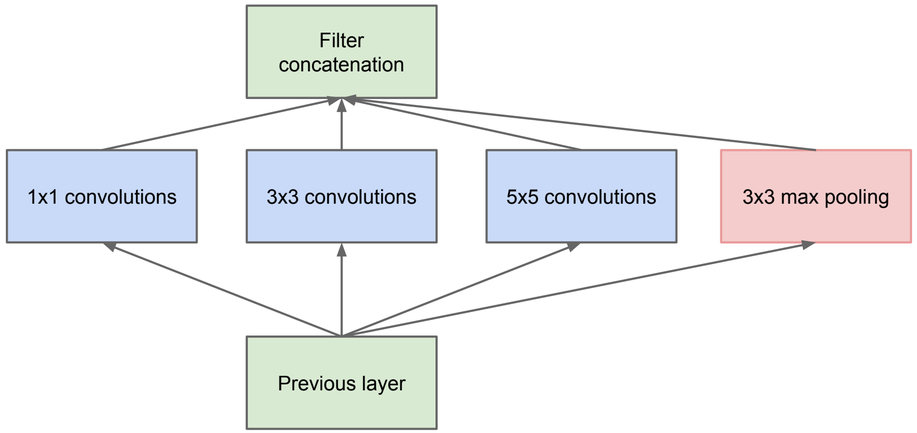




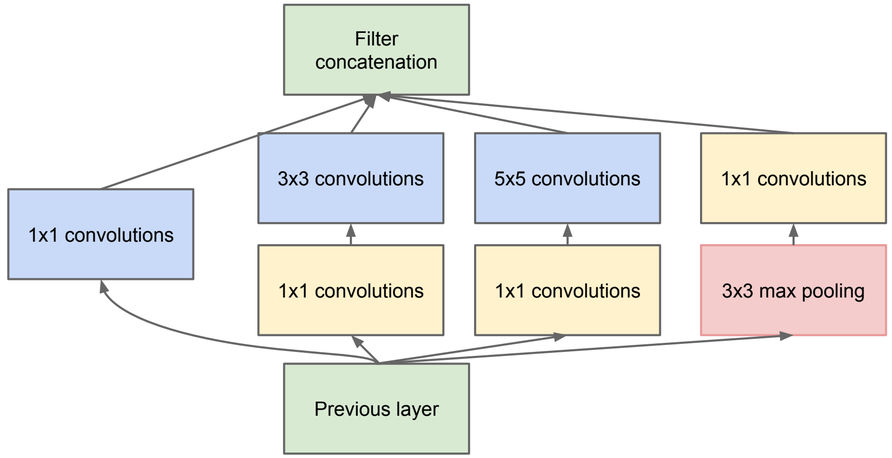
Inception Module

- *Rather than going deeper and deeper, inception architectures go wider*
- *Several conv layers, with different filter size, process the same inputs*
- *This way, more features can be detected from the same image*
- *The outcome of this parallel processing is then recombined through a concatenation step as channels of an image*





(a) Inception module, naïve version

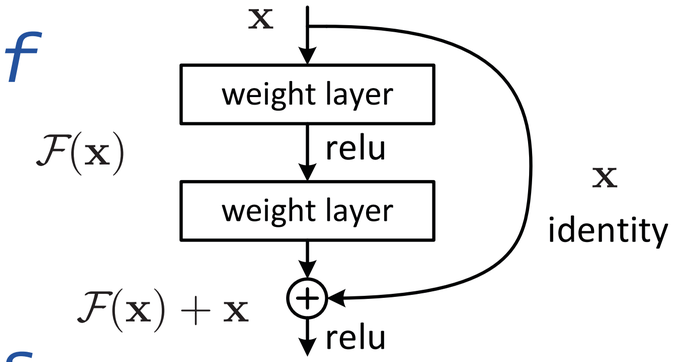


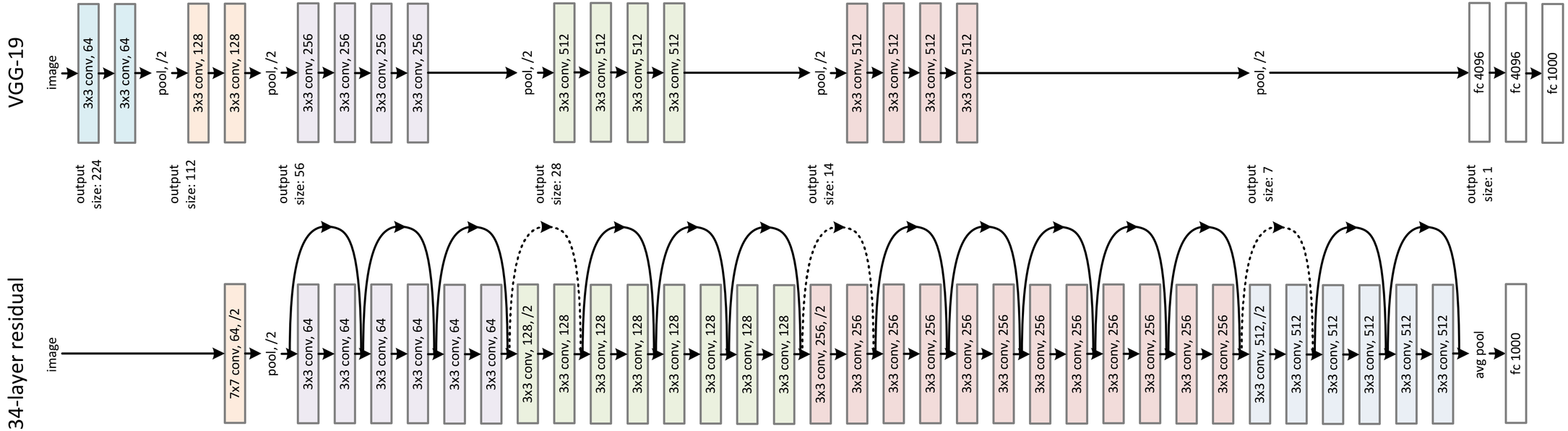
(b) Inception module with dimension reductions

A history of ConvNNs

- *VGGNet(2014):exploit small filters (3x3, 1x1), previously considered not optimal in a stack of Conv layers*

- *ResNet (2015):implemented skip connections, which allows to focus the learning on residuals (difference between inputs and outputs)*

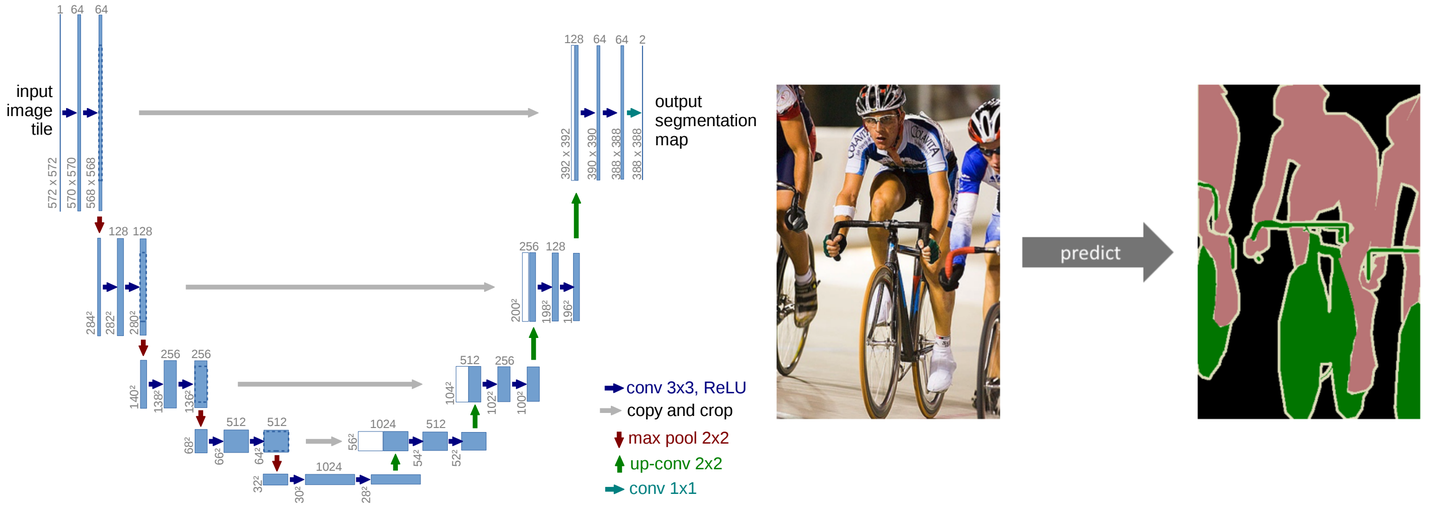




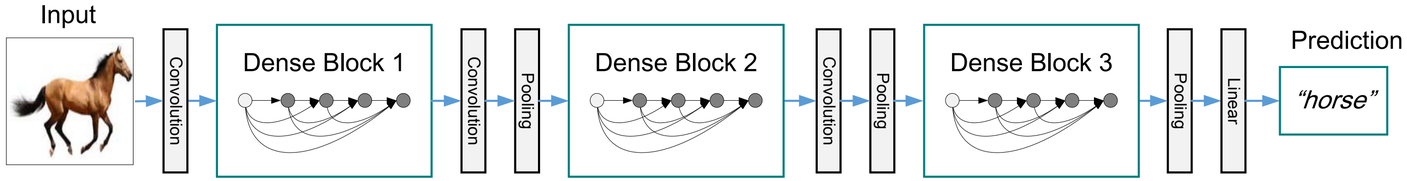
Check this blog entry on ConvNNs

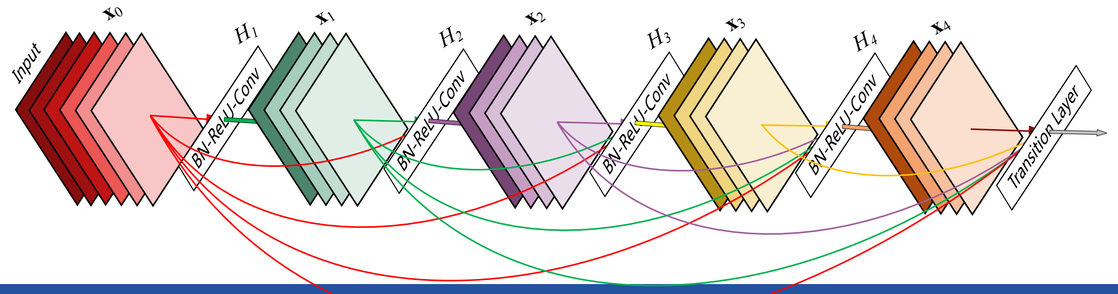
A history of ConvNNs

- *U-net (2015): conv layers with skip connections, in a downsampling+upsampling U-shaped sequence. Introduced for semantic segmentation*



- *DenseNet (2016): uses skip connections between a given layer and all the layers downstream in a dense block, with Conv layers in between*

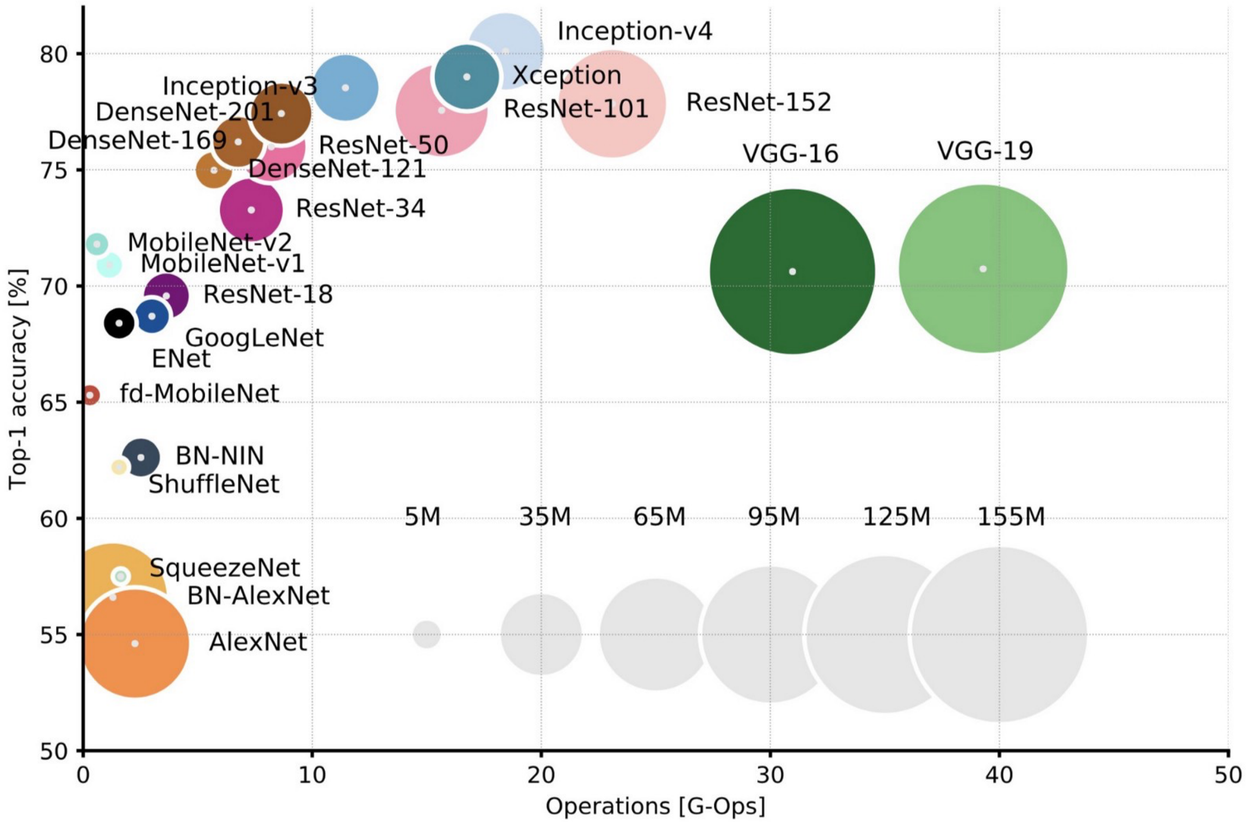




The computational cost

- *In this evolution, computing-vision networks drastically improved in performance*
- *This came at a big cost in complexity (number of parameters and operations)*
- *The inference with this network became particularly slow*
  - *big interest in optimize these networks on dedicated resources, e.g. FPGAs*
- *Many cloud providers provide optimized versions of these networks:*
  - *you can just use them (re-training if needed), rather than inventing your own one*

https://towardsdatascience.com/neural-network-architectures-156e5bad51ba

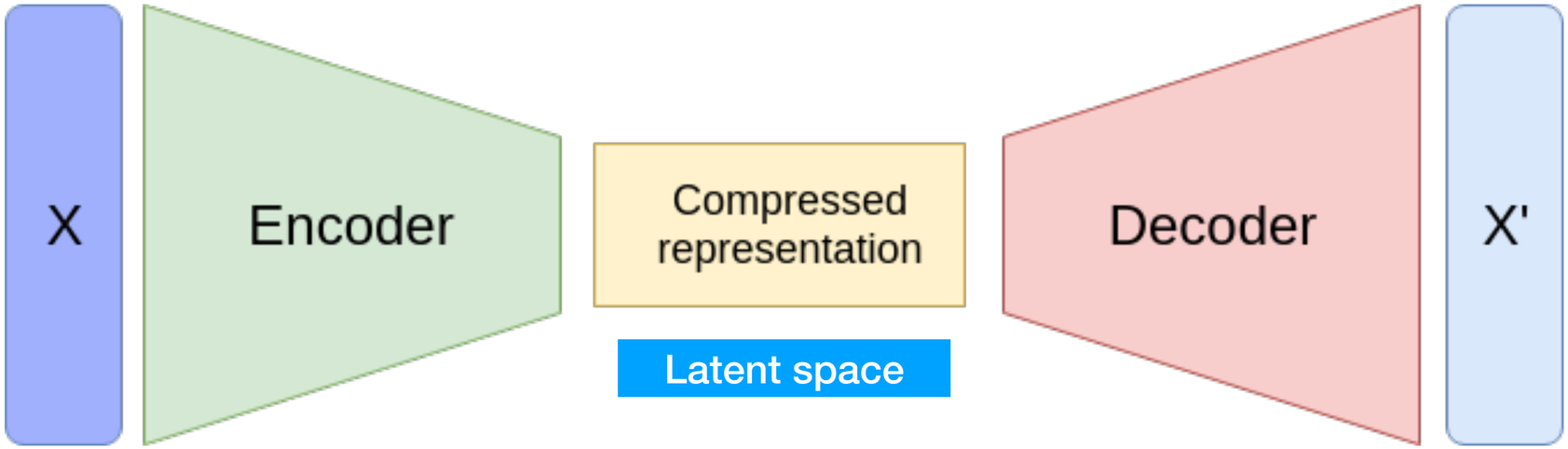


Program for Today

Date	Topic	Tutorial
Sep 16	Intro & class description	
Sep 23	probability Theory + Basic of machine learning + Dense NN	*Basic jupyter + DNN on mnist (give jet dnn as homework)*
Sep 30	Statistics + Convolutional NN	*Convolutional NNs with MNIST*
Oct 7	Training in practice: regularization, optimization, etc	*Free Practice*
Oct 14	Unsupervised learning and anomaly detection	*Autoencoders with MNIST*
Oct 21	Graph NNs	*Tutorial on Graph NNs*
Oct 28	Generative models: GANs, VAEs, etc	*TBD*
Nov 4	Normalizing flows	*TBD*
Nov 11	Transformers	
Nov 18	Network compression (pruning, quantization, Knoweldge Distillation)	
Nov 25	Tutorial on hls4ml **To Be Confirmed**	
Dec 2	To Be Decided	
Dec 9	Quantum Machine Learning	
Dec 16	**Exams To Be Confirmed**	

Autoencoders

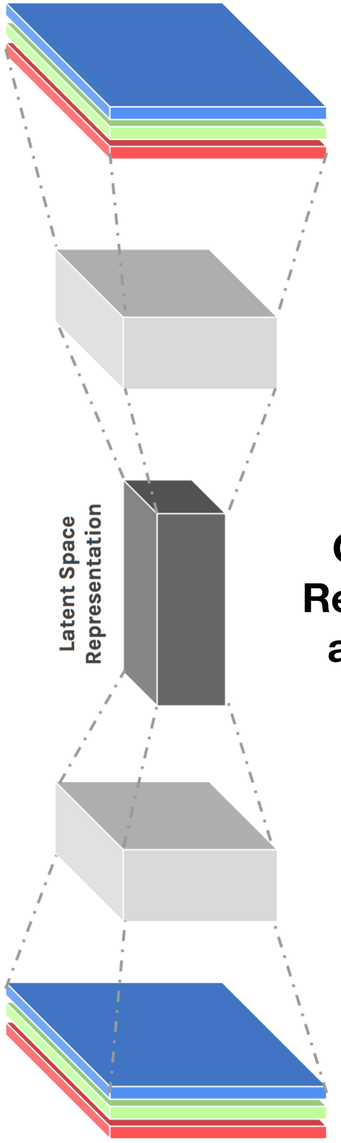
- *Autoencoders are networks trained to copy its input to its output*
  - *They consist of an encoder* $z = q(x)$ *and a decoder* $x' = p(z) = p(q(x))$
  - *They have a typical "bottleneck" structure (under complete autoencoders): they go from* $\mathbb{R}^n \rightarrow \mathbb{R}^k$ *and from* $\mathbb{R}^k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



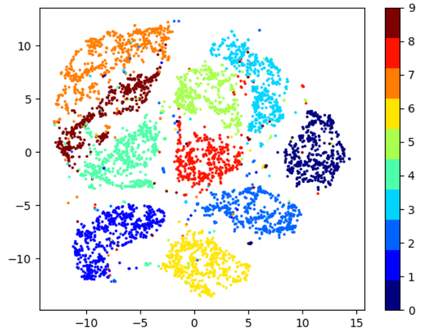
Importance of Bottleneck

- *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 prioritise the relevant aspects and discard the rest.This is what force the network to learn from data as opposed to just return a network in which the encoder mirrors the encoder (trivial solution).*
- *This make autoencoders useful for*
  - *Data compression*
  - *Clustering*
  - *Anomaly Detection*

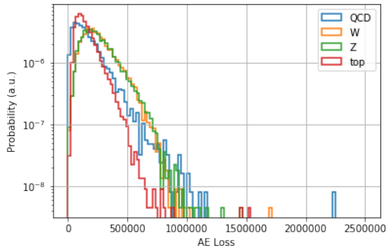
UNSUPERVISED TASKS



Compressed Representation, also good for clustering

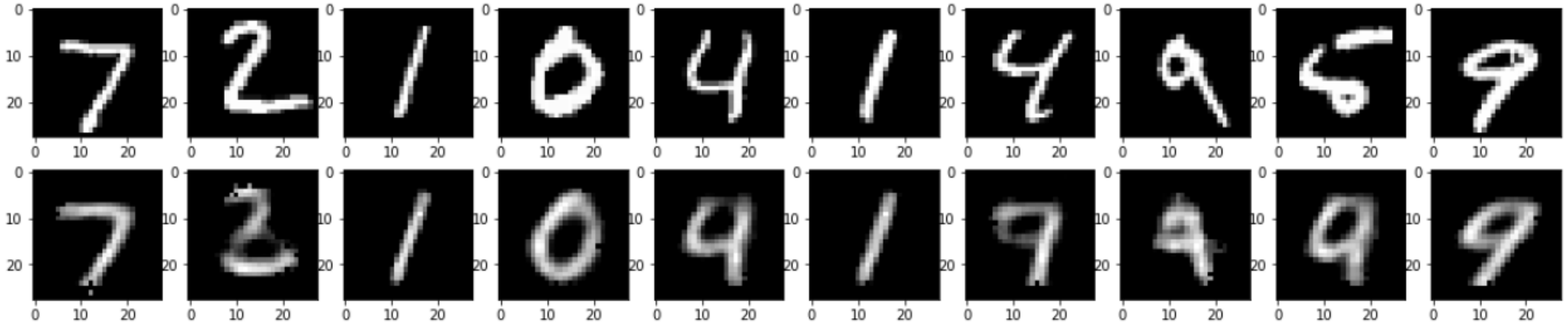


Distance to input for Anomaly Detection



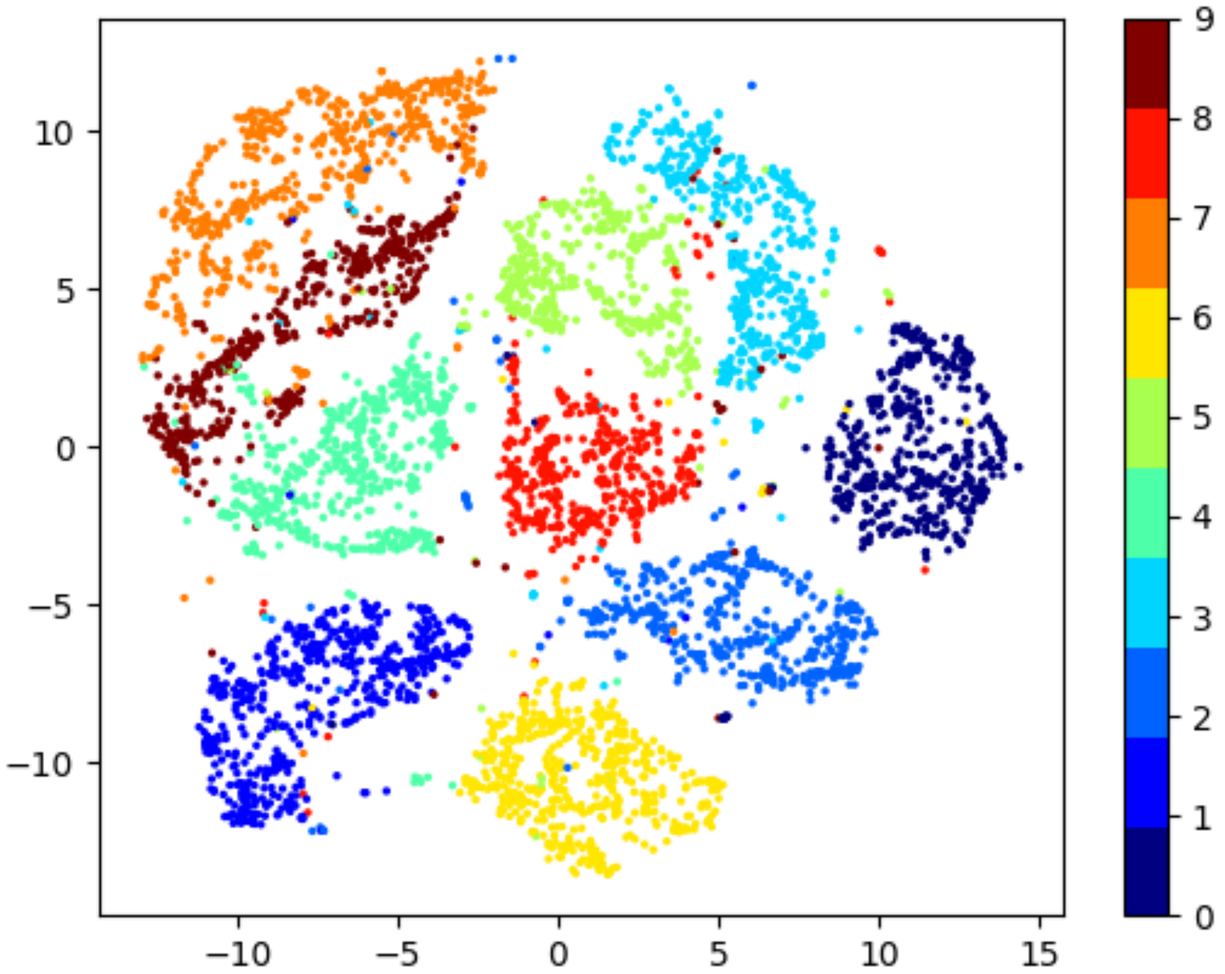
Compression

- *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*



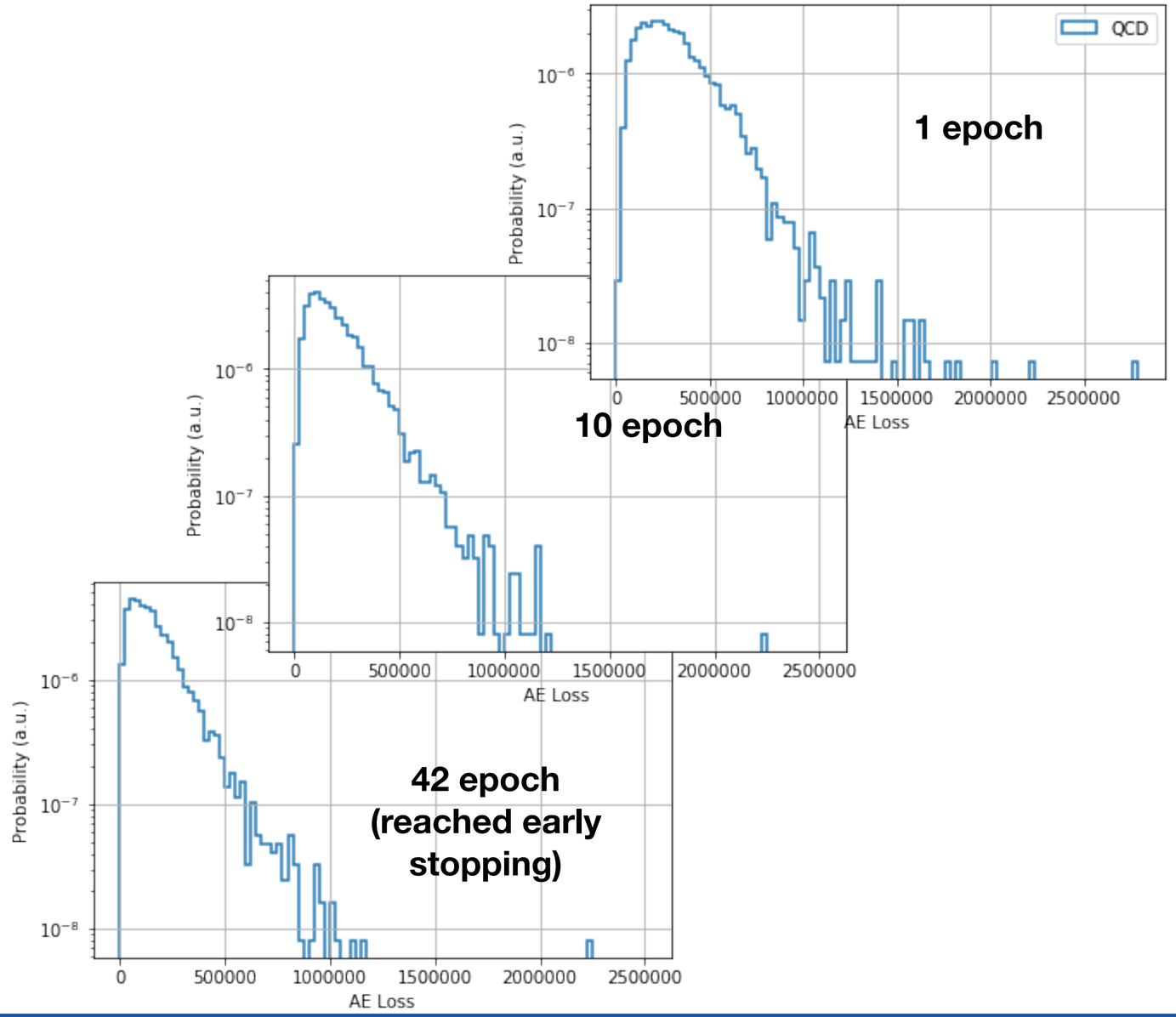
Clustering

- *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*



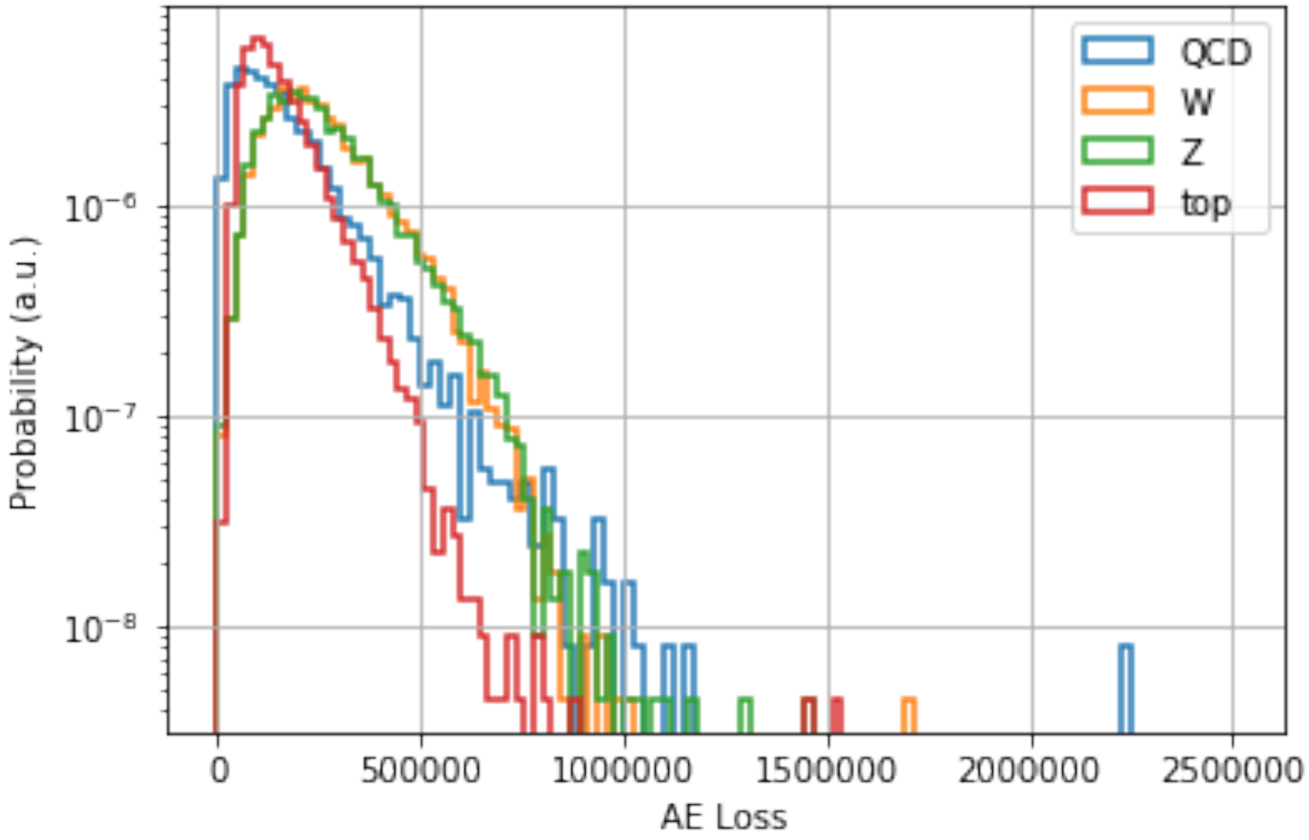
Anomaly Detection

- *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*



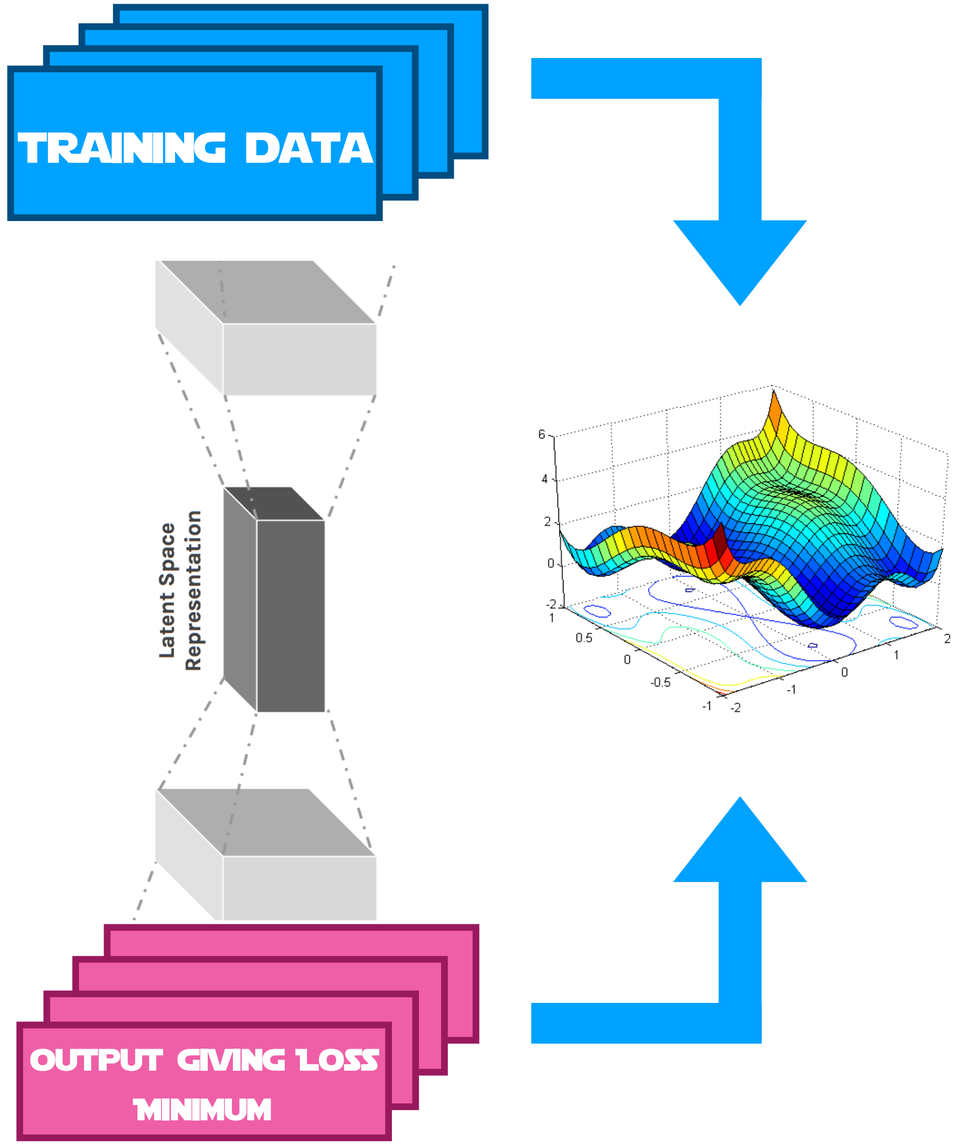
Anomaly Detection

- *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*



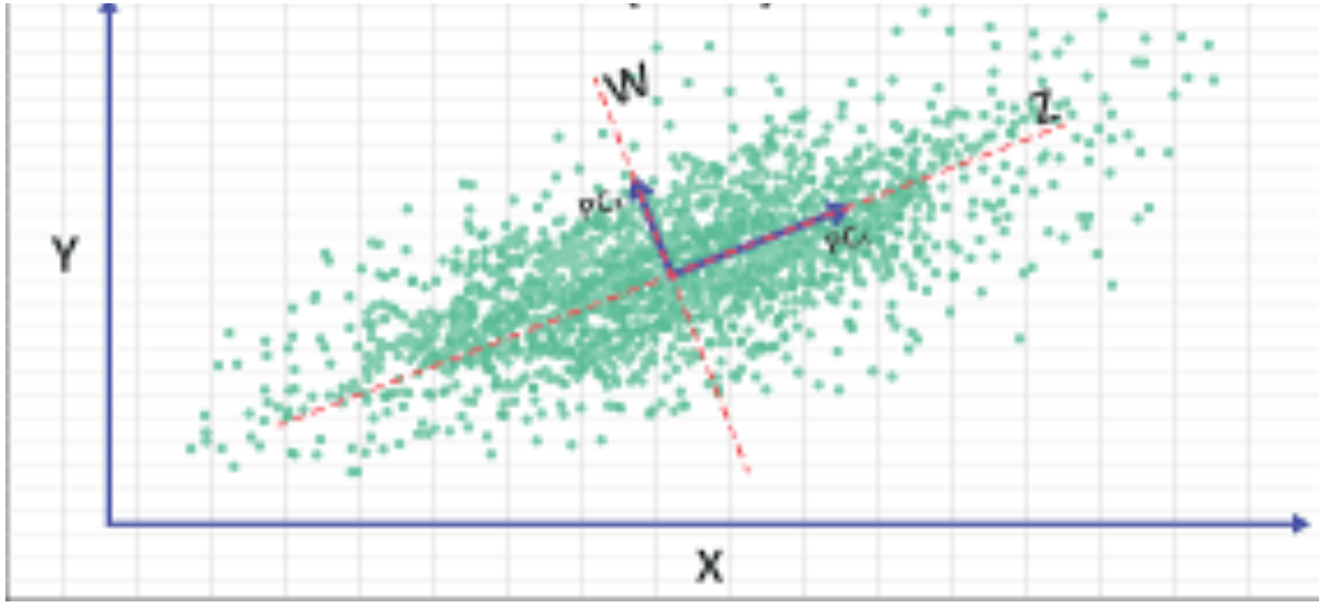
Training and Autoencoder

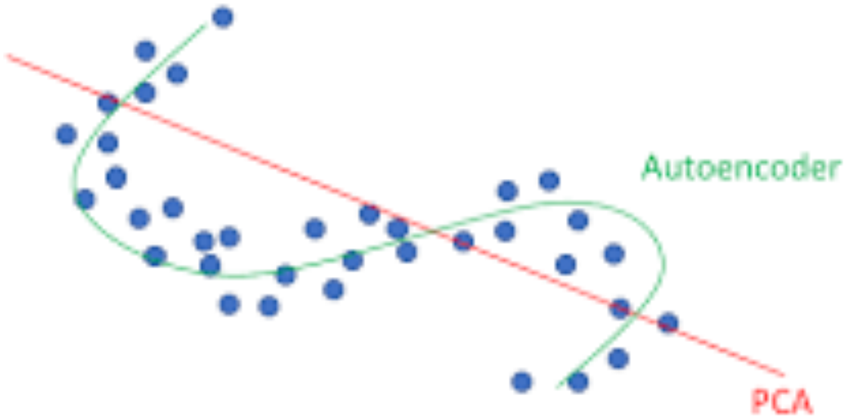
- *Ingredients:*
  - *A training dataset* x
  - *A model returning an output* x' *with the same dimensionality as* x
  - *A loss function consisting of a metric that penalises* x' *if far away from* x
    - *For example,* x *and* x' *could be images and the loss could be the pixel-by-pixel MSE (mind the underlying assumptions)*
  - ***No ground truth y**: this is an **UNSUPERVISED** training*



Autoencoder vs PCA

- *Principal Component Analysis (PCA) is a linear process used to characterized data in a smaller dataset*
  - *Diagonalize the covariance matrix* and take the first k eigenvectors*
- *Autoencoders generalise this procedure*
  - *A **linear** autoencoder with MSE performs basically a PCA*
  - *A non-linear autoencoder can learn more from the data (more effective compression, etc.)*





*covariance matrix: go back to the stat classes mid you don't remember what it is

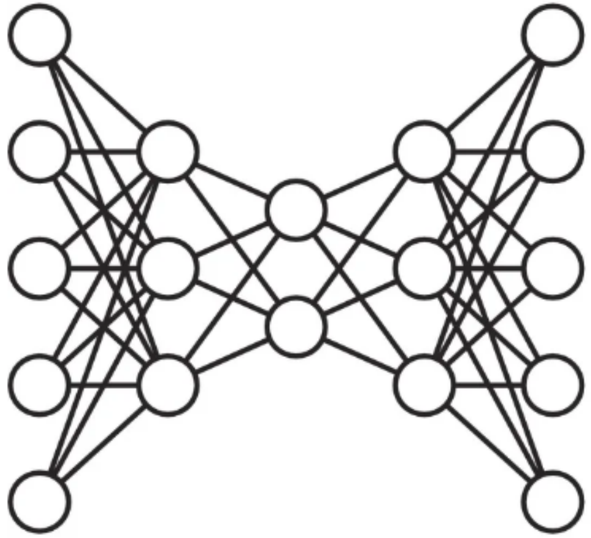
Do We Need a Bottleneck?

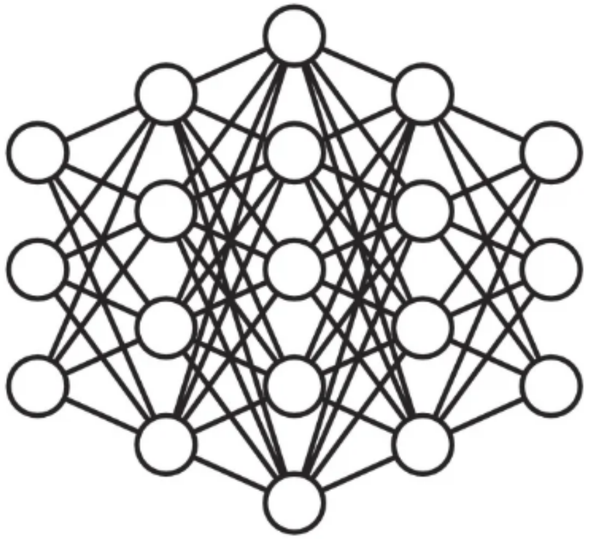
- *Autoencoders can be undercomplete (inner dimensionality = input) or over complete (inner dimensionality > input)*

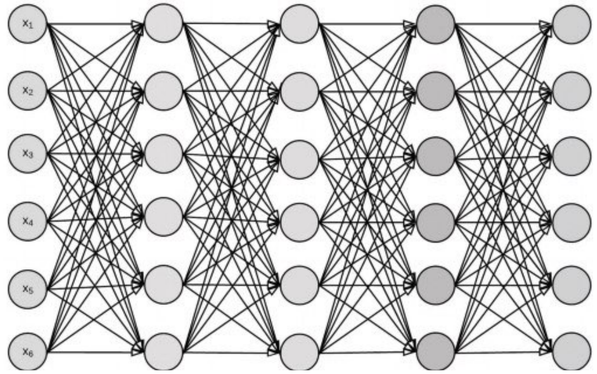
- *One can still force the autoencoder towards a lossy compression using regularization: add to the loss a term that limits the model capacity*

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

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

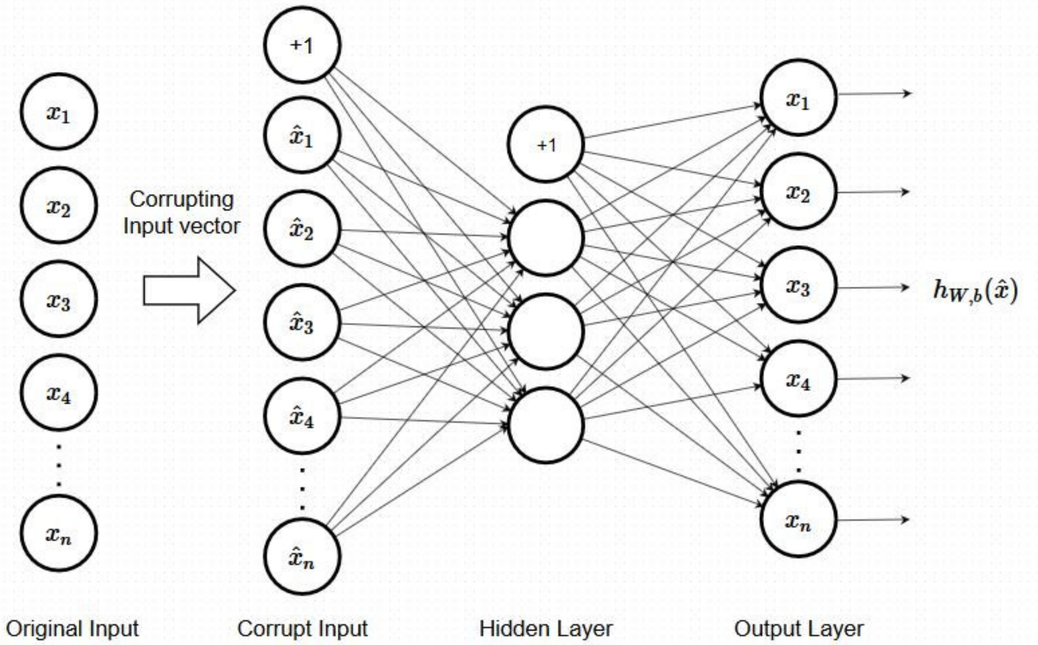


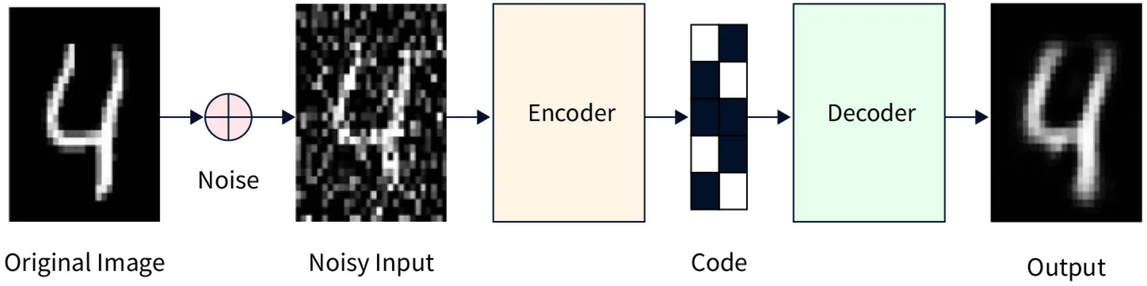




Denoising Autoencoders

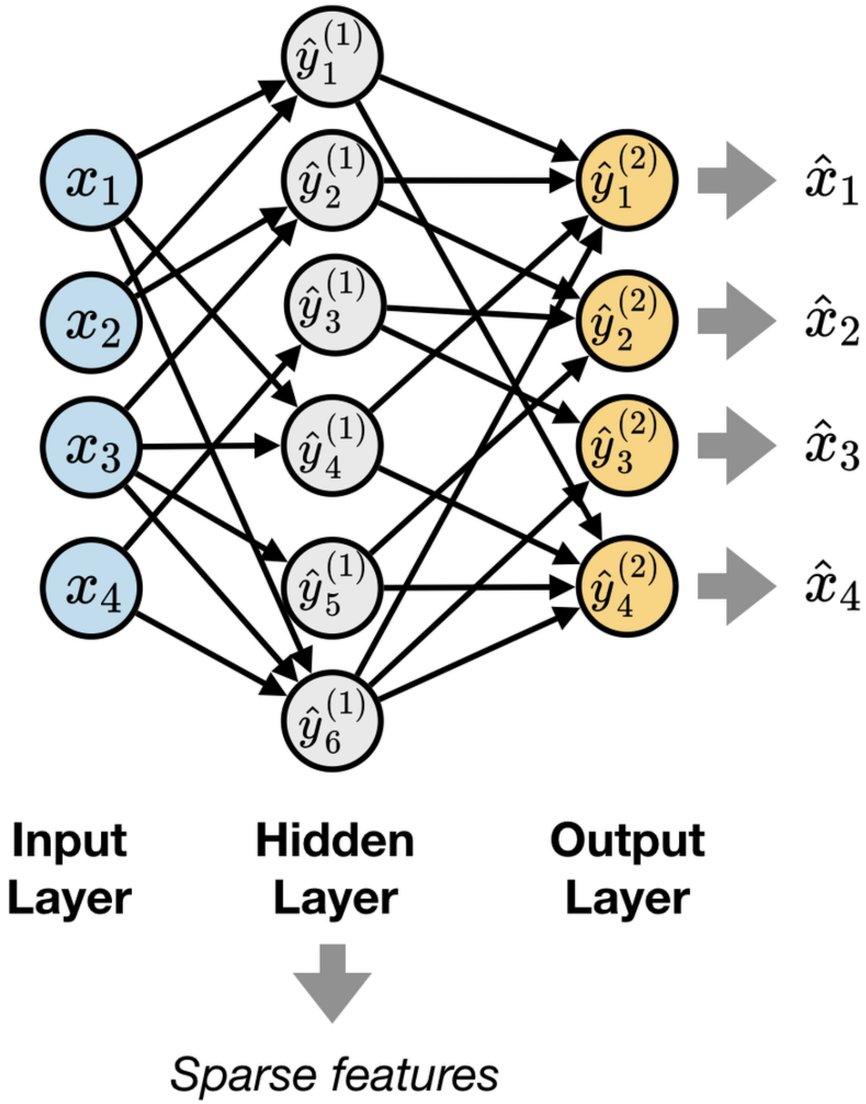
- A denoising AE is an AE trained to be robust against a preturbation of the data
- This is done minimizing the loss $L(x, p(q(\tilde{x})))$, where $\tilde{x}$ is obtained adding some noise to x
- Because of the noise added
  - The network is regularized so that it is prevented from learning the identity
  - 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 as feature engineering 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* $\pi(z) \propto e^{-\lambda|z_i|}$ *and then obtain*

The sparsity loss term

$$-\log p(x) = -\sum_z \log p(x\,|\,z) + \lambda \sum_i |z_i|$$

The usual loss term, keeping in mind that $z = q(x)$

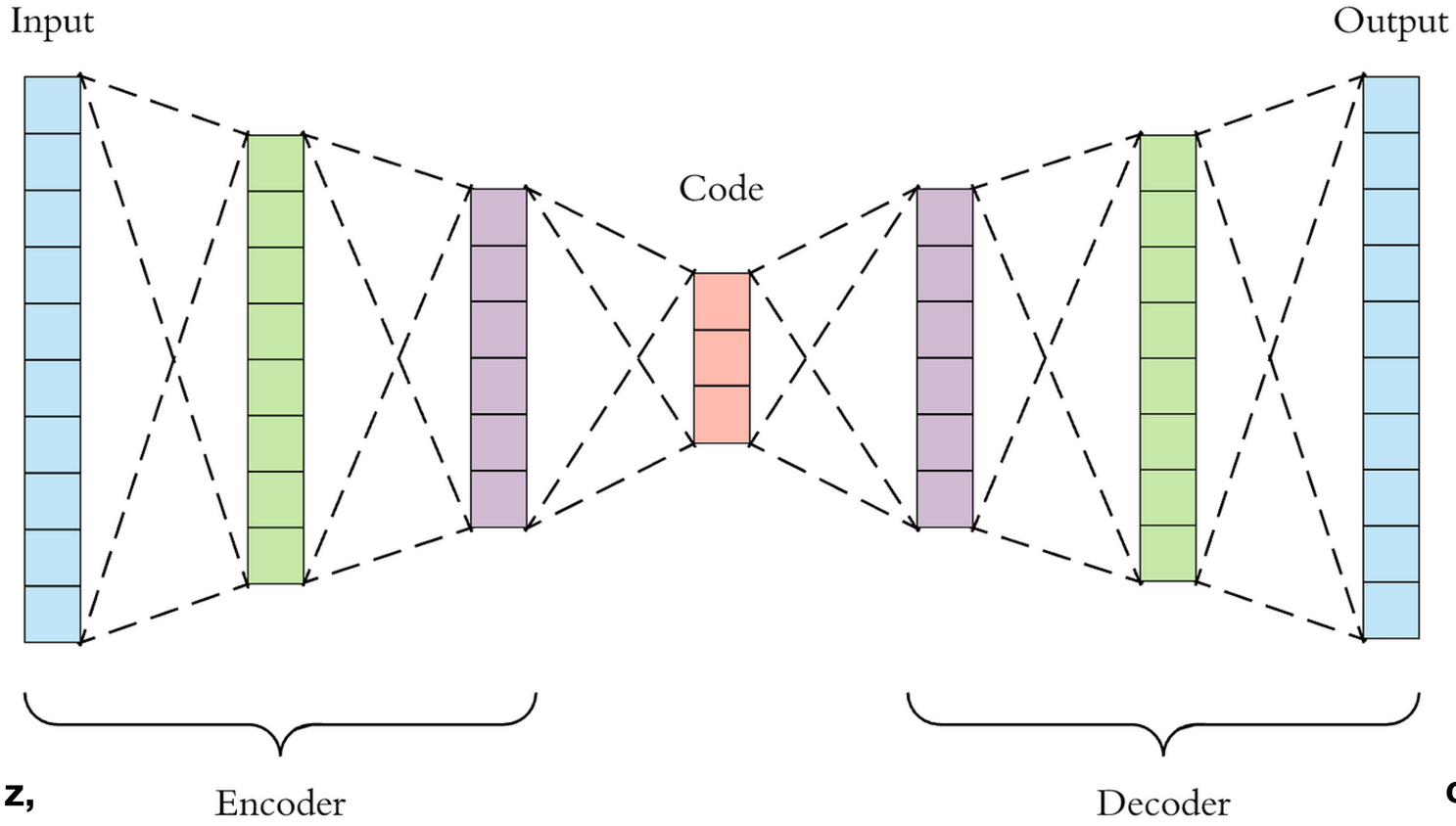
hyperparameter

A perfectly trained plain AEs

The loss is some function of $\vec{x}$, given the latent-space quantities $\vec{z}$

$$L(x) = -\log p(x \mid z)$$

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



Perfectly trained implies that $x' = x$

There is no knowledge of the probability distribution of z

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

$$z = q(x)$$

We use the decoder to compute the conditional probability

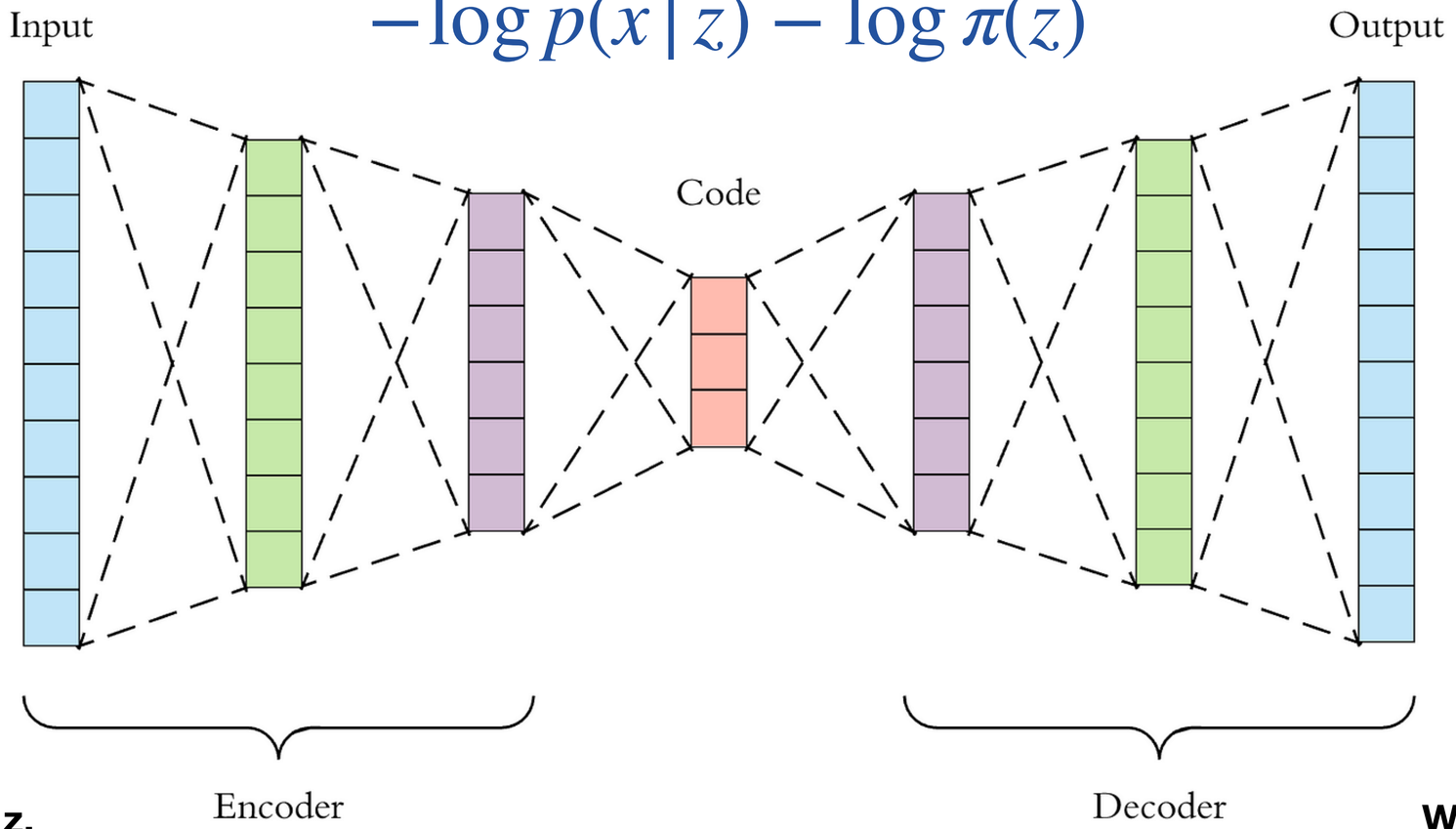
$$p(x \mid z) = p(x \mid q(x))$$

From AEs to Generative AEs

The loss is still trying to approximate the probability of x

$$L(x) = -\log p(x) = -\log p(x \mid z)\pi(z) =$$
$$-\log p(x \mid z) - \log \pi(z)$$

The regularization term takes into account the distribution of z (the choice of z is indirectly an arbitrary choice of a prior)



Perfectly trained implies that $x' = x$

The regularization term introduces a prior on the probability distribution of z

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

$$z = q(x)$$

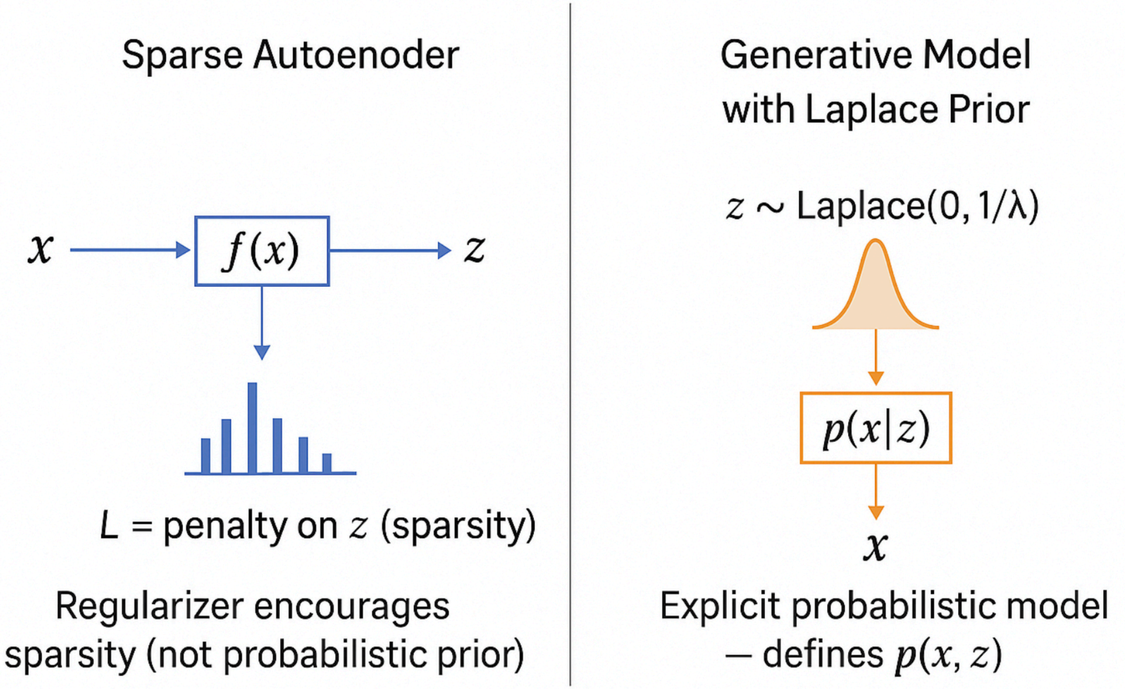
$$p(x \mid z) = p(x \mid q(x))$$

We use the decoder to compute the conditional probability

Almost a Generative Model

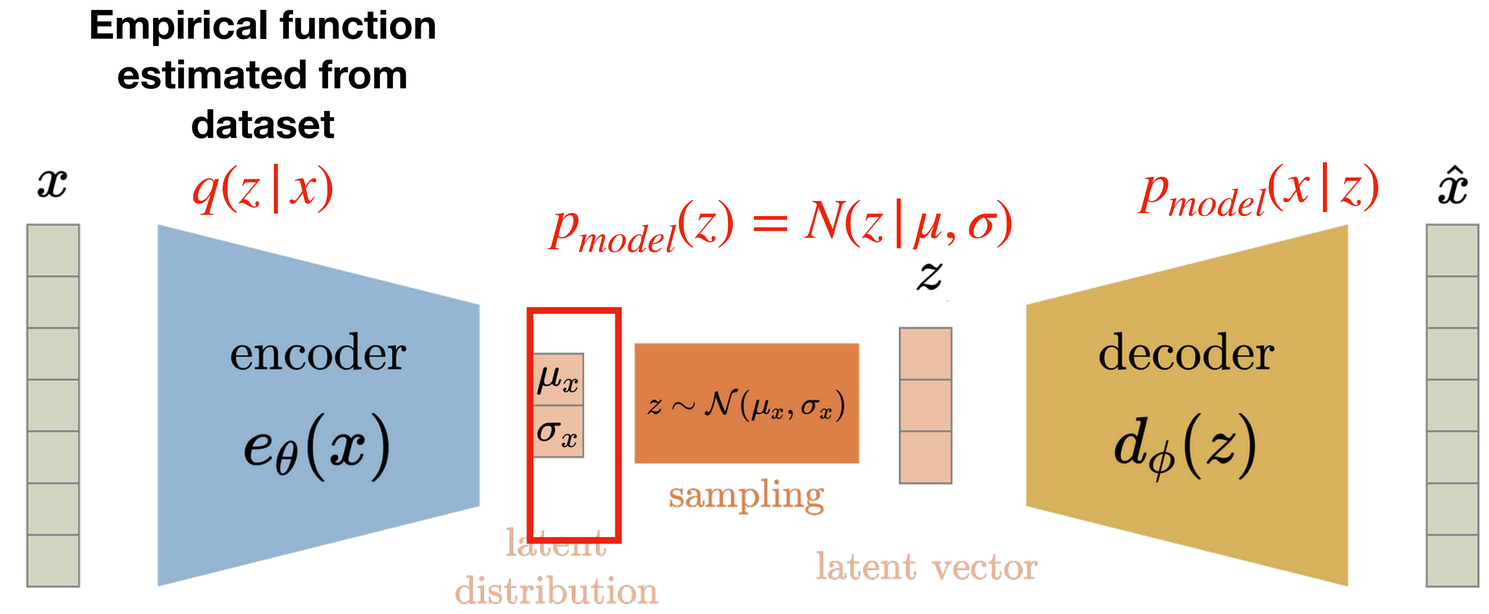
- With a generator, one can create new instances of your data (e.g., start from z and create new images from a decoder of a Convolutional AE)
  - Sample a point z from the prior $\pi(z)$
  - Apply the decoder to that point z
  - You will obtain an image x' sampled with probability $p(x) = p(x|x)\pi(z)$
- Ideally, a sparse AE looks like a generative model in which $\pi(z) \propto e^{-\lambda|z_i|}$
  - It is not that yet, because we don't have a prior distribution but only a deters,inistic outcome (for a given x, we get the same z and the same output $p(z) = p(q(x))$ every time)
  - Instead, we would like to have a network that gives different outputs (with an underlying distribution) from the same z inputs

Sparse Autoencoder vs. Generative Model with Laplace Prior



Variational Autoencoders

- *The VAE is a conceptually similar to a sparse AE, but with elaborate regularisation technique*
  - *For a given x, we want to learn a distribution p(z) in the latent space, and not just a point estimate for z*
  - *We do that changing the network structure and the regularisation strategy*
    - *The regulation term forces the approximate latent distribution p(z|x) to be as close as possible to a target function, minimizing the Kullback-Leibler divergence between the learned function and the target function*
- *This procedure is reflected in the architecture (modified wrt AE: the latent state nodes are the parameters of the function) and the loss function (extra regularisation term)*



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

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

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

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

Shannon Entropy and Kullback-Leibler Divergence

- Shannon Entropy: $\mathcal{H}[p(x)] = -\mathbb{E}[log(p(x))] = -\int dx p(x) \log p(x)$ (which becomes $\mathcal{H}[p(x)] = -\sum p(x) \log p(x)$ when approximating the integral by a sum, as in our case)
  - When the log is in base 2, Shannon Entropy has units of bits and is a measurement of amount of information
- 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

$$D_{KL}(p(x), q(x)) = \mathcal{H}_p(q) - \mathcal{H}_p(p)$$

- 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 $-\log p(x)$, or maximizing $\log p(x)$, where we want to enforce the dependence on some z with some prior
  - We don't know the prior and we want to approximate it via the encoder $z \sim q(x)$
- An effective training strategy consists in maximizing a function $\mathscr{L}(q) \leq \log p(x)$ (q being the encoder function $q(z|x)$). Doing so, p(x) will also be pushed to its maximum
- In particular we choose

$$\mathscr{L}(q) = \int dz q(z|x) \log p(z,x) + \mathscr{H}(q(z|x)) = \int dz q(z|x)[\log p(z|x)p(x)] - \int dz q(z|x) \log q(z|x)$$

$$= \int dz q(z|x)[\log p(x)] + \int dz q(z|x) \log \frac{p(z|x)}{q(z|x)} = \log p(x) - \int dz q(z|x) \log \frac{q(z|x)}{p(z|x)} =$$

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

- 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, keeping in mind that we have a discrete set of z distributed as $z \sim q(z|x)$ and that $\int dz q(z|x) f(z) = E_{z\sim q(z|x)} f(z)$

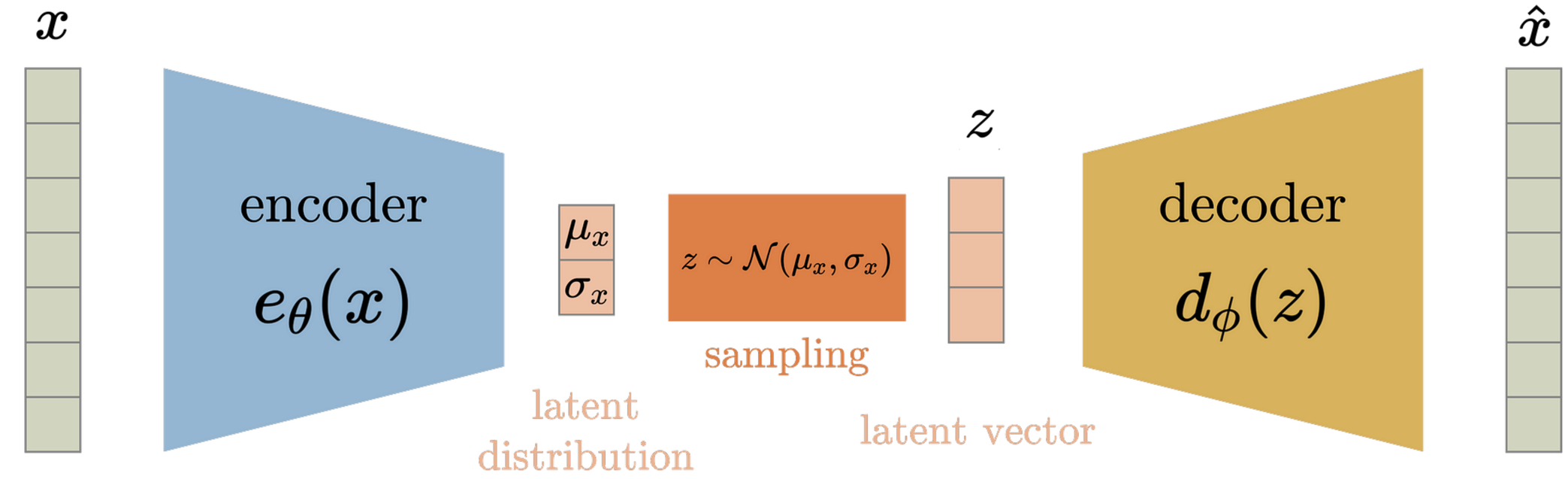
$$\mathscr{L}(q) = \mathbb{E}_{z\sim q(z|x)} \log p(z,x) + \mathscr{H}(q(z|x)) = \mathbb{E}_{z\sim q(z|x)}[\log p(x|z)p(z)] - \mathbb{E}_{z\sim q(z|x)} \log q(z|x)$$

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

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

- 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



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

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

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

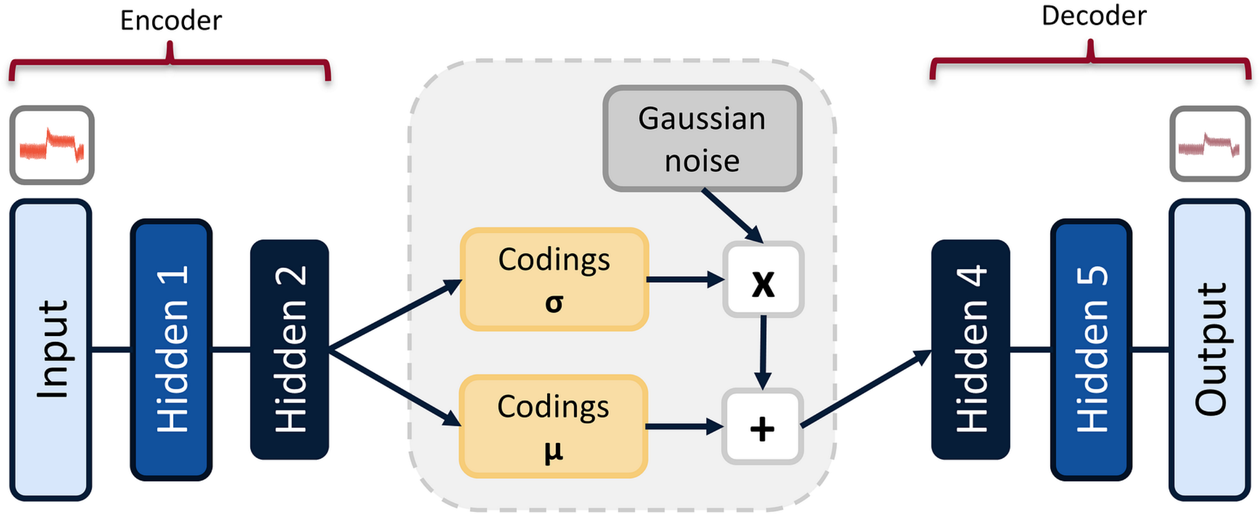
$$loss = reconstruction\ loss + similarity\ loss$$

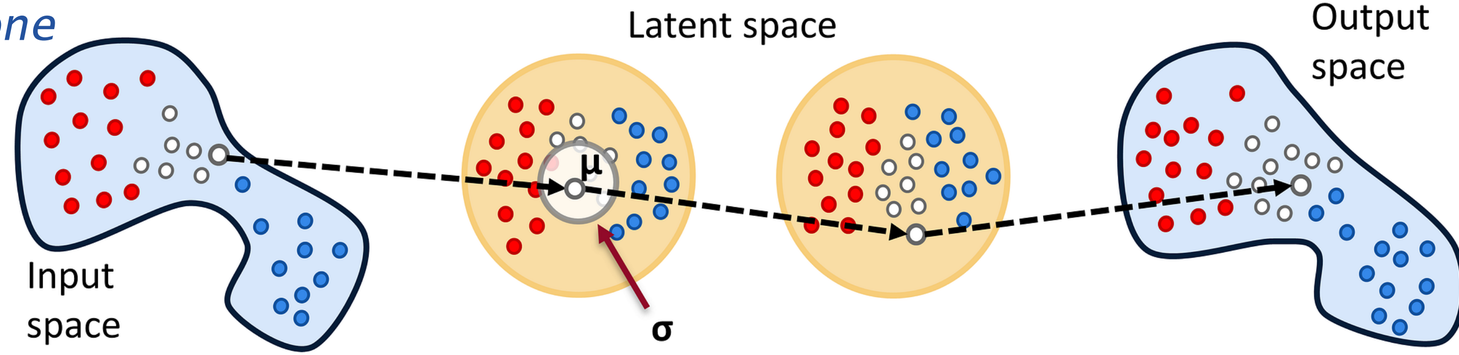
What the VAE loss does

- 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

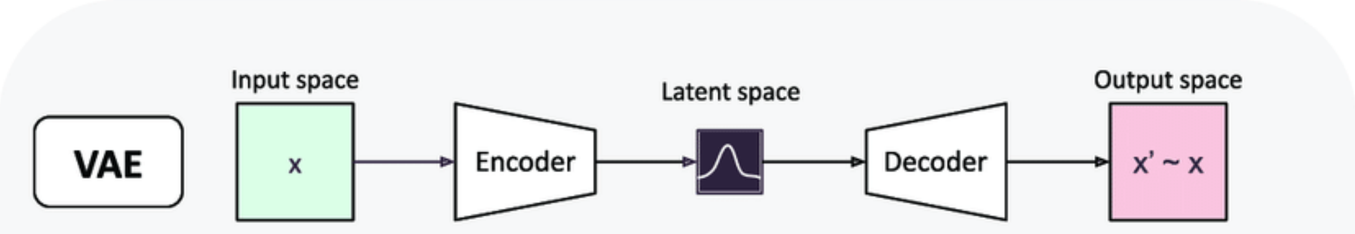
- *The idea of anomaly detection for (V)AEs is that*
  - *The network learns to decode standard data with high fidelity*
  - *The decoding fails when data not belonging to the original dataset (outliers/anomalies) are processed*
  - *A large loss indicates an anomaly, so it can be used as an anomaly 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*

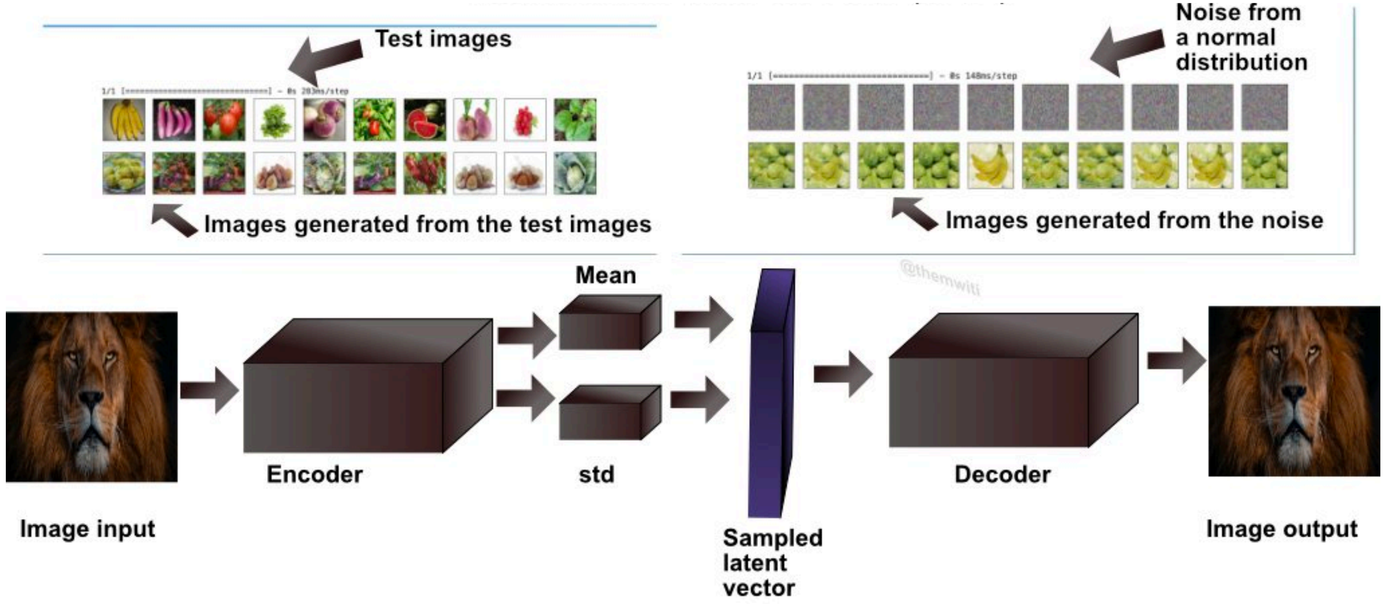




VAEs as Generative Models

- *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*

Conclusions

- *We saw a first example of unsupervised training, with autoencoders*
  - *good for clustering, compression, and anomaly detection*
- *We discussed the role of the bottleneck and alternative methods to enforce the learning process from data (as opposed to trivial solutions)*
- *We discussed the generalization to Variational Autoencoders and the ELBO*
- *We saw how VAEs generalize Sparse AEs and can be used as generators*