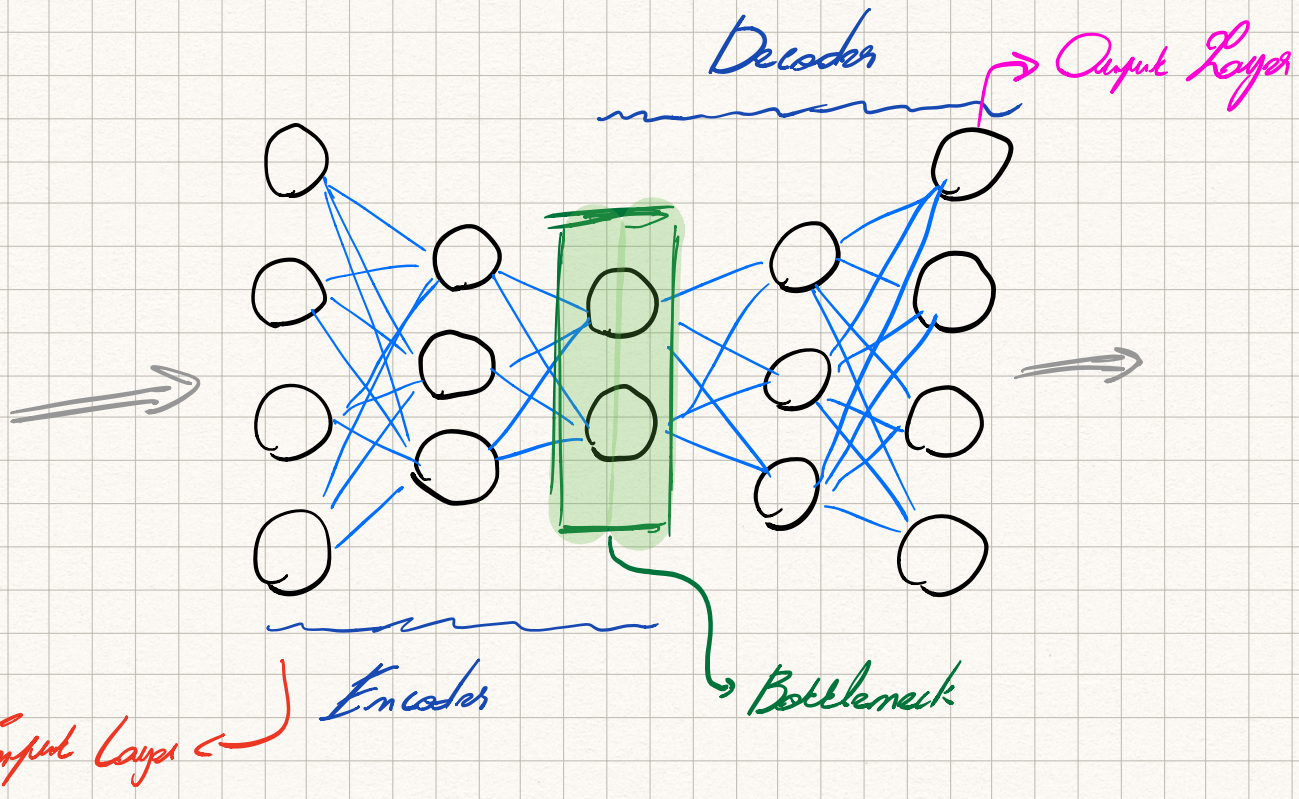
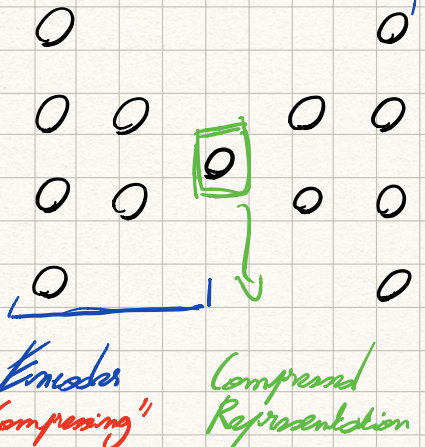
Based on: “Introduction to autoencoders’

Autoencoder:





Decoder = "Decompressing"

Autoencoders are neural networks that create

try to learn = lower data dimension " representations.

By training a double element network:

- Encoder
- Decoder (could be used to generate future = never seen on training " images?)

⚠ It can not be used to generate new data, satisfactory

↳ We do not know the = compressed data representation", i.e. which distribution follows our = encoder' to generate numbers.

↳ VAEs try to adress this limitation.

The goal in autoencoders is to minimize the reconstruction error;

$\mathcal{L}(x, \bar{x})$ = loss of $\bar{x}$ with respect to x.

⚠ If we create an AE with no activation functions (no non-linearity introduced) we would observe some kind of PCA (Principal Component Analysis)

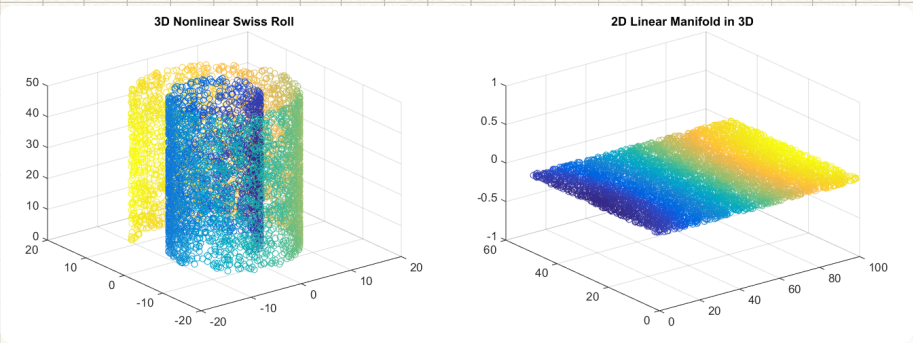
PCA has, though, a difference, there is the implicit restriction of orthogonality, not present in AE. ⇒ PCA more efficient; except huge dataset.

Usually, in order to avoid network overfitting, we use regularization:

$$\mathcal{L}(x, \bar{x}) + \text{regularizer}$$

The "Bottleneck" reduced dimension "ensures" it.

Manifold → "Continuous, non-intersecting surface.



Visual representation of a non-linear dimensionality reduction resulting in a 2D linear manifold.

Sparse Autoencoders:

Even if we restrict the "Bottleneck" layer enough, the neural network may still memorize the whole train set.

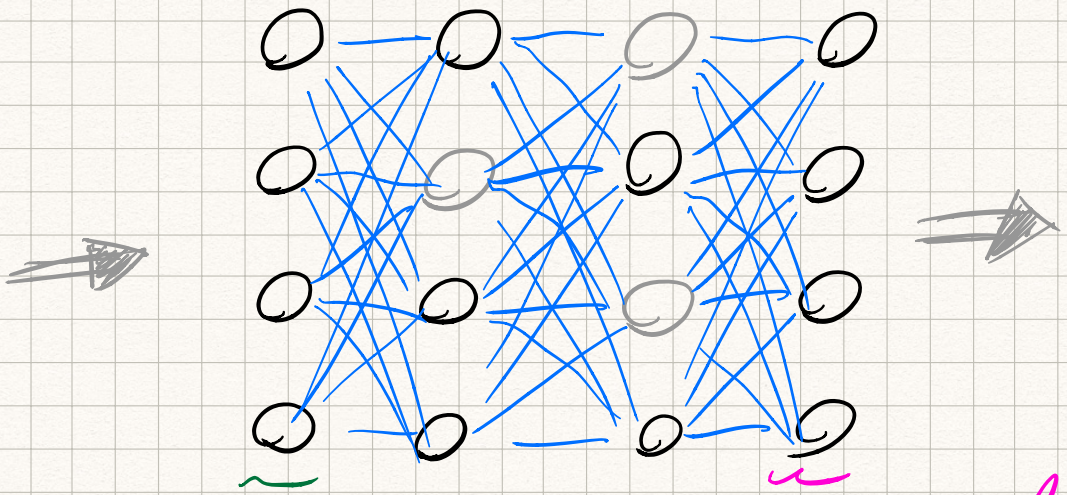
That's why "SAE" (Sparse Autoencoders) exist, we try to train a model that is "data-dependent" ⇒ Different neural paths will be activated for different data.

AE

Uses full network activation

SAE

Allow to create specific paths depending on the input.



↳ Input layer

↳ Output layer

○ ≃ Active Neuron

○ ≃ Inactive Neuron

The ways to obtain are:

- L1-Regularization: also called "Lasso regression"

$$\lambda \sum_{j=1}^{p} |\beta_j|$$

⇒ Shrinks some feature coefficients (neuron weights) to 0 ⇒ Sparse coercion.

- L2-Regularization: Ridge Regression

$$\lambda \sum_{j=1}^{p} \beta_j^2$$

⇒ Penalizes "big values" of features

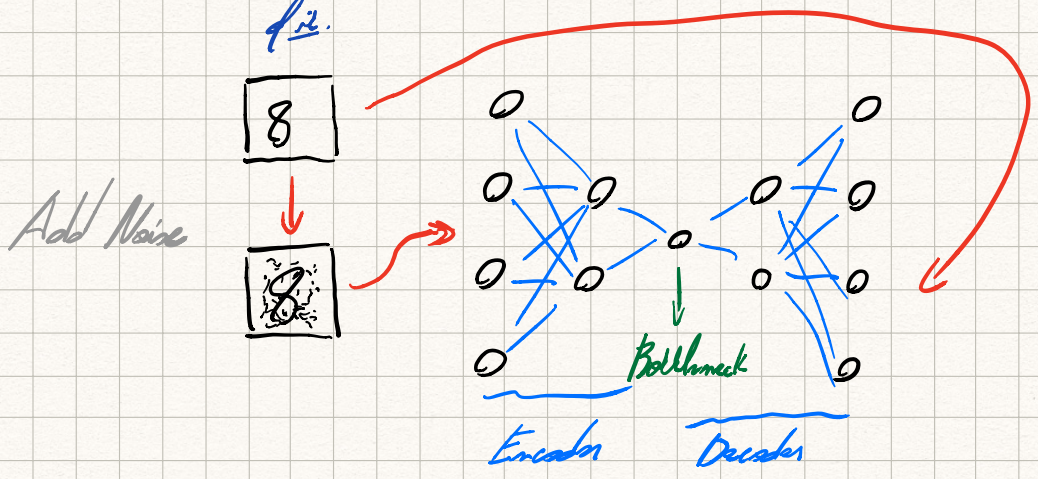
↳ Weights are smaller, i.e. weaker connection

- KL-Divergence: difference between 2 probability distributions.

$$\mathcal{L}(x, \tilde{x}) + \sum_{j} KL(p \| \hat{p}_j)$$

$$KL \rightarrow \sum_{j=1}^{|h|} \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}$$ (KL-Div 2 Bernoulli)

Denoising autoencoder: AE where the data has been corrupted at the start of the model, in order to "fool the model" to learn the true representation of it.



Contractive AE: AE where we enforce that for similar input, the learned encoding should also be similar.

↳ AE learns how to "contract" similar inputs in the same way in the embedding layer.

Constructing a loss therm which penalizes "large derivatives" of our hidden layers with respect to input examples.

↳ Squared Frobenius norm $\|A\|_F$ (L2-Norm) of the Jacobian matrix J of hidden layer activation with respect to input observation. ↳ First order partial derivatives

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}$$