Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

MSE | MASTER OF SCIENCE IN ENGINEERING

Machine Learning

T-MachLe

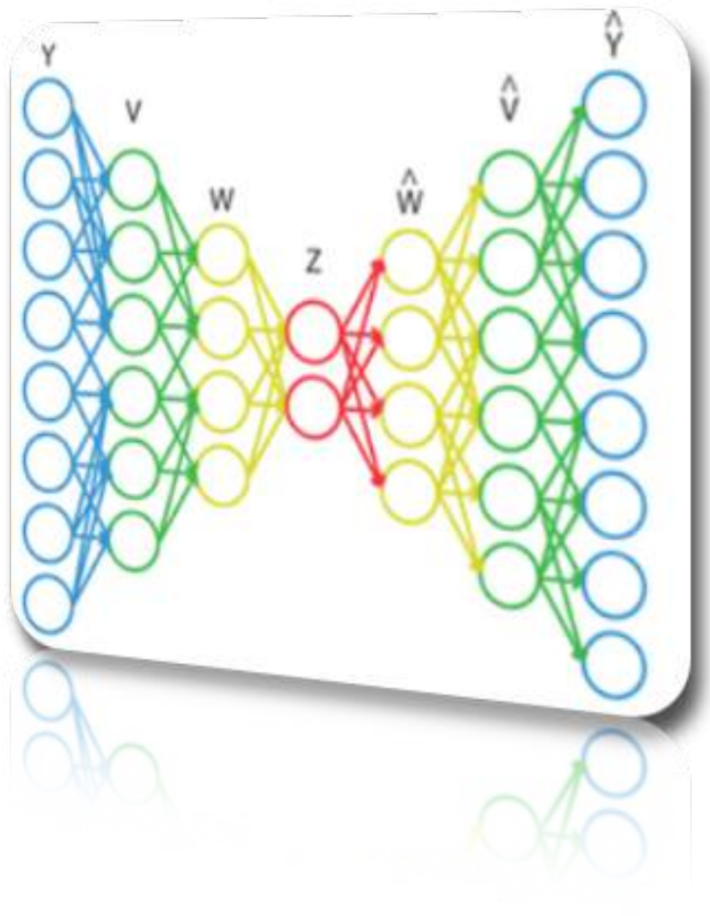
12. Autoencoders

Jean Hennebert
Andres Perez Uribe

Plan

1. Principles of autoencoders
2. Uses of autoencoders
3. Generative models
4. Towards Self-supervised systems

Practical Work 12



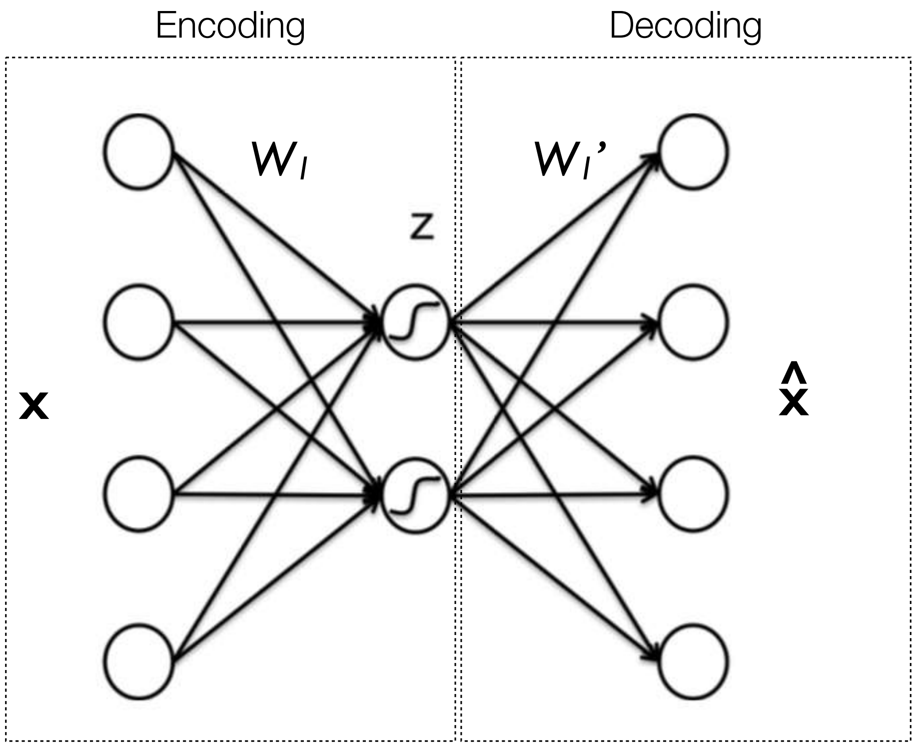
Source: Max Bernier, Konrad P. Kording, "Deep networks for motor control functions", frontiers in Computational Neuroscience, March 2015, Vol. 9

Definition

An **autoencoder** is a neural network used for unsupervised learning and able to discover “features” and “efficient codings” of the input space.

- The simplest form of an autoencoder is a feedforward, non-recurrent neural network similar to the MLP.
- In such network, the output layer has the same number of nodes as the input layer.
- The mapping function $h_\theta(\mathbf{x})$ is trained to reconstruct its own inputs instead of predicting a target value.

$$\widehat{\mathbf{x}} = h_\theta(\mathbf{x})$$

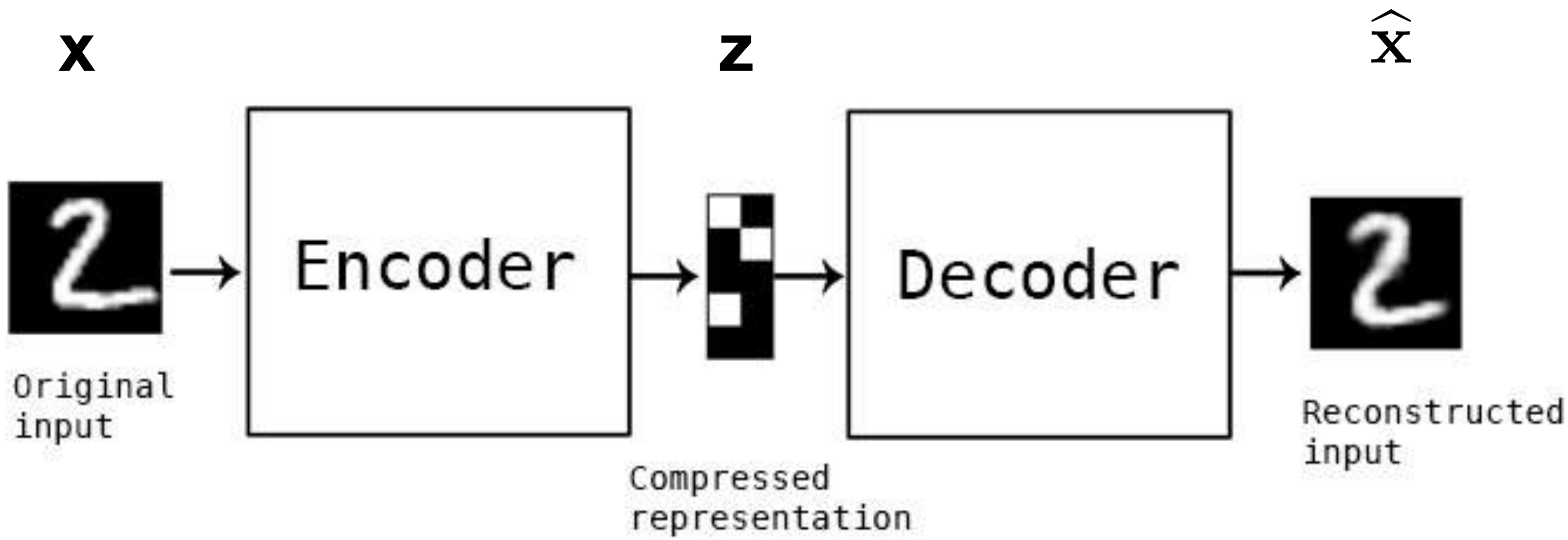


$\mathbf{z} = E_{\theta_E}(\mathbf{x})$ $\widehat{\mathbf{x}} = D_{\theta_D}(\mathbf{z})$

$$\widehat{\mathbf{x}} = h_\theta(\mathbf{x})$$

- In order to learn something else than the unity function, the network is often composed in such a way that the number of nodes in the hidden layer is smaller than the number of nodes in the input and output layers, forming a so-called “**diabolo**” network.
- The principle is to “cut” the network into two pieces after training

The basic concept



- To train such a network we do not need labels, that is why they can be used for unsupervised learning
- However, they are trained as if we had a supervised learning problem by using any variant of Backpropagation and minimizing the difference between $\hat{\mathbf{x}}$ and $\mathbf{x}$.

from Ricky Han blog

Simplest auto-encoder (in Keras)

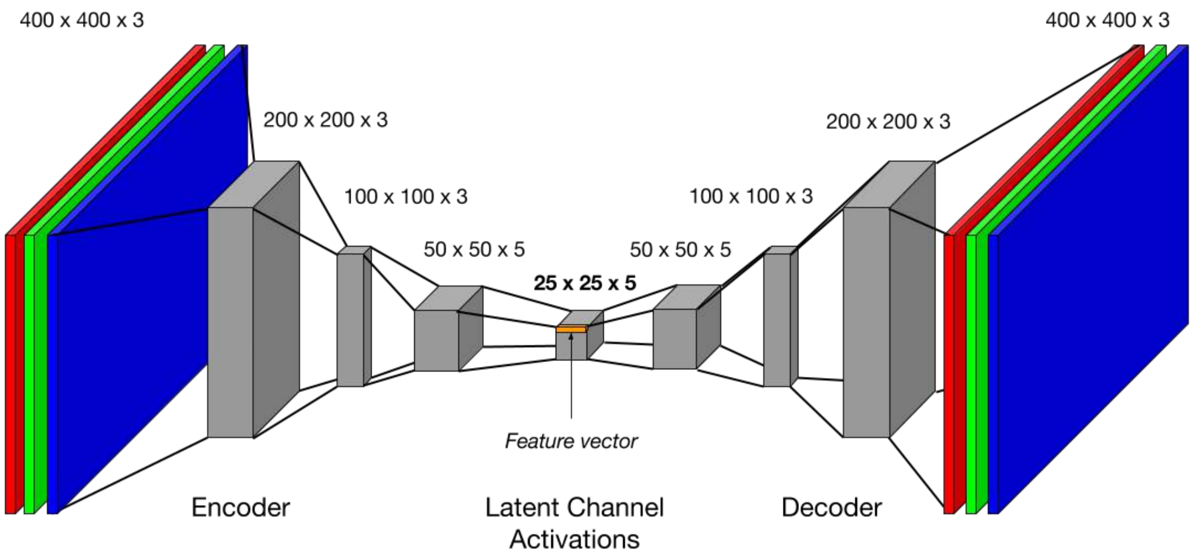
```python
import keras
from keras import layers

# This is the size of our encoded representations
encoding_dim = 32  # 32 floats -> compression of factor 24.5, assuming the input is 784 floats

# This is our input image
input_img = keras.Input(shape=(784,))
# "encoded" is the encoded representation of the input
encoded = layers.Dense(encoding_dim, activation='relu')(input_img)
# "decoded" is the lossy reconstruction of the input
decoded = layers.Dense(784, activation='sigmoid')(encoded)

# This model maps an input to its reconstruction
autoencoder = keras.Model(input_img, decoded)
```

- This auto-encoder uses an MLP architecture with a single hidden layer composed of 32 units.
- It reconstructs the 28x28 pixels of an MNIST image as a one-dimensional vector of size 784

from https://blog.keras.io/building-autoencoders-in-keras.html

Deep convolutional autoencoders

- The encoder and the decoder can be convolutional and deconvolutional (or upsampling) neural networks respectively
- The inner-layer is called the “latent space representation” of the data set



A convolutional auto-encoder (in Keras)

```
import keras
from keras import layers

input_img = keras.Input(shape=(28, 28, 1))

x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(input_img)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
encoded = layers.MaxPooling2D((2, 2), padding='same')(x)

# at this point the representation is (4, 4, 8) i.e. 128-dimensional

x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(encoded)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(16, (3, 3), activation='relu')(x)
x = layers.UpSampling2D((2, 2))(x)
decoded = layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = keras.Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```


from https://blog.keras.io/building-autoencoders-in-keras.html

Objective function

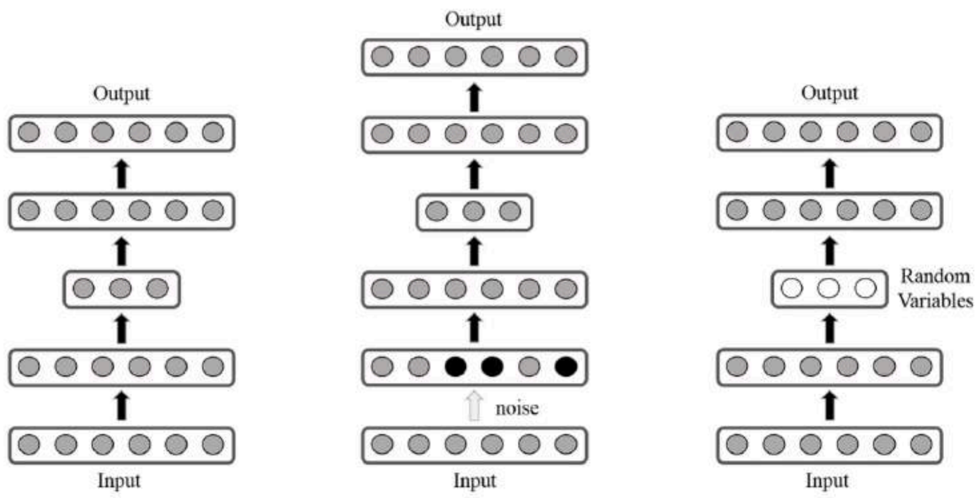
- The objective function evaluates how close the reconstruction is to the given input, e.g., by using the MSE loss function:

$$
\begin{aligned}
J(\theta) = J(\theta_E, \theta_D) &= \frac{1}{2N}\sum_{n=1}^{N}(h_\theta(\mathbf{x}_n) - \mathbf{x}_n)^2 \\
&= \frac{1}{2N}\sum_{n=1}^{N}(\widehat{\mathbf{x}}_n - \mathbf{x}_n)^2
\end{aligned}
$$

- Other loss functions like binary cross-entropy are also frequently used.

The surprising usefulness of autoencoders

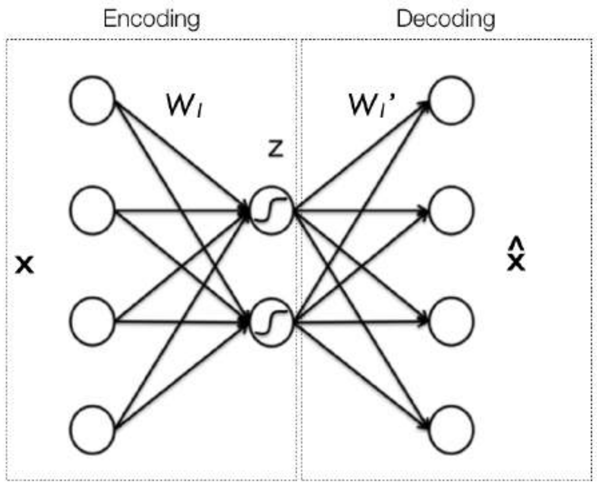
1. Compression
2. Dimensionality reduction / manifold learning
3. Denoising
4. Initializing deep neural networks
5. Anomaly detection
6. Generator



basic denoising variational

1. Data compression

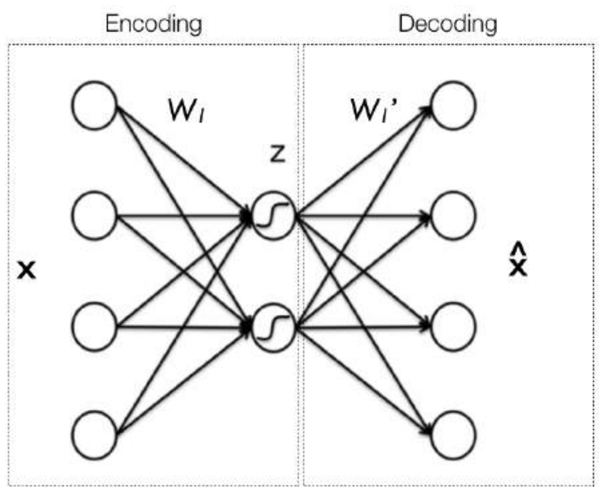
- Principles:
  - Train an autoencoder to output $\hat{X}$ as close as possible to X.
  - Transmit the W_1' to your party
  - Encode: transmit the z values to your party instead of x
  - Decode: compute the reconstructed values $\hat{X}$
- If linear activations are used, or only a single sigmoid hidden layer, then the optimal solution to an autoencoder is strongly related to principal component analysis (PCA*)



* Dimensionality reduction chapter

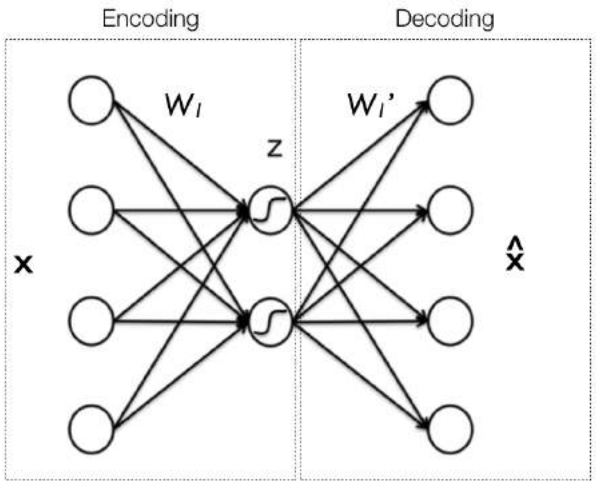
2. Dimensionality reduction

- Principles:
  - Train an autoencoder to output $\hat{X}$ as close as possible to x.
  - When a low reconstruction error is achieved the z values represent the x input values in a lower dimensional space.
  - In ML jargon we say that we learned a nonlinear manifold of the x input data.
  - The lower representation of the input data (z values) can be used to train a classifier, to establish the similarity of input data, and to discover structure in the input data (e.g., using clustering algorithms).

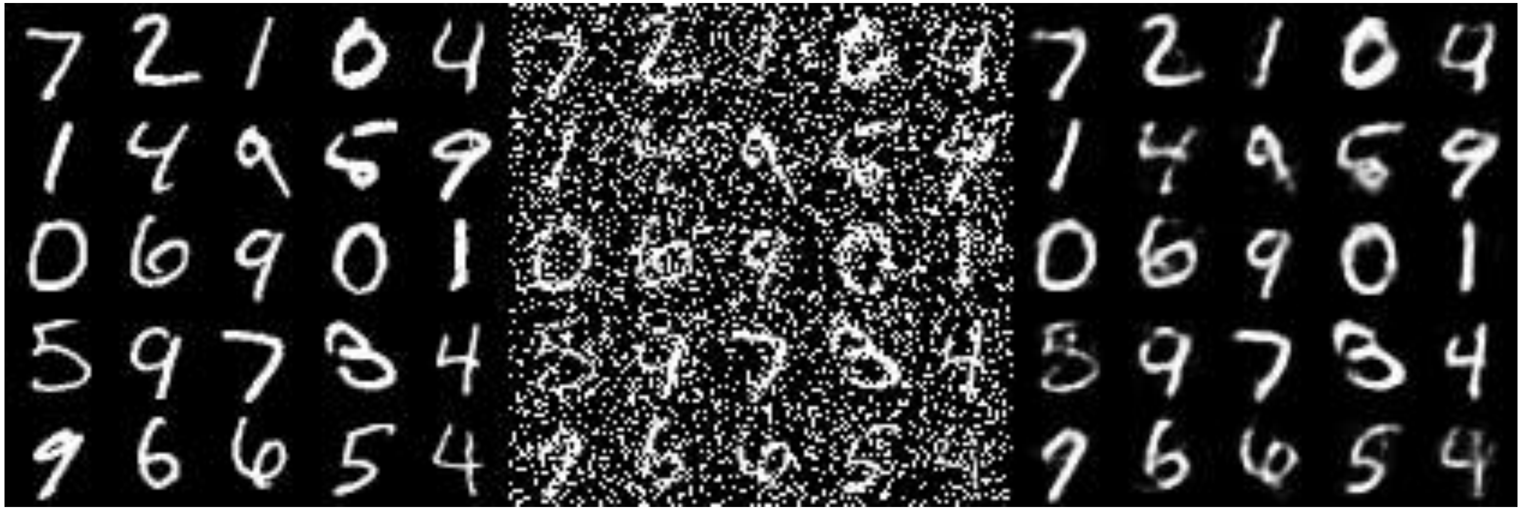


3. Denoising autoencoders

- Principles:
  - Train an autoencoder on your “clean” training input data
  - To denoise a noisy input (e.g., an image), use the encoding part to compute z and then use the decoder part to compute $\hat{X}$
  - Since $\hat{X}$ is a reconstruction based on the learned latent variables z, it should correspond to a denoised version of X.
  - <u>Refinement:</u> add noise to your training data x and attempt to reconstruct the clean data version

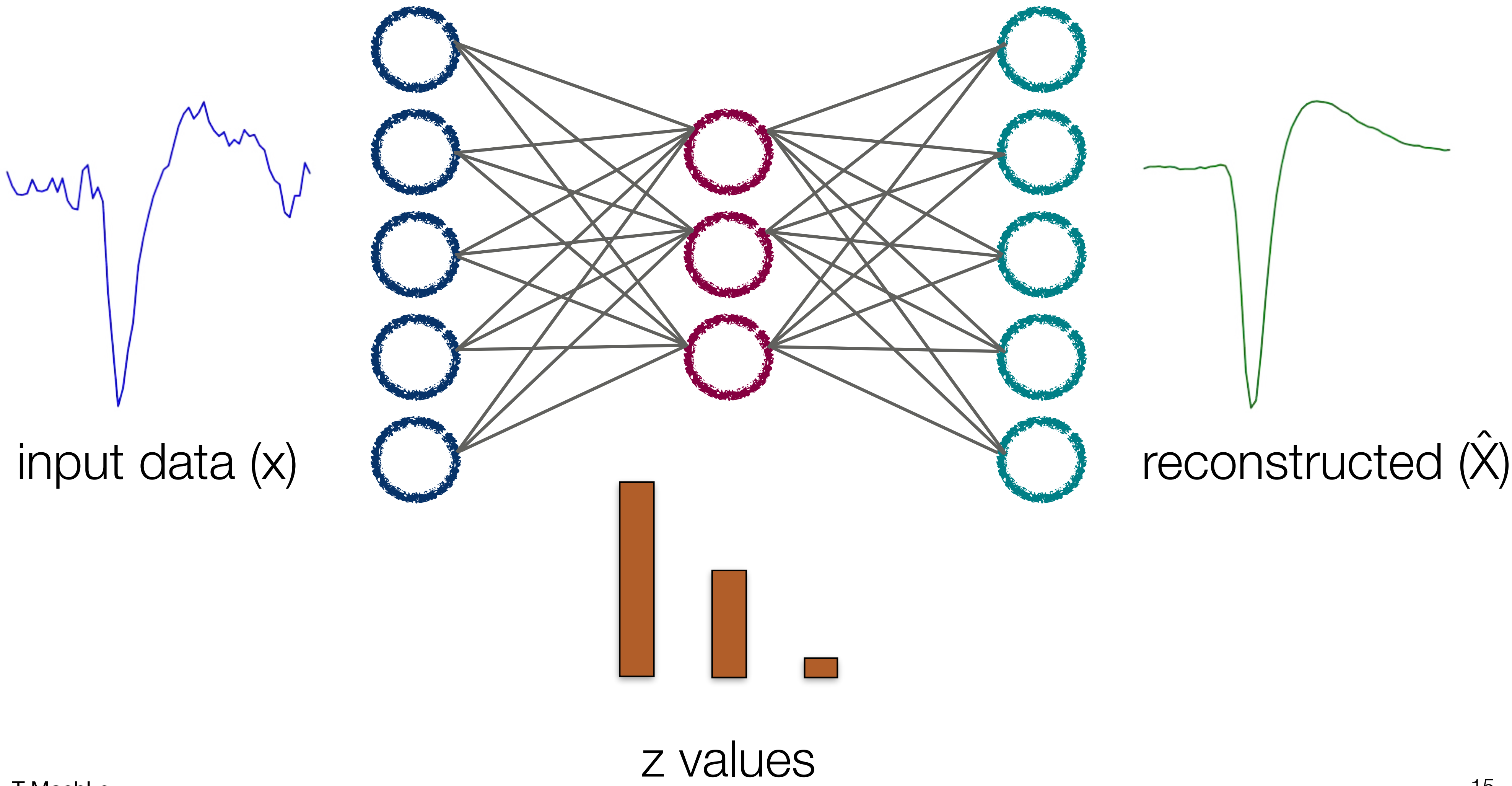


Example: image denoising



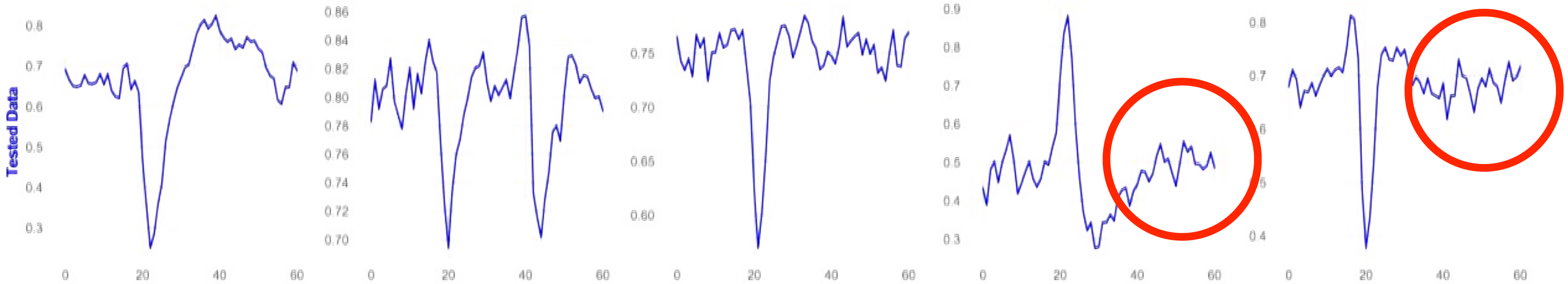
input data noisy data denoised data

Example: time-series denoising

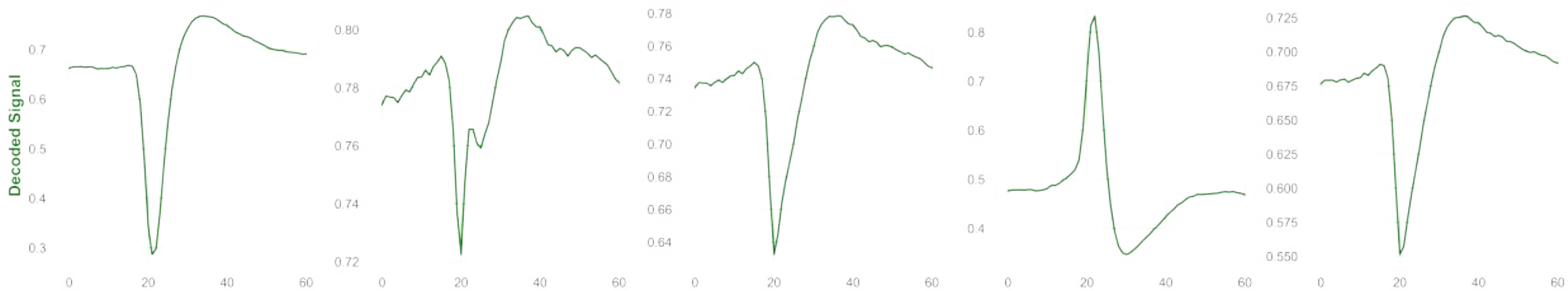


Denoising: further examples

- input signals:

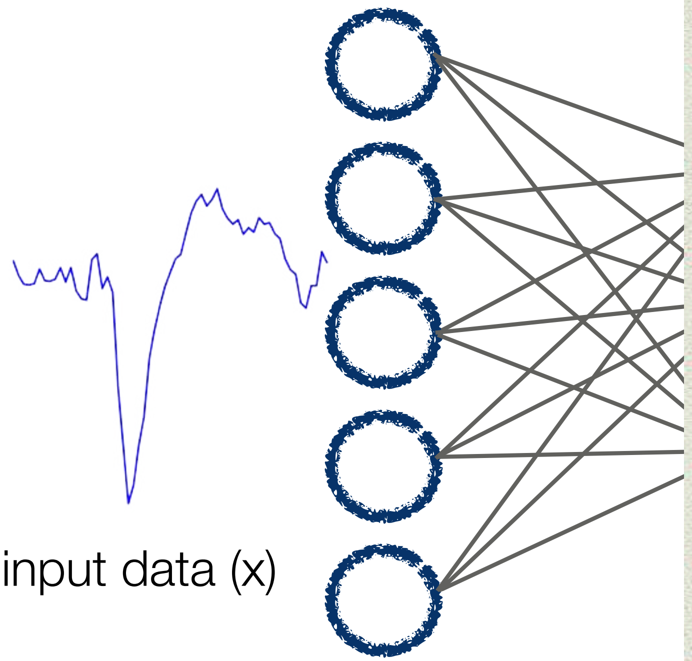


- denoised outputs:



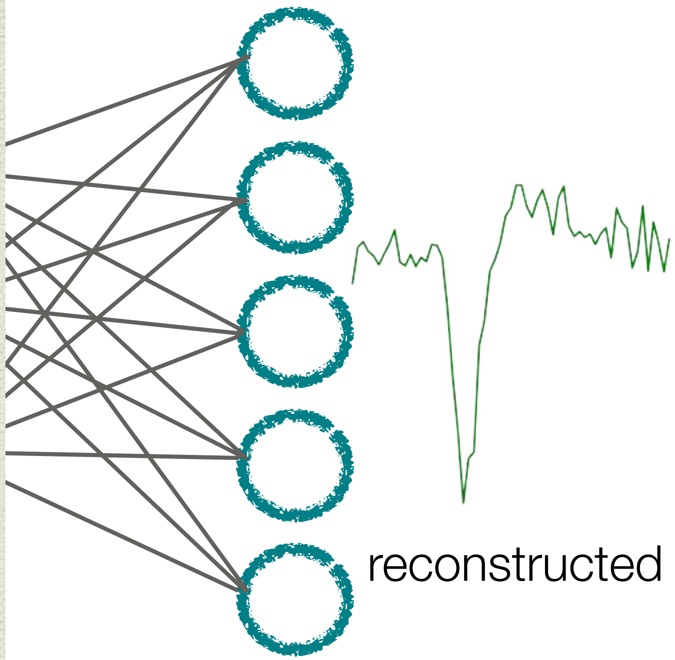
If the model is too simple, we might loose important information

Deep auto-encoder



input data (x)

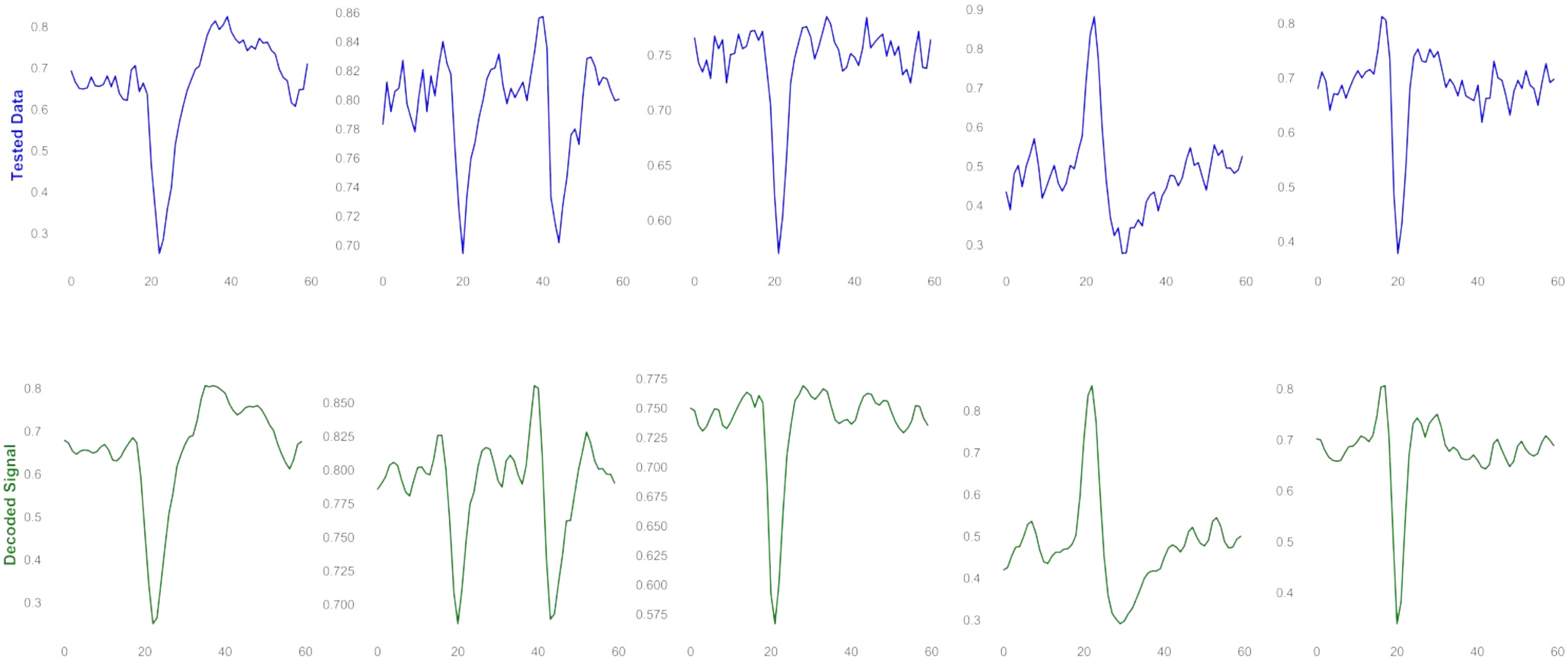
Layer (type)	Output Shape	Param #
input_117 (InputLayer)	(None, 60, 1)	0
conv1d_513 (Conv1D)	(None, 60, **60**)	240
max_pooling1d_197 (MaxPoolin	(None, 30, 60)	0
conv1d_514 (Conv1D)	(None, 30, **30**)	5430
max_pooling1d_198 (MaxPoolin	(None, 15, 30)	0
conv1d_515 (Conv1D)	(None, 15, 30)	2730
max_pooling1d_199 (MaxPoolin	(None, 8, 30)	0
conv1d_516 (Conv1D)	(None, 8, 30)	2730
up_sampling1d_184 (UpSamplin	(None, 16, 30)	0
conv1d_517 (Conv1D)	(None, 16, 30)	2730
up_sampling1d_185 (UpSamplin	(None, 32, 30)	0
conv1d_518 (Conv1D)	(None, 31, 60)	3660
up_sampling1d_186 (UpSamplin	(None, 62, 60)	0
conv1d_519 (Conv1D)	(None, 60, 1)	181



reconstructed ($\hat{X}$)

CNN auto-encoder

Deep auto-encoder denoising

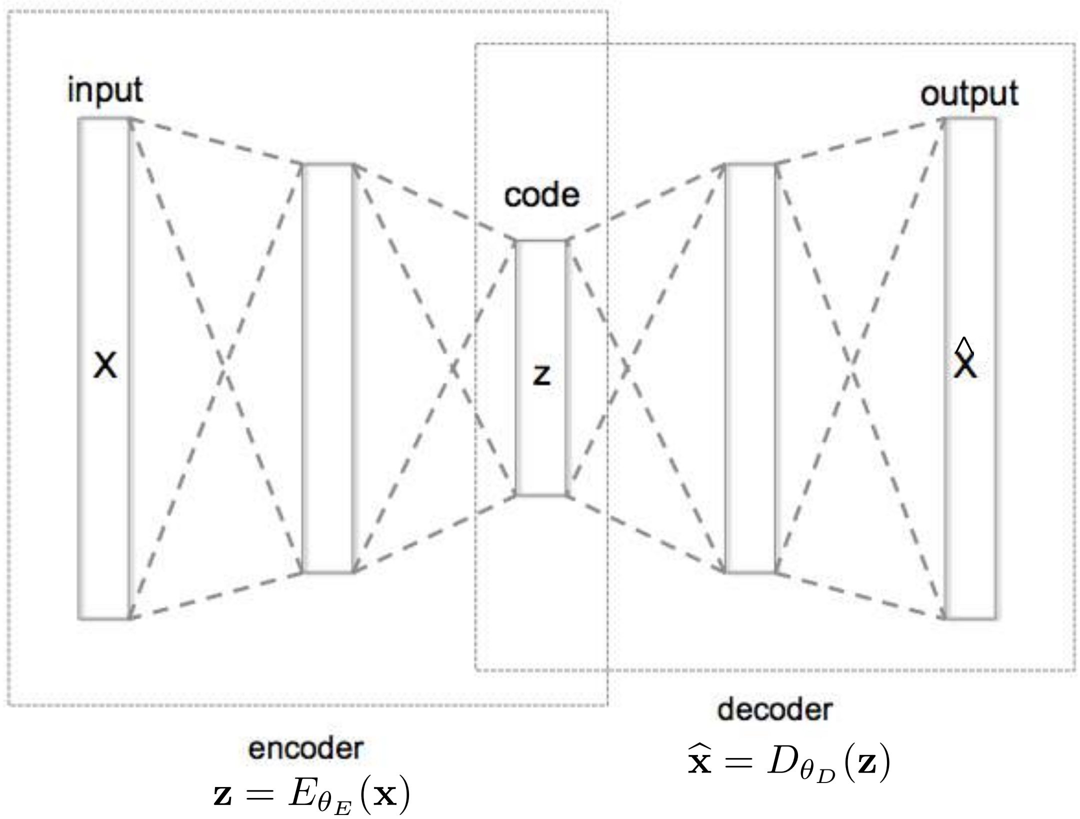


4. Initialization of deep networks for supervised learning

- Principles:
  - Pre-training step: train a sequence of autoencoders, greedily one layer at a time, using unsupervised data
  - Fine-tuning step 1: train the last layer using supervised data,
  - Fine-tuning step 2: use back-propagation to fine-tune the entire network using supervised data
- Researchers have shown that this pre-training idea helps deep neural networks converge more rapidly.
- From 2006 to 2011, this approach gained much attraction because it makes use of inexpensive unlabeled data.
- Since 2012, this research direction however has gone through a relatively quiet period, because unsupervised learning is less relevant when a lot of labeled data are available.

From Quoc V. Le, Google Brain, “A Tutorial on Deep Learning. Part 2.”

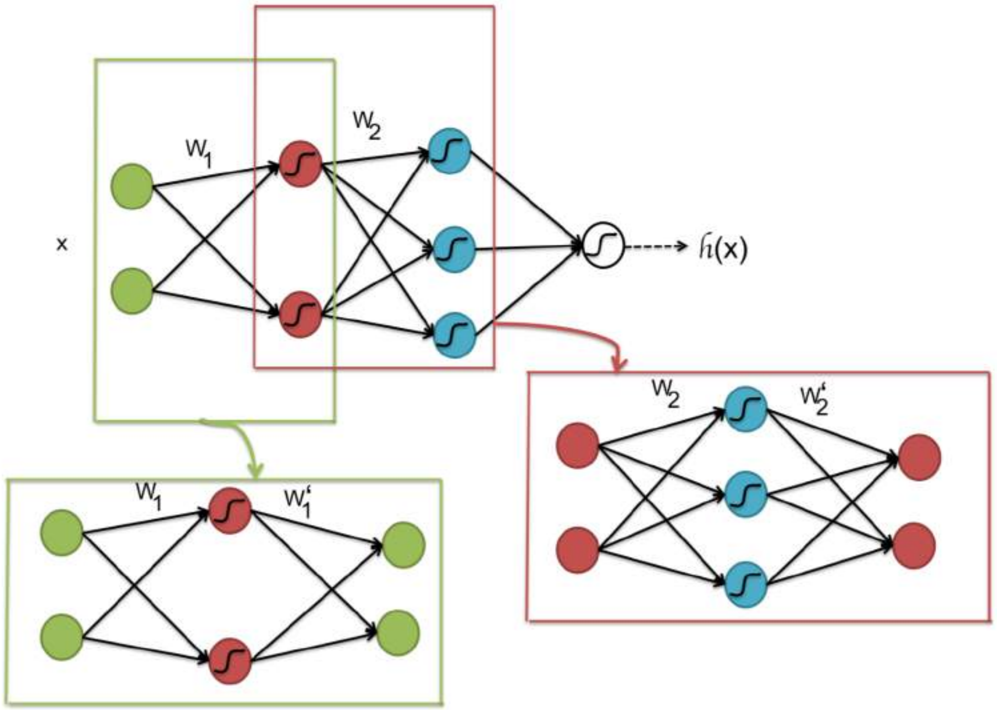
How to train deep networks?



- “stacked” autoencoders
- To handle the vanishing gradient problem, we train layer by layer

Training layer by layer

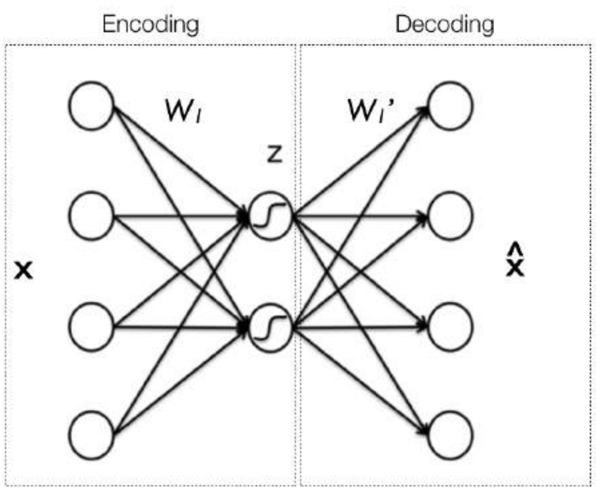
- To train the red neurons, we will train an autoencoder that has parameters W1 and W1′.
- After this, we will use W1 to compute the values for the red neurons for all of our data
- The parameters of the decoding process W1′ are then discarded.
- The subsequent autoencoder uses the values for the red neurons as inputs, and trains an autoencoder to predict those values by adding a decoding layer with parameters W2′.
- The parameters of the decoding process W2’ are then discarded.
- An so forth…



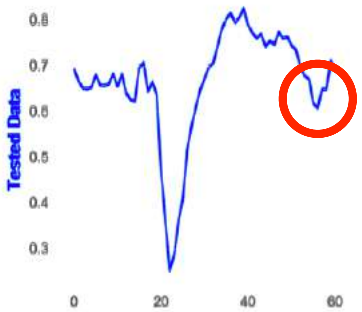
From Quoc V. Le, Google Brain, “A Tutorial on Deep Learning. Part 2.”

5. Anomaly detection

- Principles:
  - Train an autoencoder on your training input data (with no anomalies)
  - apply the autoencoder to a test dataset
  - inputs with high reconstruction error (e.g., based on a threshold) are likely to be outliers

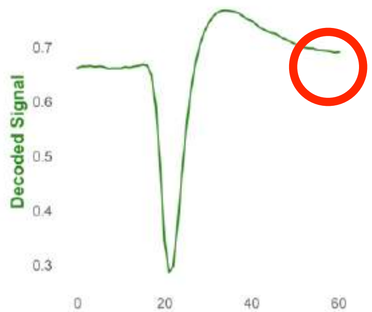


- What is a high reconstruction error ?
  - It depends on the task
  - You should determine an error threshold



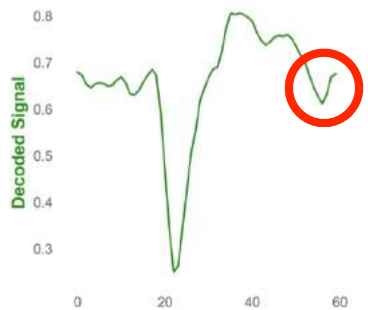
original signal

a)



If this signal captures all the important features of the original one, set a given reconstruction threshold

b)

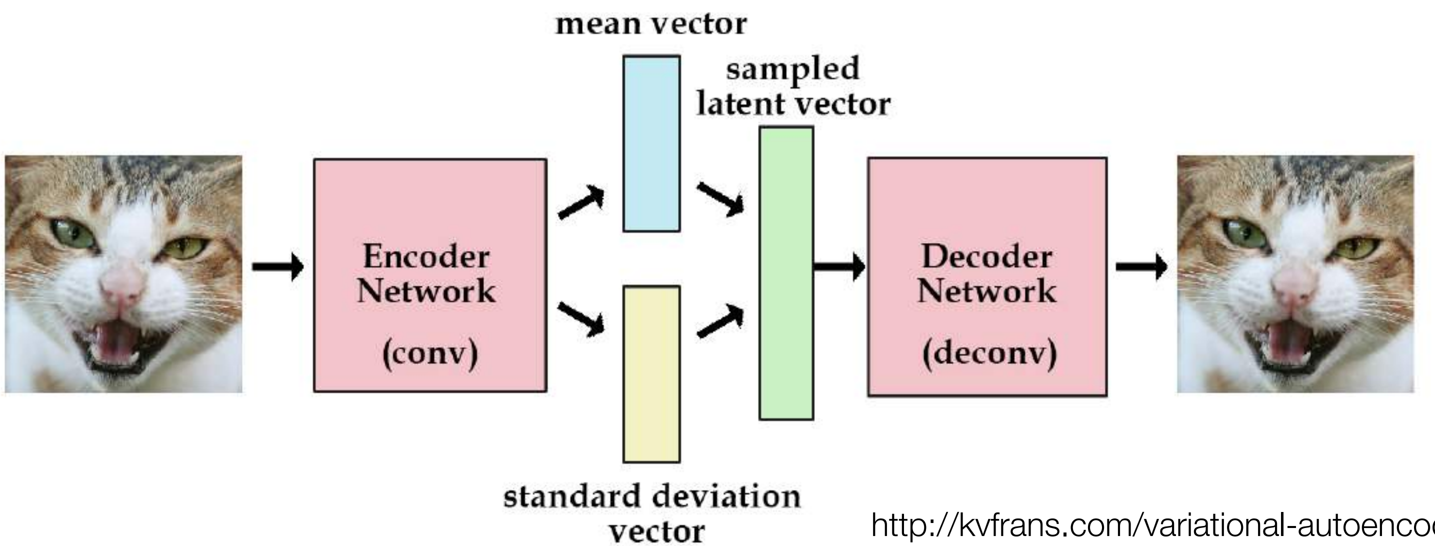


If not, use a more complex model and set a lower reconstruction threshold to spot anomalies

6. Autoencoders as generators

- Use of a variational autoencoder (VAE)
- We add a constraint on the encoding network, that forces it to generate latent vectors that roughly follow a unit gaussian distribution.
- We consider two losses to learn a vector of means and a vector of standard deviations:

```
generation_loss = mean(square(generated_image - real_image))
latent_loss = KL-Divergence(latent_variable, unit_gaussian)
loss = generation_loss + latent_loss
```



http://kvfrans.com/variational-autoencoders-explained/

Variational Auto-Encoder (VAE)

- Minimize the reconstruction error (difference between the input and the reconstructed image) and try to find a Gaussian distribution on the bottleneck layer minimizing the Kullback-Leibler divergence between the latent variable P and a unit normal distribution Q.

- Kullback-Leibler divergence == difference between two distributions

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

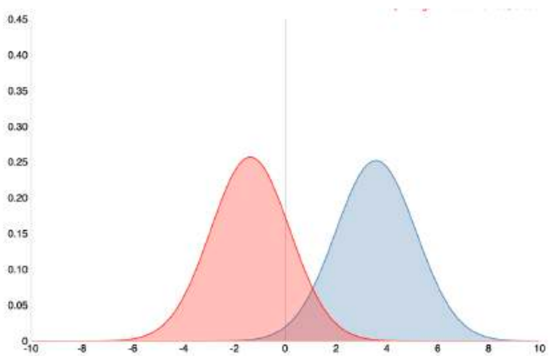
- Example:

```
# calculate the kl divergence (given two vectors p and q)
def kl_divergence(p, q):
    return sum(p[i] * log2(p[i]/q[i]) for i in range(len(p)))
```

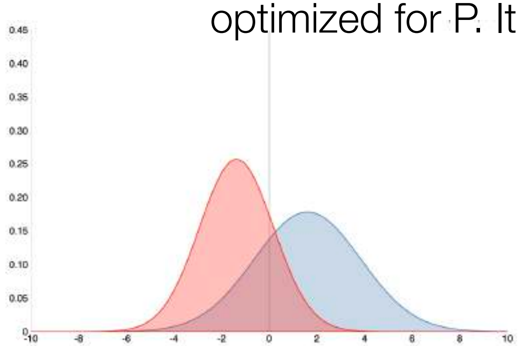
Kullback-Leibler divergence (intuition)

The KL divergence is an information theory concept and is related to the entropy and cross-entropy concepts. Let’s get an intuition of it using these examples:

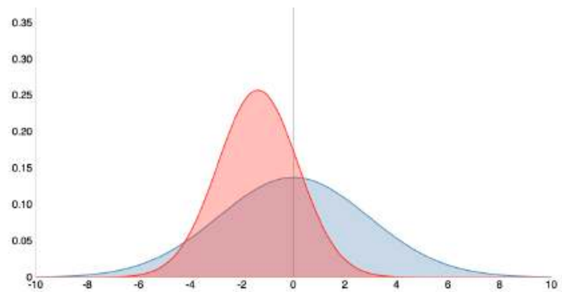
The KL divergence measures the extra number of bits required for encoding samples of P using a code optimized for Q rather than one optimized for P. It also has a Bayesian interpretation.



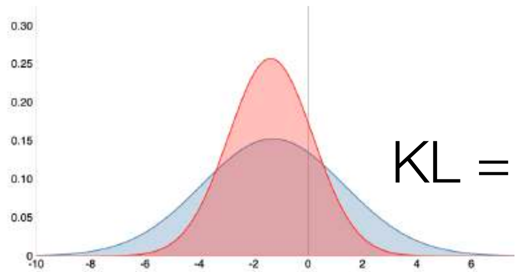
KL = 5



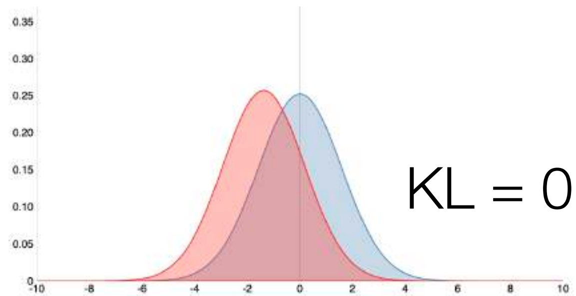
KL = 2



KL = 1



KL = 0.4



KL = 0.4

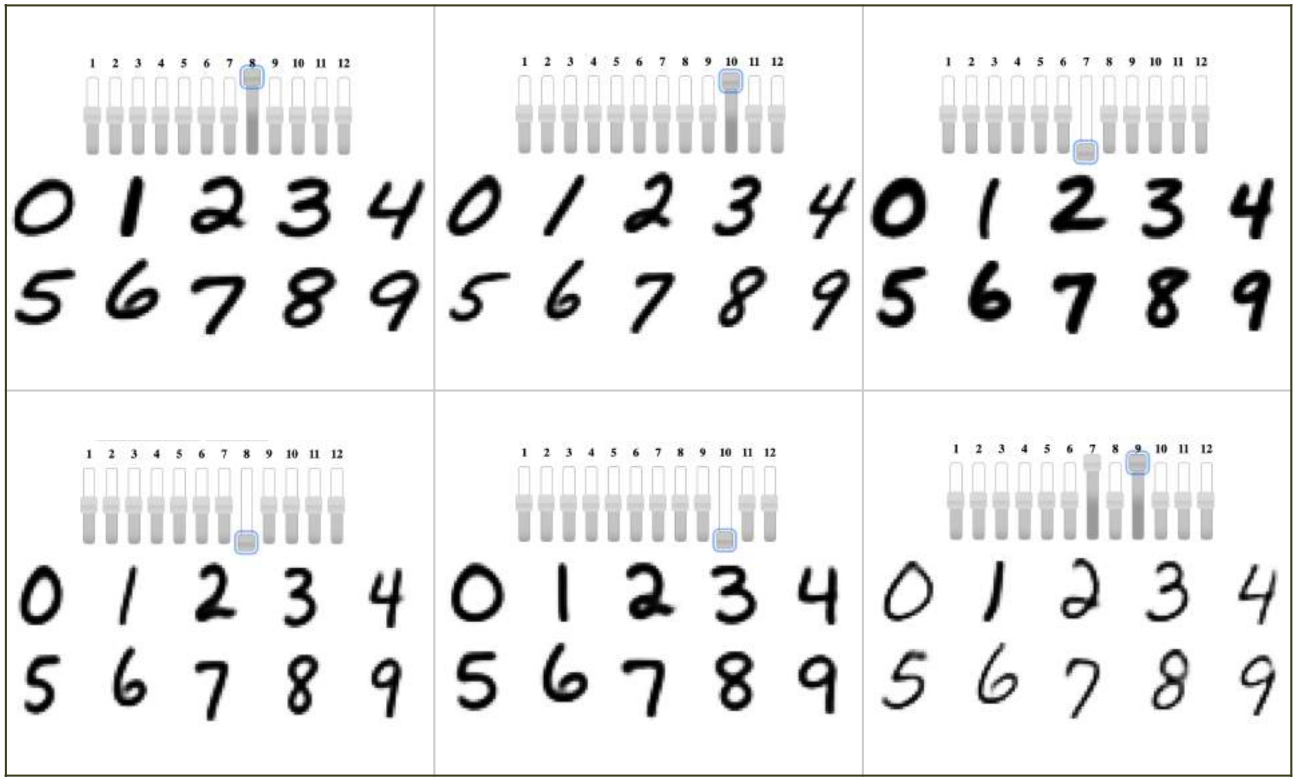
https://www.mdcrosby.com/blog/kldivergence.html

Application: data augmentation

training data

VAE

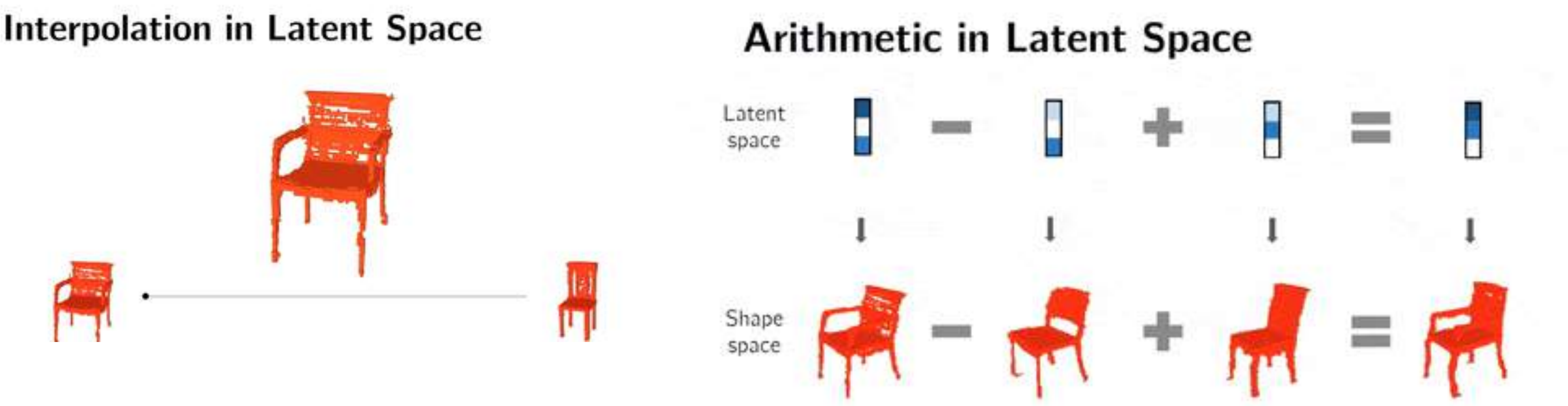
latent space variation



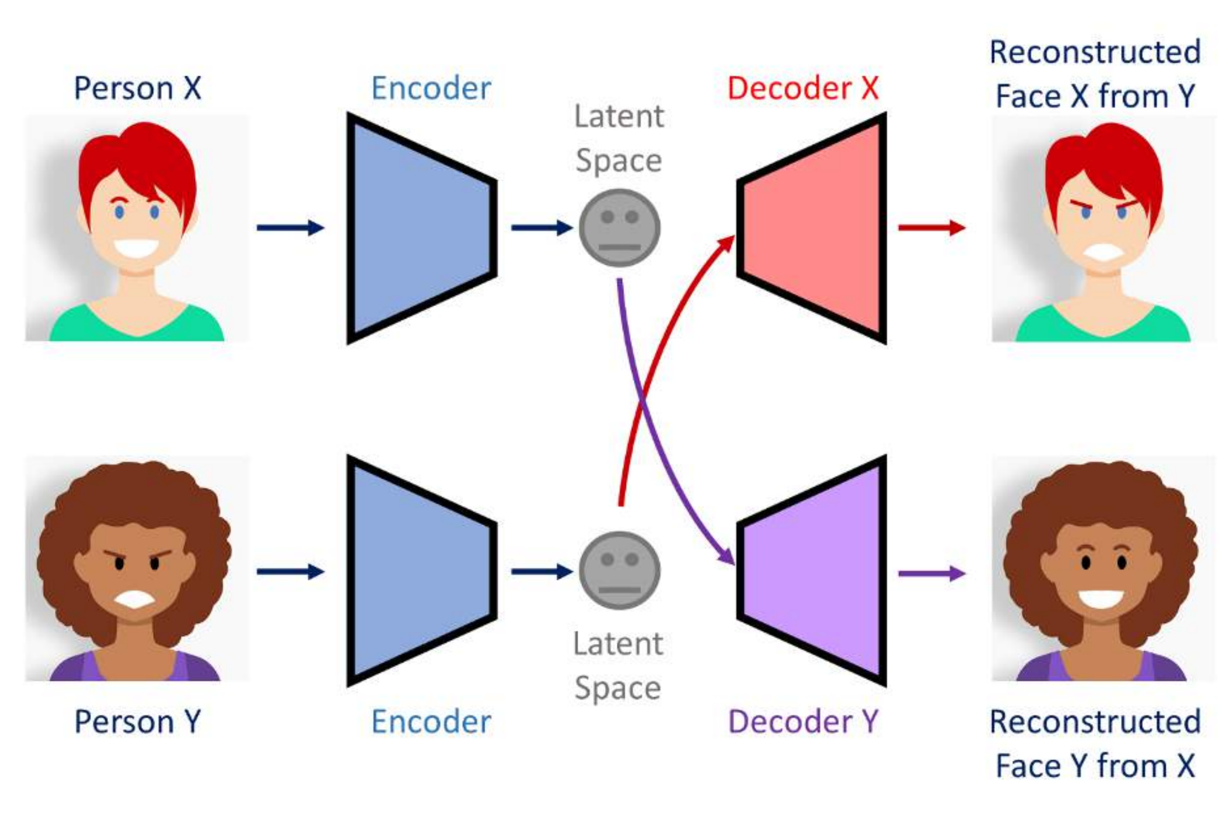
http://dpkingma.com/sgvb_mnist_demo/demo.html

Example: generating new images

- Generating new images is now easy: all we need to do is sample a latent vector from the unit gaussian and pass it into the decoder.
- Examples:



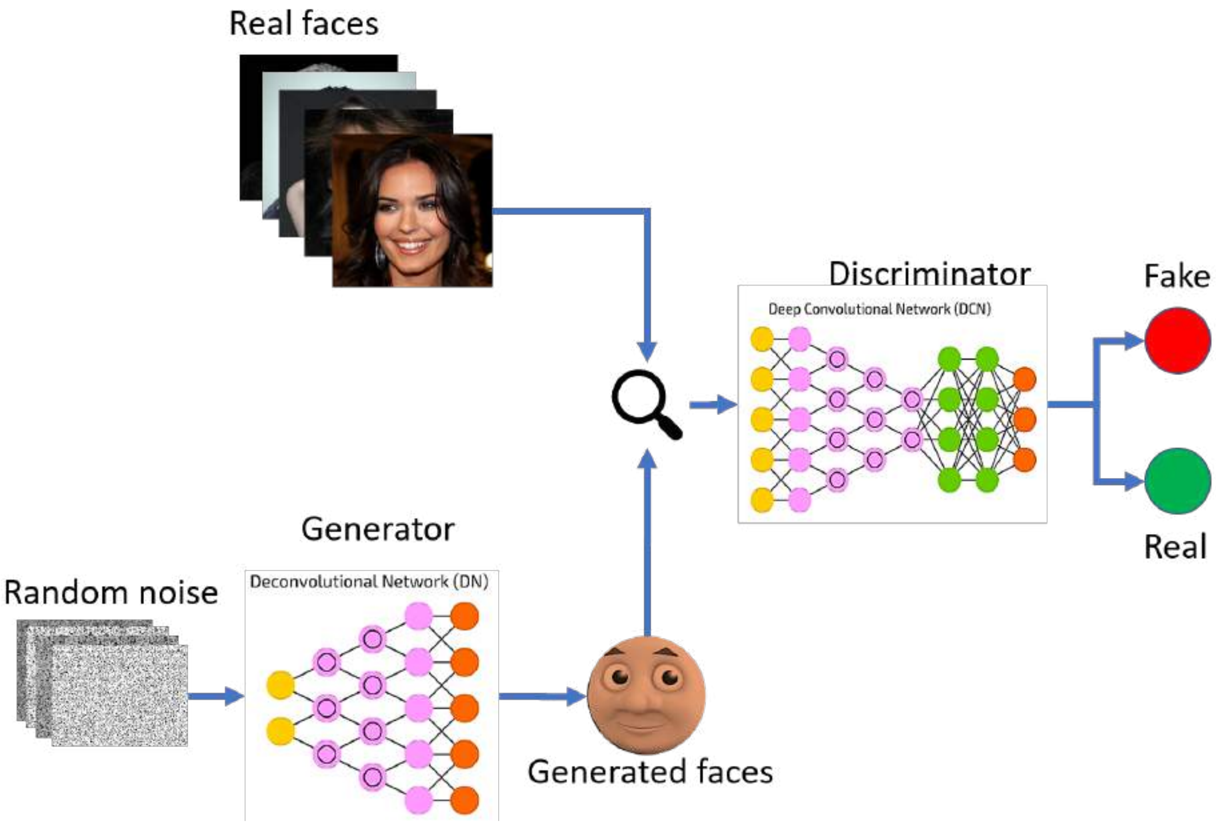
Deep fakes using auto-encoders



by Shaan Subramaniam (science-union.org)

Generative Adversarial Networks (GANs)

- A new architecture introduced by Ian Goodfellow et al., from the University of Montreal in 2014



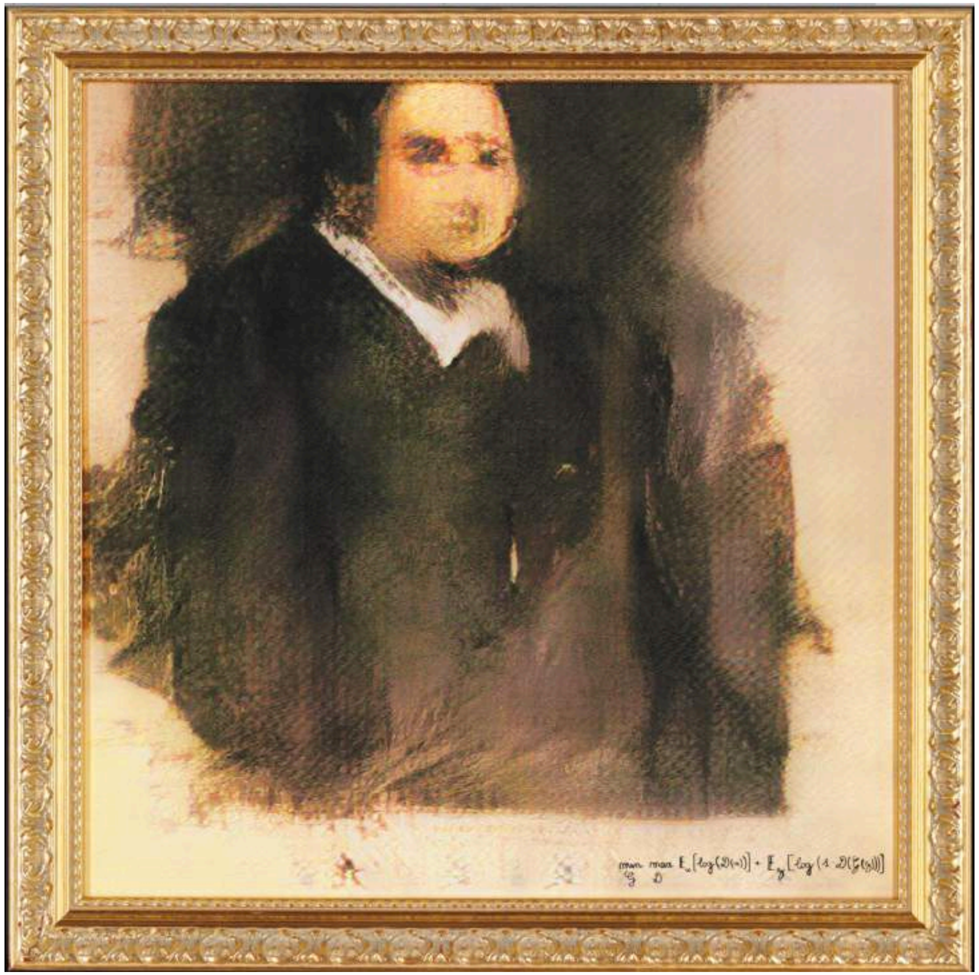
- Wasserstein GANs where instead of real/fake we want to match a certain distribution

thispersondoesnotexist.com





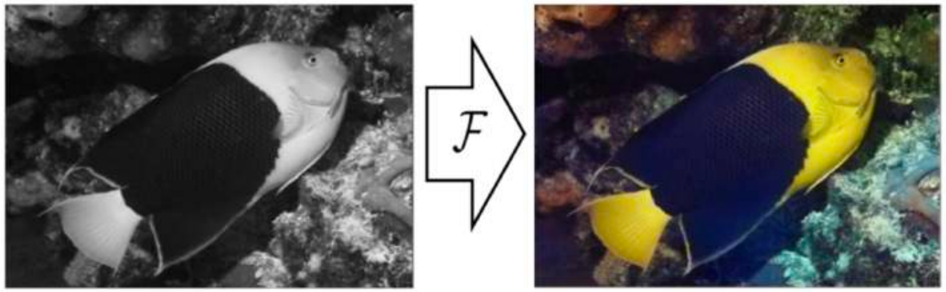

Edmond Belamy portrait



- The Obvious group, composed of researchers and artists in Paris created a fictif family and generated portraits of its members.
- The portrait of Edmond Belamy was sold by Christies in New York for $432’500, on 25.10.18

Self-supervised learning

- How to profit from huge available datasets (e.g., images on the web, videos from Youtube) even if we cannot get labels for all available data ?
- Use a pretext task to learn data representations in the latent space in an unsupervised way (similar to the principle of auto-encoders). Then, use those representations to process new inputs or in different tasks to be learned in a supervised way.
    - **Example**: use a DB of color images, generate the grayscale version of each image and train a model to go from greyscale to color. Then use the trained model to colorize new images.



Conclusions

- Autoencoders =
  - Neural networks trained to reproduce its input
  - Diabolo topology
  - “Stacked” deep learning autoencoder : training layer by layer
  - 5 applications:
    - data compression / dimensionality reduction / feature extraction
    - denoising
    - initializing deep networks (pre-training)
    - anomaly detection
    - generative models
- Unsupervised learning = no labels required, discover a good “latent representation” of the input dataset