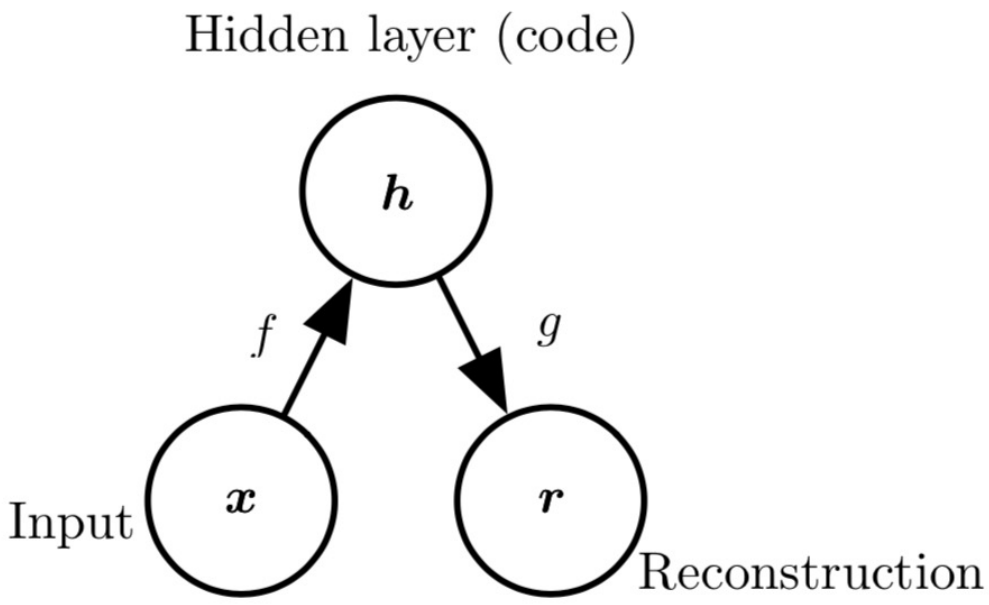
Lecture 17: Autoencoders

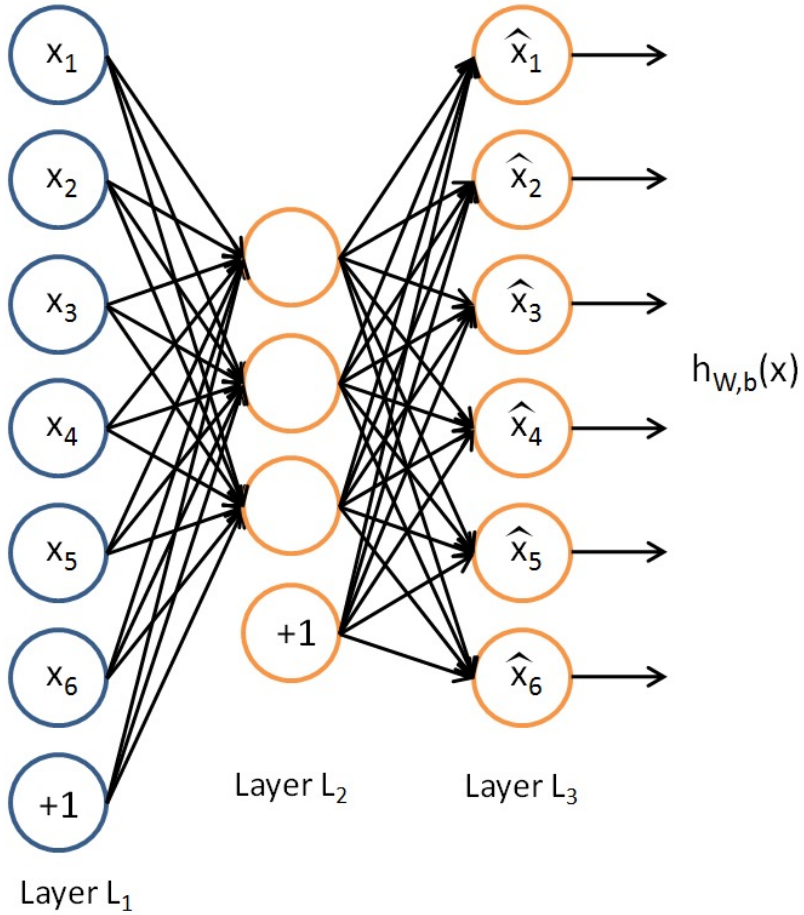
Andreas Wichert
Department of Computer Science and Engineering
Técnico Lisboa

Structure of an Autoencoder

- An autoencoder is a neural network that is trained to attempt to copy its input to its output.
- It has a hidden layer ***h*** that describes a code used to represent the input.
- The network may be viewed as consisting of two parts:
  - An encoder function ***h*** = f(***x***)
  - A decoder that produces a reconstruction ***r*** = g(***h***).



- Unsupervised Learning
  - Data: no labels!
  - Goal: Learn the structure of the data
- Traditionally, autoencoders were used for dimensionality reduction or feature learning.



Avoiding Trivial Identity

- Undercomplete autoencoders
  - $\boldsymbol{h}$ has lower dimension than $\boldsymbol{x}$
  - f or g has low capacity (e.g., linear g)
  - Must discard some information in $\boldsymbol{h}$

- Overcomplete autoencoders
  - $\boldsymbol{h}$ has higher dimension than $\boldsymbol{x}$
  - Must be regularized

Undercomplete Autoencoders

- An autoencoder whose code dimension is less than the input dimension is called undercomplete
- Minimising the Loss function

$$L(\mathbf{x}, g(f(\mathbf{x})))$$

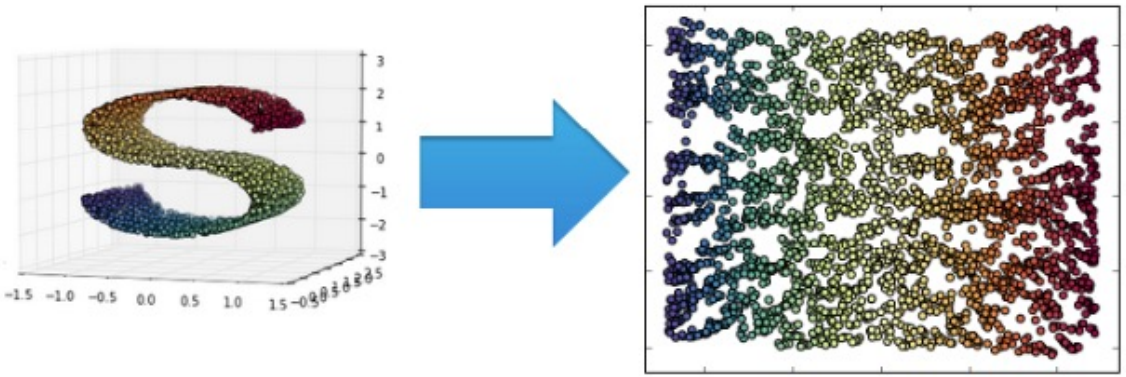
such as for example

$$E(\mathbf{w}) = \frac{1}{2} \cdot \sum_{k=1}^{N} (\mathbf{x}_k - g(f(\mathbf{x}_k)))^2$$

- If our input is interpreted as *bit vectors* or vectors of bit probabilities the *cross entropy* can be used

$$E(\mathbf{w}) = H(p, q) = \sum_{k=1}^{N} \sum_{bits} \mathbf{x}_k \cdot \log(g(f(\mathbf{x}_k)))$$

Applications

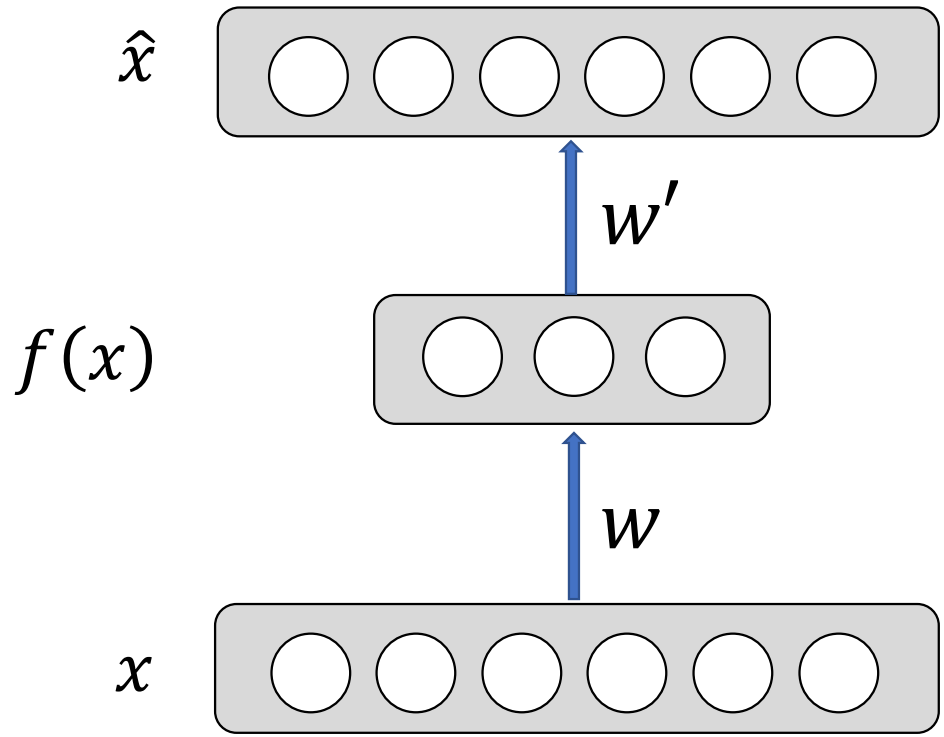


- Dimensionality reduction
- Visualization
- Feature extraction
- How to learn binary codes
- How good are 256-bit codes for retrieval of small color images?

Undercomplete AE

- Hidden layer is **Undercomplete** if smaller than the input layer
  - Compresses the input
  - Compresses well only for the training dist.

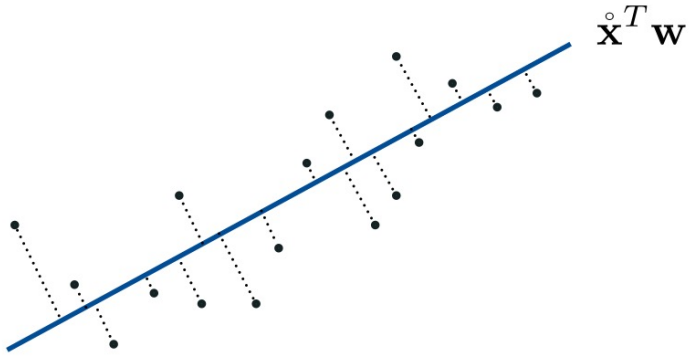
- Hidden nodes will be
  - Good features for the training distribution.
  - Bad for other types on input

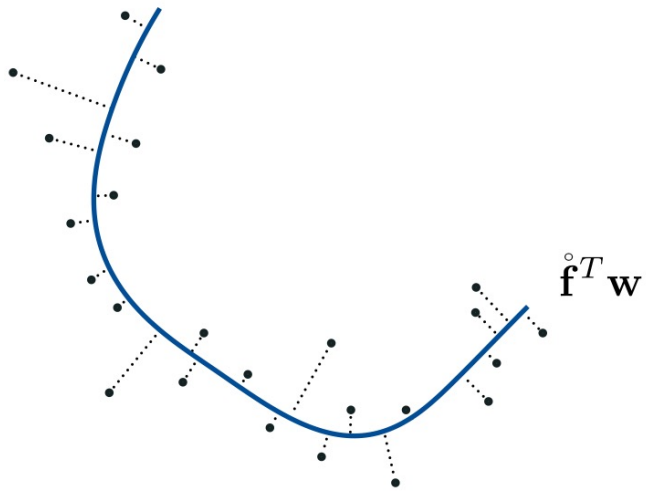


$$L(\mathbf{x}, g(f(\mathbf{x})))$$

- trained with backpropagation using minibatches

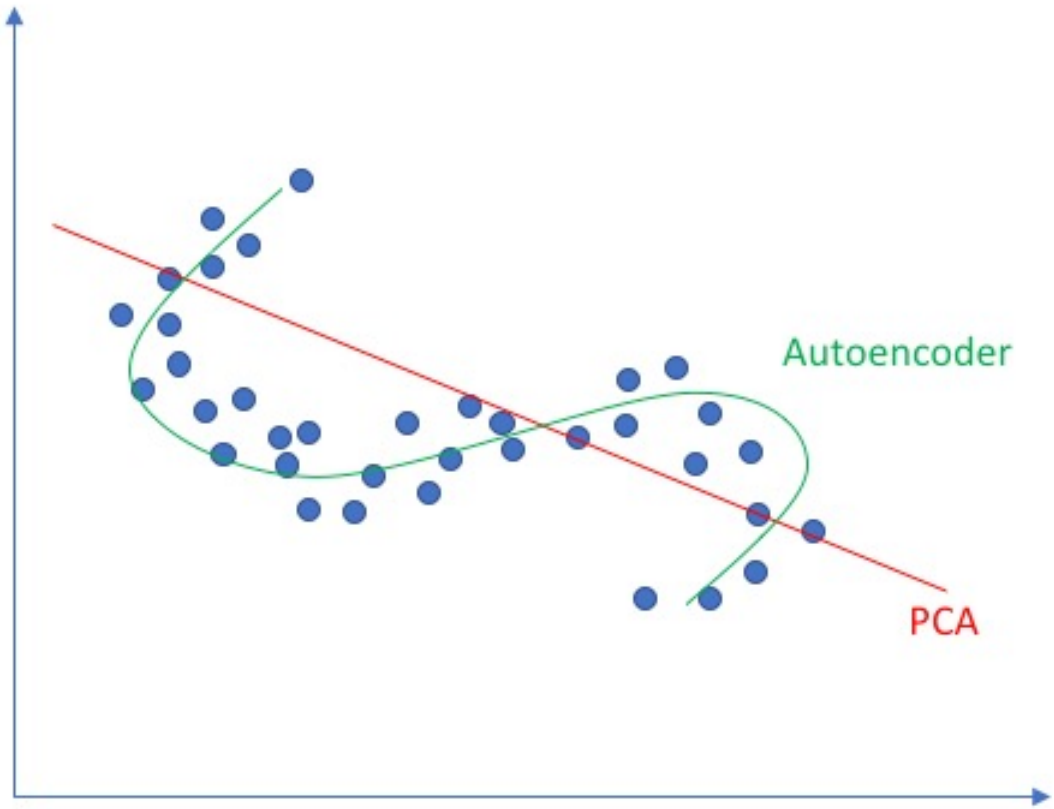
- Learning an undercomplete representation forces the autoencoder to capture the **most important features** of the training data.
- When the decoder is linear and L is the mean squared error, an under- complete autoencoder learns to span the same subspace as PCA





- Autoencoders with nonlinear encoder functions ***f*** and nonlinear decoder functions ***g*** can thus learn a more powerful nonlinear generalization of PCA (later)

Linear vs nonlinear dimensionality reduction



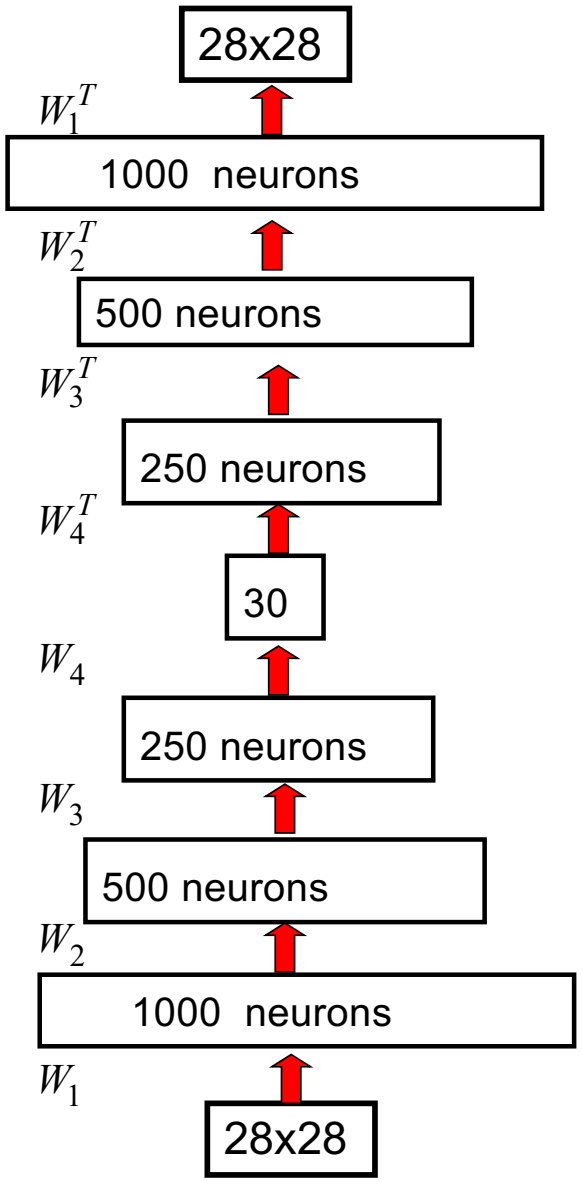
Too much Capacity

- An autoencoder with a one-dimensional code but a very powerful nonlinear encoder could learn to represent each training example $\boldsymbol{x}^{(i)}$ with the code i.
- The decoder could learn to map these integer indices back to the values of specific training examples.
- An autoencoder trained to perform the copying task can fail to learn anything *useful* about the dataset if the **capacity** of the autoencoder is allowed to become **too great**.

Deep Autoencoders

(Geoffrey Hinton with Ruslan Salakhutdinov)

- They always looked like a really nice way to do non-linear dimensionality reduction:
- But it is very difficult to optimize deep autoencoders using backpropagation.
- Layer-wise training

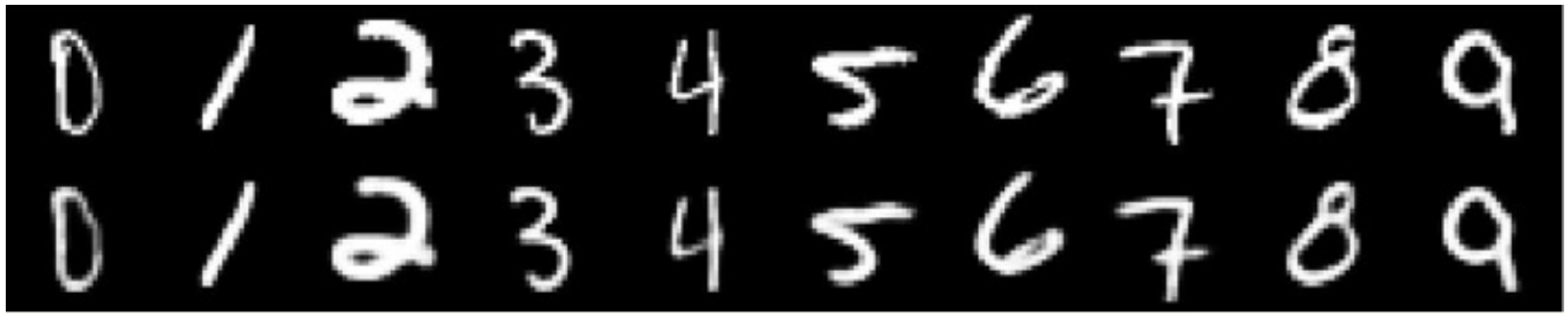


Layer-wise

- Pretraining involves successively adding a new hidden layer to a model and refitting
- Allowing the newly added model to learn the inputs from the existing hidden layer, often while keeping the weights for the existing hidden layers fixed.
- This gives the technique the name "*layer-wise*" as the model is trained one layer at a time.

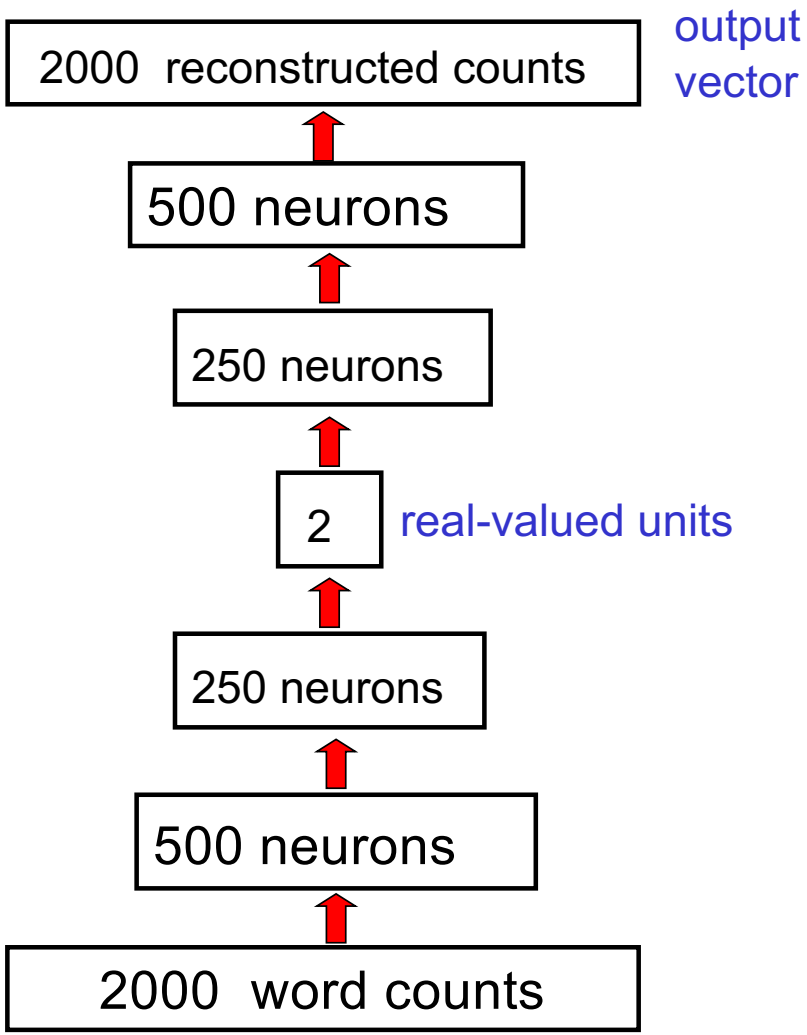
A comparison of methods for compressing digit images to 30 real numbers.

- Real Data

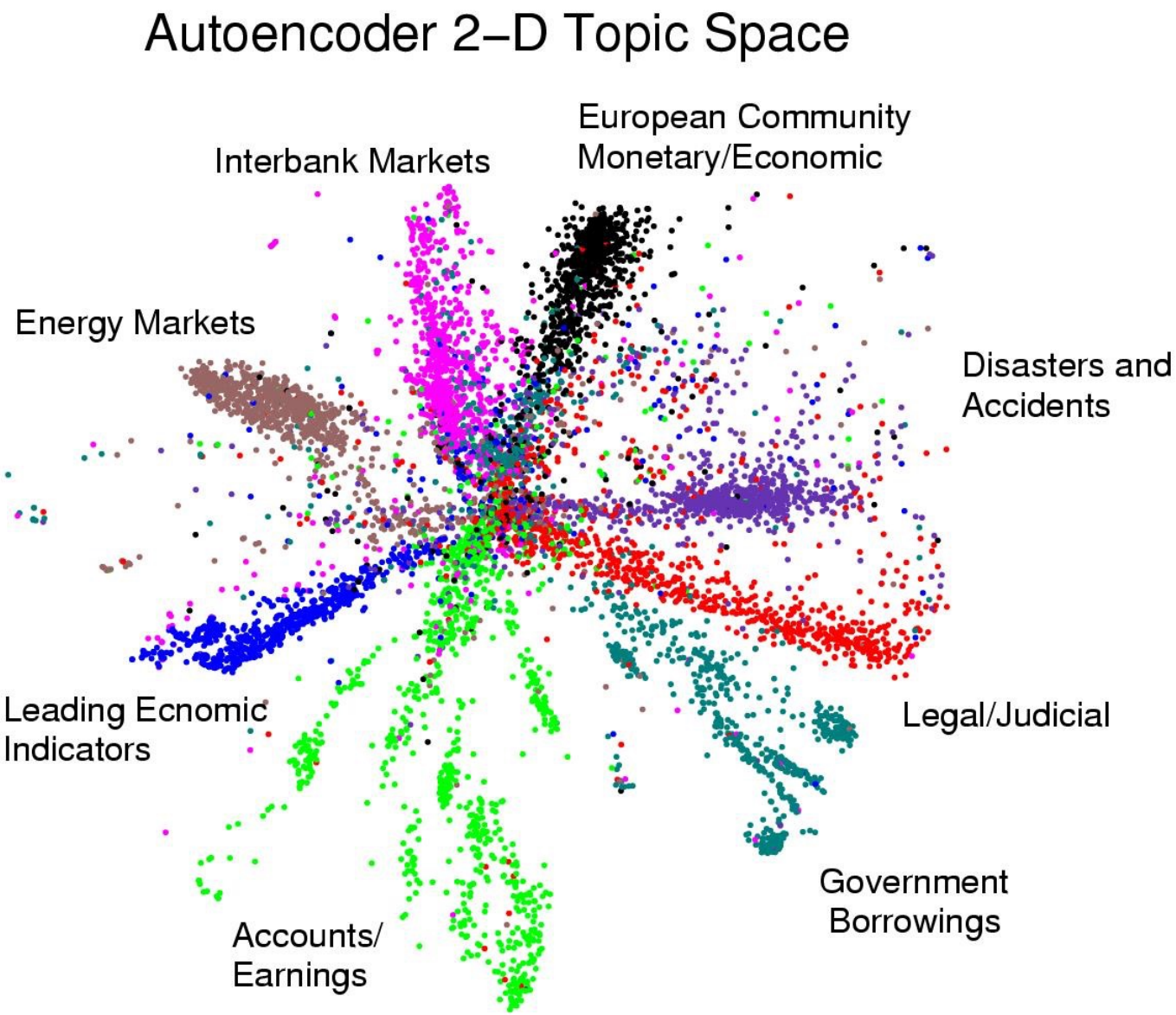


- 30-D deep auto

Compressing a document count vector to 2 numbers



- We train the autoencoder to reproduce its input vector as its output
- This forces it to compress as much information as possible into the 2 real numbers in the central bottleneck.
- These 2 numbers are then a good way to visualize documents.



- We train the autoencoder to reproduce its input vector as its output
- This forces it to compress as much information as possible into the 2 real numbers
- Then use different colors for different document categories

Overcomplete Autoencoders

- One way to obtain useful features from the autoencoder is to constrain ***h*** to have bigger dimension than ***x***.
- An autoencoder whose code dimension is bigger than the input dimension is called overcomplete
- For Overcomplete Autoencoders a linear encoder and linear decoder can learn to copy the input to the output without learning anything useful about the data distribution.
- A **regularized** autoencoder can be nonlinear and overcomplete but still learn something useful about the data distribution even if the model capacity is great enough to learn a trivial identity function.

Sparse Autoencoders

- Limit capacity of autoencoder by adding a term to the cost function penalizing the code for being larger
- Special case of variational autoencoder

- Probabilistic model
- Laplace prior corresponds to l_1 sparsity penalty Dirac variational posterior

Sparse Autoencoders

$$L(\mathbf{x}, g(f(\mathbf{x})) + \Omega(\mathbf{h}))$$

with sparsity penalty $\Omega(\mathbf{h})$, $g(\mathbf{h})$ is the decoder output and typically we have $\mathbf{h} = f(\mathbf{x})$

For example

$$E(\mathbf{w}) = \frac{1}{2} \cdot \sum_{k=1}^{N} (\mathbf{x}_k - g(f(\mathbf{x}))_k)^2 + \lambda \cdot \|\mathbf{w}\|_1$$

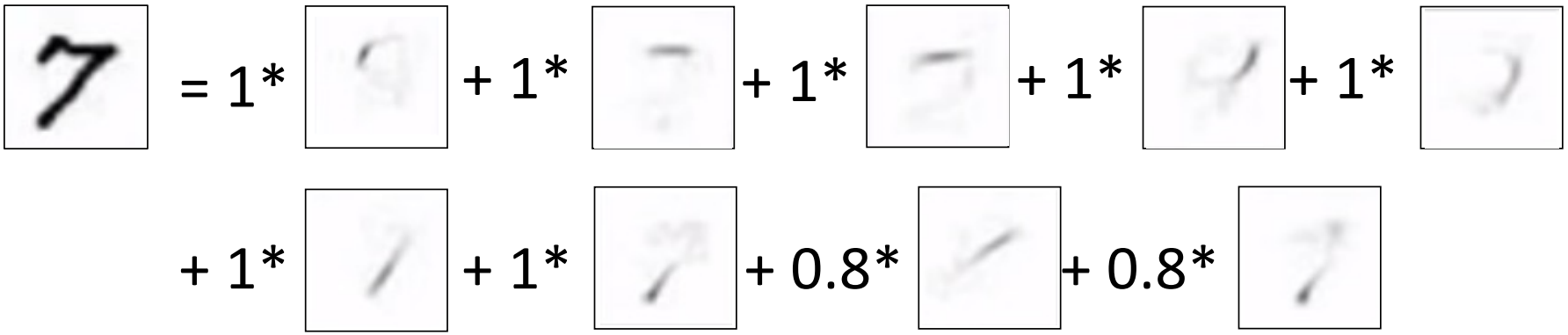
for one hidden layer. We can use $\lambda \cdot \|\mathbf{w}\|_1$ only for $\mathbf{h}$

Sparse Autoencoders

- An autoencoder that has been regularized to be sparse must respond to unique statistical features of the dataset it has been trained on, rather than simply acting as an identity function.
- In this way, training to perform the copying task with a sparsity penalty can yield a model that has learned useful features
- One way to achieve actual zeros in $\boldsymbol{h}$ for sparse autoencodersis to use rectified linear units to produce the code layer.
- With a prior that actually pushes the representations to zero $\lambda \cdot \|\boldsymbol{w}\|_1$, one can indirectly control the average number of zeros in the representation.

Sparsely Regulated Autoencoders

- We want our learned features to be as **sparse** as possible.
- With sparse features we can generalize better.



Sparse Codes

- Why sparse codes?
- Biologically Plausible

- Sparse coding: Very small number of 1s is equally distributed over the coordinates of the vectors
- "*En Passant*" we introduce the Whilshaw Associative Memory

Sparse Codes: Associative Memory

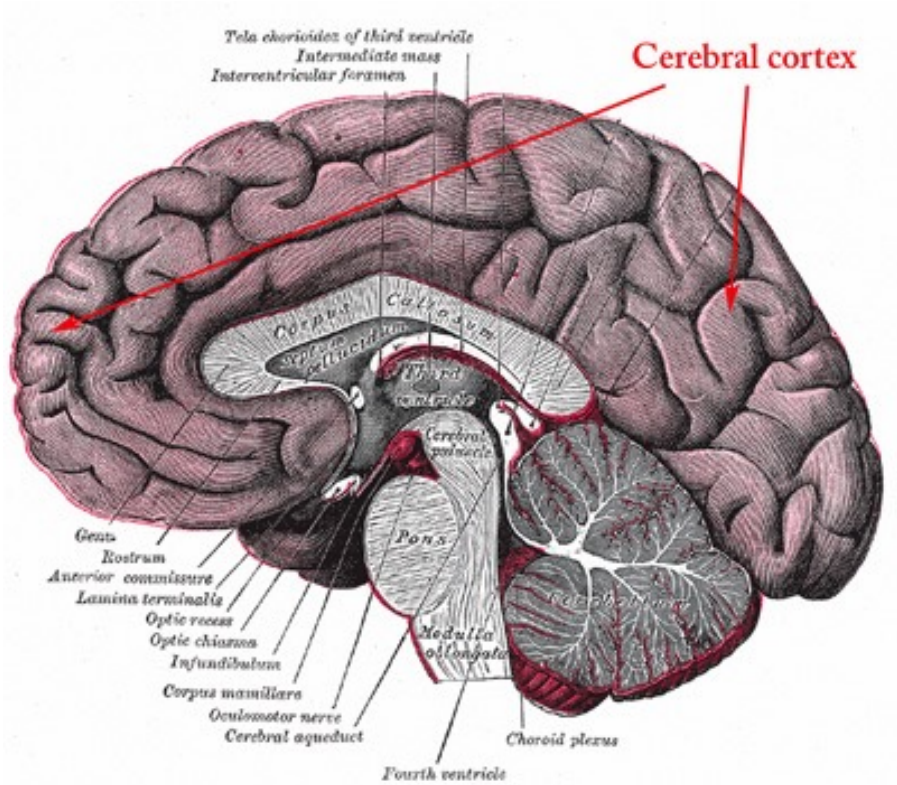
- Human memory is based on associations with the memories it contains
  - ... Just a snatch of well-known tune is enough to bring the whole thing back to mind
  - ... A forgotten joke is suddenly completely remembered when the next-door neighbor starts to tell it again
- This type of memory has previously been termed content-addressable, which means that one small part of the particular memory is linked - associated -with the rest.

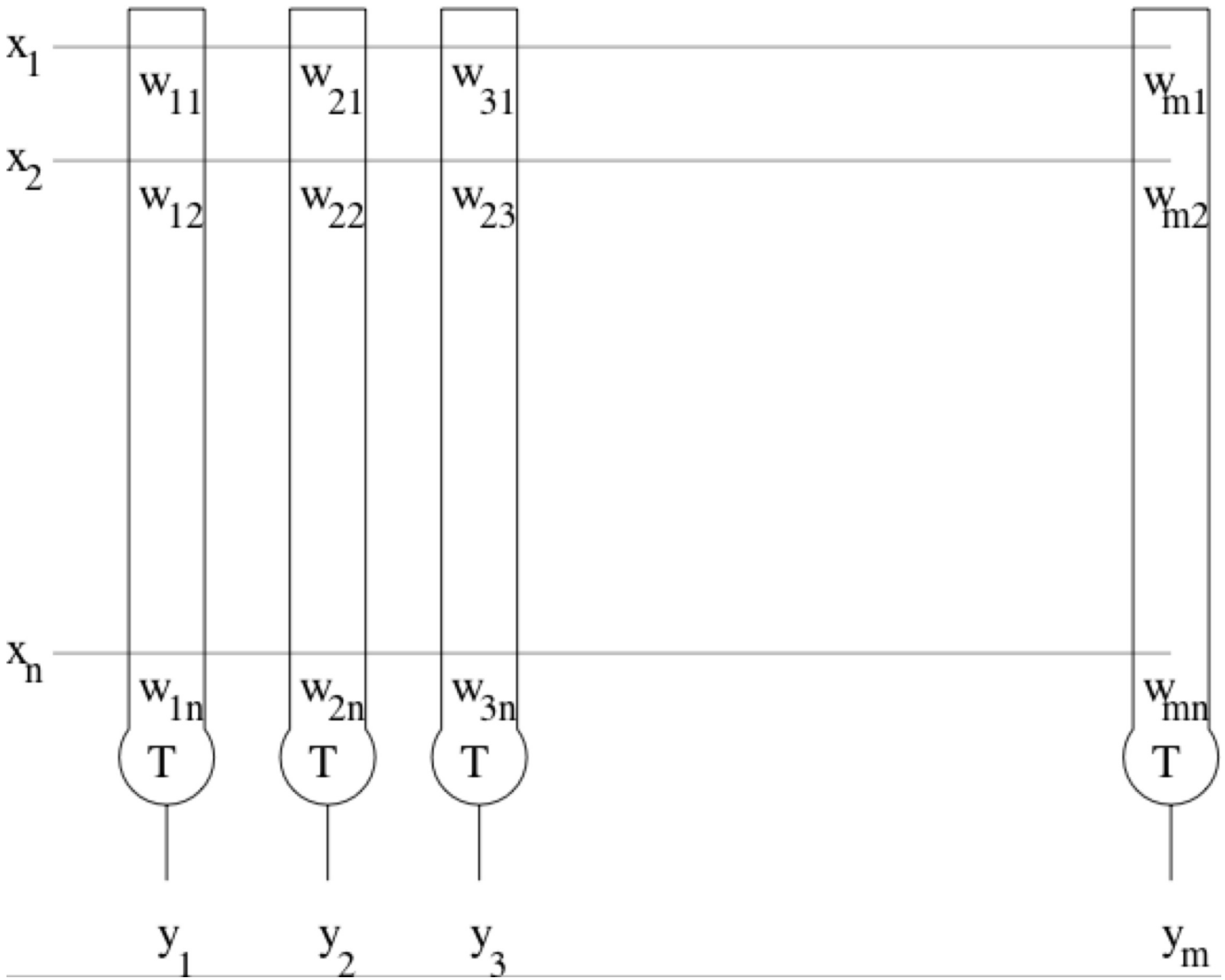
Sparse Codes: Associative Memory

- The ability to correct faults if false information is given
- To complete information if some parts are missing
- To interpolate information, that means if a pattern is not stored the most similar stored pattern is determined

Sparse Codes

- The cerebral cortex is a huge associative memory
- or rather a large network of associatively connected topographical areas
- Associations between patterns are formed by Hebbian learning




x1
x2
xn
w11
w21
w31
wm1
w12
w22
w23
wm2
w1n
w2n
w3n
wmn
T
T
T
T
y1
y2
y3
ym

Sparse Codes

- The patterns are represented by binary **sparse** vectors
- The presence of a feature is indicated by a one component of the vector, its absence through a zero component of the vector

- Always two pairs of these vectors are associated
- This process of the association is called learning

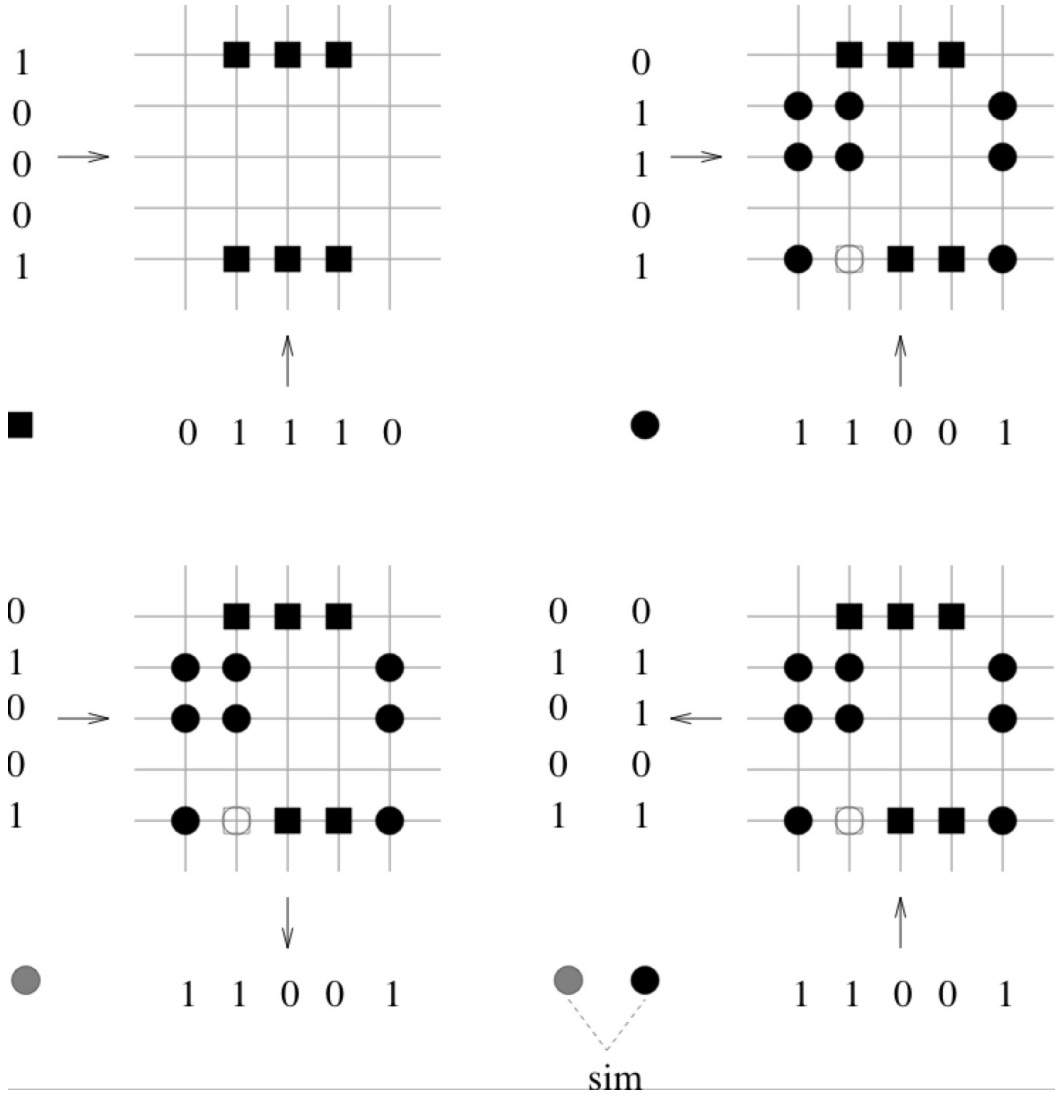
Sparse Codes

- The first of the two vectors is called the question vector and the second the answer vector
- After the learning the question vector is presented to the associative memory and the answer vector is determined

- This process is called:

  - **association** provided that the answer vector represents the reconstruction of the disturbed question vector

  - **heteroassocation** if both vectors are different


1
0
0
0
1
0 1 1 1 0
0
1
1
0
1
1 1 0 0 1
0
1
0
0
1
1 1 0 0 1
0 0
1 1
0 1
0 0
1 1
1 1 0 0 1
sim

Learning

- In the initialization phase of the associative memory no information is stored;
  - because the information is represented in the ***w*** weights they are all set to zero

- In the learning phase, binary vector pairs are associated
- Let ***x*** be the question vector and ***y*** the answer vector, so that the learning rule
- is:

$$w_{ij}^{new} = 1 \quad if\ y_i \cdot x_j = 1$$
$$w_{ij}^{new} = w_{ij}^{old}\ otherwise$$

- This rule is called the binary **Hebb rule**

- In the *one-step* retrieval phase of the associative memory
- A fault tolerant answering mechanism recalls the appropriate answer vector for a question vector $\boldsymbol{x}$
- To the presented question vector $\boldsymbol{x}$ the most similar learned $\boldsymbol{x}^l$ question vector regarding the *Hamming distance* is determined
  - Hamming distance indicates how many positions of two binary vectors are different
- The appropriate answer vector $\boldsymbol{y}$ is identified

Retrieval

$$y_i = \begin{cases} 1 \ \sum_{j=1}^{n} w_{ij} x_j \geq T \\ 0 \text{ otherwise.} \end{cases}$$

- *T* is the threshold of the unit
  - In the hard threshold strategy, the threshold *T* is set to the number of "one" components in the question vector
    - If one uses this strategy it is quite possible that no answer vector is determined
  - In soft threshold strategy, the threshold is set to the maximum sum

$$T :=_{max\ i} \sum_{j=1}^{n} \delta(w_{ij} x_j) \qquad \delta(x) = \begin{cases} 1 \text{ if } \ x > 0 \\ 0 \text{ if } \ x = 0 \end{cases}$$

Sparse Codes: Storage Analysis

- For an estimation of the asymptotic number ***L*** of vector pairs (***x***, ***y***) which can be stored in an associative memory before it begins to make mistakes in retrieval phase.
- It is assumed that both vectors have the same dimension ***n***
- It is also assumed that both vectors are composed of ***M*** *1*s, which are likely to be in any coordinate of the vector

Sparse Codes: Storage Analysis

- The optimum value for *M* is approximately

$$M \doteq \log_2(n/4)$$

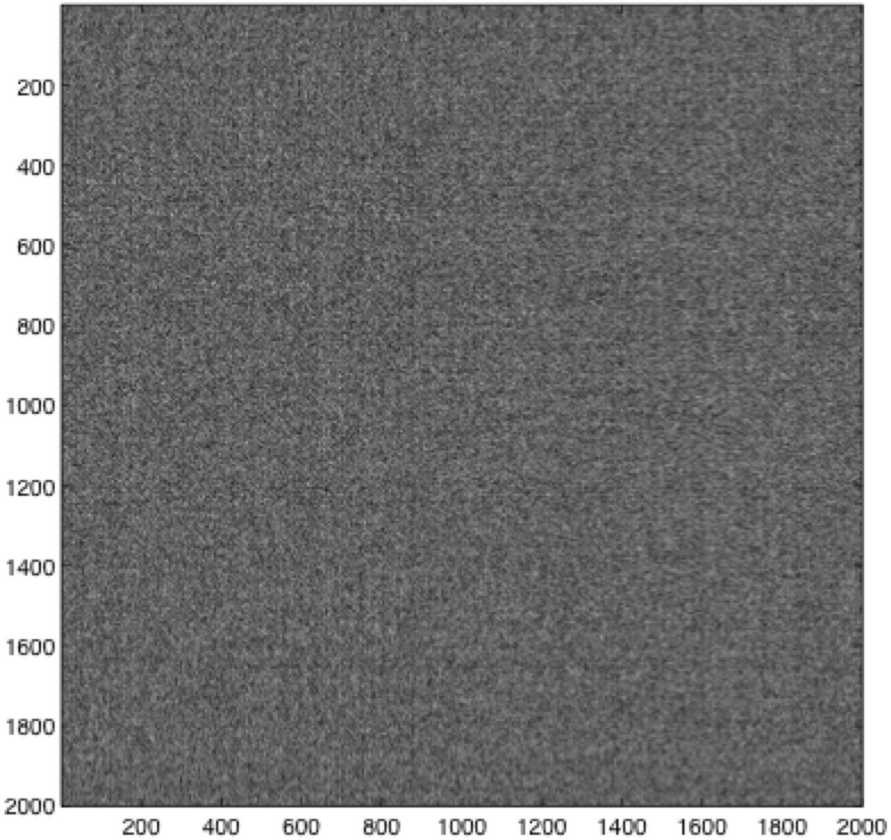
- *L* vector pairs can be stored in the associative memory

$$L \doteq (\ln 2)(n^2/M^2)$$

- This value is much ***greater*** then *n* if the optimal value for ***M*** is used

Sparse Codes: Storage Analysis

- *L* is much ***greater*** then *n* if the optimal value for ***M*** is used
- Storage of data ***and*** fault tolerant answering mechanism!
  - Sparse coding: Very small number of 1s is equally distributed over the coordinates of the vectors
  - For example, in the vector of the dimension *n=1000000 M=18*, ones should be used to code a pattern
  - The real storage capacity value is lower when patterns are used which are not sparse



- The weight matrix after learning of 20000 test patterns, in which ten ones were randomly set in a 2000 dimensional vector represents a high loaded matrix with equally distributed weights

- One of the Holy Grails of Neuroscience
- **Sparse coding** is the representation of items by the strong activation of a relatively small set of neurons
- Sparse coding is also relevant to the amount of energy the brain needs to use to sustain its function.
- The total number of action potentials generated in a brain area is inversely related to the sparseness of the code
  - The total energy consumption decreases with increasing sparseness.

Denoising Autoencoders

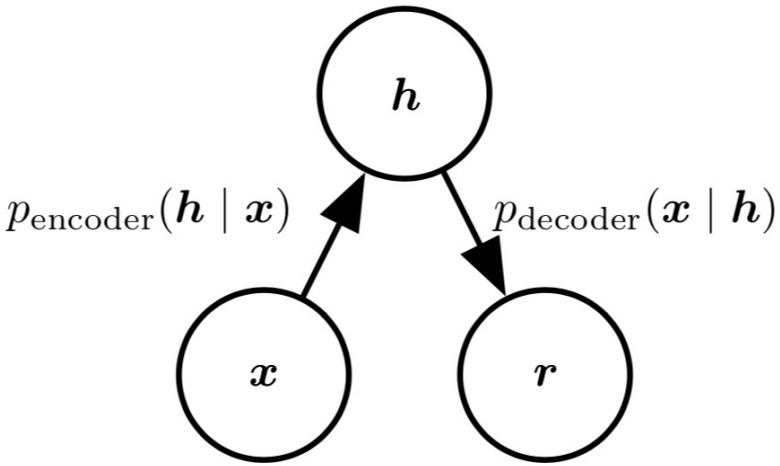
A denoising autoencoder or DAE instead minimizes

$$L(\mathbf{x}, g(f(\tilde{\mathbf{x}})) + \Omega(\mathbf{h}))$$

where $\tilde{\mathbf{x}}$ is a copy of $\mathbf{x}$ that has been corrupted by some form of noise.

Denoising autoencoders must therefore undo this corruption rather than simply copying their input.

Stochastic Encoders and Decoders



Stochastic Encoders

$$p_{encoder}(\mathbf{h}|\mathbf{x})$$

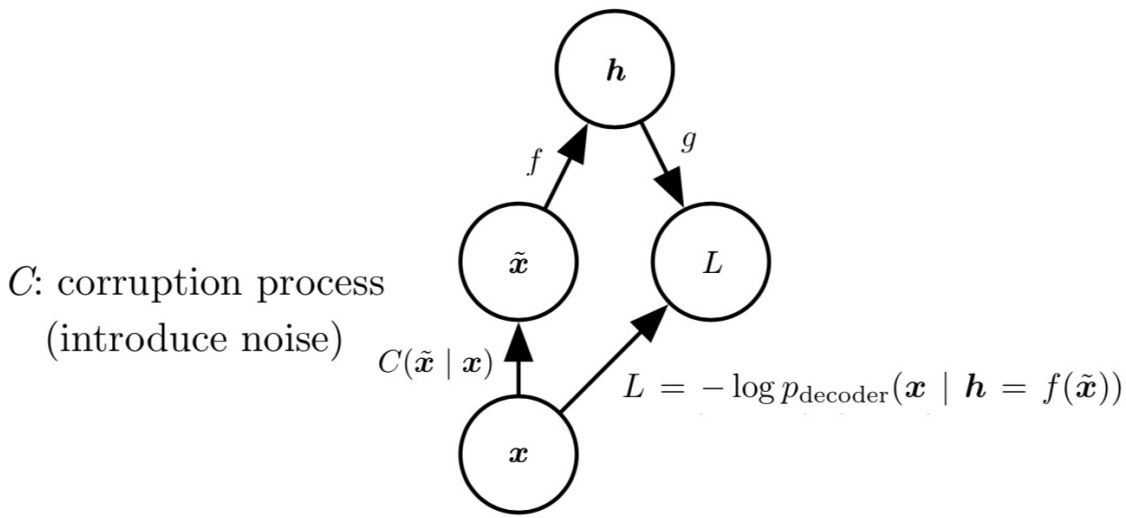
Stochastic Decoder

$$p_{decoder}(\mathbf{x}|\mathbf{h})$$

We may then train the autoencoder by minimizing

$$-\log p_{decoder}(\mathbf{x}|\mathbf{h})$$

Denoising Autoencoders



The denoising autoencoder (DAE) is an autoencoder that receives a corrupted data point as input and is trained to predict the original, uncorrupted data point as its output.
A corruption process $C(\tilde{\mathbf{x}}, \mathbf{x})$ represents a conditional distribution over corrupted samples $\tilde{\mathbf{x}}$ given a data sample $\mathbf{x}$

The autoencoder then learns a reconstruction distribution $p_{reconstruct}(\mathbf{x}|\tilde{\mathbf{x}})$ from samples $(\tilde{\mathbf{x}}.\mathbf{x})$

Sample a training example x from the training data $\mathbf{x}$
Sample a corrupted version $\tilde{\mathbf{x}}$ from $C(\tilde{\mathbf{x}}, \mathbf{x})$

Use $(\tilde{\mathbf{x}}, \mathbf{x})$ s a training example for estimating the autoencoder reconstruction distribution

$$p_{reconstruct}(\mathbf{x}|\tilde{\mathbf{x}}) = p_{decoder}(\mathbf{x}|\mathbf{h})$$

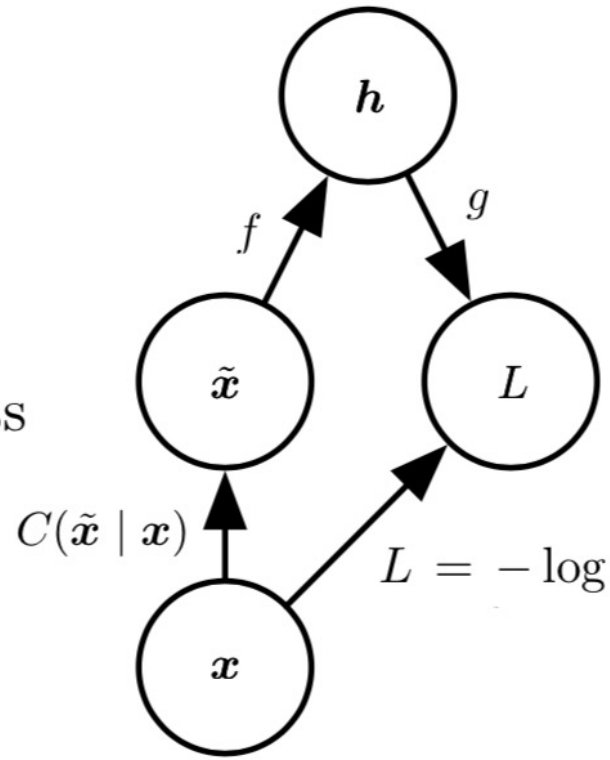
with $\mathbf{h}$ the output of the encoder $\mathbf{h} = f(\tilde{\mathbf{x}})$

and $p_{decoder} = g(\mathbf{h})$

A denoising autoencoder is trained to map a corrupted data point $\tilde{\mathbf{x}}$ back to the original data point $\mathbf{x}$

Denoising Autoencoder

C: corruption process
(introduce noise)



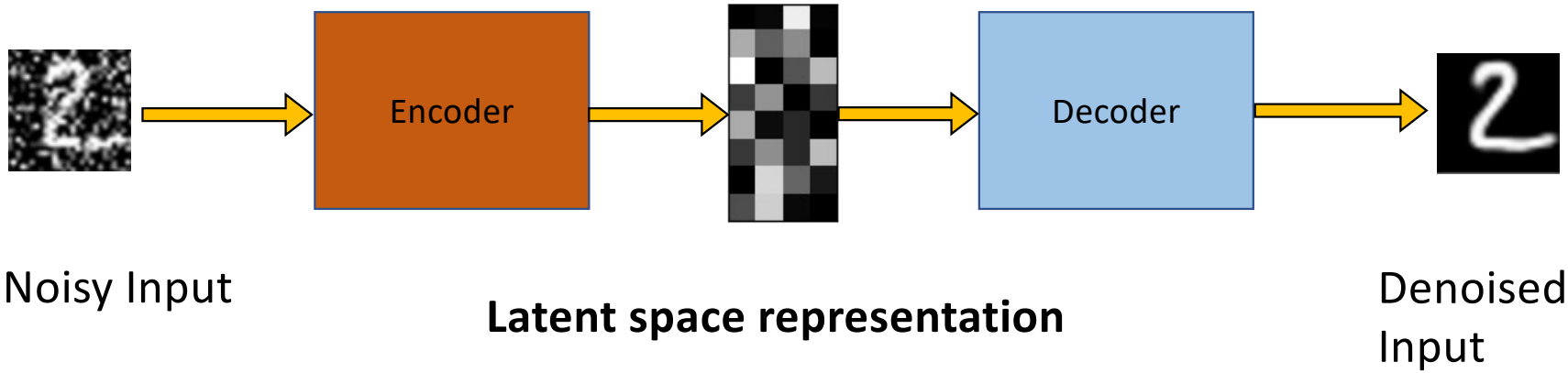
$$L = -\log p_{\text{decoder}}(\boldsymbol{x} \mid \boldsymbol{h} = f(\tilde{\boldsymbol{x}}))$$

Denoising Autoencoders

<u>Intuition:</u>

- We still aim to encode the input and to NOT mimic the identity function.
- We try to undo the effect of *corruption* process stochastically applied to the input.

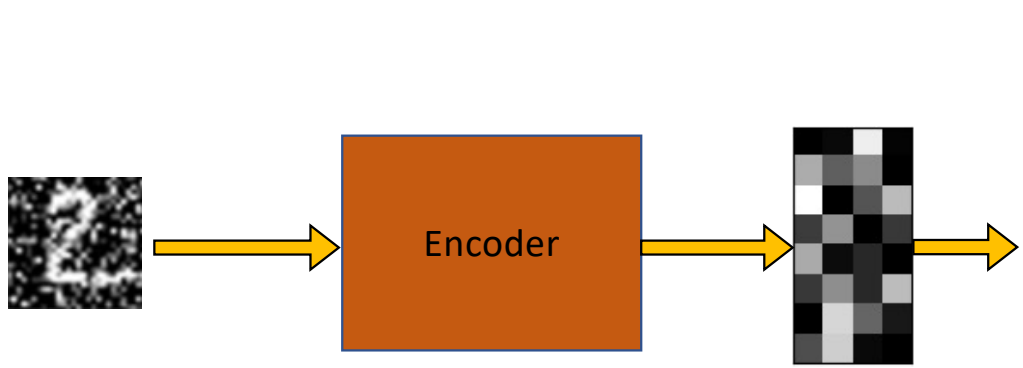
A more robust model



Denoising Autoencoders

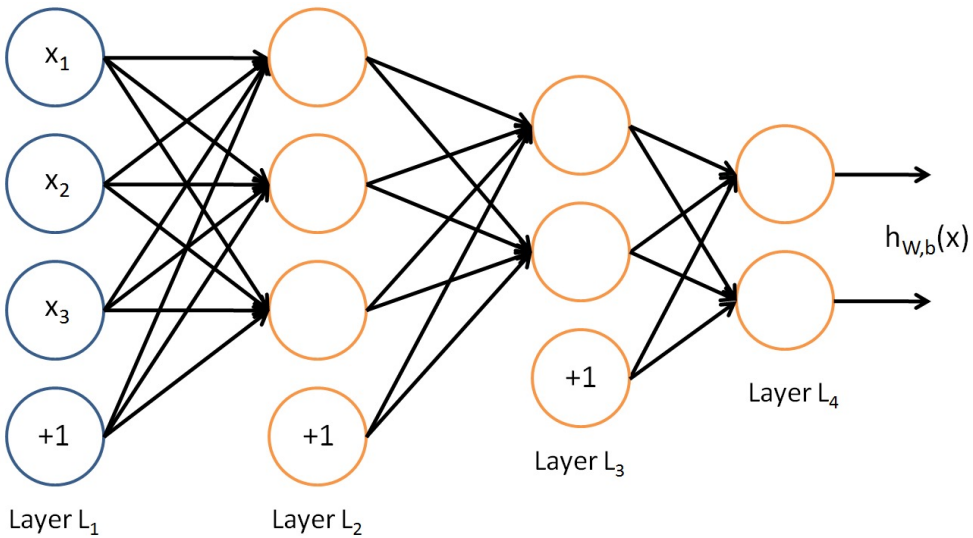
Use Case:

- Extract robust representation for a NN classifier.



Noisy Input

Latent space representation



Denoising Autoencoders

Instead of trying to mimic the identity function by minimizing:

$$L\left(x, g\left(f(x)\right)\right)$$

where L is some loss function

A **DAE** instead minimizes:

$$L\left(x, g\left(f(\tilde{x})\right)\right)$$

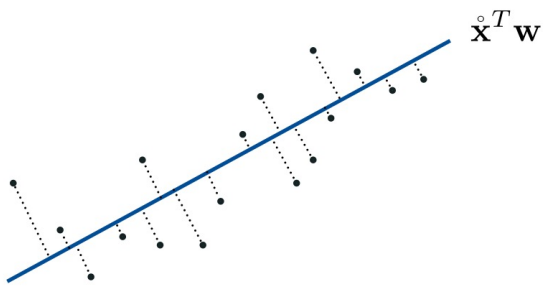
where $\tilde{x}$ is a copy of x that has been corrupted by some form of noise.

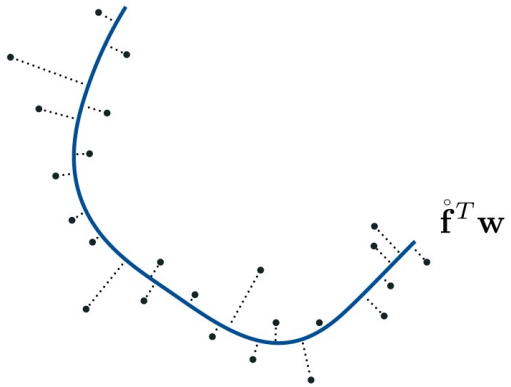
Stochastic Encoders and Decoders

- Data concentrates around a low-dimensional manifold or a small set of such manifolds
- Manifold is a topological space (which may also be a separated space) which locally resembles real n-dimensional space in, for example the real coordinate space Rn is the prototypical n-manifold, a circle is a compact 1-manifold.
- Autoencoders take this idea further and aim to learn the structure of the manifold.

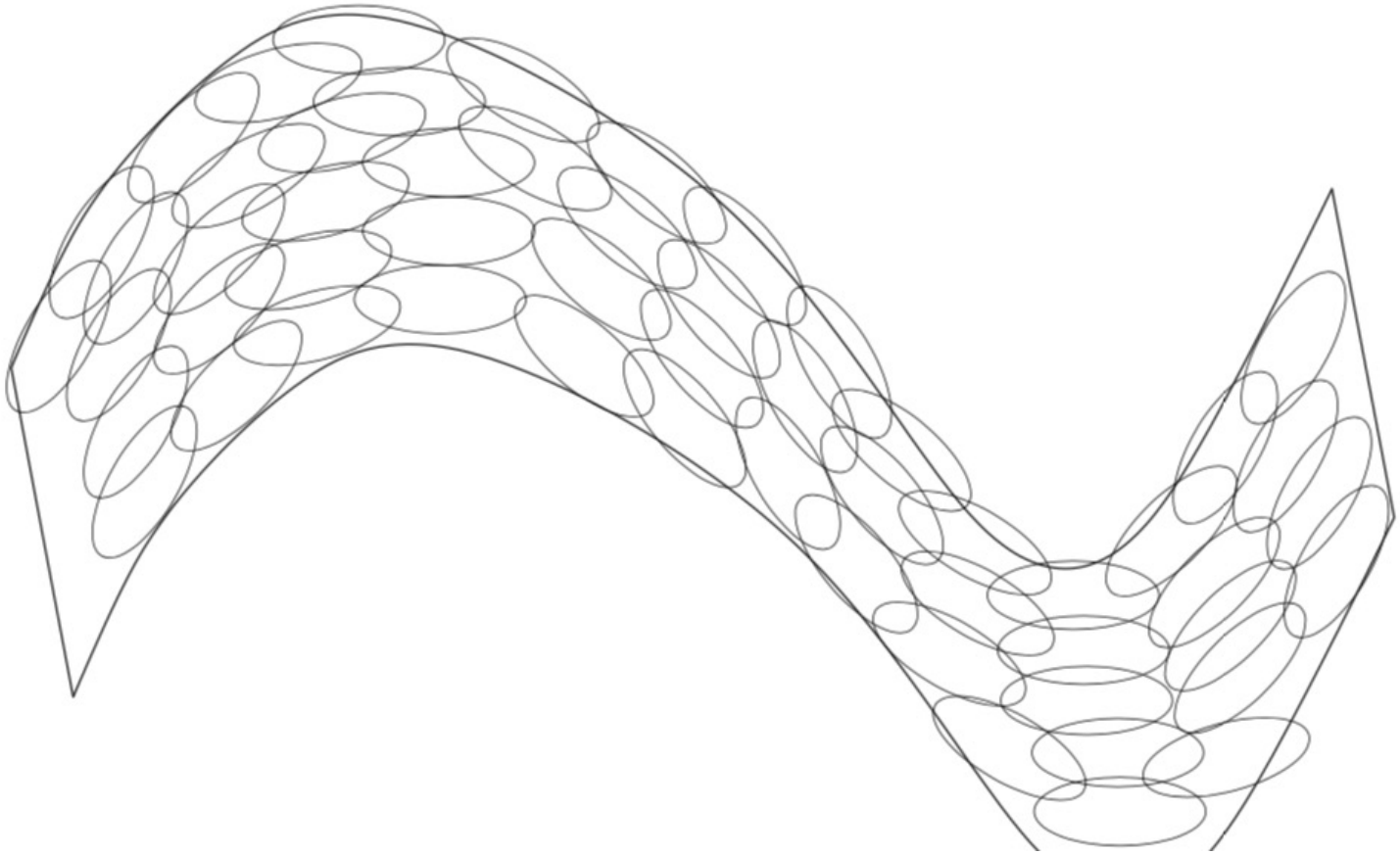
Stochastic Autoencoders

- An important characterization of a manifold is the set of its tangent planes. At a point ***x*** on a ***d***-dimensional manifold, the tangent plane is given by d basis vectors that span the local directions of variation allowed on the manifold

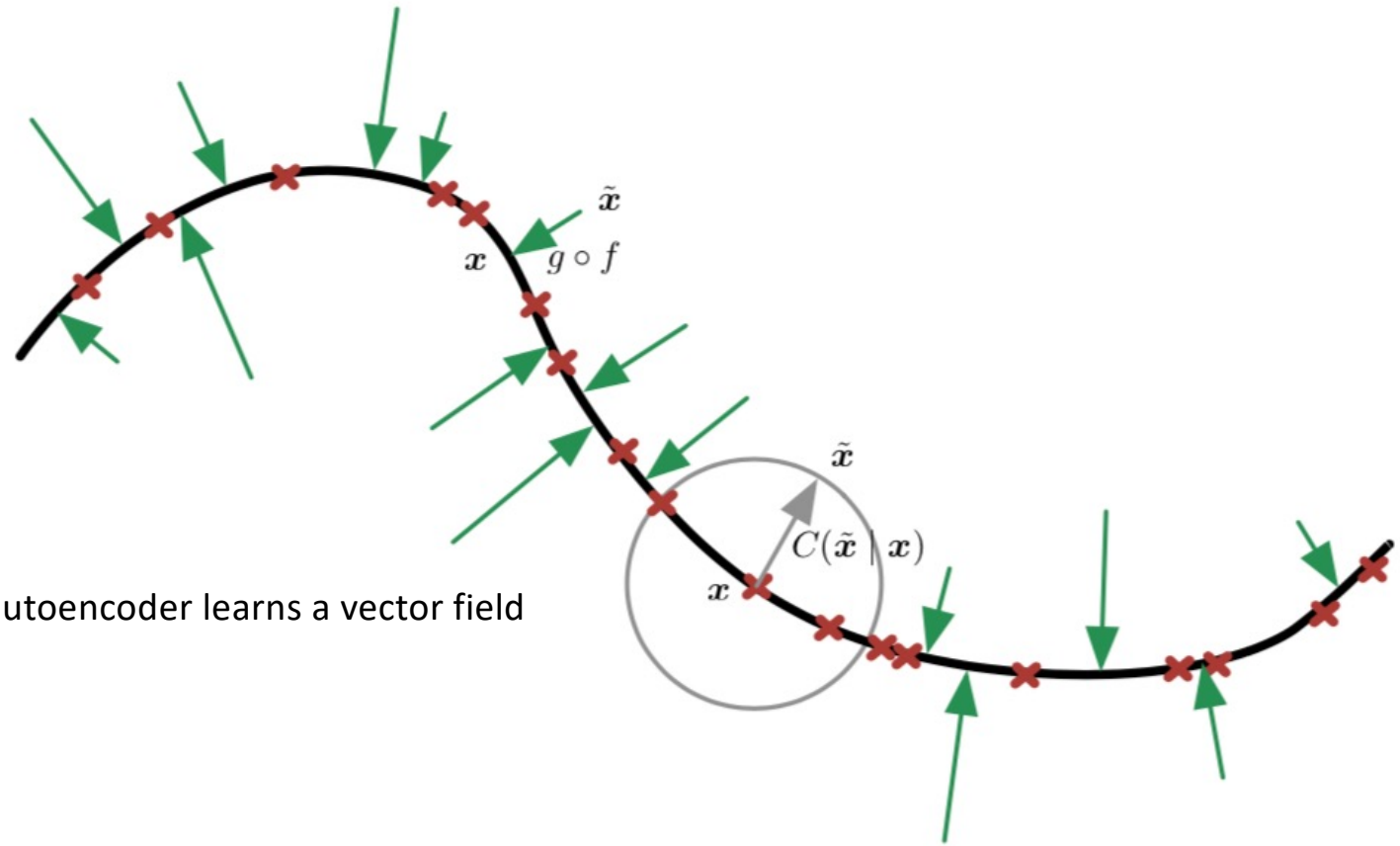




2-dimensional manifold in 3-dimensions



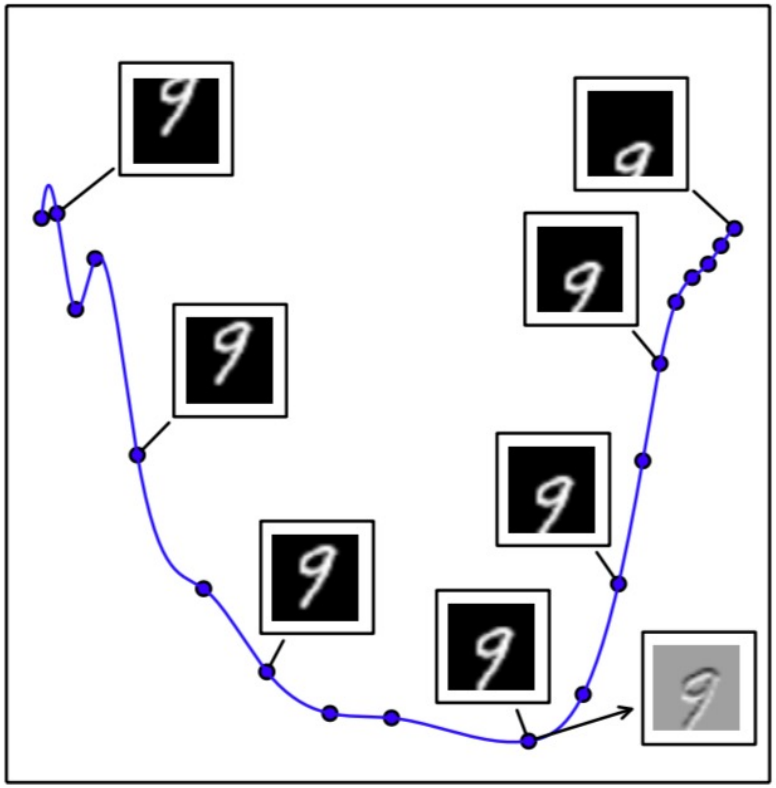
Denoising Autoencoders Learn a Manifold



The autoencoder learns a vector field

- An illustration of the concept of a tangent hyperplane.
- Here we create a one-dimensional manifold in 784-dimensional space.
- We take an MNIST image with 784 pixels and transform it by translating it vertically.
- The amount of vertical translation defines a coordinate along a one-dimensional manifold that traces out a curved path through image space.

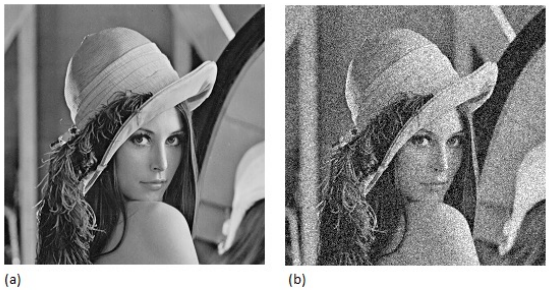
- 1-D manifold obtained by vertically translating image.
- In space of first 2 principal components
  with tangent line (gray image)

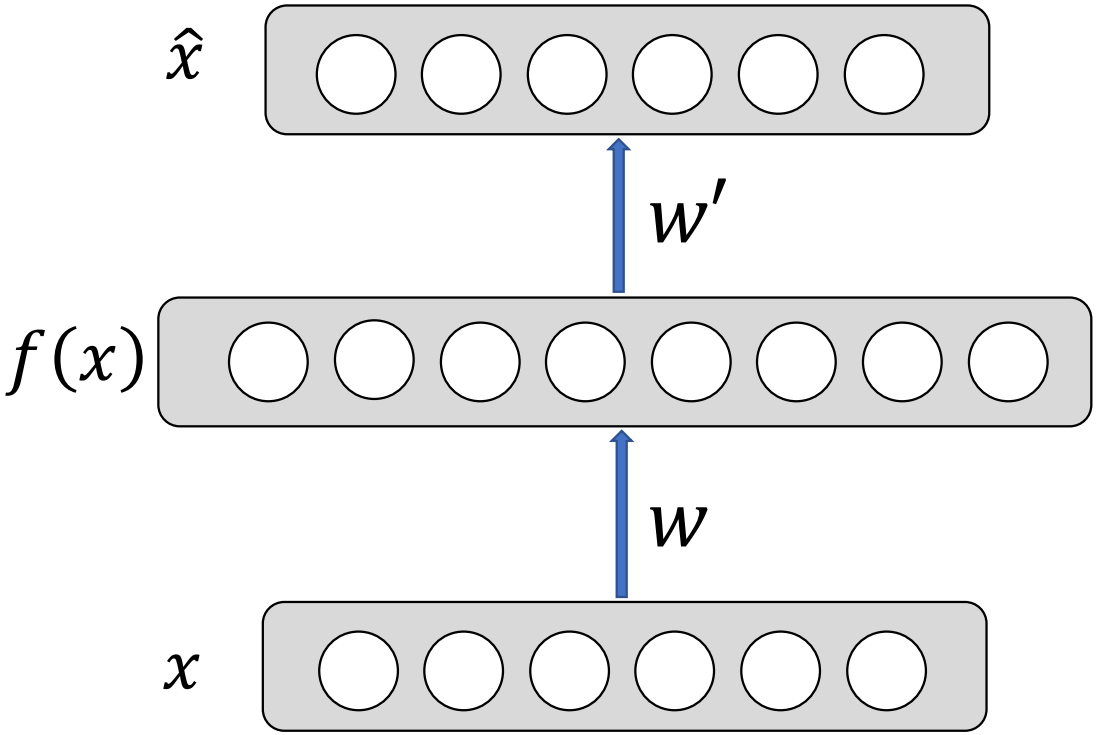


Denoising Autoencoders

Idea: A robust representation against noise:

- Random assignment of subset of inputs to 0, with probability v.
- Gaussian additive noise.



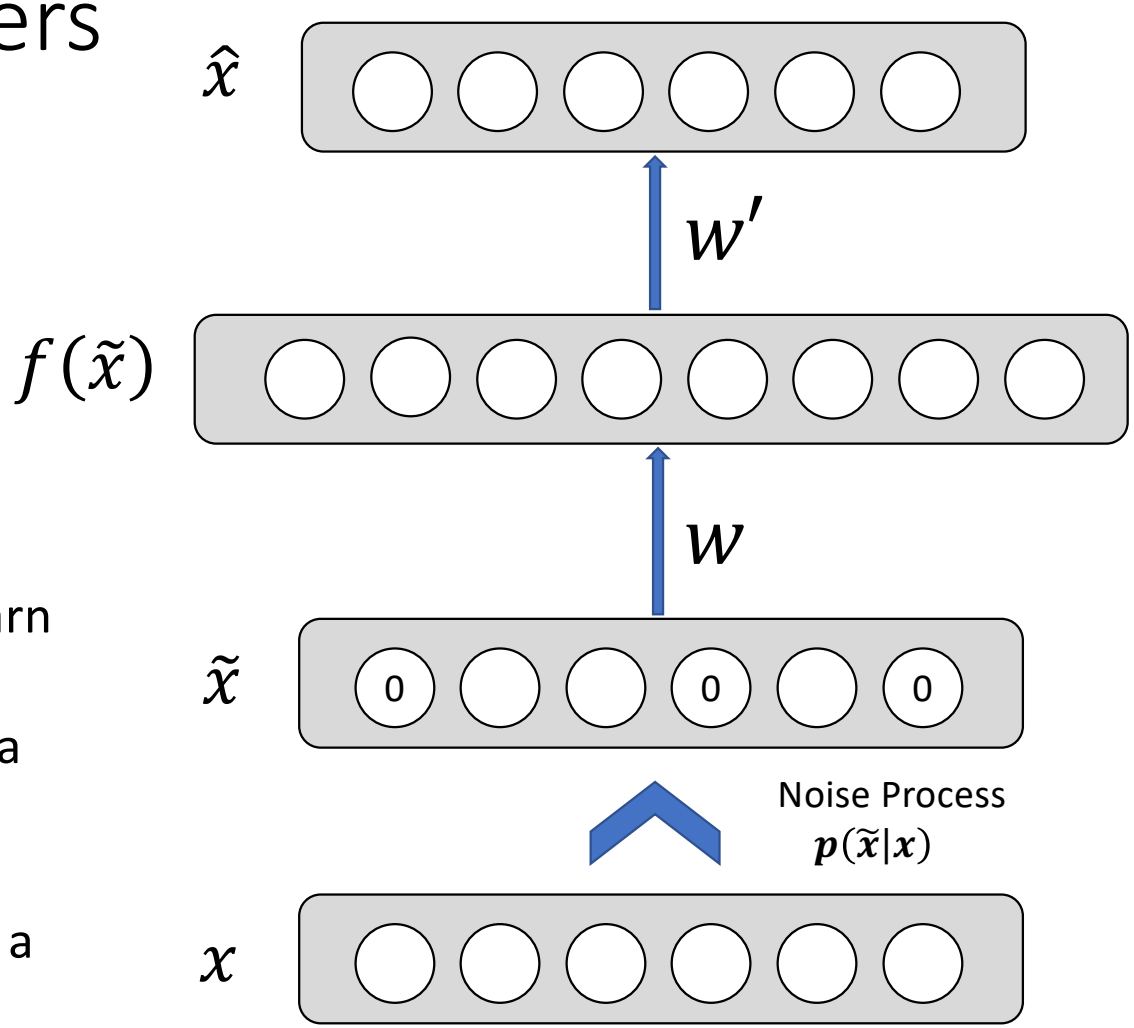


Denoising Autoencoders

- Reconstruction $\hat{x}$ computed from the corrupted input $\tilde{x}$.
- Loss function compares $\hat{x}$ reconstruction with the noiseless x.

- The autoencoder cannot fully trust each feature of x independently so it must learn the correlations of x's features.
- Based on those relations we can predict a more 'not prune to changes' model.

- We are forcing the hidden layer to learn a generalized structure of the data.



Denoising Autoencoders - process

Taken some input x

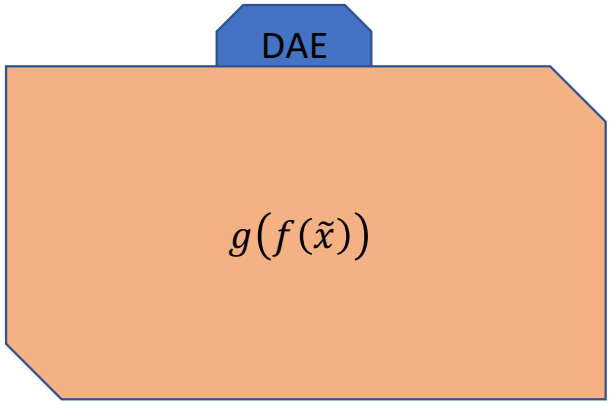
Apply Noise

$\tilde{x}$

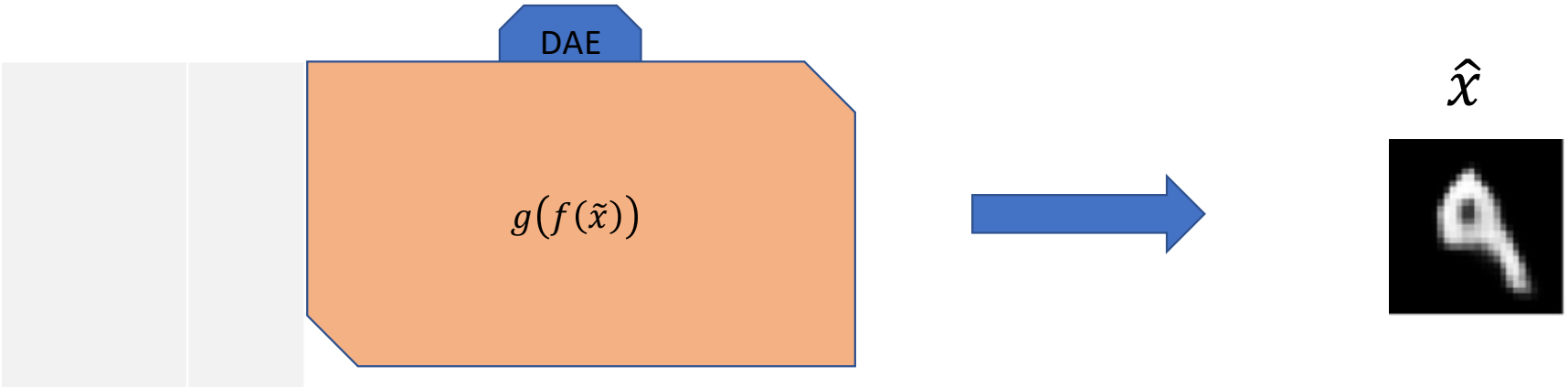
Denoising Autoencoders (DAE)

$\tilde{x}$

Encode And Decode

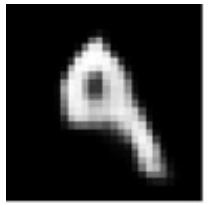


Denoising Autoencoders - process



Denoising Autoencoders - process

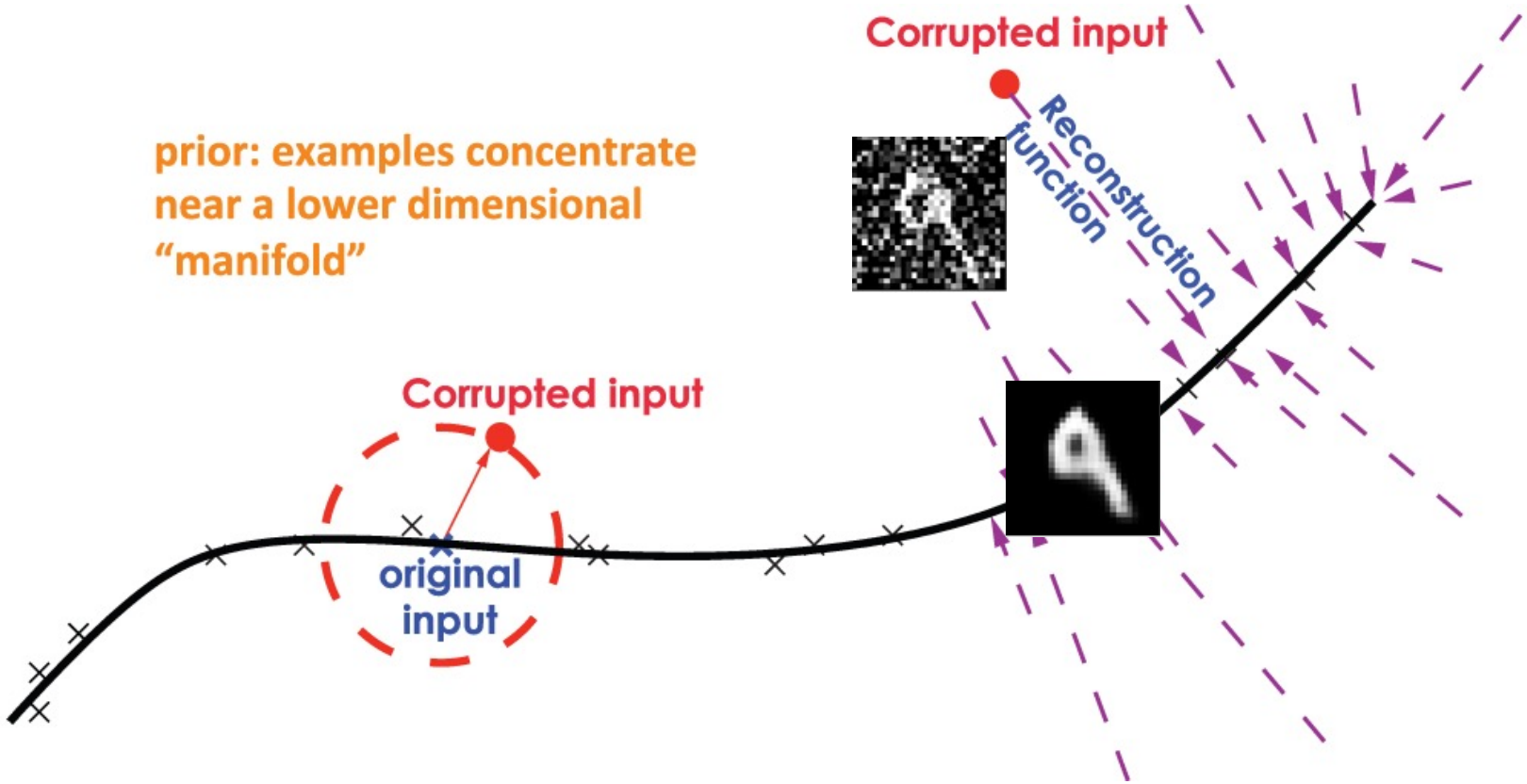
$\hat{x}$



Compare

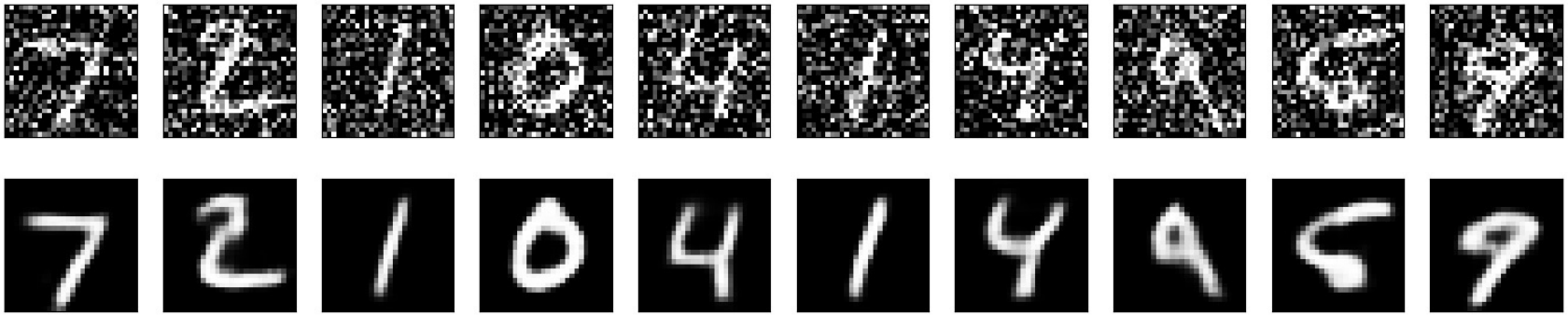
x

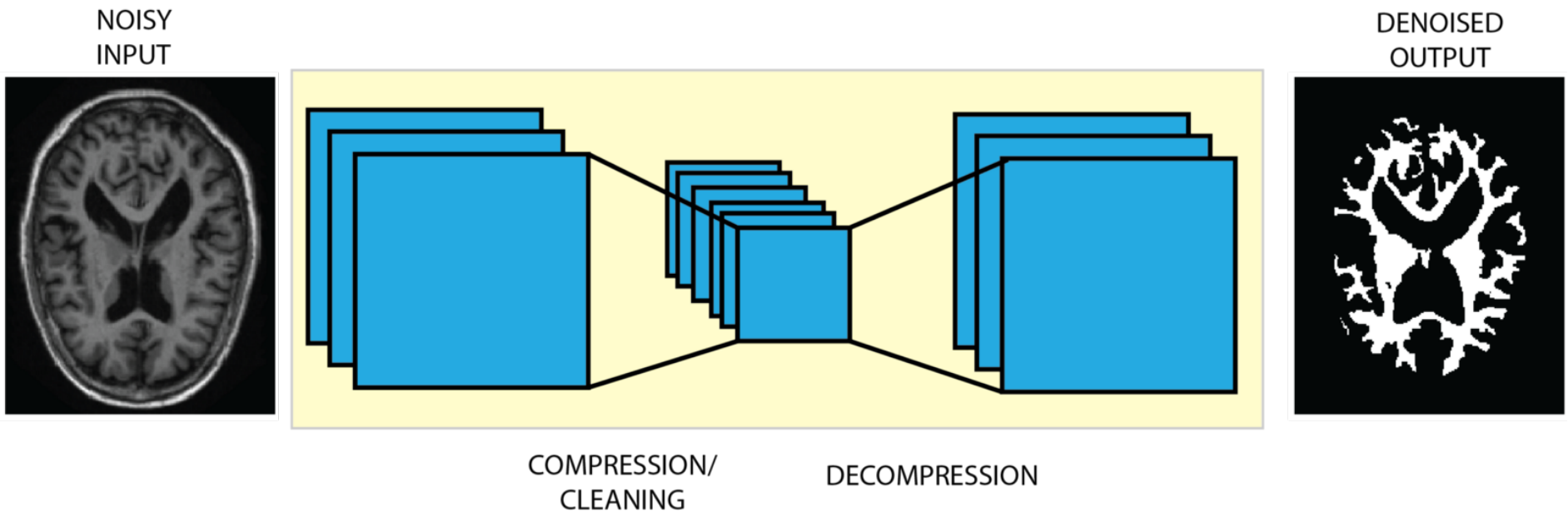
Denoising autoencoders



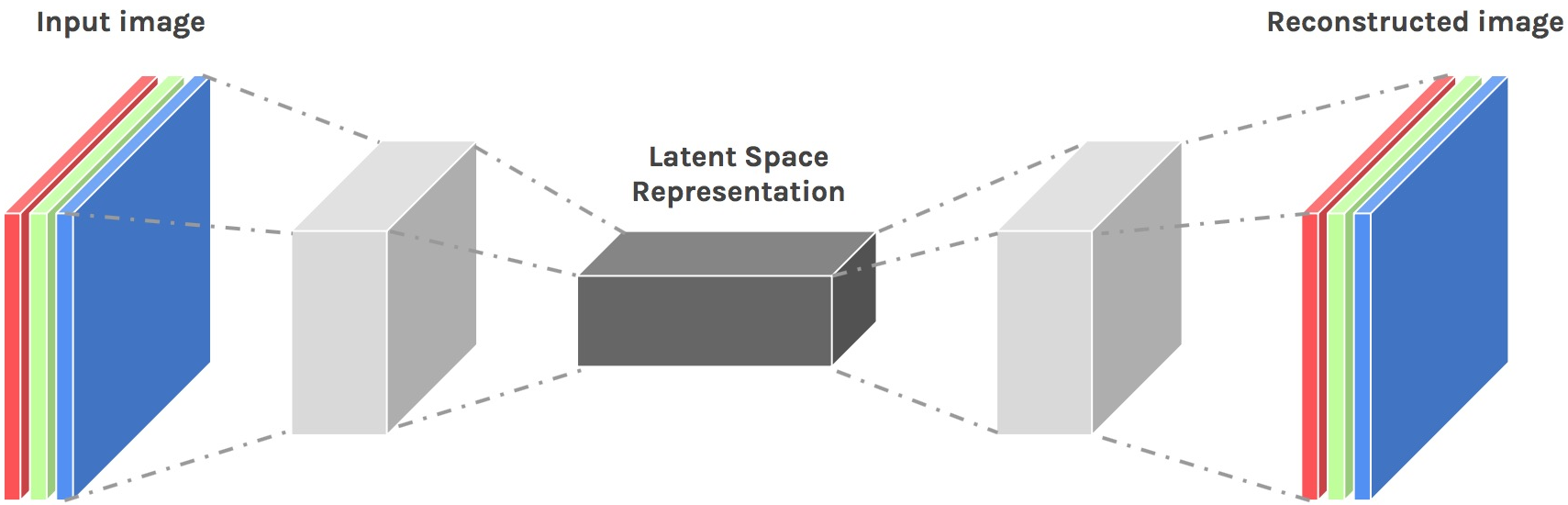
Denoising convolutional AE – keras

- 50 epochs.
- Noise factor 0.5
- 92% accuracy on validation set.



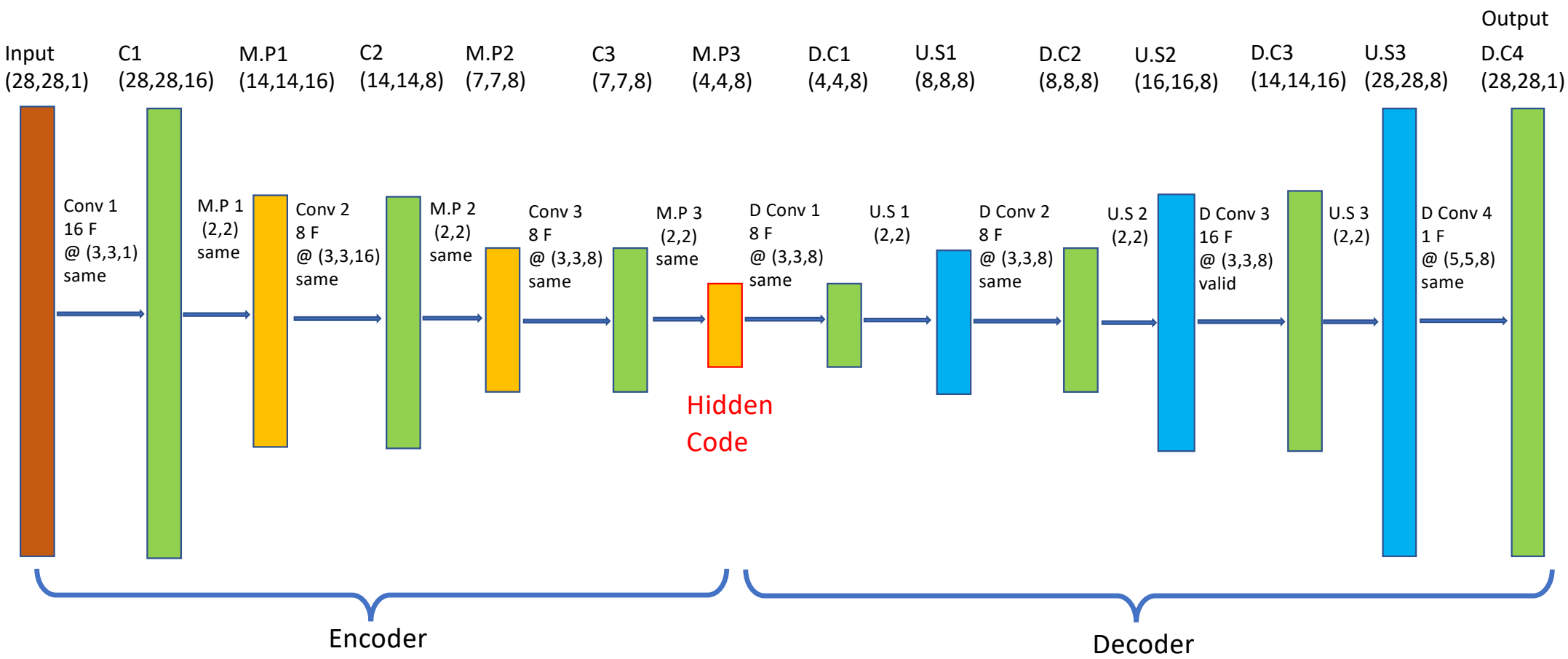

NOISY
INPUT
DENOISED
OUTPUT
COMPRESSION/
CLEANING
DECOMPRESSION

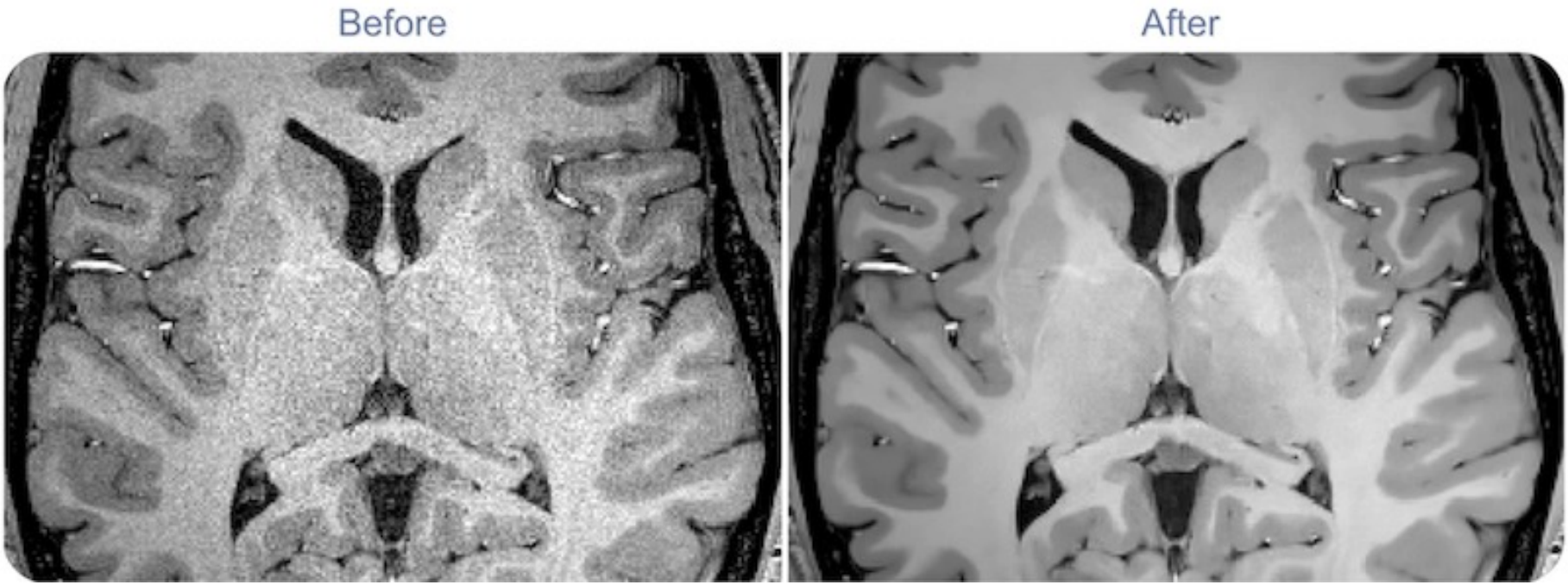
Convolutional AE

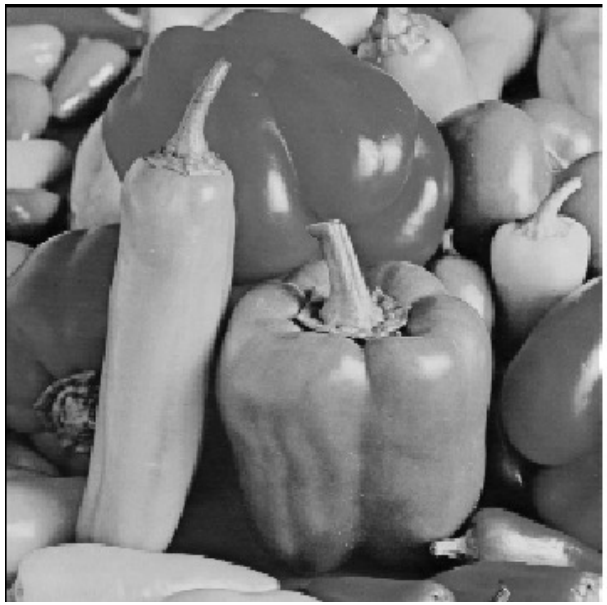


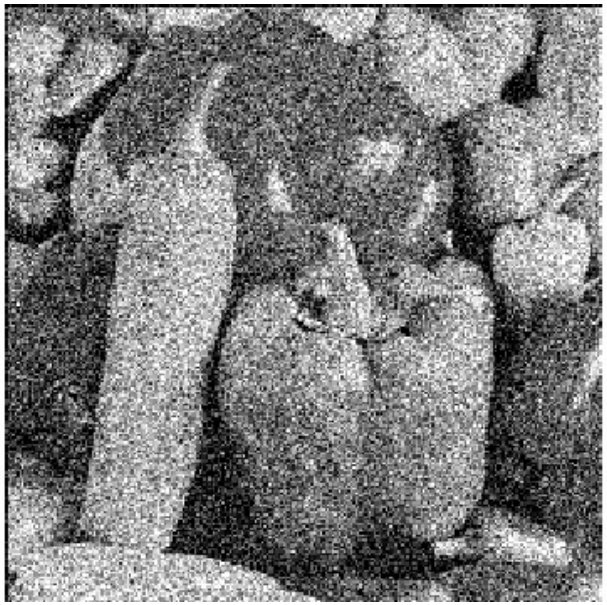
Convolutional AE

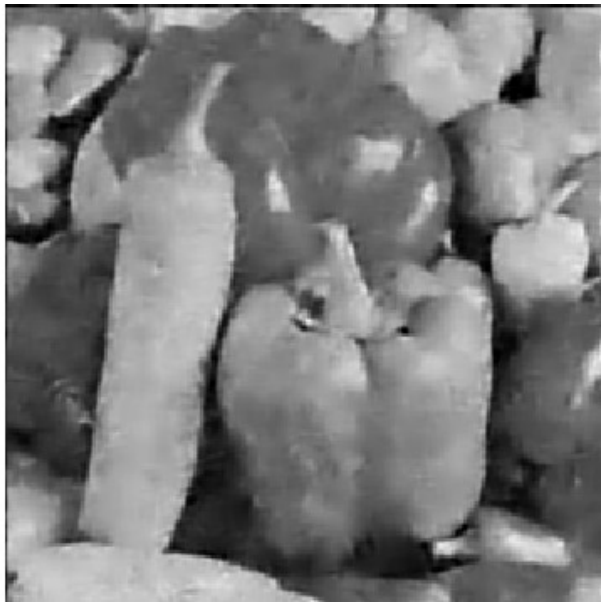
* Input values are normalized
* All of the conv layers activation functions are relu except for the last conv which is sigm

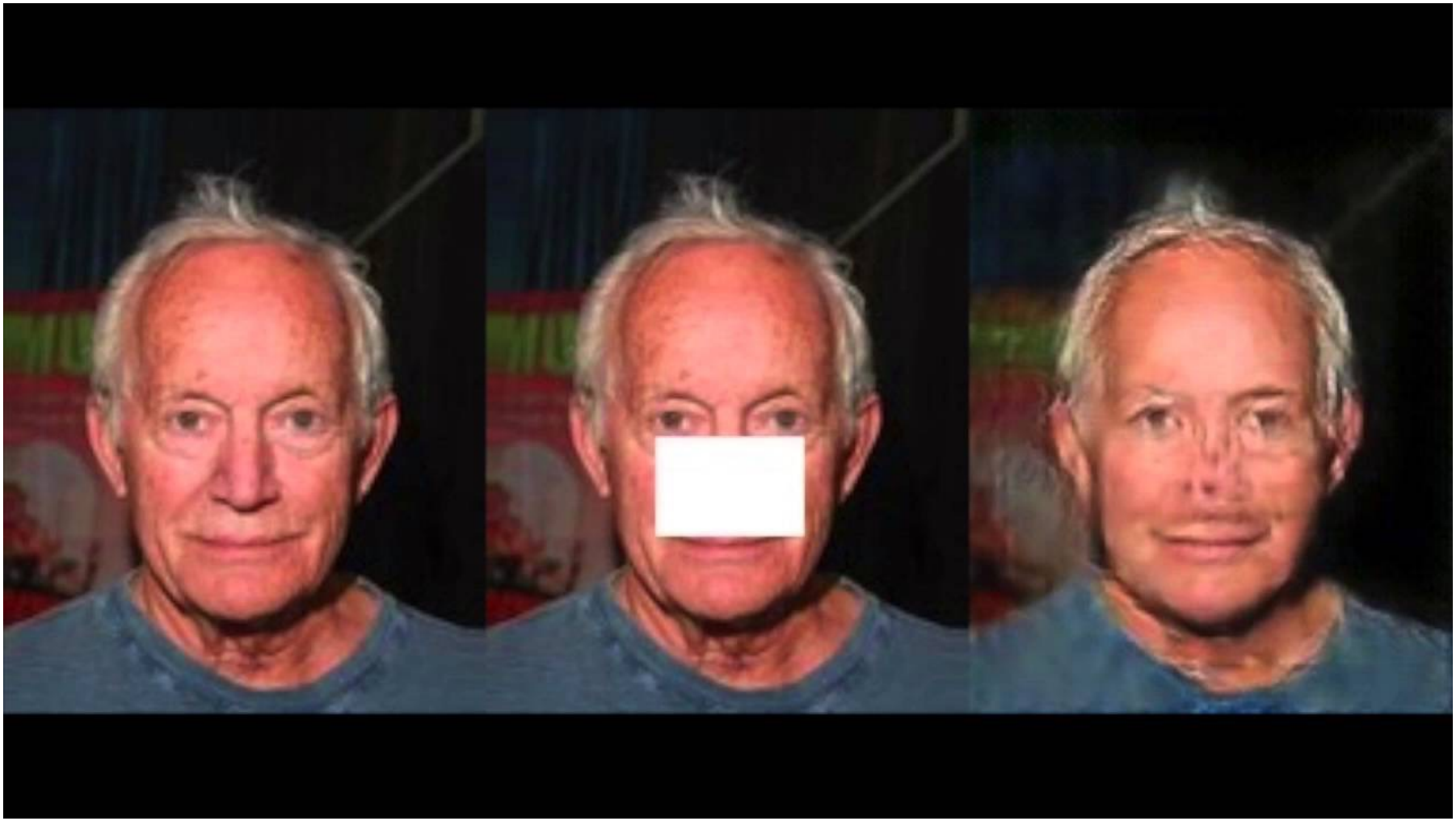



Before
After

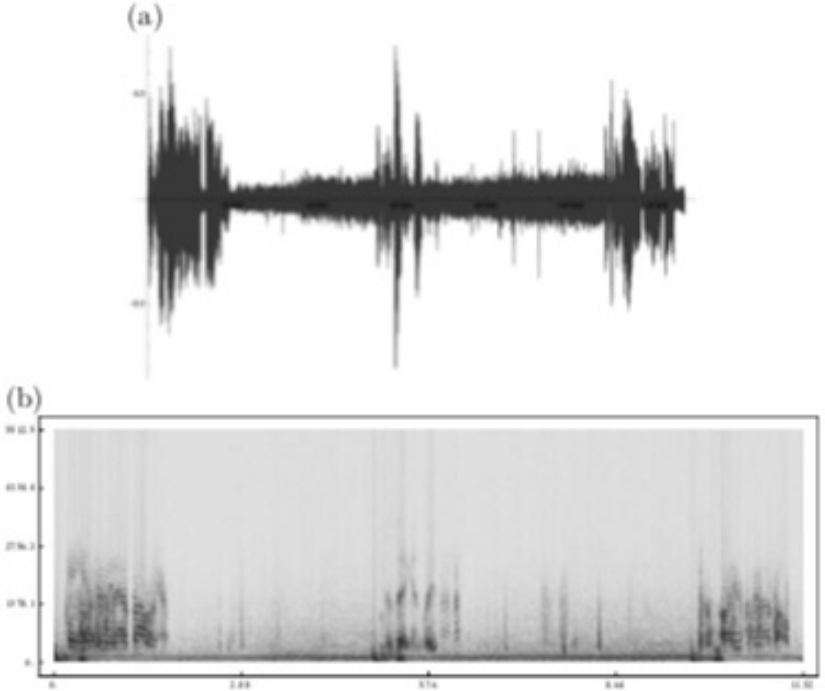


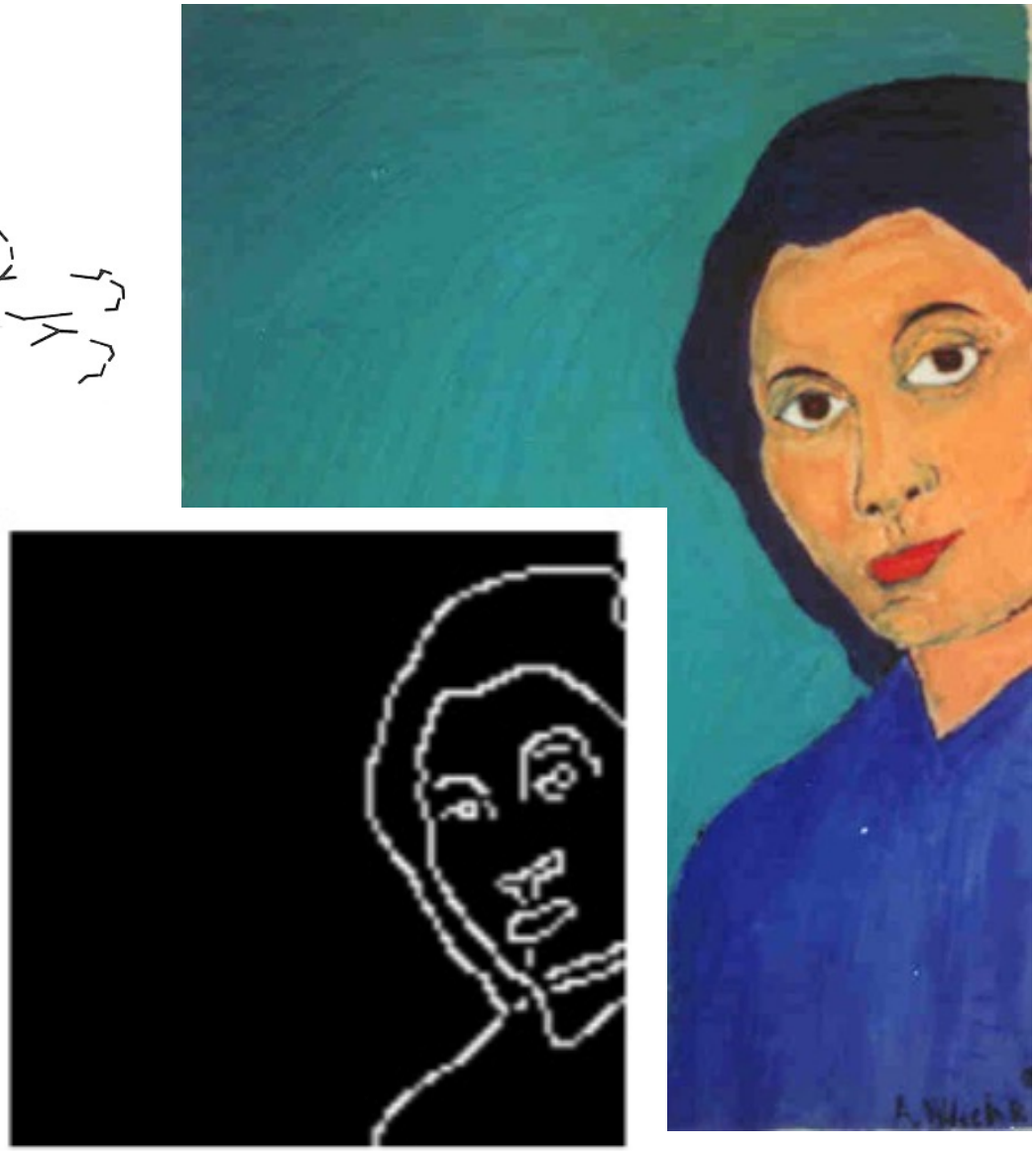




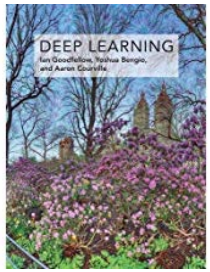


- Next: *Feature Extraction*

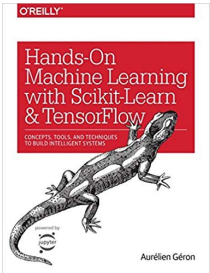




Literature

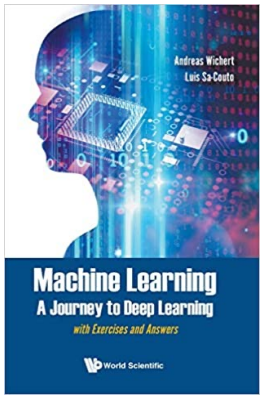


- Deep Learning, I. Goodfellow, Y. Bengio, A. Courville MIT Press 2016
  - Chapter 14



- Hands-On Machine Learning with Scikit-Learn and TensorFlow, Aurélien Géron , O'Reilly Media; 1 edition, 2017
  - Chapter 15

Literature



- Machine Learning - A Journey to Deep Learning, A. Wichert, Luis Sa-Couto, World Scientific, 2021
  - Chapter 15