Structure learning

with deep neuronal networks

6th Network Modeling Workshop, 6/6/2013

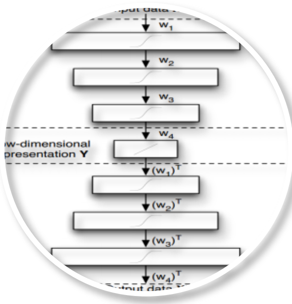
Patrick Michl

RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG

BioQuant MODEL base of LIFE

dkfz. GERMAN CANCER RESEARCH CENTER IN THE HELMHOLTZ ASSOCIATION

Agenda



Autoencoders

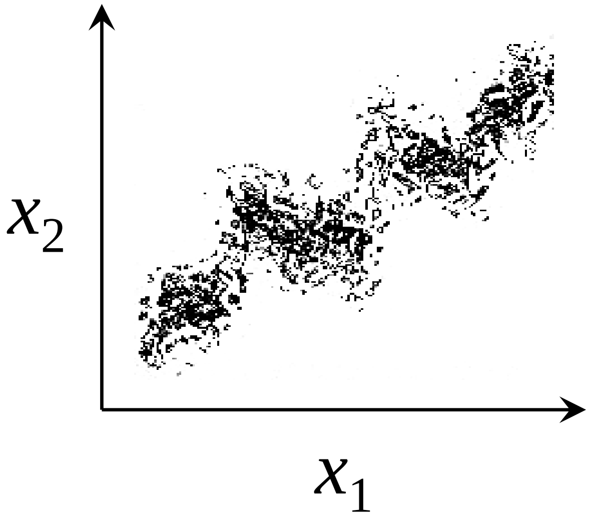
Biological Model

Validation & Implementation

Autoencoders

Dataset

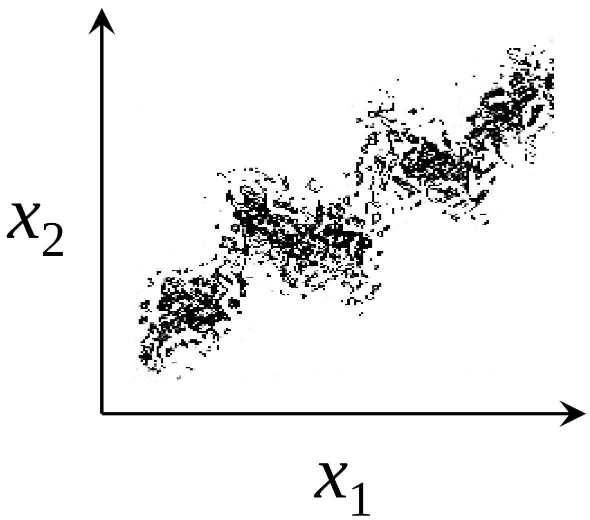
Model



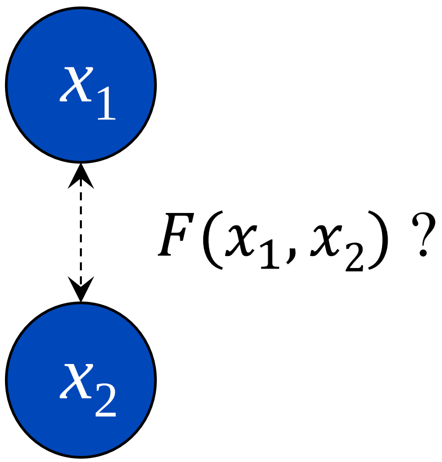
Real world data usually is **high dimensional** …

Autoencoders

Dataset



Model

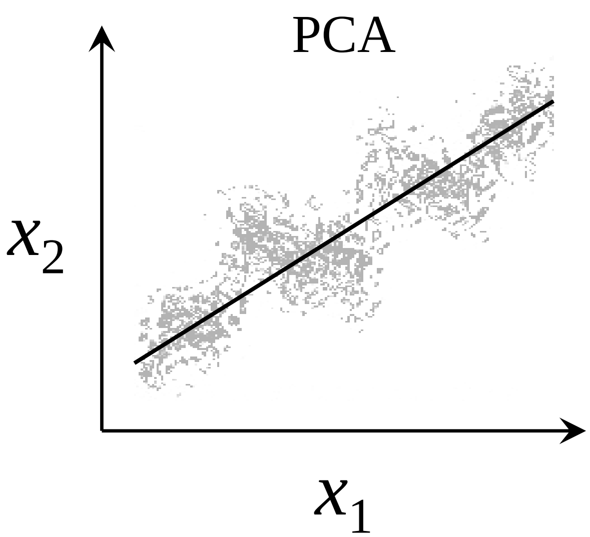


… which makes **structural analysis** and modeling complicated!

Autoencoders

Dataset

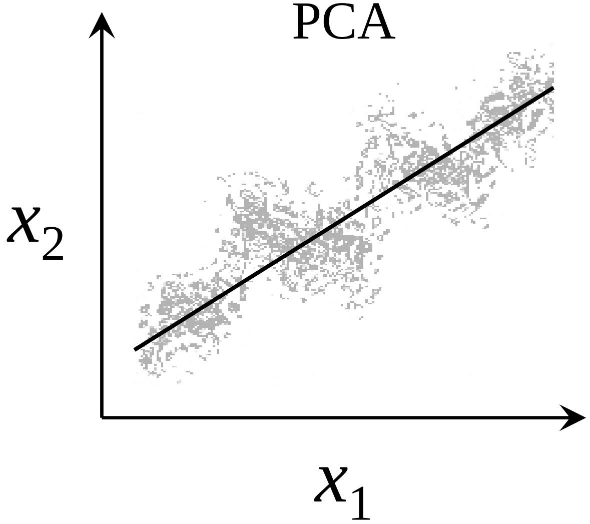
Model



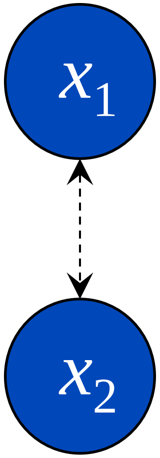
Dimensionality reduction techinques like **PCA** …

Autoencoders

Dataset



Model



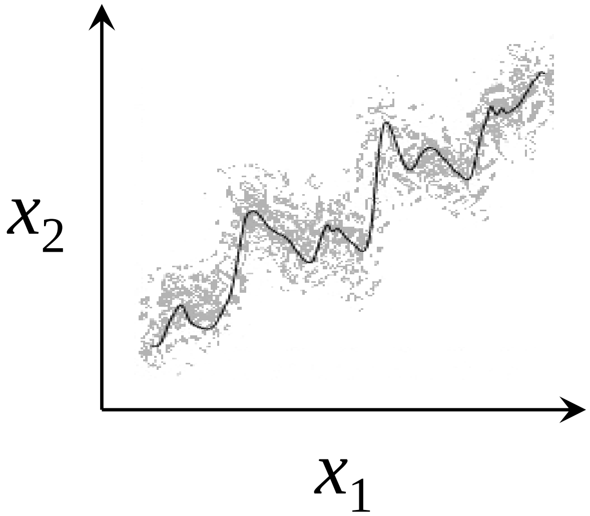
$$x_2 = \alpha x_1 + \beta$$

… can not preserve **complex structures**!

Autoencoders

Dataset

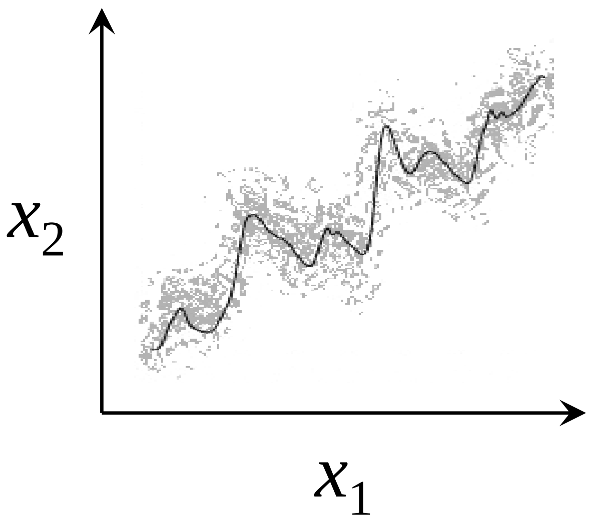
Model



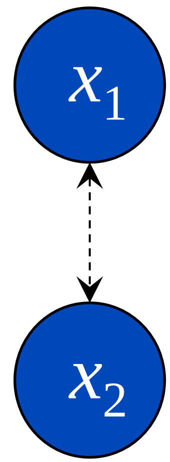
Therefore the analysis of **unknown structures** …

Autoencoders

Dataset



Model



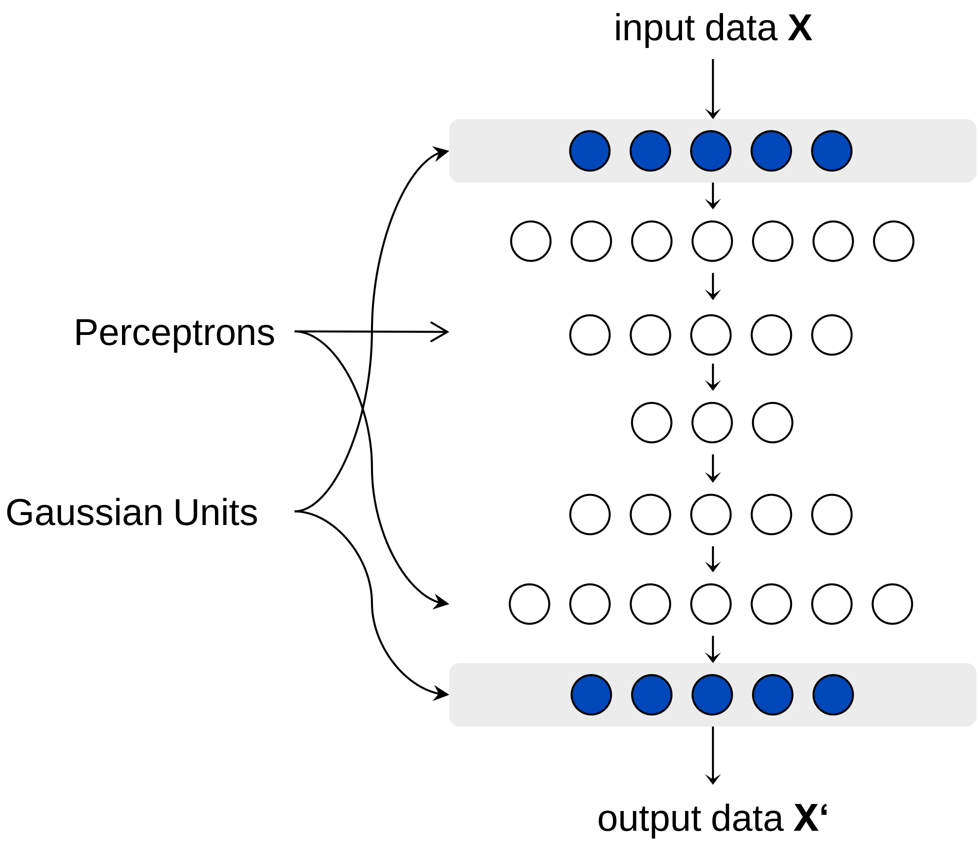
$$x_2 = f(x_1)$$

… needs more considerate **nonlinear techniques**!

Autoencoders

Autoencoder

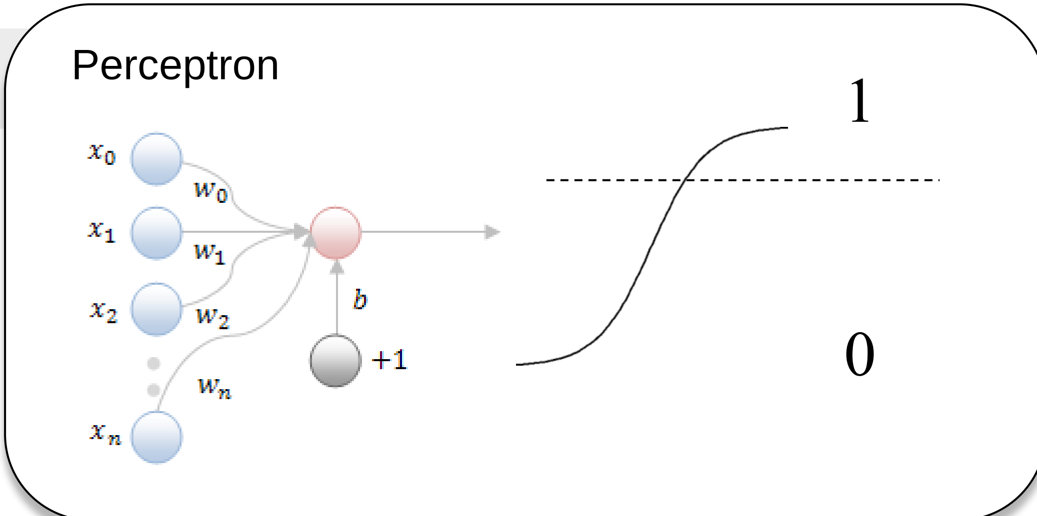
- Artificial Neuronal Network

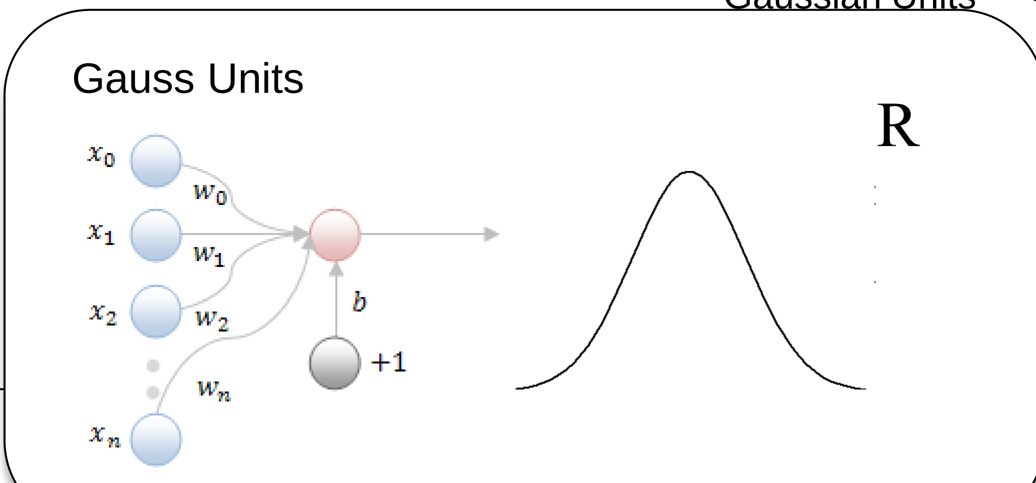


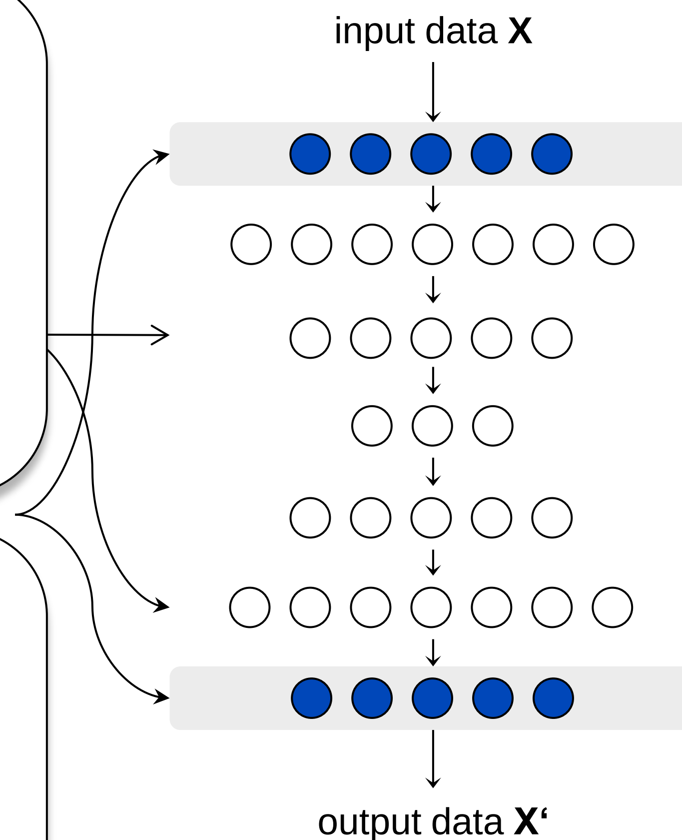
Autoencoders are **artificial neuronal networks** …

Autoencoders

•





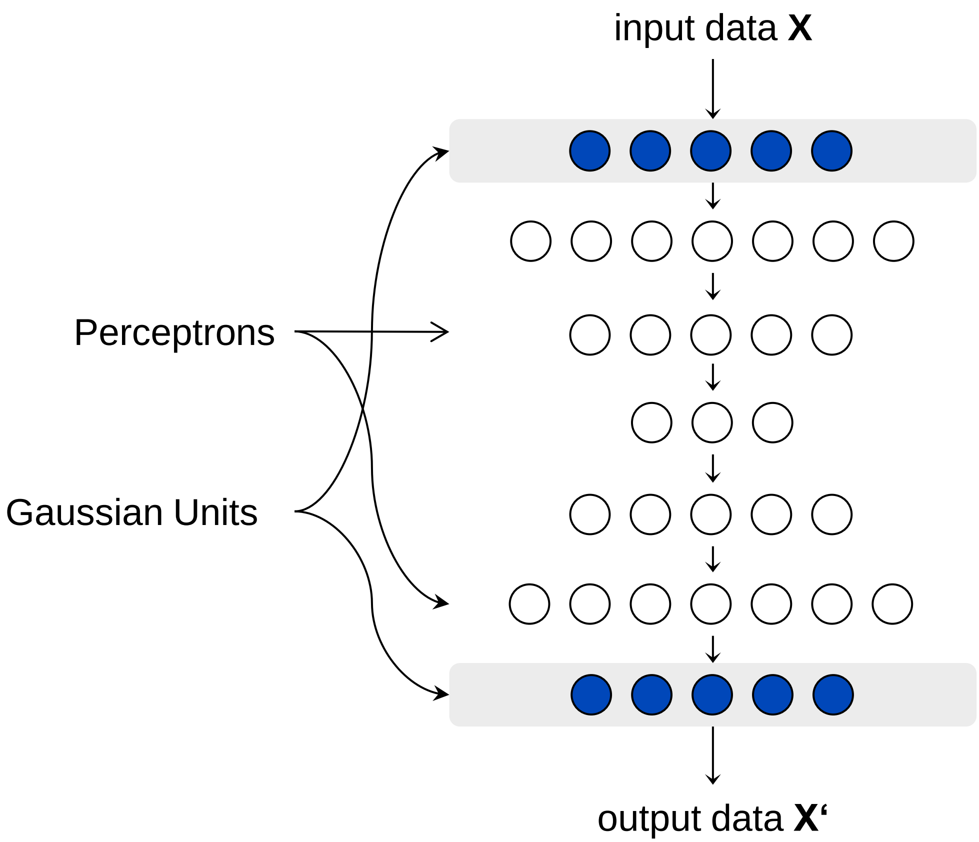


…nal networks …

Autoencoders

Autoencoder

- Artificial Neuronal Network

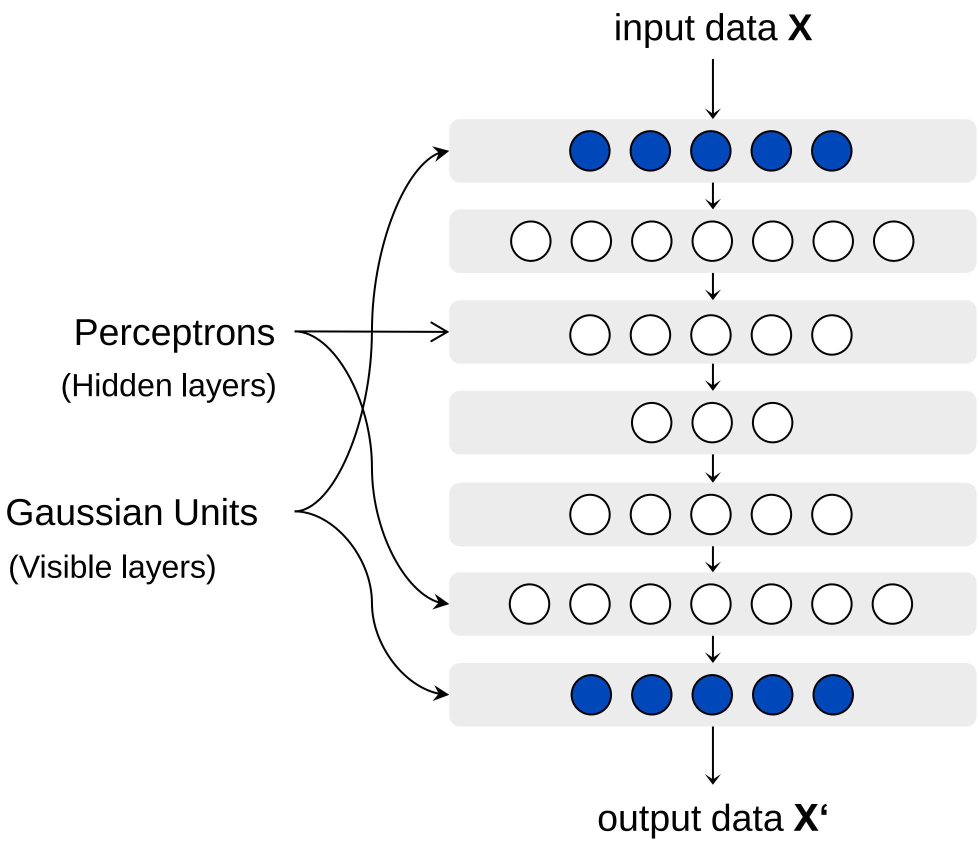


Autoencoders are **artificial neuronal networks** …

Autoencoders

Autoencoder

- Artificial Neuronal Network
- Multiple hidden layers

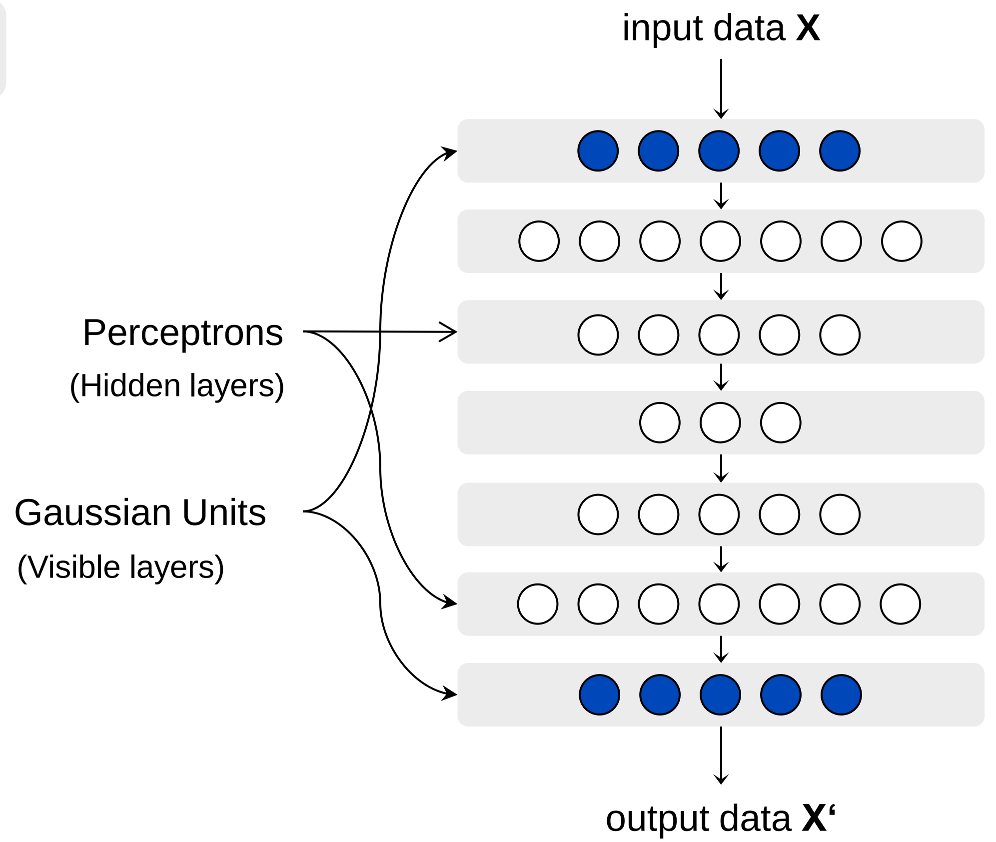


… with **multiple hidden layers**.

Autoencoders

Autoencoder

- Artificial Neuronal Network
- Multiple hidden layers

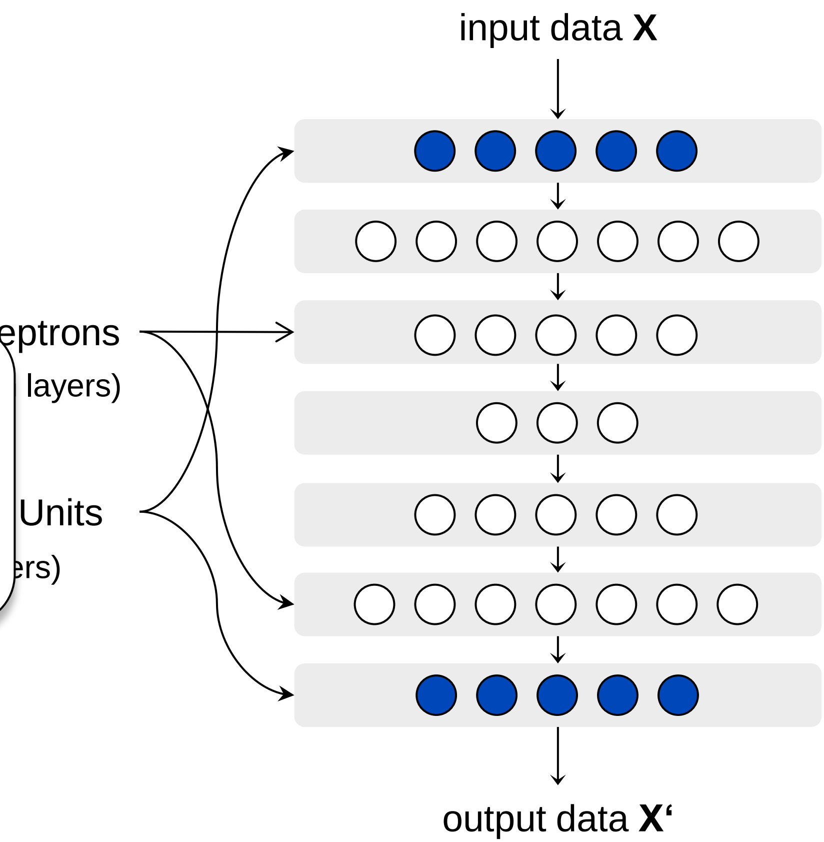


Such networks are called **deep networks**.

Autoencoders

Autoencoder

- Artificial Neuronal Network
- Multiple hidden layers



Definition (*deep network*)

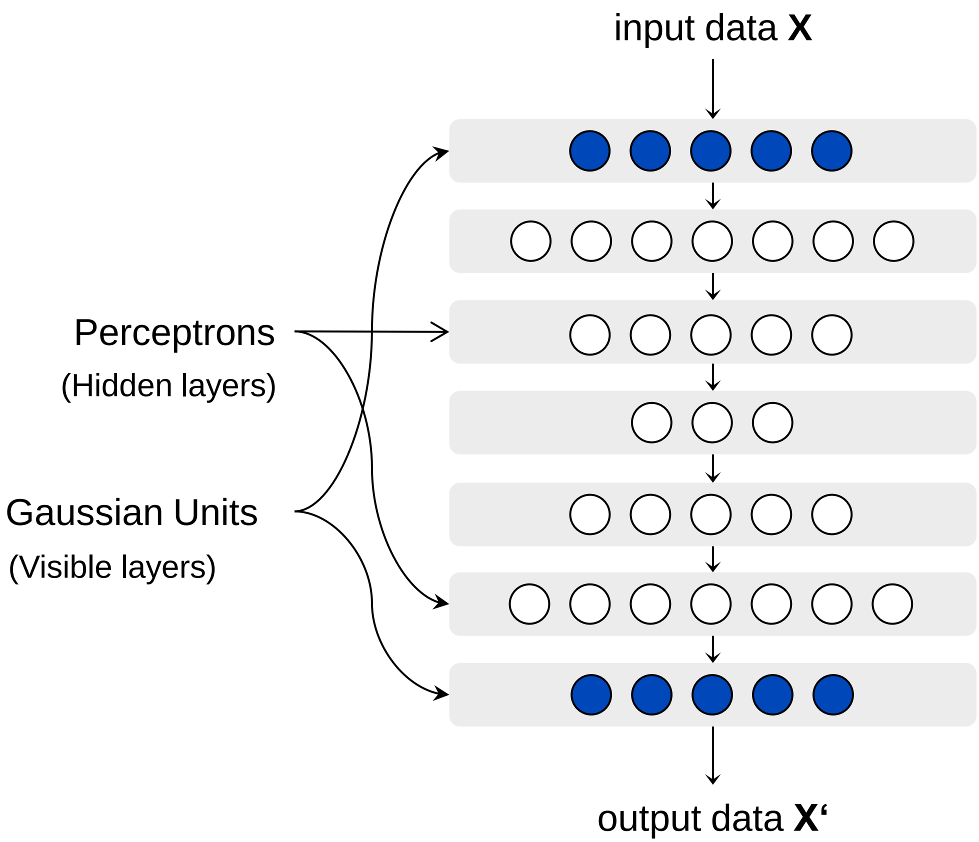
Deep networks are artificial neuronal networks with multiple hidden layers

Such networks are called **deep networks**.

Autoencoders

Autoencoder

- Deep network

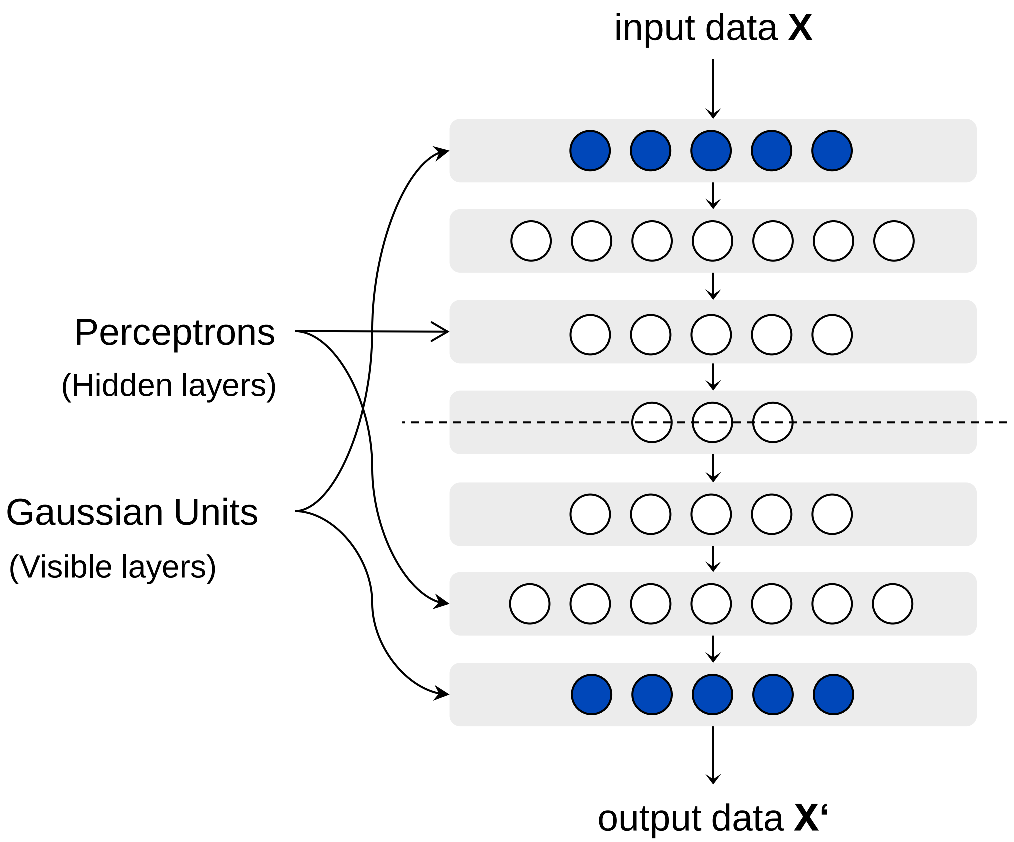


Such networks are called **deep networks**.

Autoencoders

Autoencoder

- Deep network
- Symmetric topology

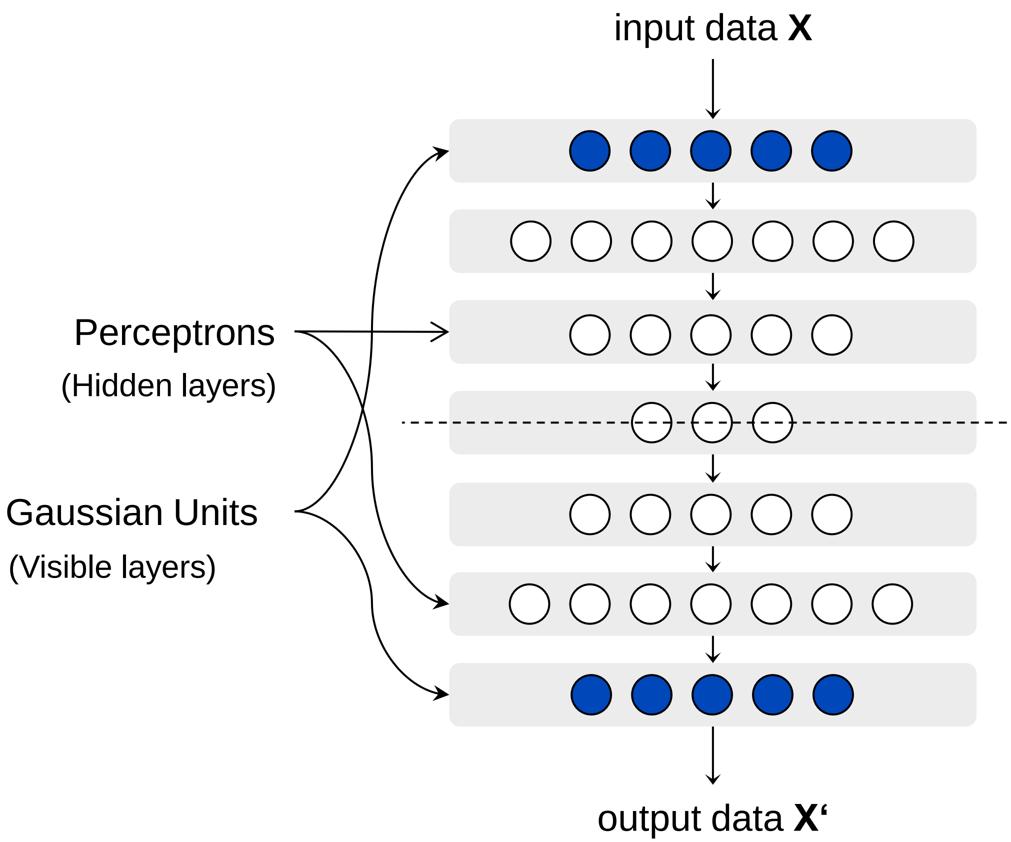


Autoencoders have a **symmetric topology** …

Autoencoders

Autoencoder

- Deep network
- Symmetric topology

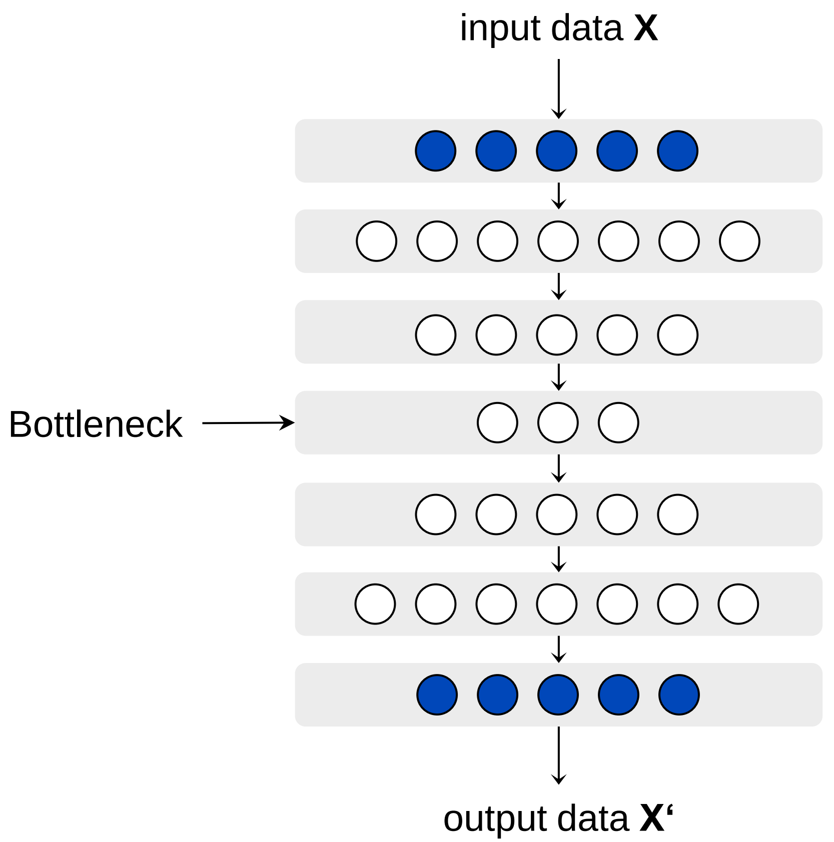


… with an **odd number** of hidden layers.

Autoencoders

Autoencoder

- Deep network
- Symmetric topology
- Information bottleneck

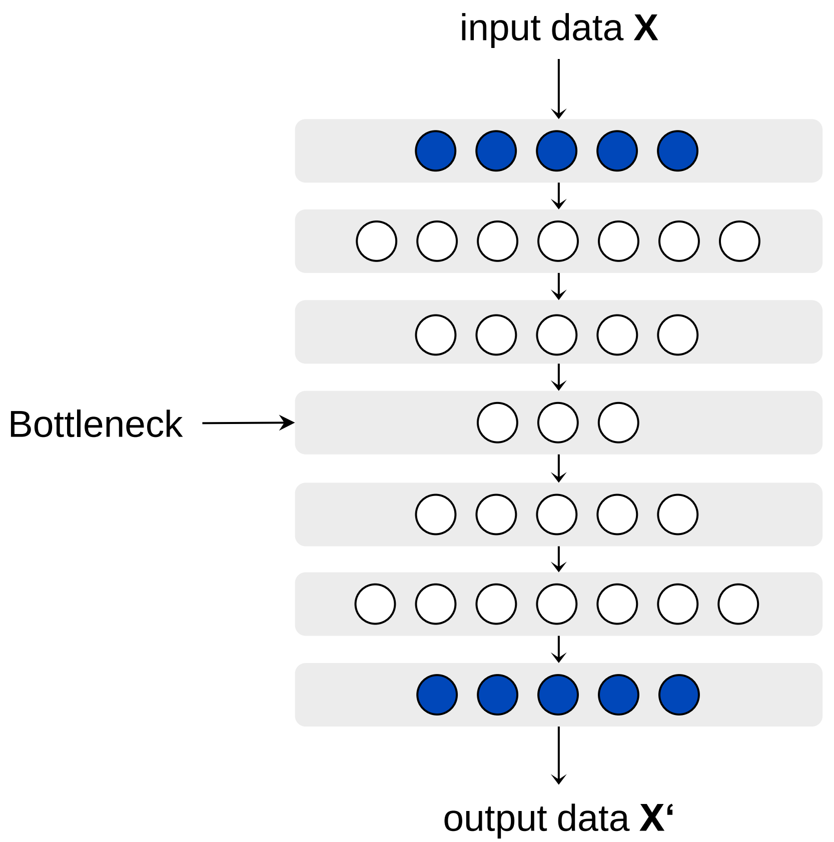


The small layer in the center works lika an **information bottleneck**

Autoencoders

Autoencoder

- Deep network
- Symmetric topology
- Information bottleneck

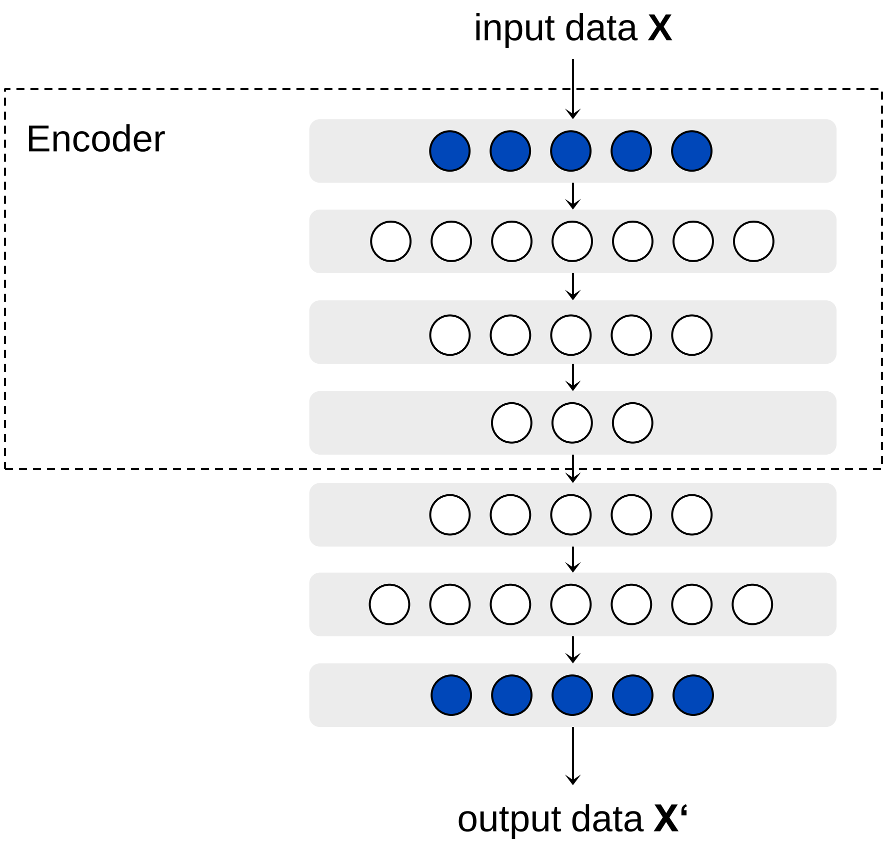


... that creates a **low dimensional code** for each sample in the input data.

Autoencoders

Autoencoder

- Deep network
- Symmetric topology
- Information bottleneck
- Encoder

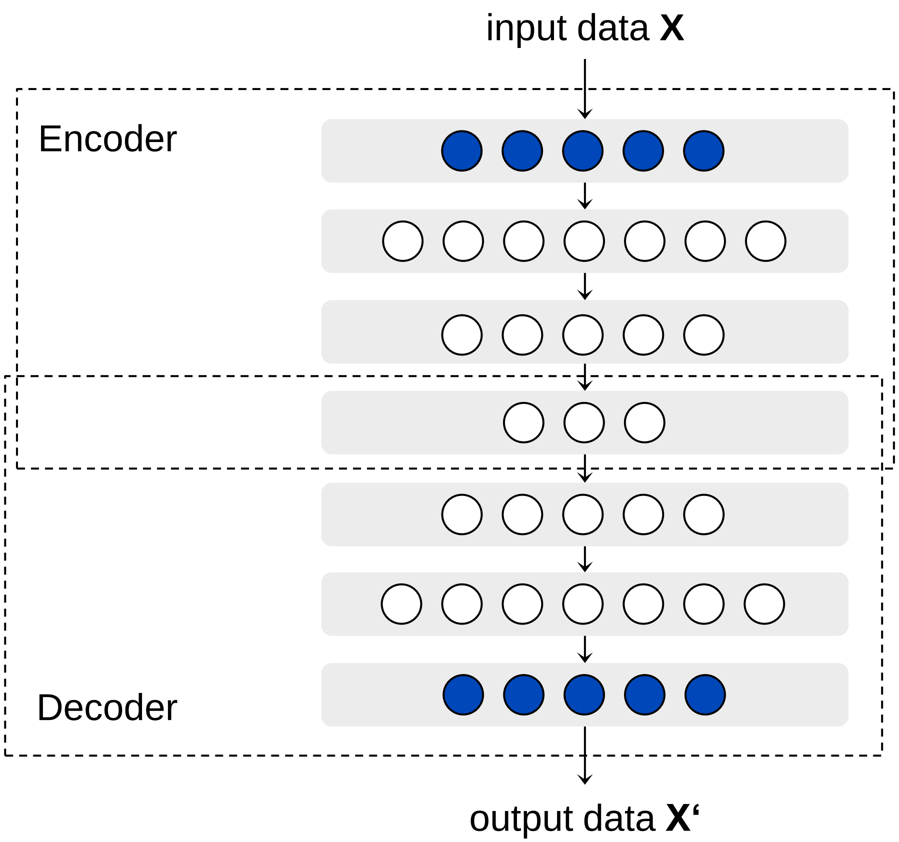


The upper stack does the **encoding** …

Autoencoders

Autoencoder

- Deep network
- Symmetric topology
- Information bottleneck
- Encoder
- Decoder



… and the lower stack does the **decoding**.

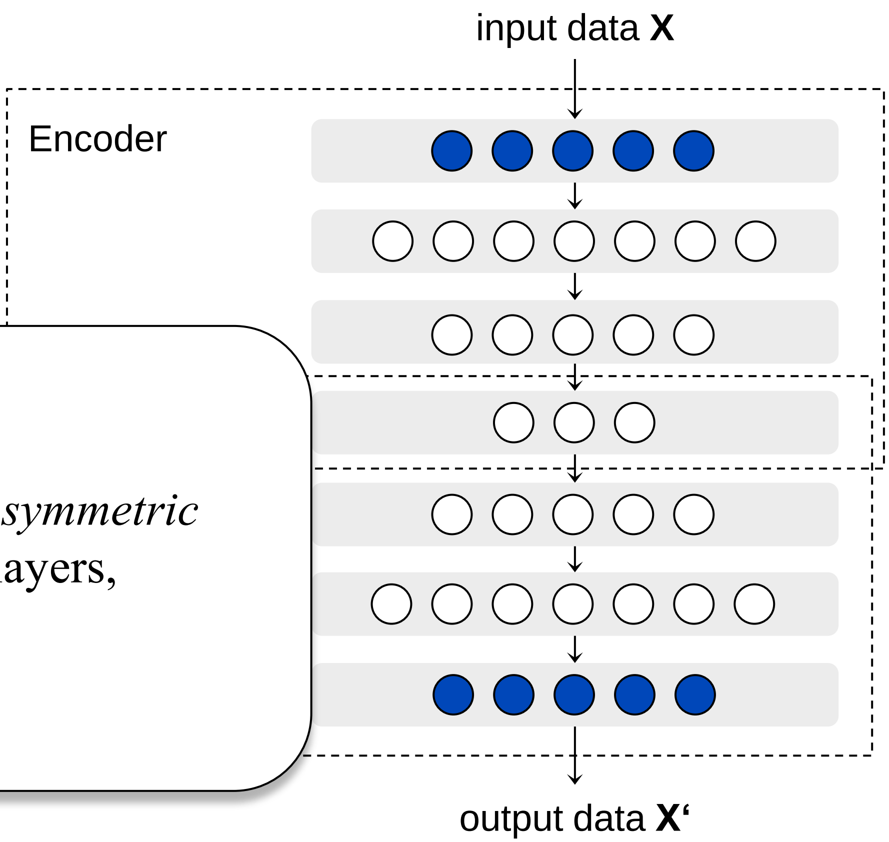
Autoencoders

Autoencoder

- Deep network
- Symmetric topology
- Information bottleneck
-
-

Definition (*autoencoder*)

Autoencoders are *deep networks* with a *symmetric topology* and an odd number of hiddern layers, containing a *encoder*, a low dimensional representation and a *decoder*.



… and the lower stack does the **decoding**.

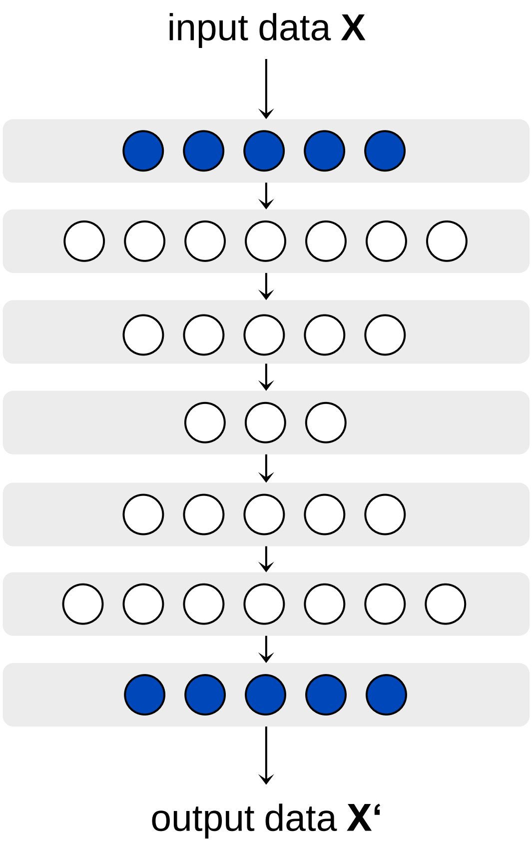
Autoencoders

Autoencoder

Problem: dimensionality of data

Idea:

1. Train autoencoder to minimize the distance between input **X** and output **X‘**
2. Encode **X** to low dimensional code **Y**
3. Decode low dimensional code **Y** to output **X‘**
4. Output **X‘** is low dimensional



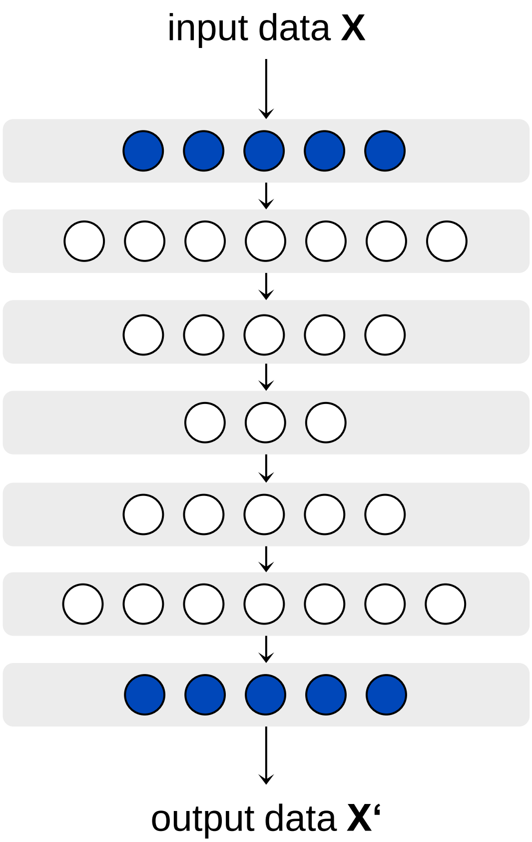
Autoencoders can be used to **reduce the dimension of data** …

Autoencoders

Autoencoder

Problem: dimensionality of data

Idea:
1. Train autoencoder to minimize the distance between input **X** and output **X‘**
2. Encode **X** to low dimensional code **Y**
3. Decode low dimensional code **Y** to output **X‘**
4. Output **X‘** is low dimensional



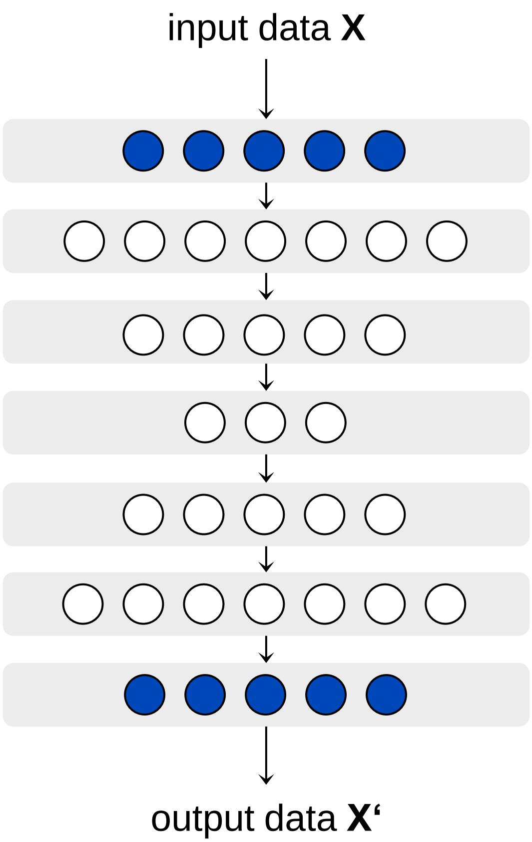
… if we can train them!

Autoencoders

Autoencoder

Training

Backpropagation

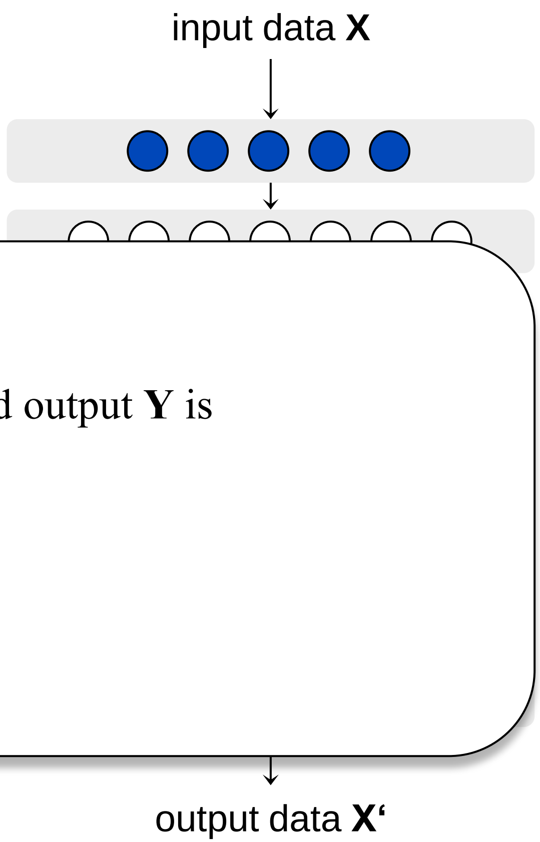


In feedforward ANNs **backpropagation** is a good approach.

Autoencoders

Autoencoder

Training



Backpropagation

(1) The distance (error) between current output **X‘** and wanted output **Y** is computed. This gives a error function

$$X' = F(X)$$
$$\text{error} = \sqrt{X'^2 - Y}$$

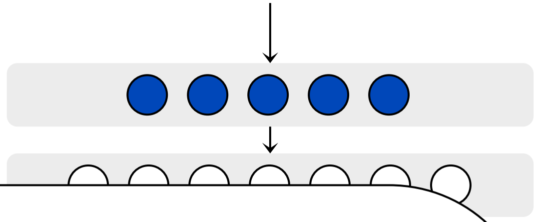
In feedforward ANNs **backpropagation** is a good approach.

Autoencoders

Autoencoder

Training

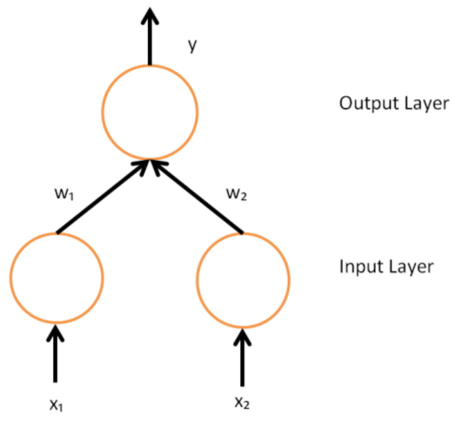
input data **X**

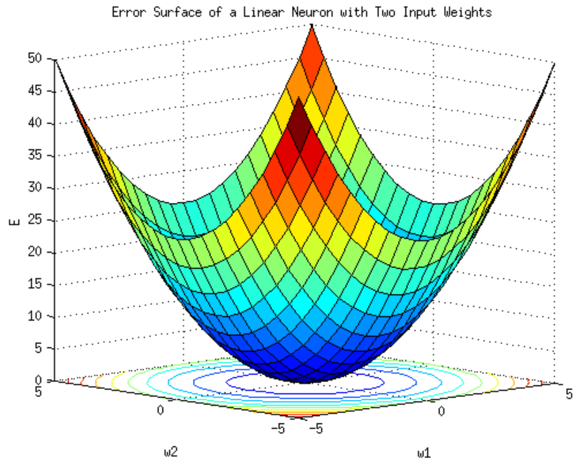


Backpropagation

(1) The distance (error) between current output **X'** and wanted output **Y** is computed. This gives a error function

Example (*linear neuronal unit with two inputs*)

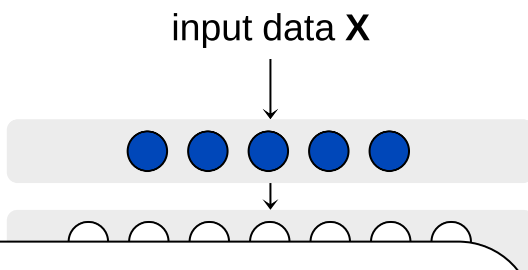




Autoencoders

Autoencoder

Training

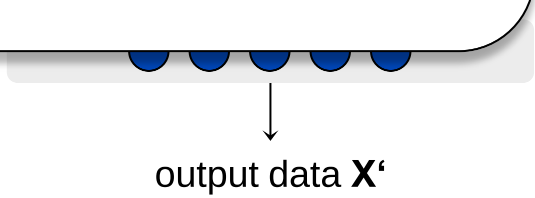


Backpropagation

(1) The distance (error) between current output **X‘** and wanted output **Y** is computed. This gives a error function

(2) By calculating $-\nabla error$ we get a vector that shows in a direction which decreases the error

(3) We update the parameters to decrease the error

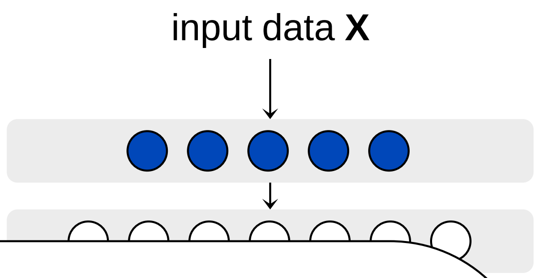


In feedforward ANNs **backpropagation** is a good approach.

Autoencoders

Autoencoder

Training



Ba

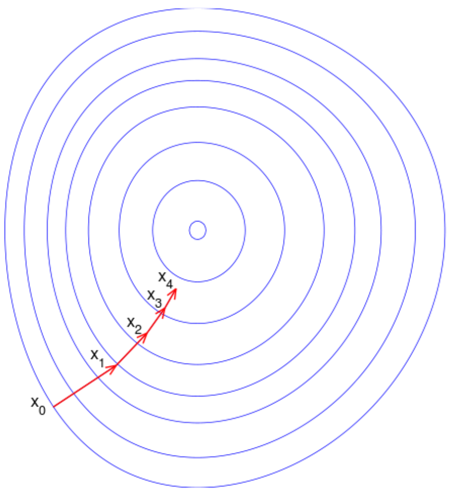
Backpropagation

(1) The distance (error) between current output **X'** and wanted output **Y** is computed. This gives a error function

(2) By calculating $-\nabla error$ we get a vector that shows in a direction which decreases the error

(3) We update the parameters to decrease the error

(4) We repeat that



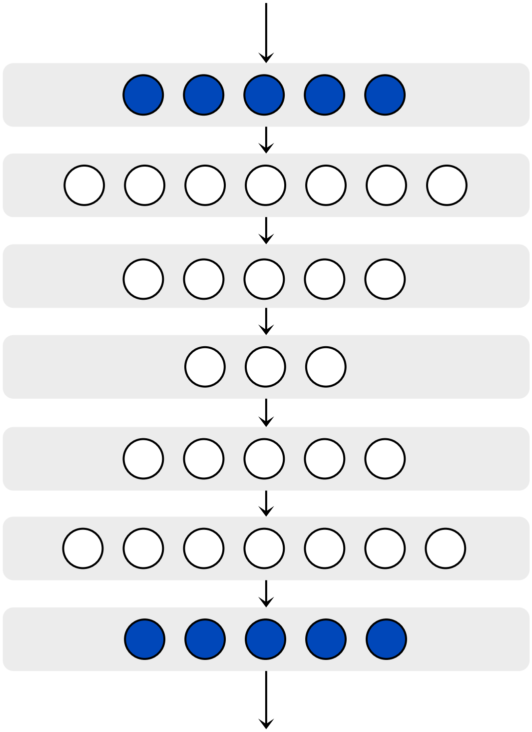
Autoencoders

Autoencoder

Training

Backpropagation

Problem: Deep Network



… the problem are the multiple hidden layers!

Autoencoders

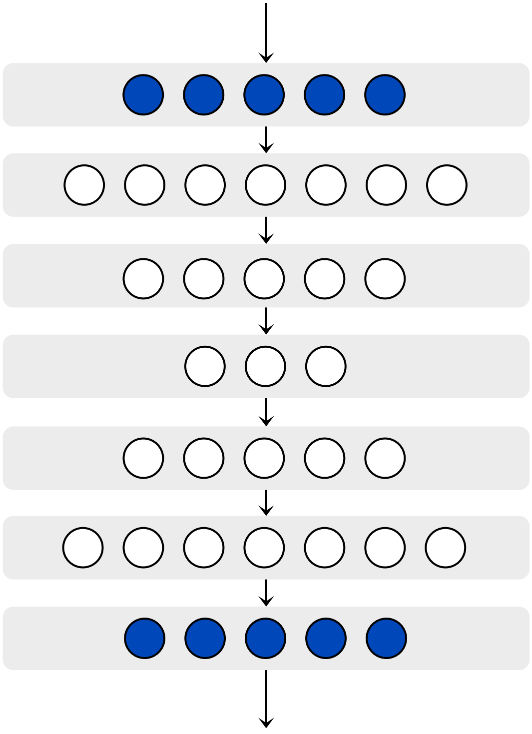
Autoencoder

Training

Backpropagation

Problem: Deep Network

- Very slow training



Backpropagation is known to be slow far away from the output layer …

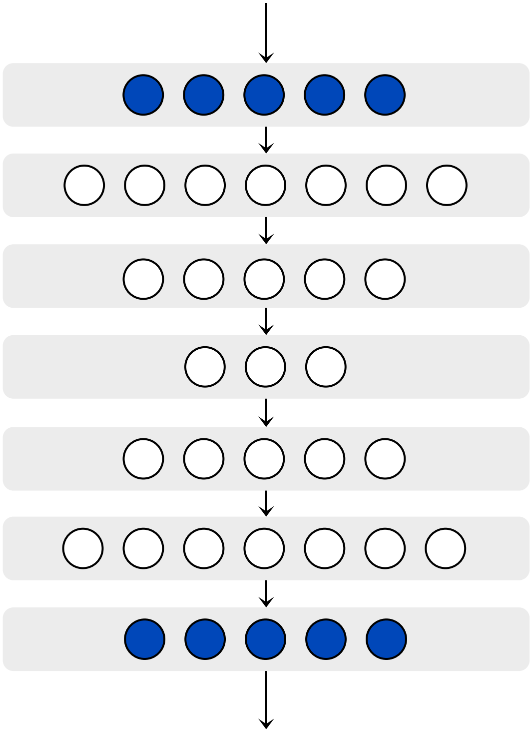
Autoencoders

Autoencoder

Training

Backpropagation

Problem: Deep Network
- Very slow training
- Maybe bad solution



… and can converge to poor **local minima**.

Autoencoders

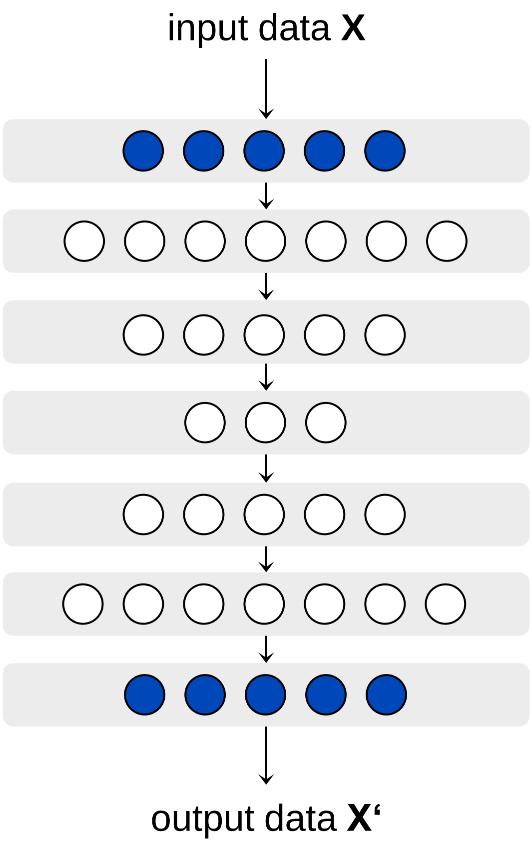
Autoencoder

Training

Backpropagation

Problem: Deep Network
- Very slow training
- Maybe bad solution

Idea: Initialize close to a good solution



The task is to **initialize the parameters** close to a good solution!

Autoencoders

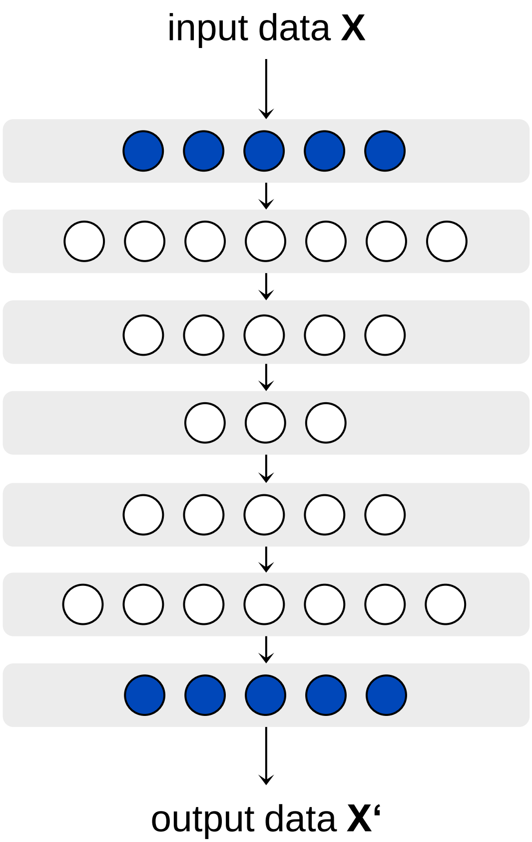
Autoencoder

Training

Backpropagation

Problem: Deep Network
- Very slow training
- Maybe bad solution

Idea: Initialize close to a good solution
- Pretraining



Therefore the training of autoencoders has a **pretraining** phase …

Autoencoders

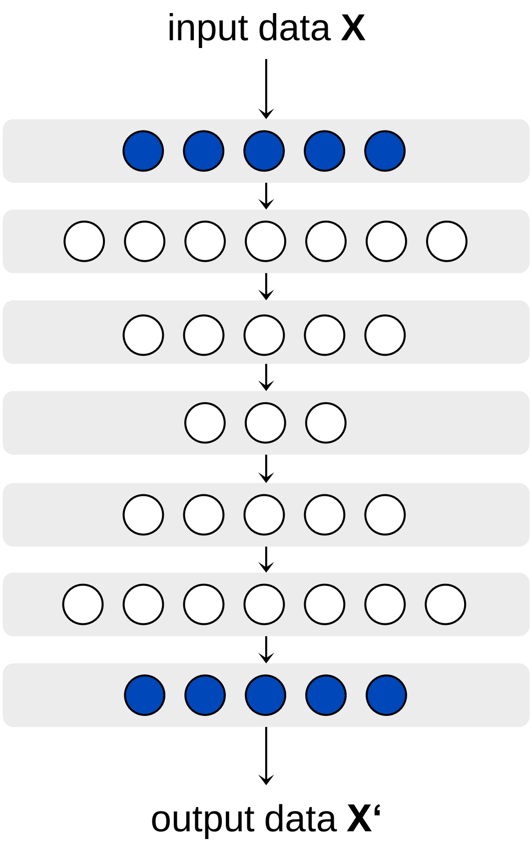
Autoencoder

Training

Backpropagation

Problem: Deep Network
- Very slow training
- Maybe bad solution

Idea: Initialize close to a good solution
- Pretraining
- Restricted Boltzmann Machines



… which uses **Restricted Boltzmann Machines** (RBMs)

Autoencoders

Autoencoder

input data **X**

Ba

Pr

•

•

Id

•

•

Restricted Boltzmann Machine

- RBMs are **Markov Random Fields**

Autoencoders

Autoencoder

input data **X**

Ba

Pr

•

•

Id

•

•

Restricted Boltzmann Machine

- RBMs are **Markov Random Fields**

Markov Random Field

Every unit influences every neighbor

The coupling is undirected

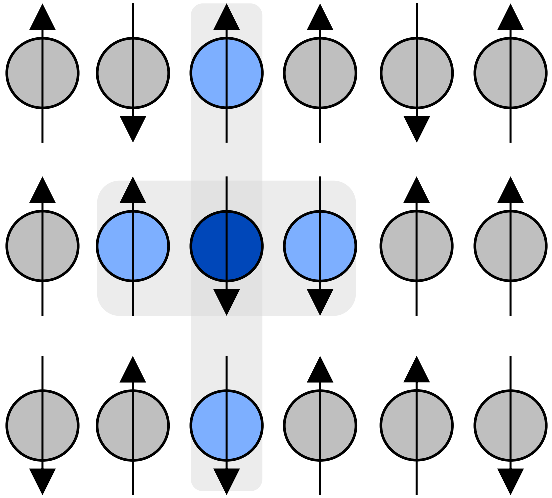
Motivation (Ising Model)

A set of magnetic dipoles (*spins*)

is arranged in a graph (lattice)

where neighbors are

coupled with a given strengt



Autoencoders

Autoencoder

input data **X**

Ba

Pr

-
-

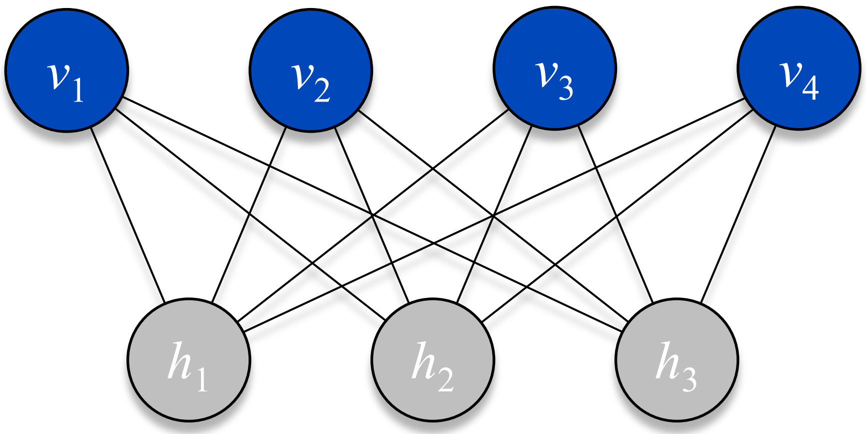
Id

-
-

Restricted Boltzmann Machine

- RBMs are **Markov Random Fields**
- Bipartite topology: **visible** (v), **hidden** (h)
- Use local **energy** to calculate the probabilities of values

Training:
contrastive divergency
(Gibbs Sampling)



Autoencoders

Autoencoder

input data **X**

Ba

Pr

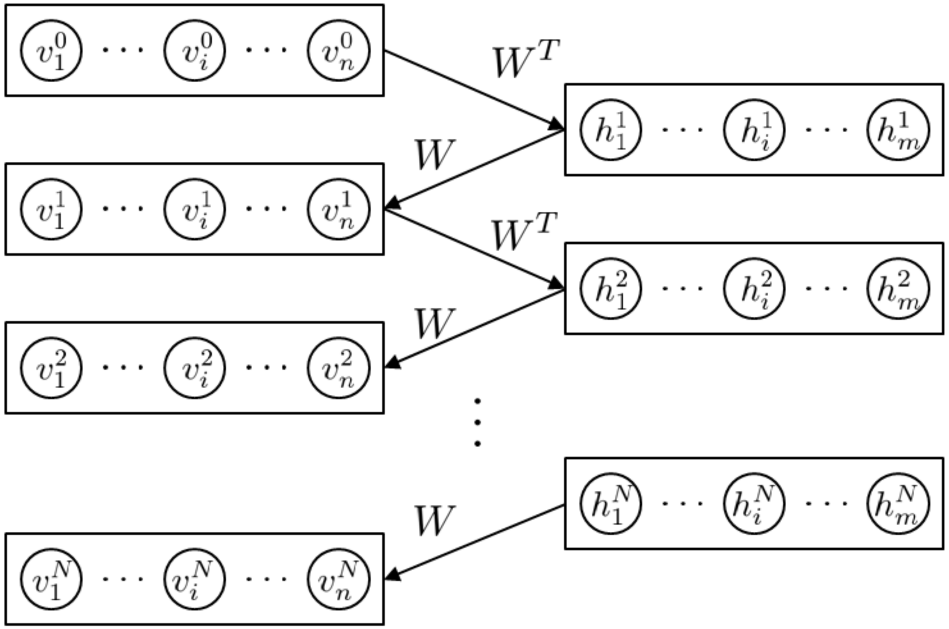
-
-

Id

-
-

Restricted Boltzmann Machine

Gibbs Sampling



Autoencoders

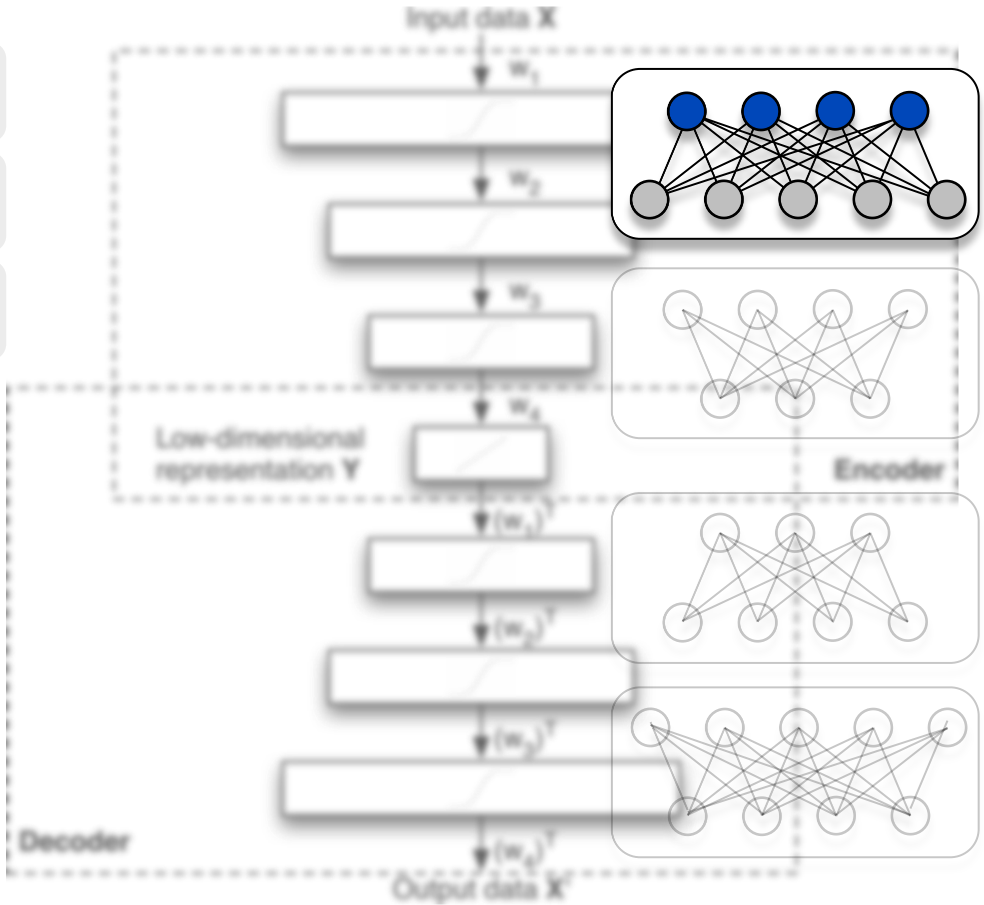
Autoencoder

Training

Top

V := set of visible units
x_v := value of unit $v, \forall v \in V$
$x_v \in \boldsymbol{R}, \forall v \in V$

H := set of hidden units
x_h := value of unit $h, \forall h \in H$
$x_h \in \{\mathbf{0}, \mathbf{1}\}, \forall h \in H$



The top layer RBM transforms **real value data** into binary codes.

Autoencoders

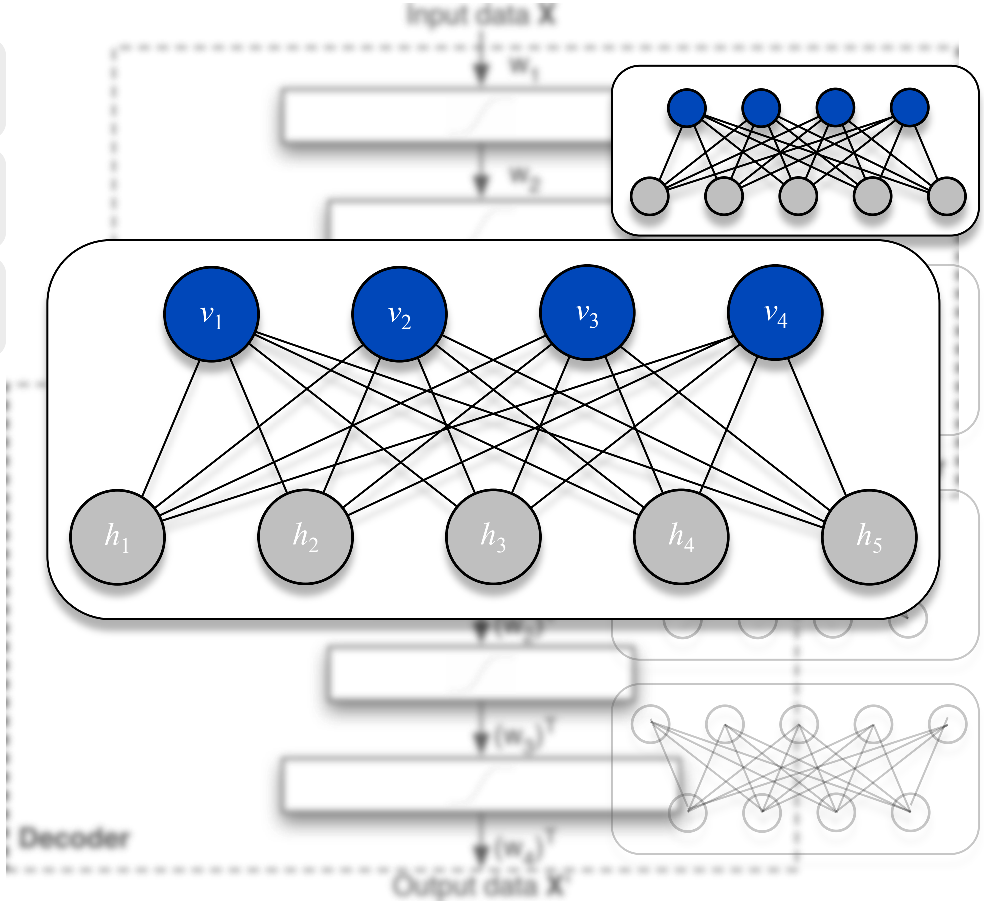
Autoencoder

Training

Top

$$x_v \sim N\left(b_v + \sum_h w_{vh}\, x_h, \sigma_v\right)$$

σ_v := std. dev. of unit v
b_v := bias of unit v
w_{vh} := weight of edge (v, h)

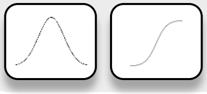


Therefore visible units are modeled with **gaussians** to encode **data** …

Autoencoders

Autoencoder

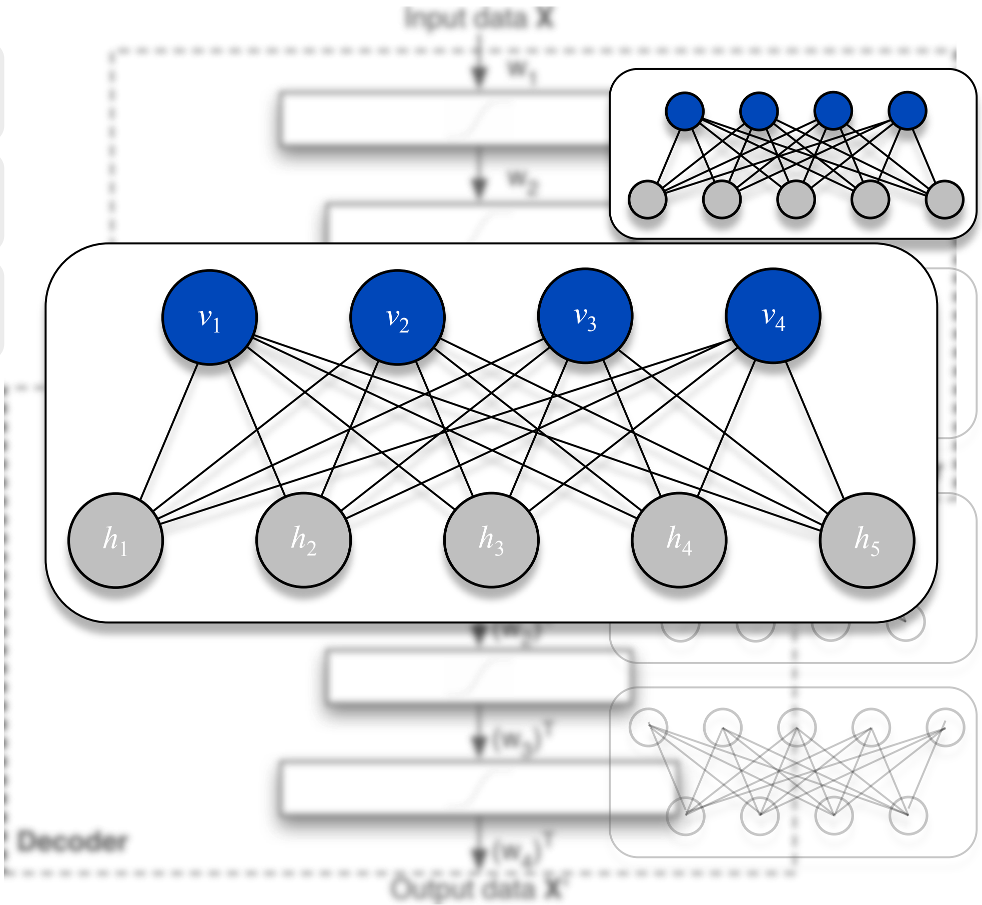
Training



Top

$$x_h \sim \mathrm{sigm}\left(b_h + \sum_v w_{vh} \frac{x_v}{\sigma_v}\right)$$

$\sigma_v :=$ std. dev. of unit v
$b_h :=$ bias of unit h
$w_{vh} :=$ weight of edge (v, h)



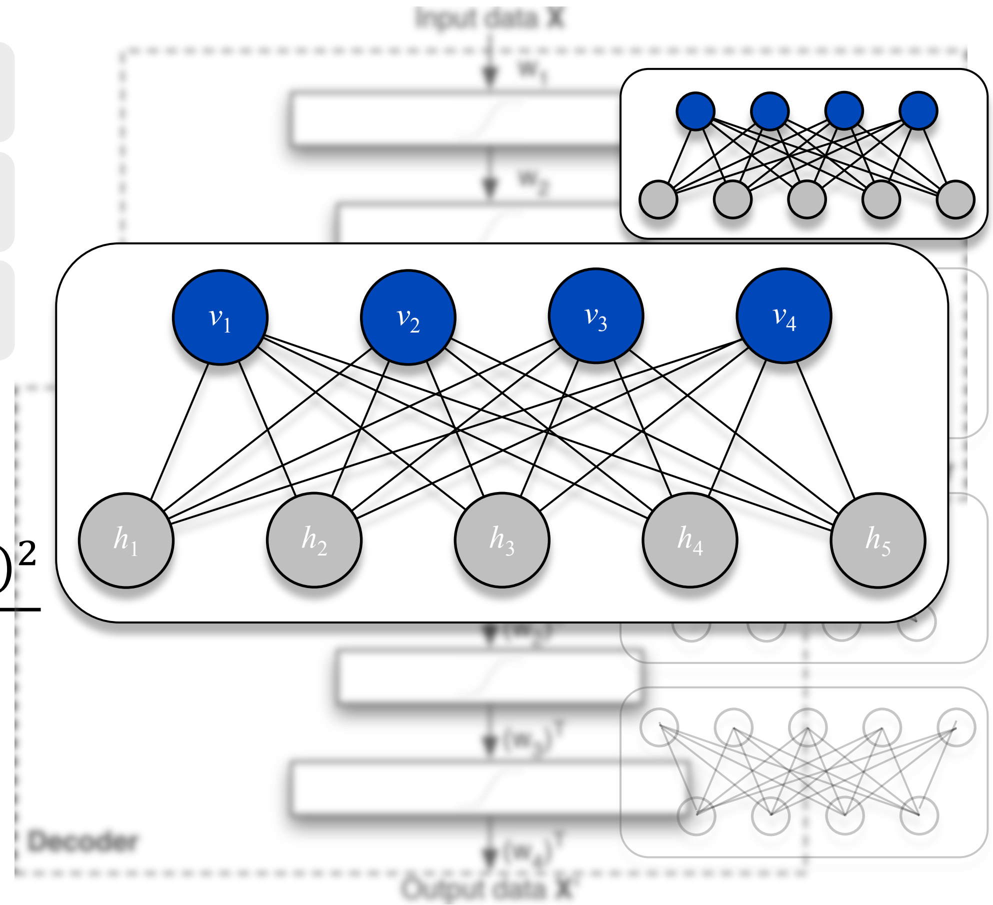
… and many hidden units with **simoids** to encode **dependencies**

Autoencoders

Autoencoder

Training

Top



Local Energy

$$E_v := -\sum_h w_{vh} \frac{x_v}{\sigma_v} x_h + \frac{(x_v - b_v)^2}{2{\sigma_v}^2}$$

$$E_h := -\sum_v w_{vh} \frac{x_v}{\sigma_v} x_h + x_h b_h$$

The **objective function** is the sum of the local energies.

Autoencoders

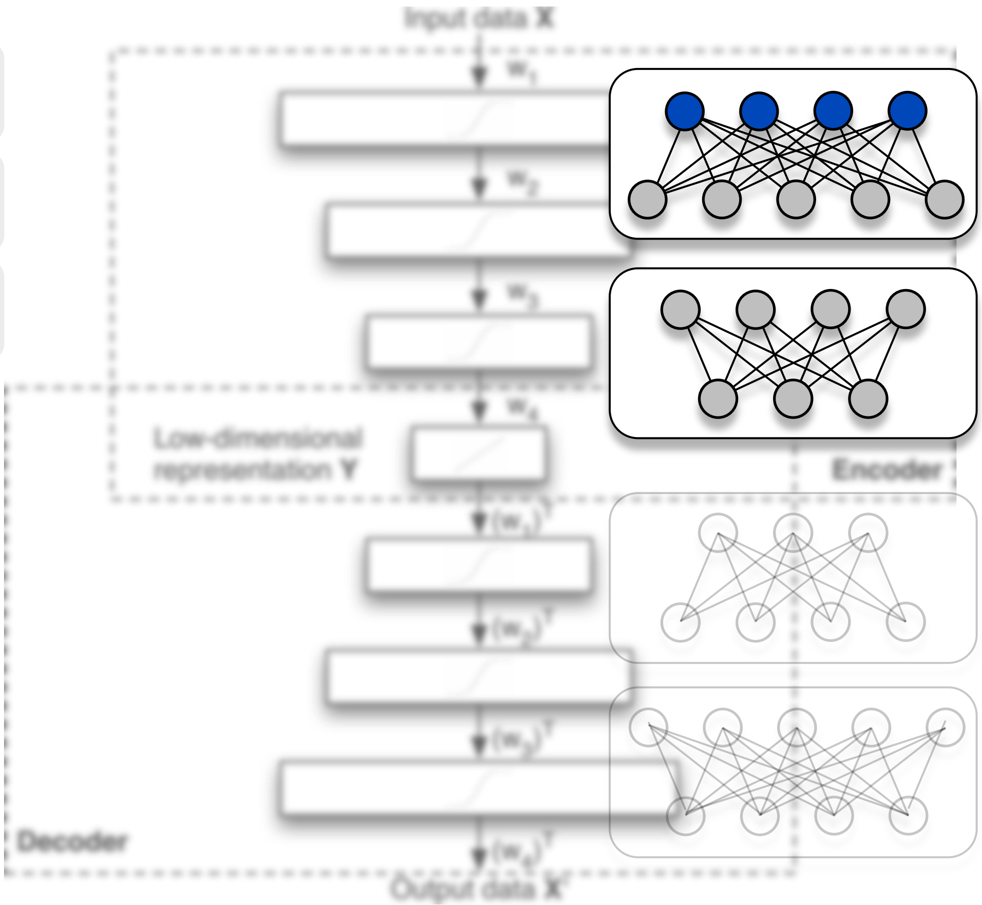
Autoencoder

Training

Reduction

V := set of visible units
x_v := value of unit $v, \forall v \in V$
$x_v \in \{\mathbf{0}, \mathbf{1}\}, \forall v \in V$

H := set of hidden units
x_h := value of unit $h, \forall h \in H$
$x_h \in \{\mathbf{0}, \mathbf{1}\}, \forall h \in H$



The next RBM layer **maps** the dependency encoding…

Autoencoders

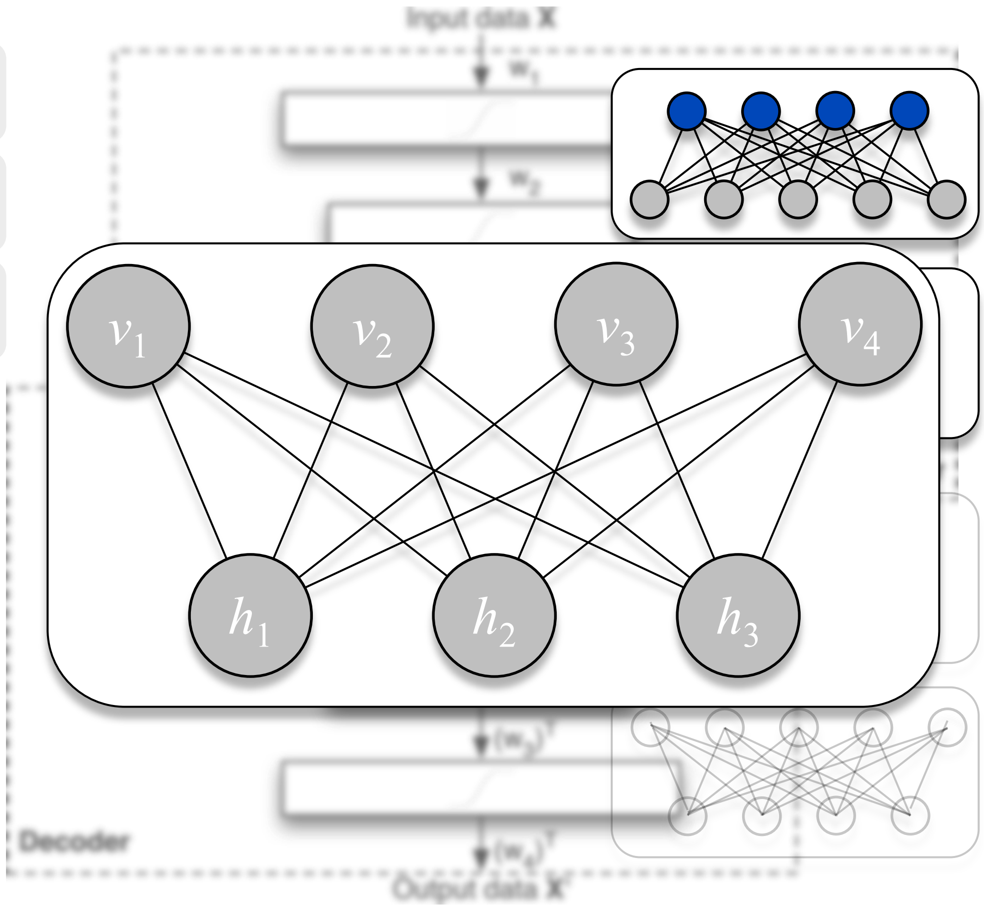
Autoencoder

Training

Reduction

$$x_v \sim \mathrm{sigm}\left(b_v + \sum_h w_{vh}\, x_h \right)$$

b_v := bias of unit v
w_{vh} := weight of edge (v, h)



… from the upper layer …

Autoencoders

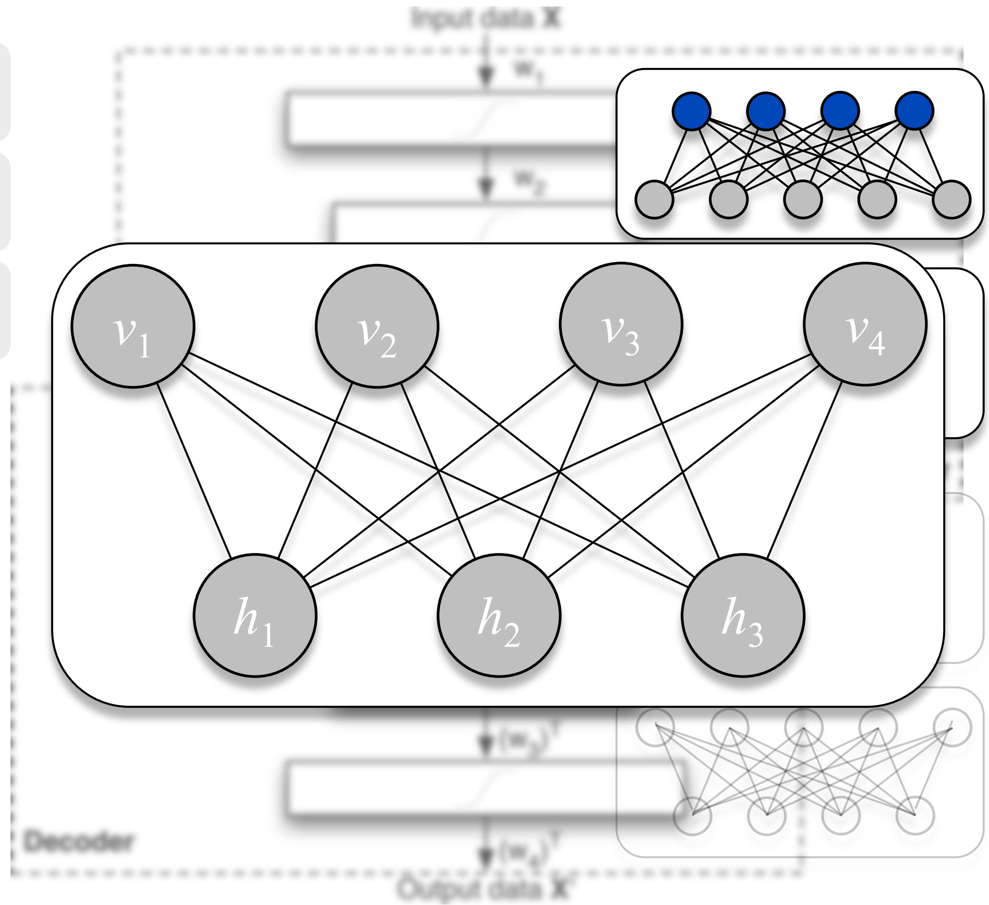
Autoencoder

Training

Reduction

$$x_h \sim \mathrm{sigm}\left(b_h + \sum_v w_{vh} \, x_v \right)$$

$b_h :=$ bias of unit h
$w_{vh} :=$ weight of edge (v, h)



… to a smaller number of **simoids** …

Autoencoders

Autoencoder

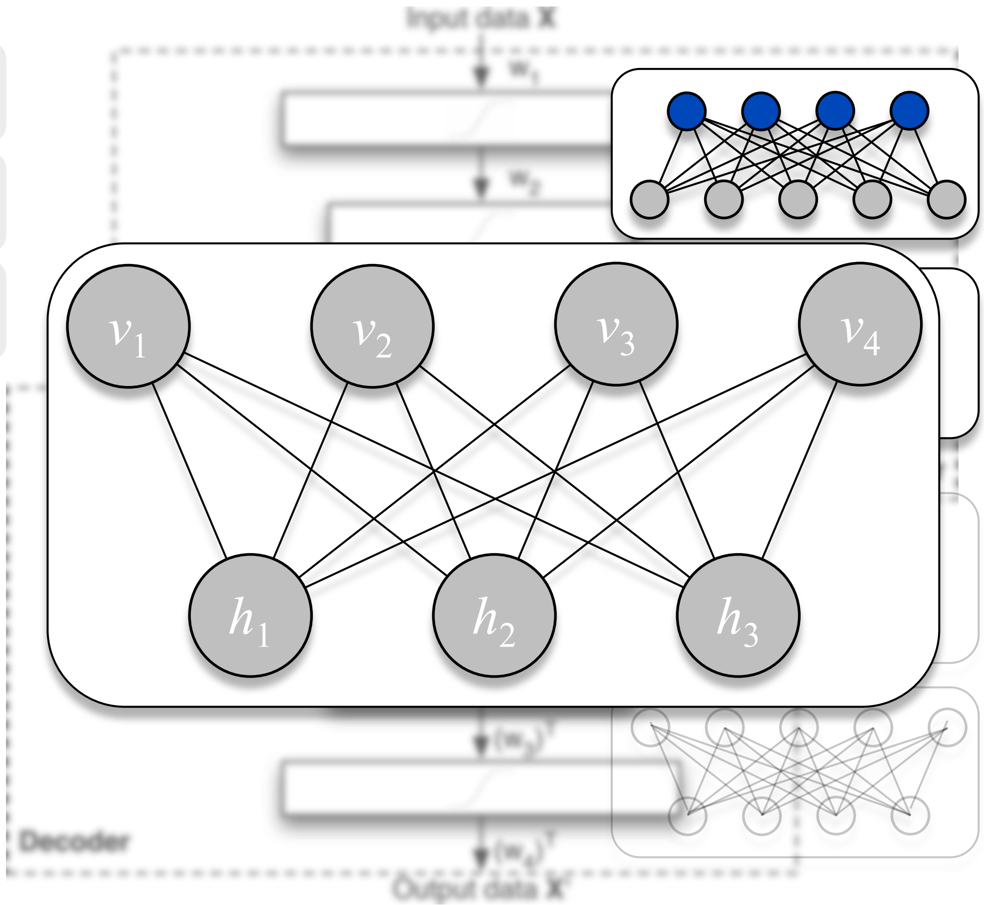
Training

Reduction

Local Energy

$$E_v := -\sum_h w_{vh}\, x_v x_h + x_h b_h$$

$$E_h := -\sum_v w_{vh}\, x_v x_h + x_v b_v$$



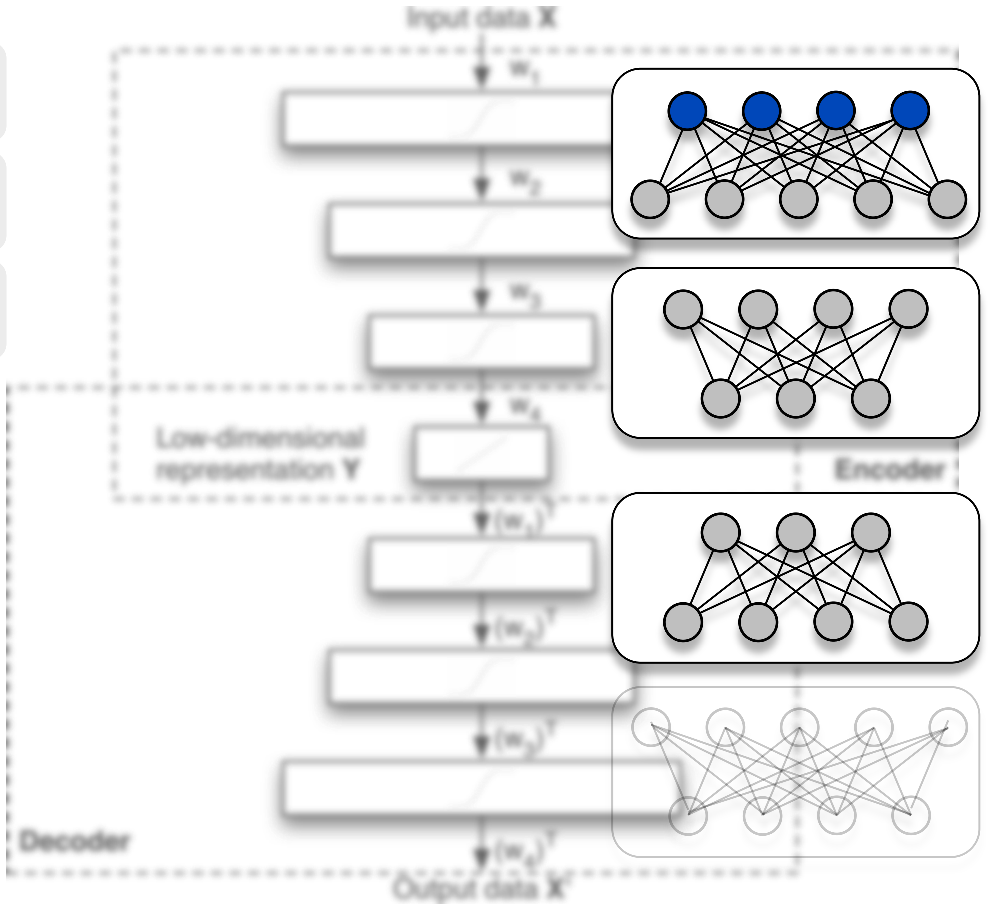
… which can be trained faster than the top layer

Autoencoders

Autoencoder

Training

Unrolling



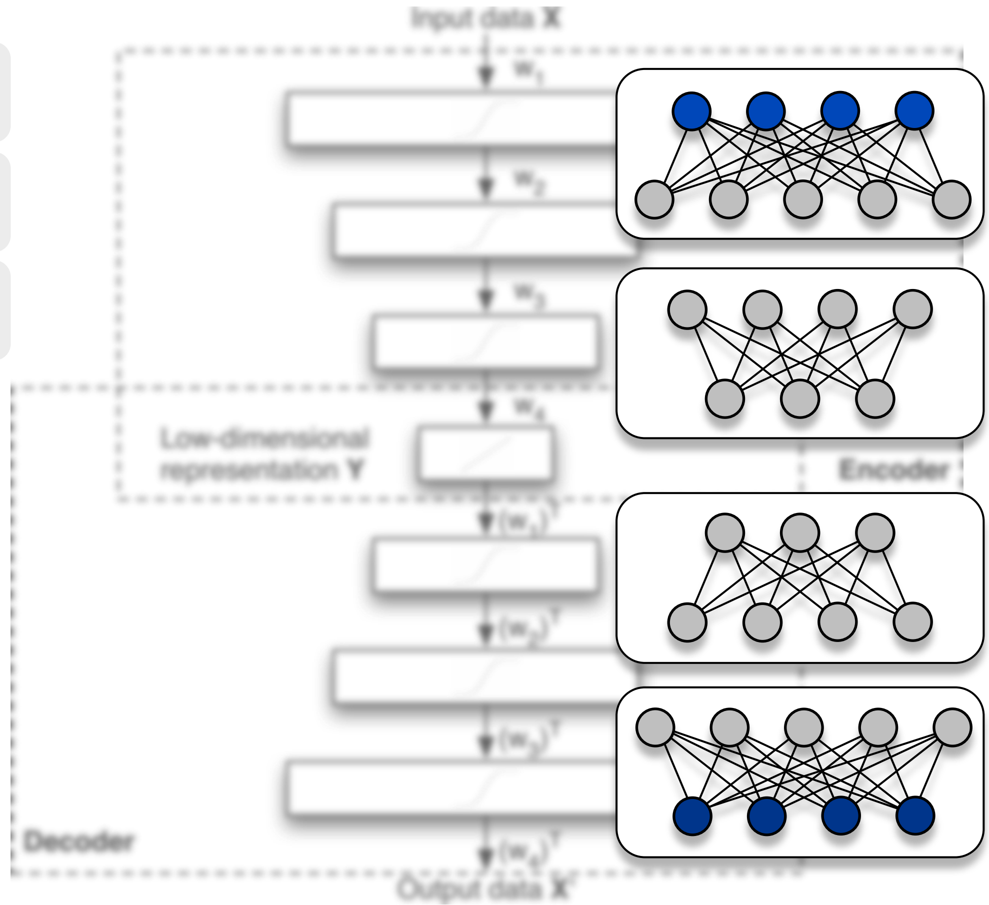
The **symmetric topology** allows us to skip further training.

Autoencoders

Autoencoder

Training

Unrolling



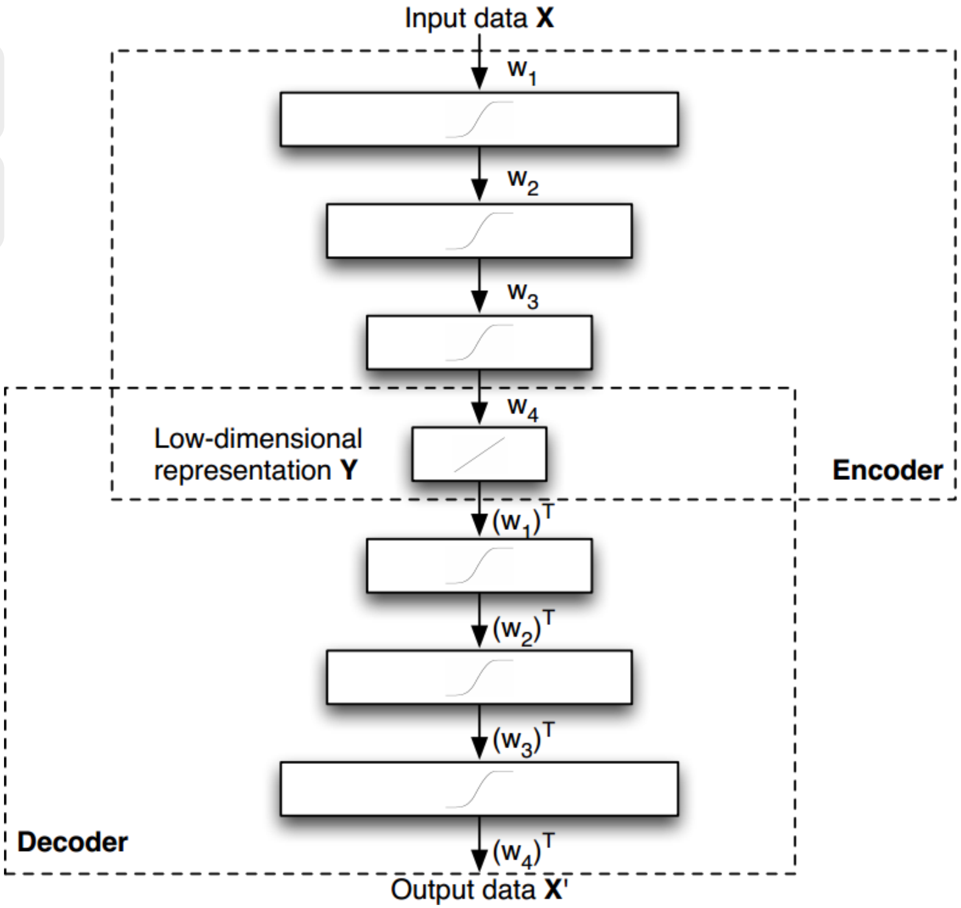
The **symmetric topology** allows us to skip further training.

Autoencoders

Autoencoder

Training

- **Pretraining**
  Top RBM (GRBM)
  Reduction RBMs
  Unrolling
- **Finetuning**
  Backpropagation



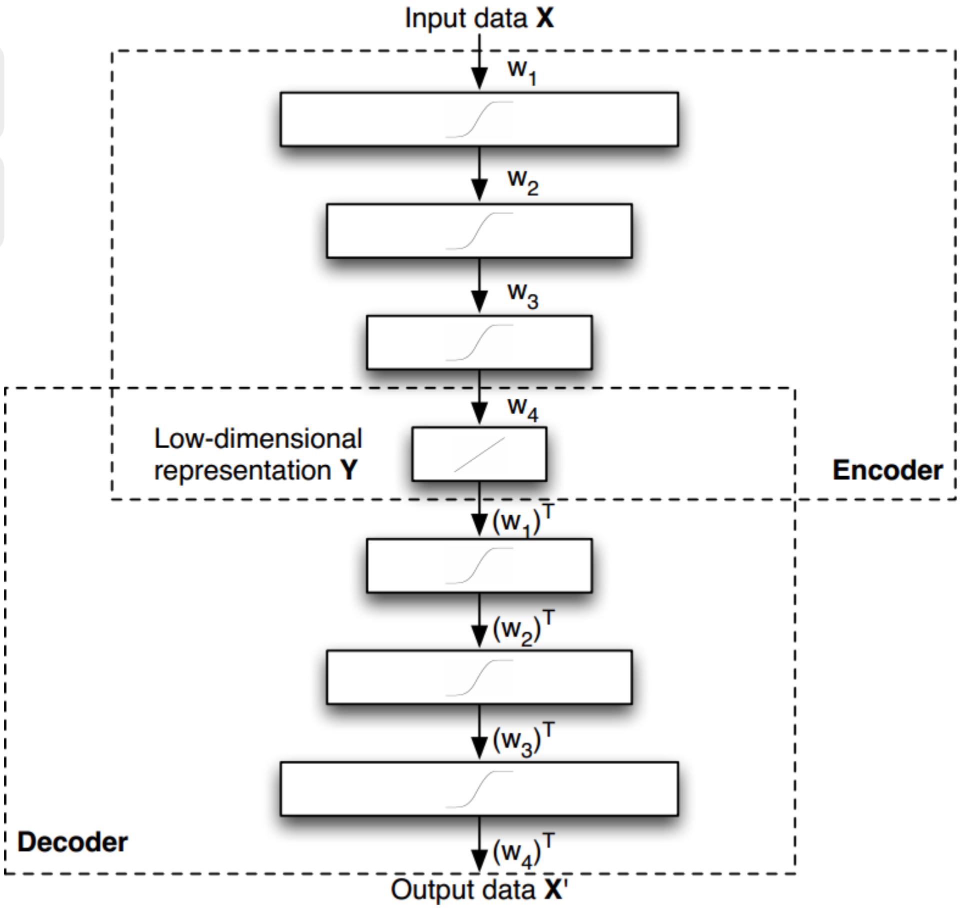
After pretraining **backpropagation** usually finds **good solutions**

Autoencoders

Autoencoder

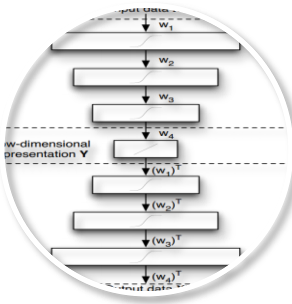
Training

- **Complexity**: O(inw)
  i: number of iterations
  n: number of nodes
  w: number of weights
- **Memory Complexity**: O(w)

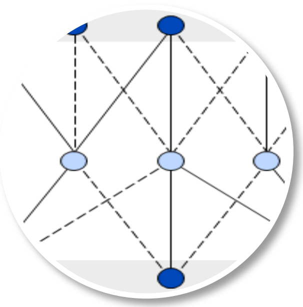


The **algorithmic complexity** of RBM training depends on the network size

Agenda



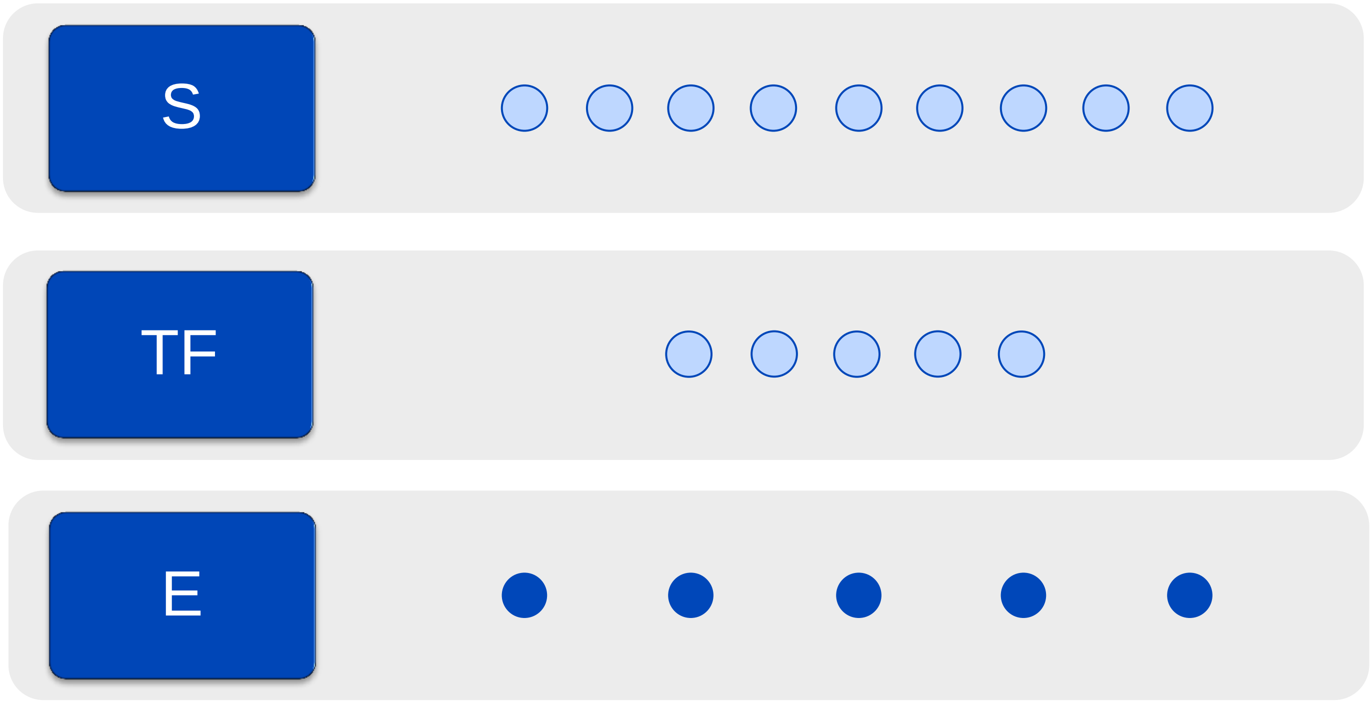
Autoencoders



Biological Model

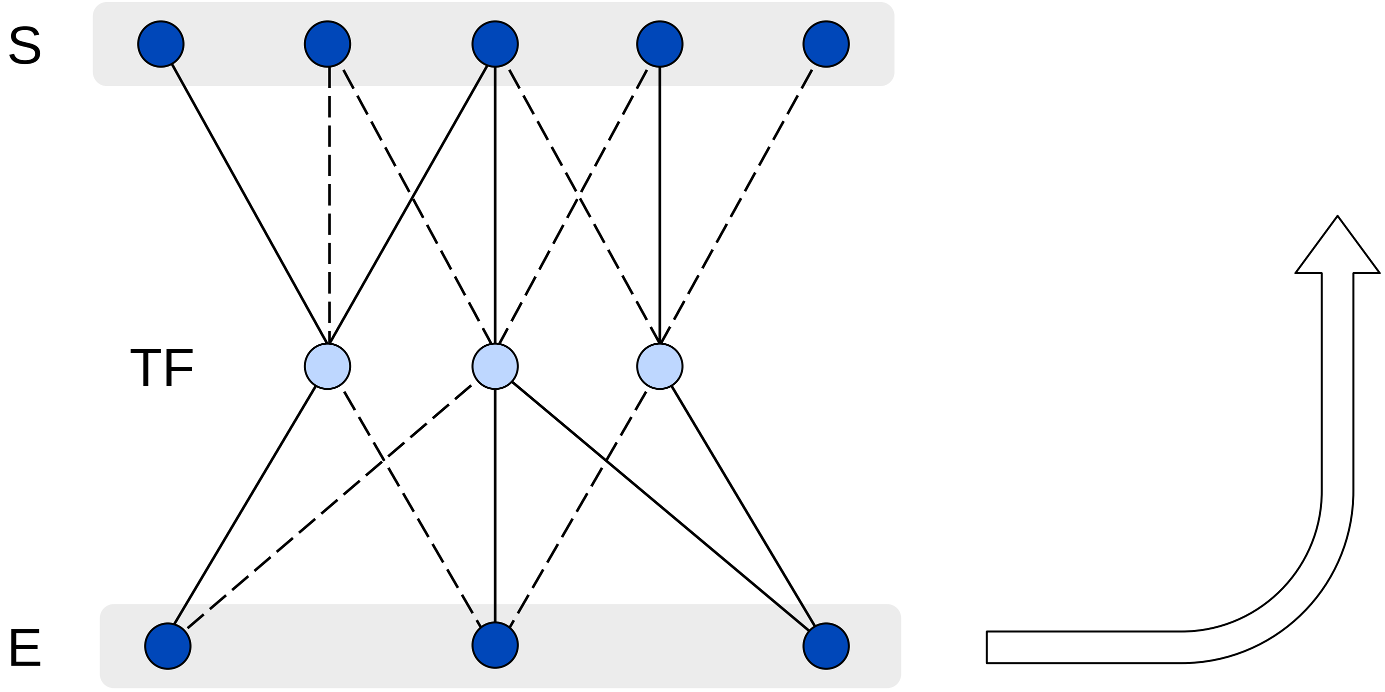
Validation & Implementation

Network Modeling
Restricted Boltzmann Machines (RBM)



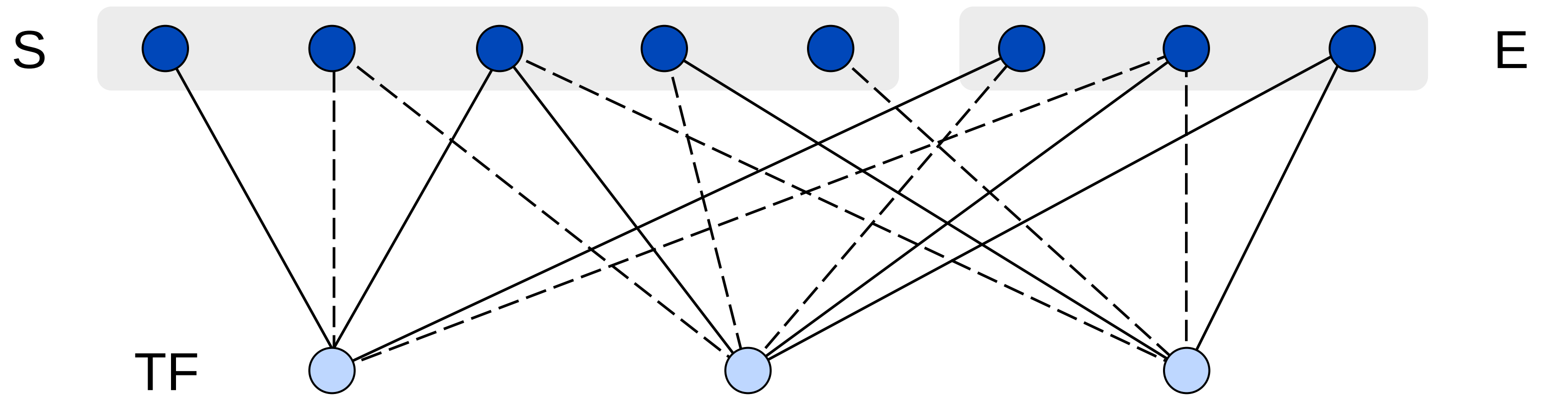
How to model the topological structure?

Network Modeling
Restricted Boltzmann Machines (RBM)



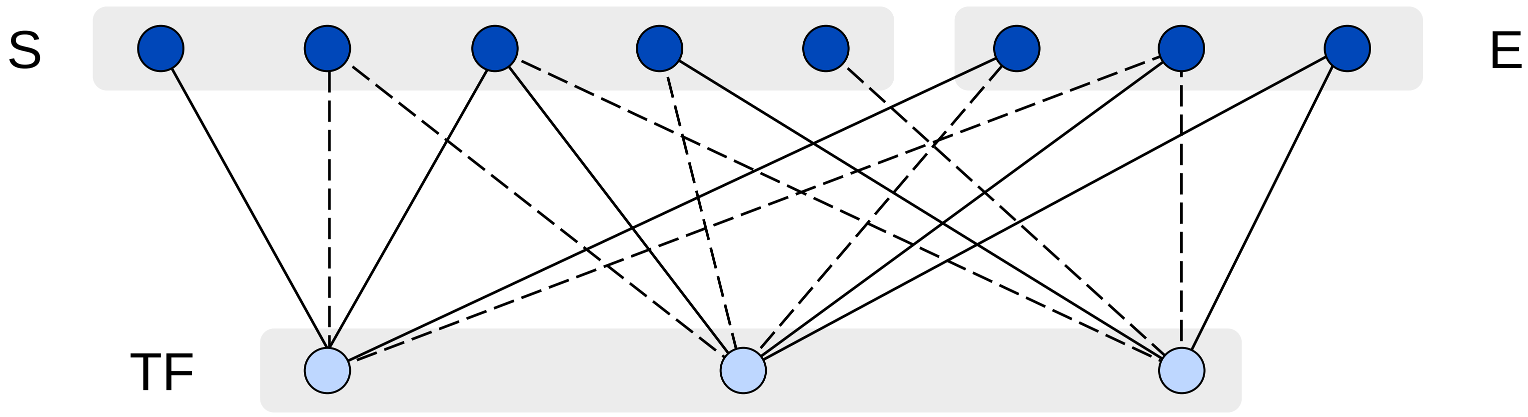
We define S and E as **visible data Layer** …

Network Modeling
Restricted Boltzmann Machines (RBM)



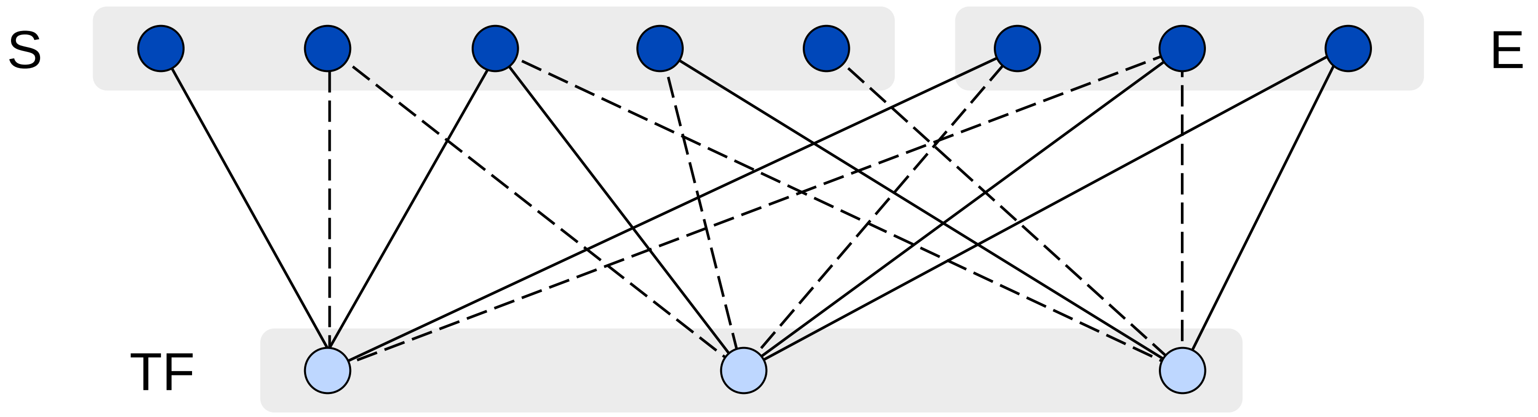
We identify S and E with the **visible layer** …

Network Modeling
Restricted Boltzmann Machines (RBM)



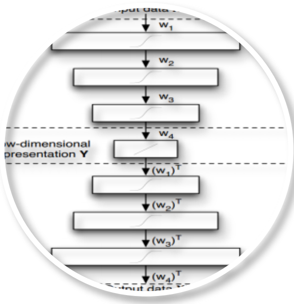
… and the TFs with the **hidden layer** in a RBM

Network Modeling
Restricted Boltzmann Machines (RBM)

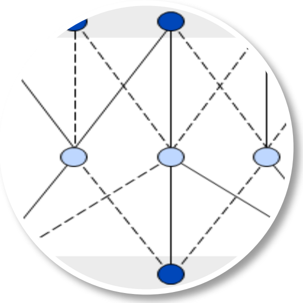


The training of the RBM gives us a model

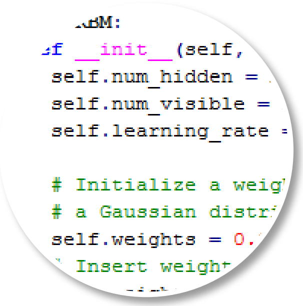
Agenda



Autoencoder



Biological Model



Implementation & Results

Results

Validation of the results

- Needs information about the true regulation
- Needs information about the descriptive power of the data

Results

Validation of the results

- Needs information about the true regulation
- Needs information about the descriptive power of the data

Without this infomation validation can only be done,
using **artificial datasets**!

Results

Artificial datasets

We simulate data in three steps:

Results

Artificial datasets

We simulate data in three steps

Step 1

Choose number of Genes (E+S) and create random bimodal distributed data

Results

Artificial datasets

We simulate data in three steps

Step 1

Choose number of Genes (E+S) and create random bimodal distributed data

Step 2

Manipulate data in a fixed order

Results

Artificial datasets

We simulate data in three steps

Step 1

Choose number of Genes (E+S) and create random bimodal distributed data

Step 2

Manipulate data in a fixed order

Step 3

Add noise to manipulated data

and normalize data

Results

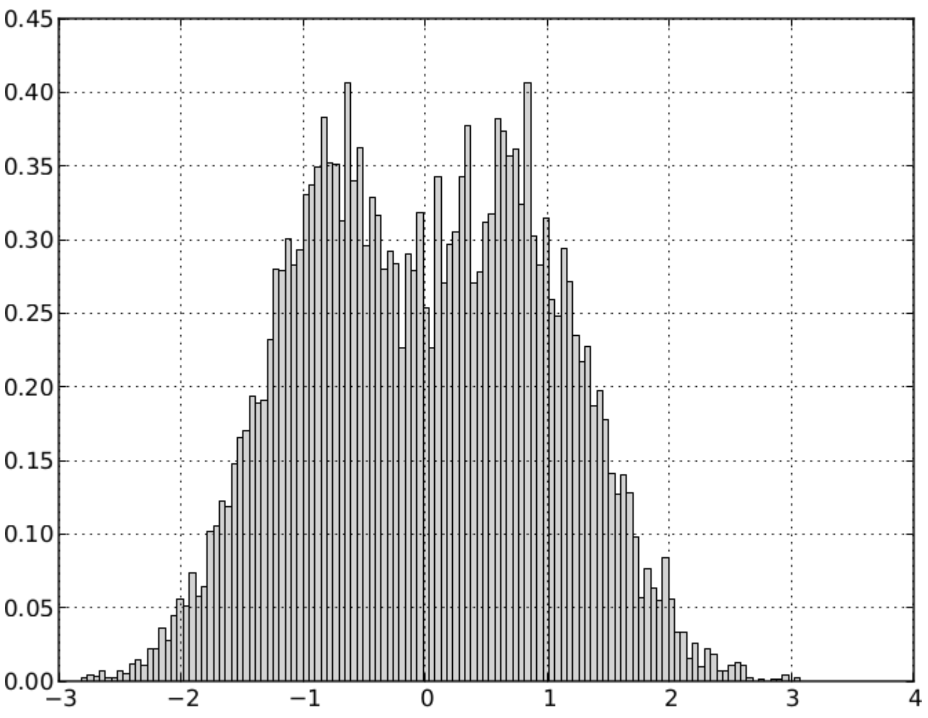
Simulation

Step 1

Number of visible nodes 8 (4E, 4S)

Create random data:

Random {-1, +1} + N(0, $\sigma = 0.5$)



Results

Simulation

Step 2

Manipulate data

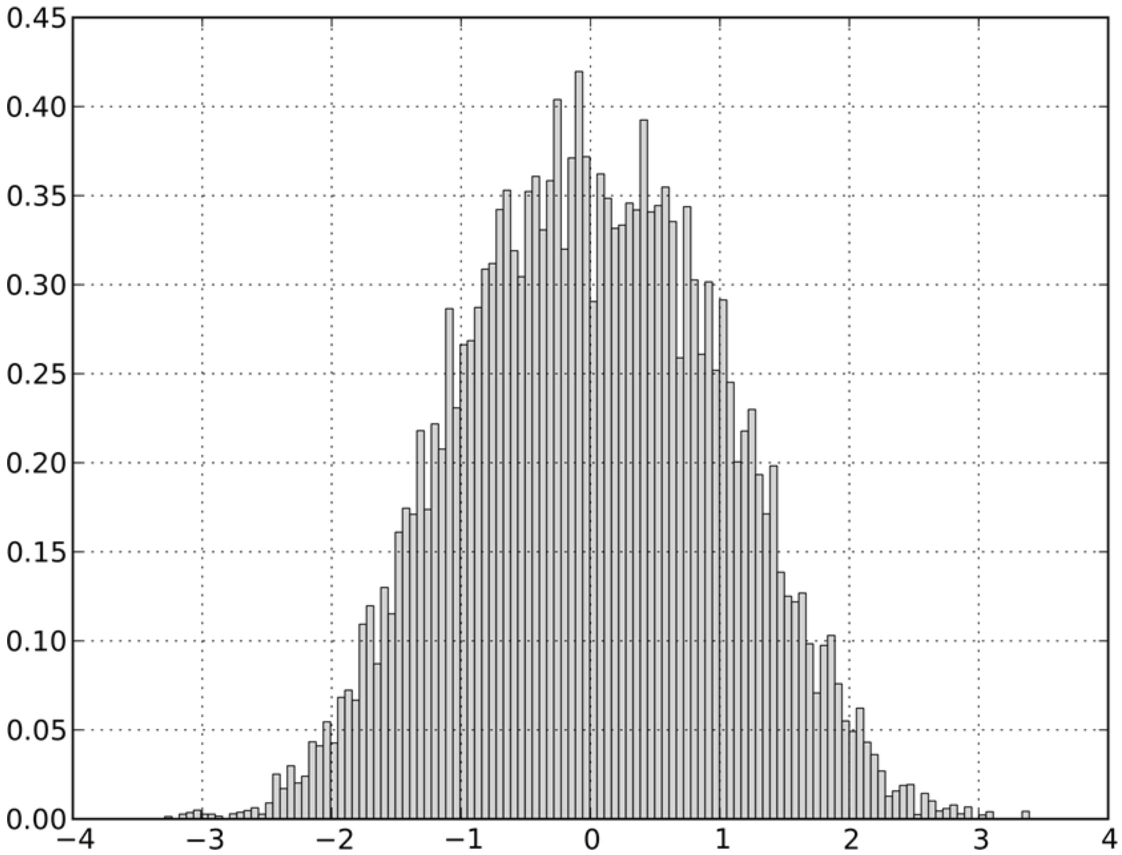
$$e_1 = 0.25s_1 + 0.25s_2 + 0.25s_3 + 0.25s_4$$
$$e_2 = 0.5s_1 + 0.5\ Noise$$
$$e_3 = 0.5s_1 + 0.5\ Noise_4$$
$$e_4 = 0.5s_1 + 0.5\ Noise$$

Results

Simulation

Step 3

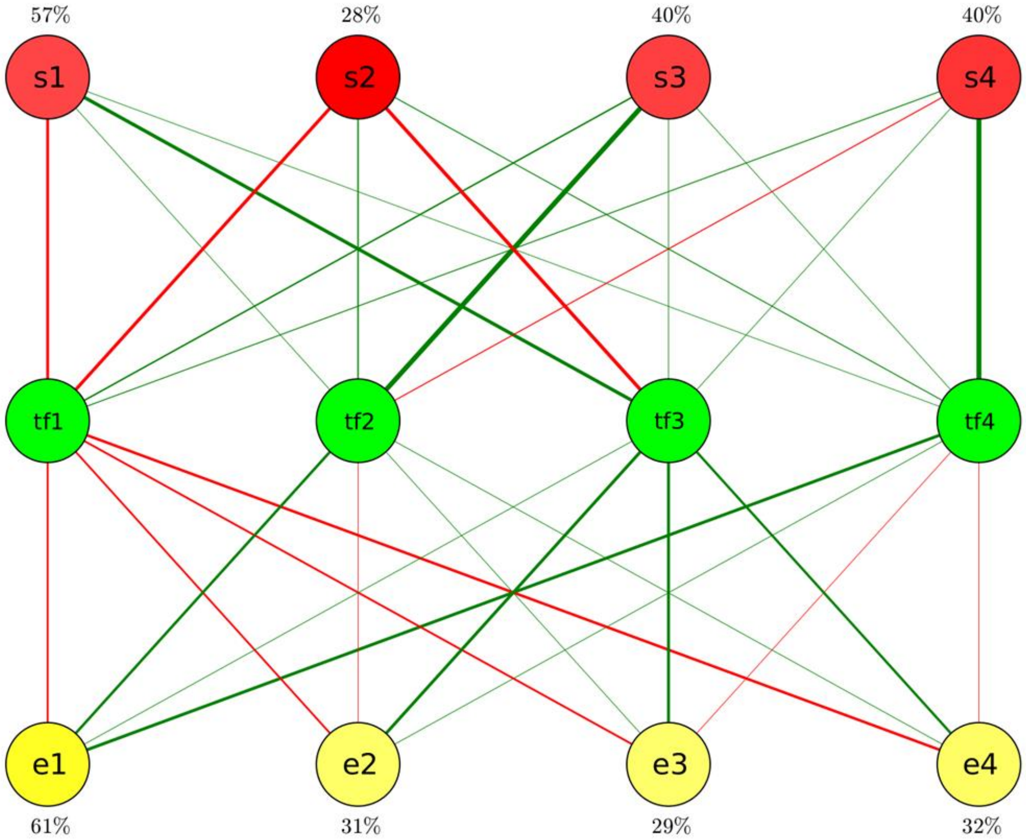
Add noise: N(0, $\sigma = 0.5$)



Results

We analyse the data **X** with an RBM

sim42: $\sigma = 0.5$, no filtering

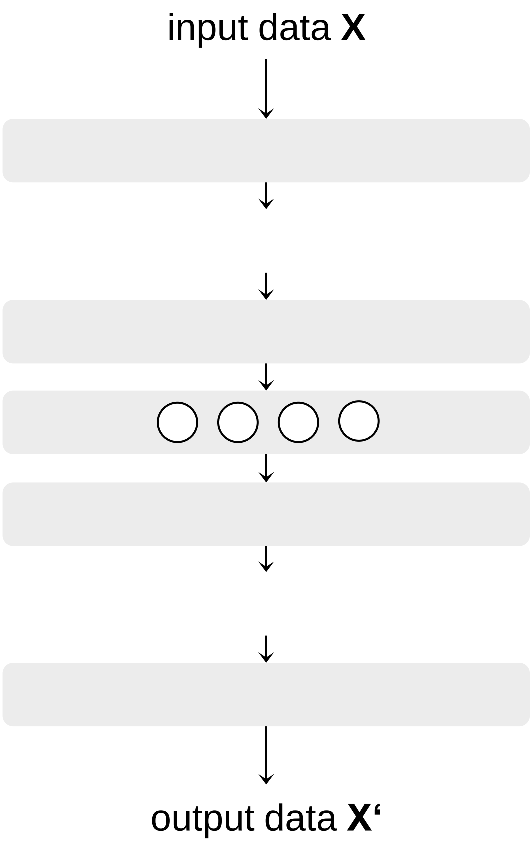


Average performance: 40.3%

Results

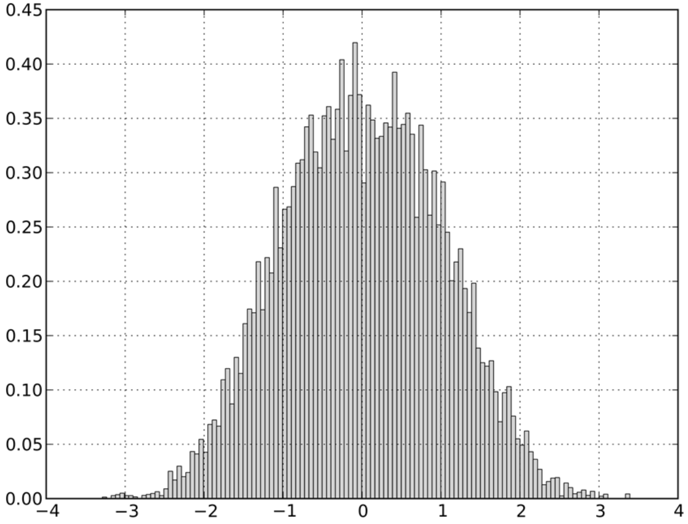
We train an autoencoder with 9 hidden layers and 165 nodes:

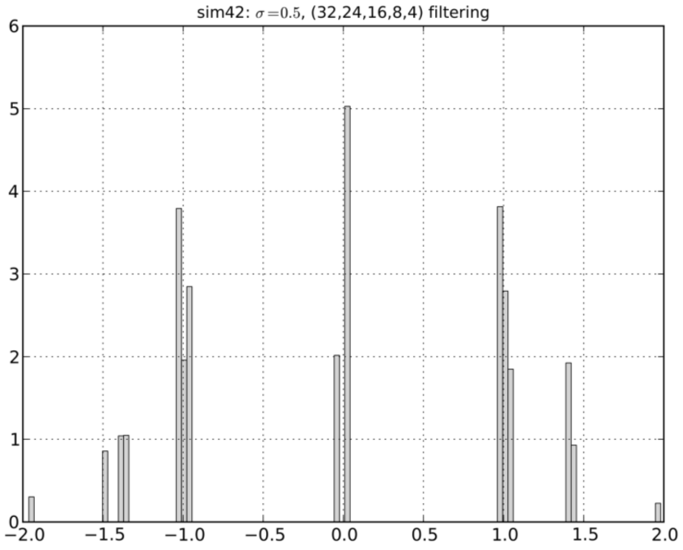
Layer 1 & 9: 32 hidden units
Layer 2 & 8: 24 hidden units
Layer 3 & 7: 16 hidden units
Layer 4 & 6: 8 hidden units
Layer 5: 5 hidden units



Results

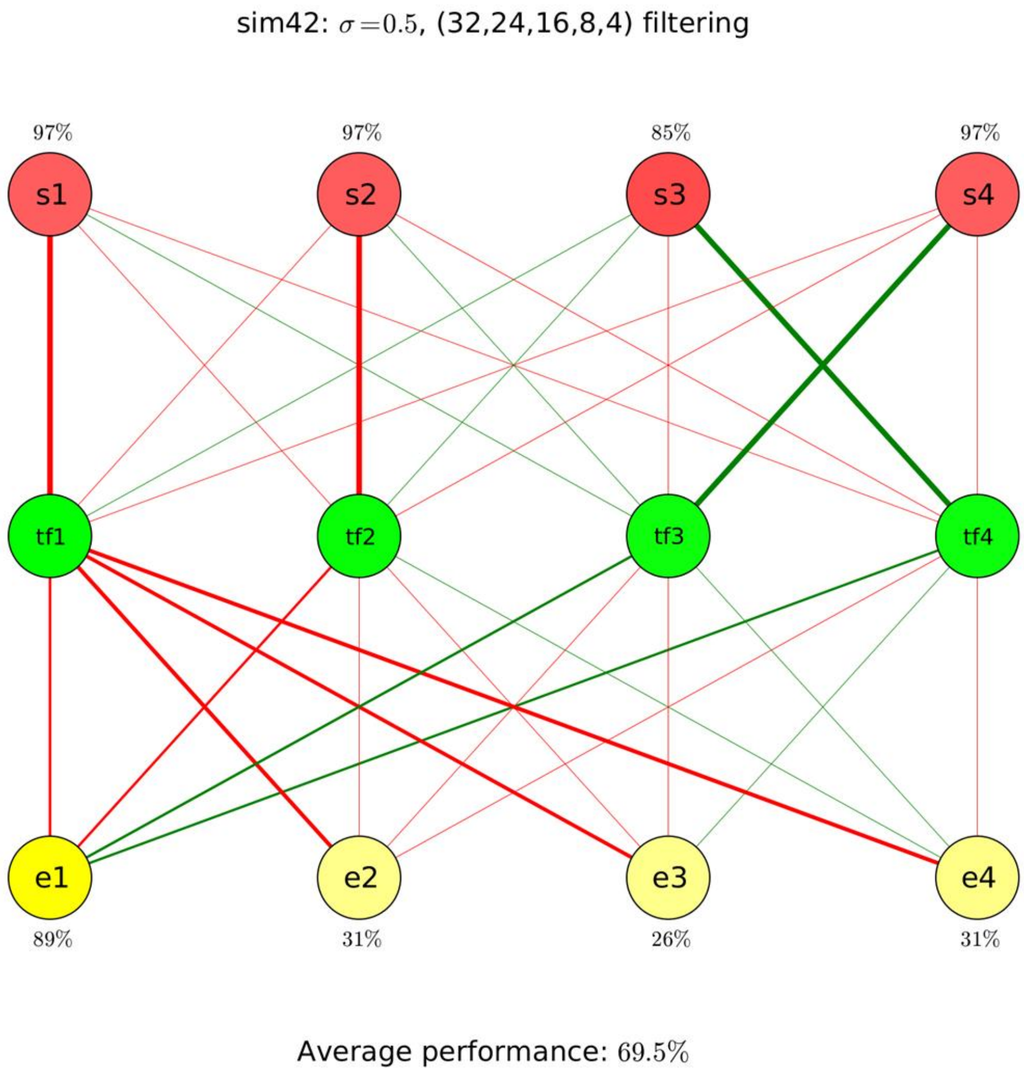
We transform the data from **X** to **X‘**
And reduce the dimensionality





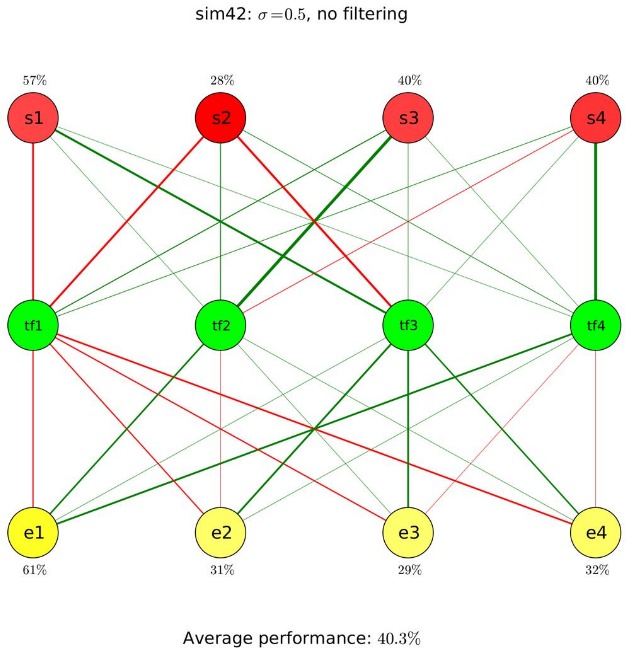
Results

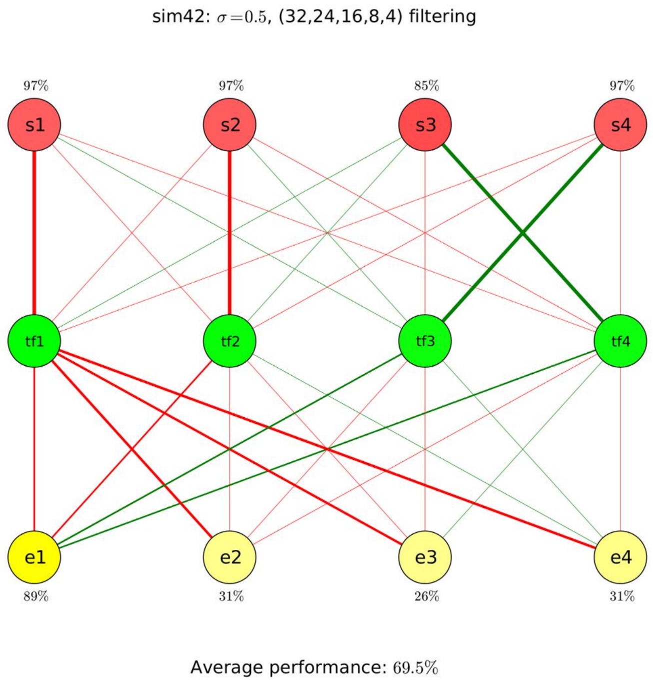
We analyse the transformed data **X'** with an RBM



Results

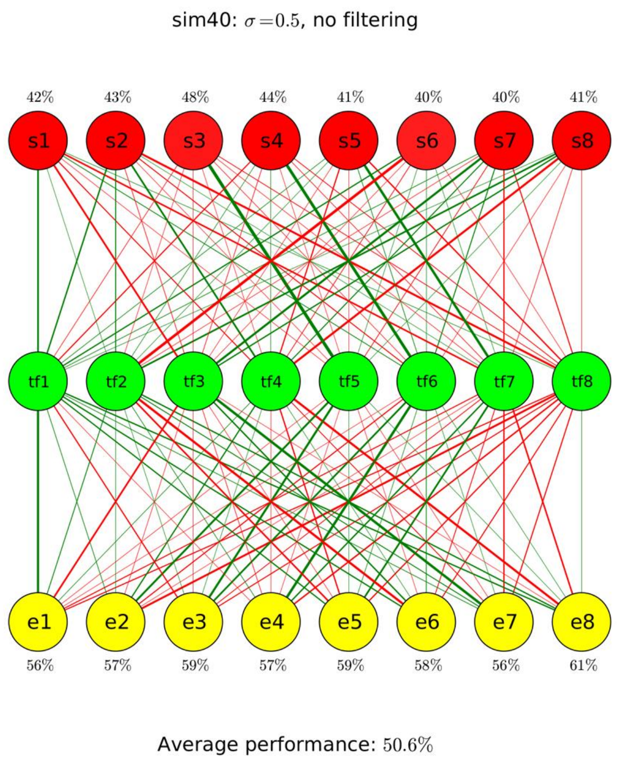
Lets compare the models

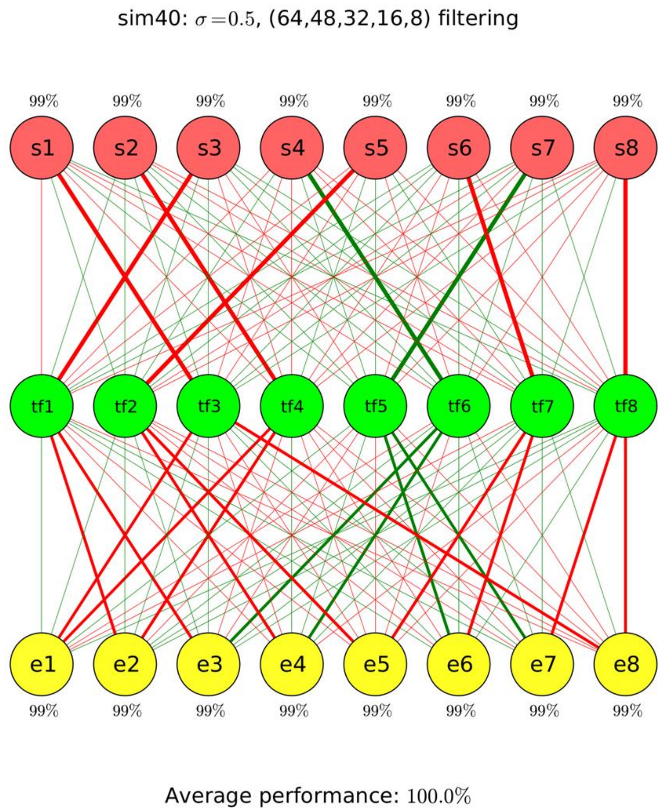




Results

Another Example with more nodes and larger autoencoder





Conclusion

Conclusion

- Autoencoders can improve modeling significantly by **reducing the dimensionality of data**
- Autoencoders **preserve complex structures** in their multilayer perceptron network. Analysing those networks (for example with knockout tests) could give more structural information
- The drawback are **high computational costs**
  Since the field of deep learning is getting more popular (Face recognition / Voice recognition, Image transformation). Many new improvements in facing the computational costs have been made.

Acknowledgement

eilsLABS

Prof. Dr. Rainer König

Prof. Dr. Roland Eils

Network Modeling Group