Advanced probabilistic learning: Deep Belief Networks

Pietro Berkes, Brandeis University

Probabilistic learning

- an unsupervised model captures the distribution of the input data
- this distribution is usually best described in term of unobserved (hidden) causes

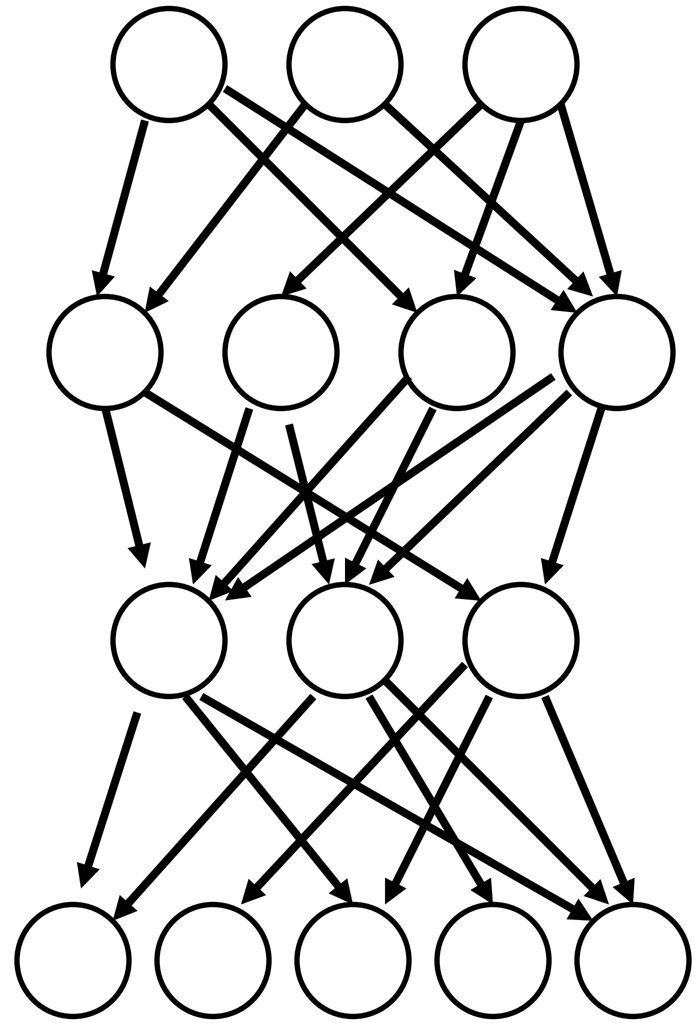
Generative models

objects

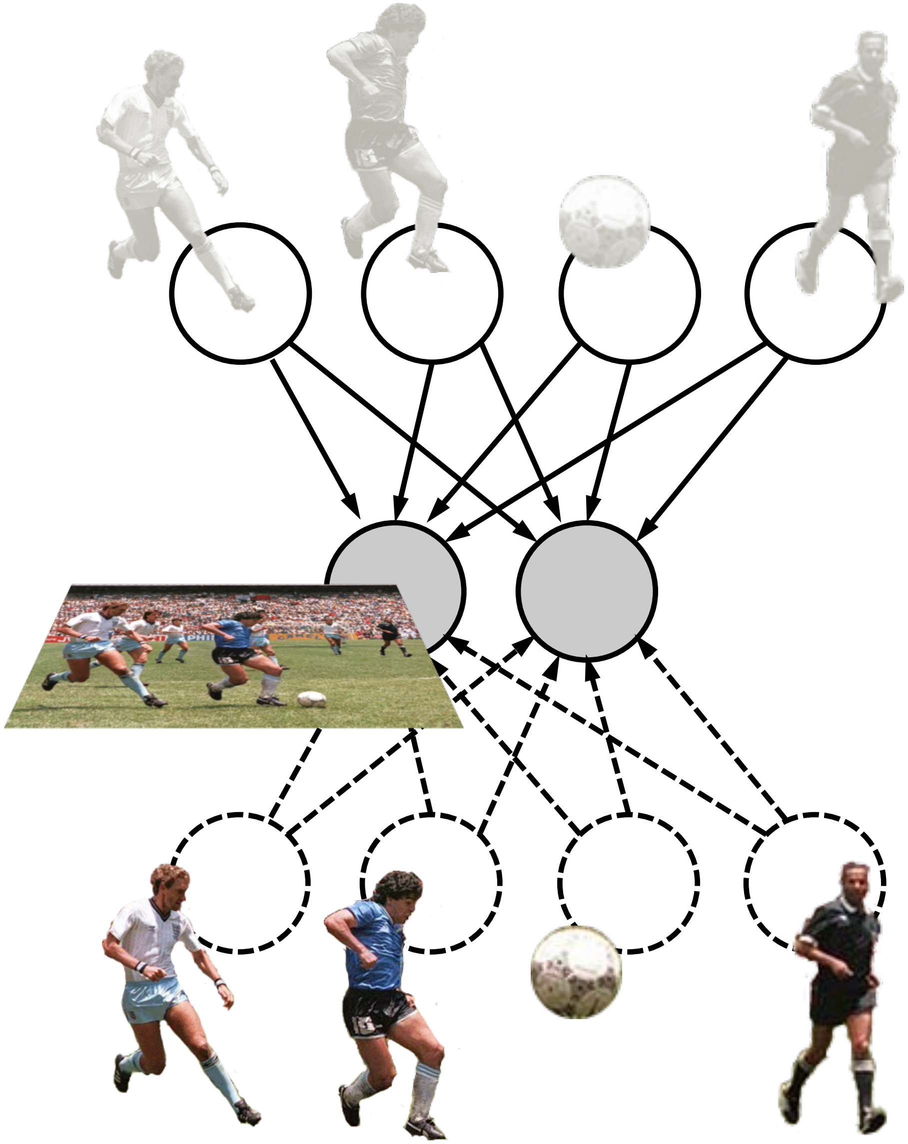
object parts

image features

pixels



Generative models and the brain

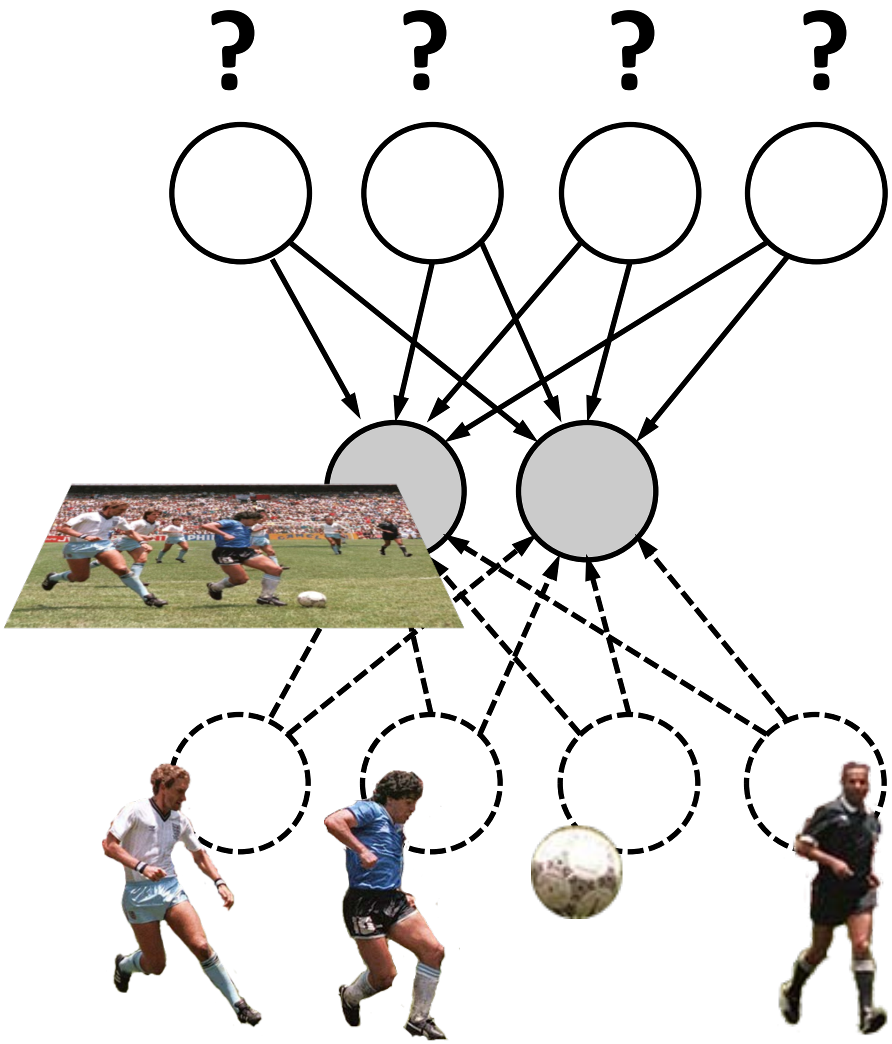


our internal model: a mirror version of the real generative process?

visual input
(light patterns)

external causes
(visual elements)

Unsupervised learning

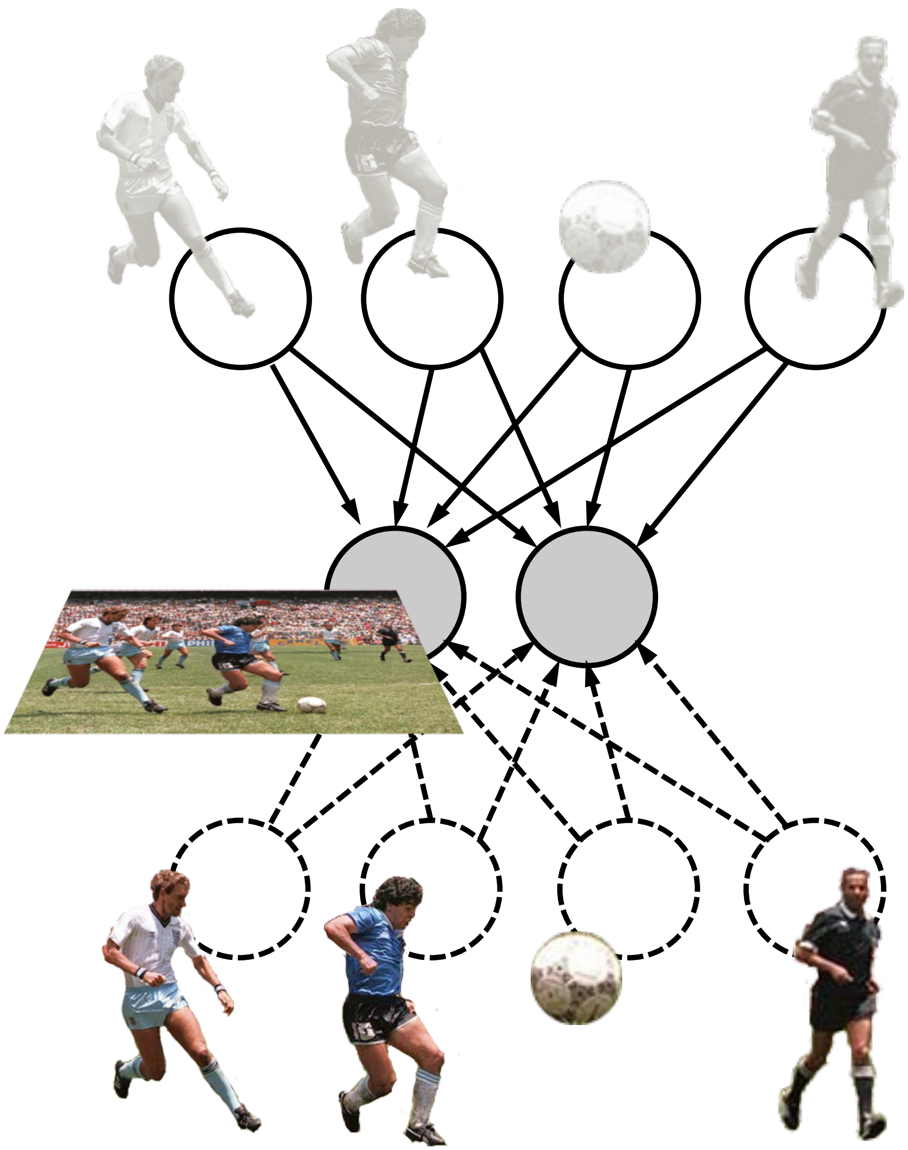


find hidden causes that make input data easy to describe

visual input
(light patterns)

external causes
(visual elements)

Inference



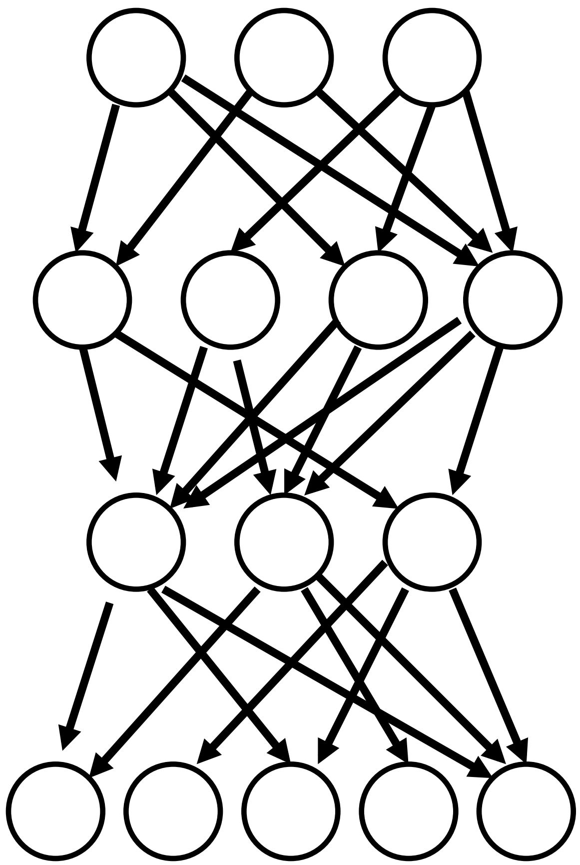
given an image, which causes are present?

visual input
(light patterns)

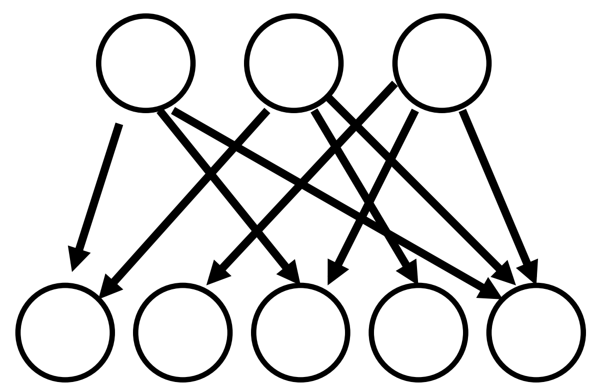
external causes
(visual elements)

Issues with hierarchical models

We would like to do this:



But we can only do this:



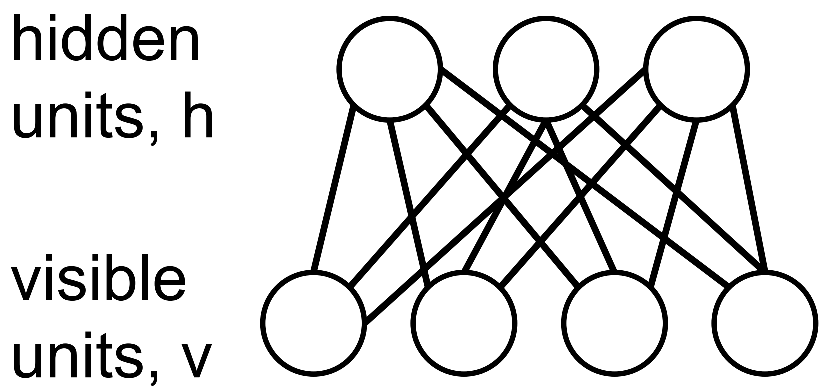
Issues with hierarchical models

- Many local minima: probability of observed data is very complicated function of parameters
- Some parameters are more sensitive than others

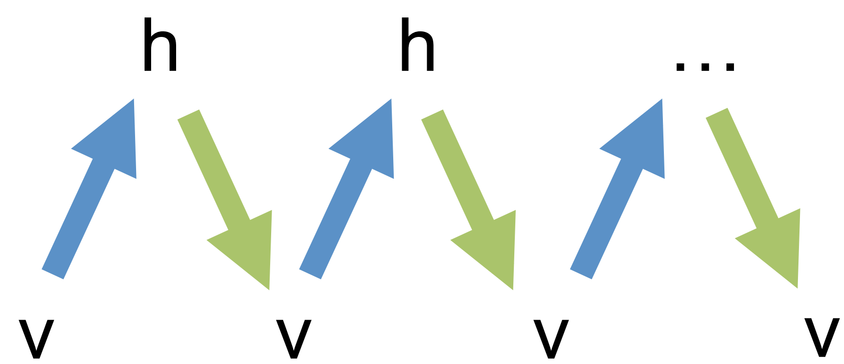
- In practice: global optimization of parameters is hell

Unrolling an RBM

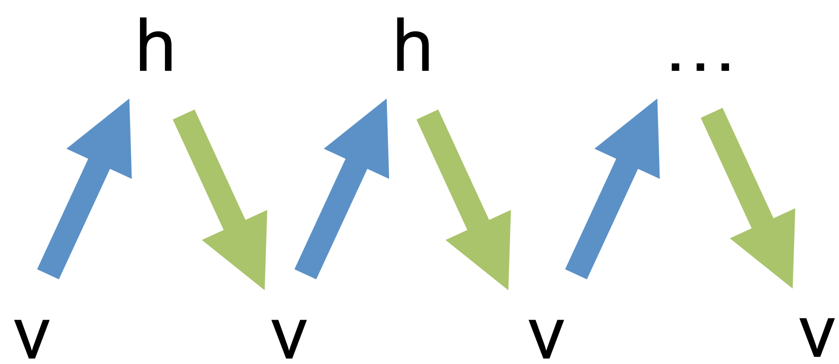
Breakthrough in 2006: start by training a hierarchical network layer-by-layer

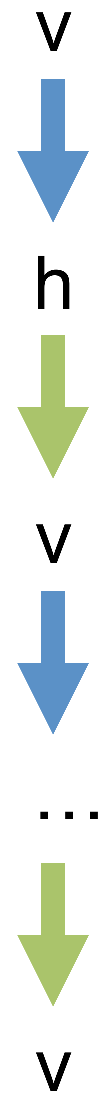


Distribution over v can be found by Gibbs sampling:



Unrolling an RBM

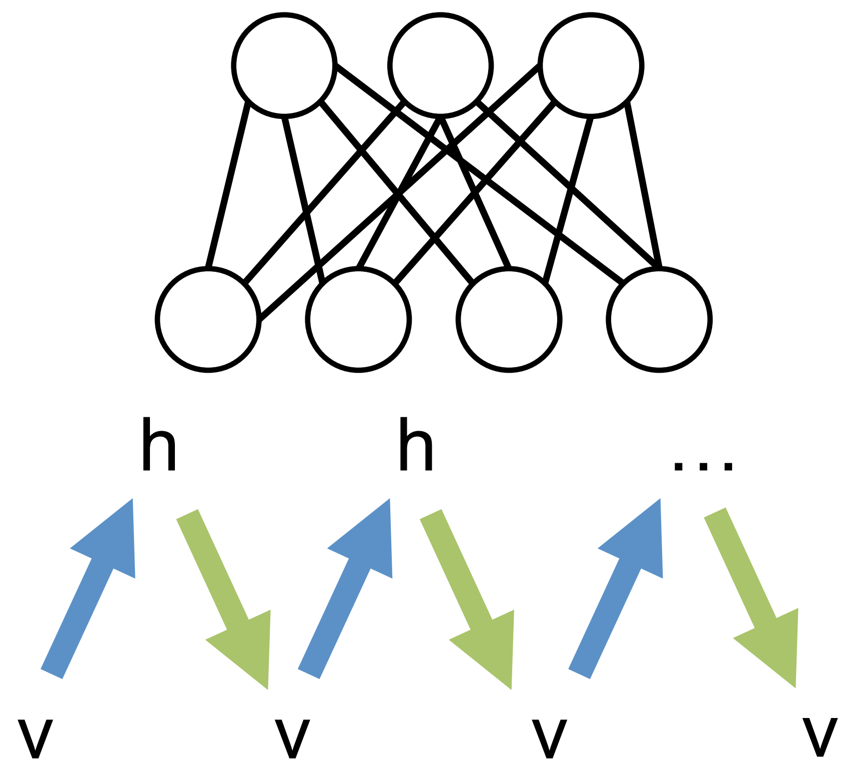


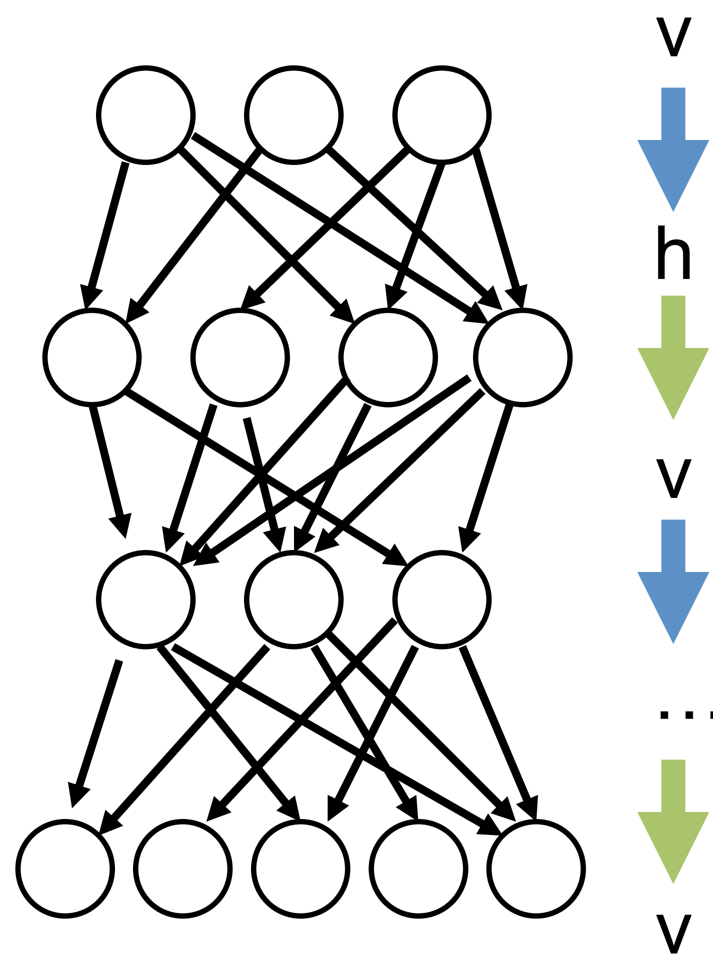


Unrolling an RBM

The RBM was unrolled in a hierarchical generative model that defines the same distribution over the observed variables.

The activation functions and the weights must be the same everywhere!



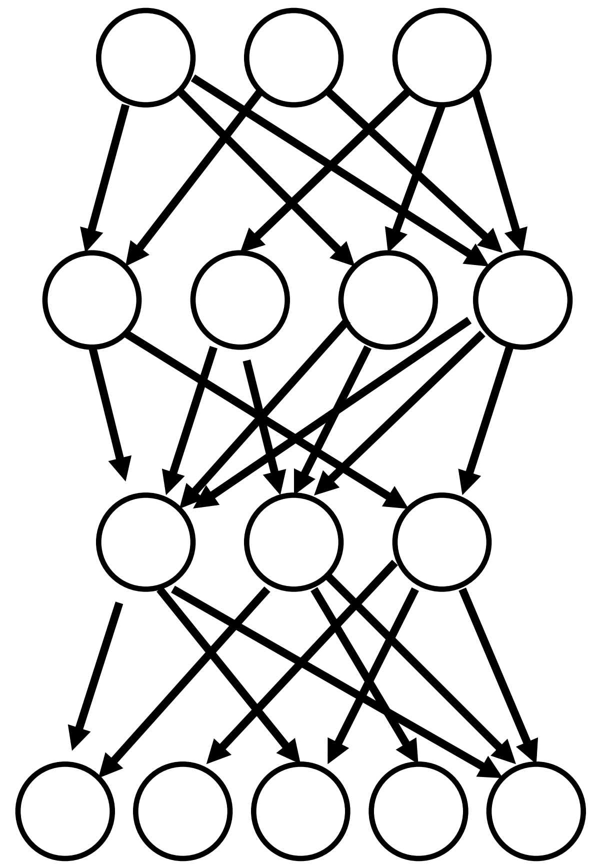


Deep Belief Networks (2006)

- Hierarchical generative model
- Activation functions as in RBM:

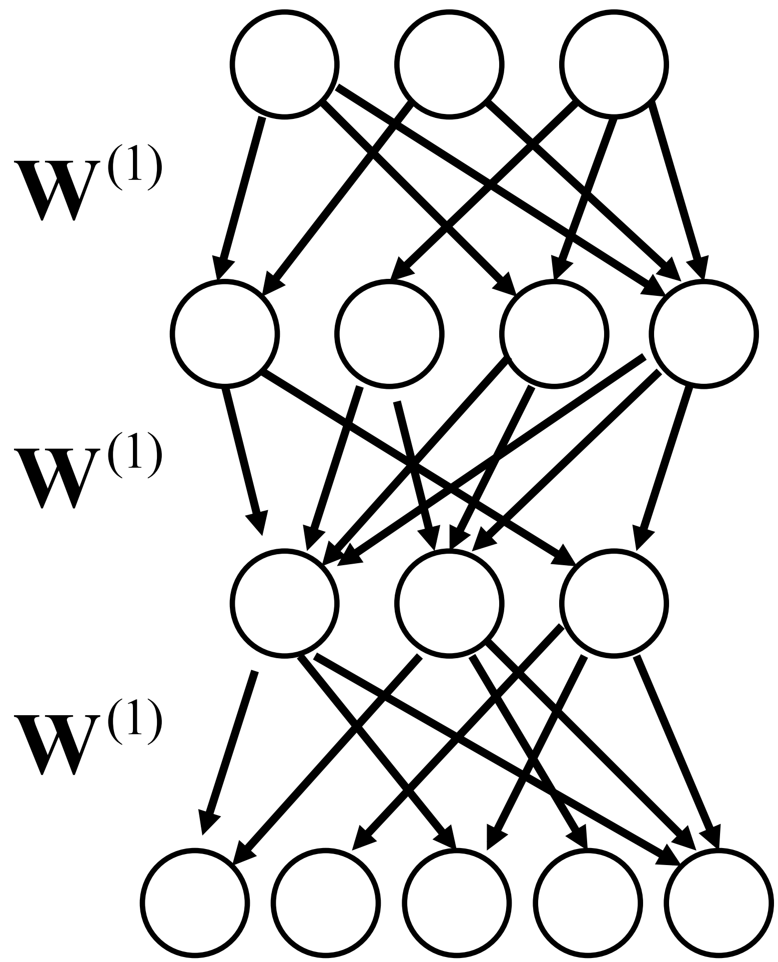
$$a_i^{(l)} = b_i^{(l)} + \sum_j w_{ij}^{(l)} x_j^{(l+1)}$$

$$P(x_i^{(l)} = 1 \mid x^{(l+1)}) = \frac{1}{1 + e^{-a_i^{(l)}}}$$



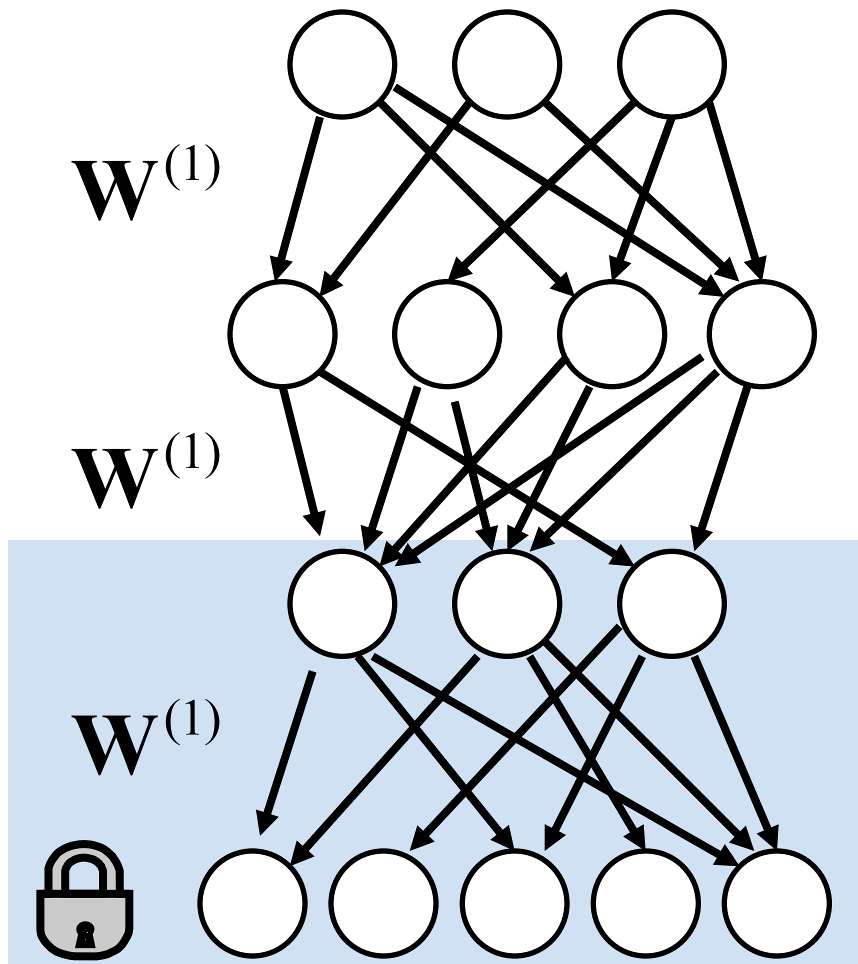
Greedy learning

- Start assuming the weights are tied at all levels => the DBN is equivalent to an RBM
- We can train the DBN easily using the RBM training rule



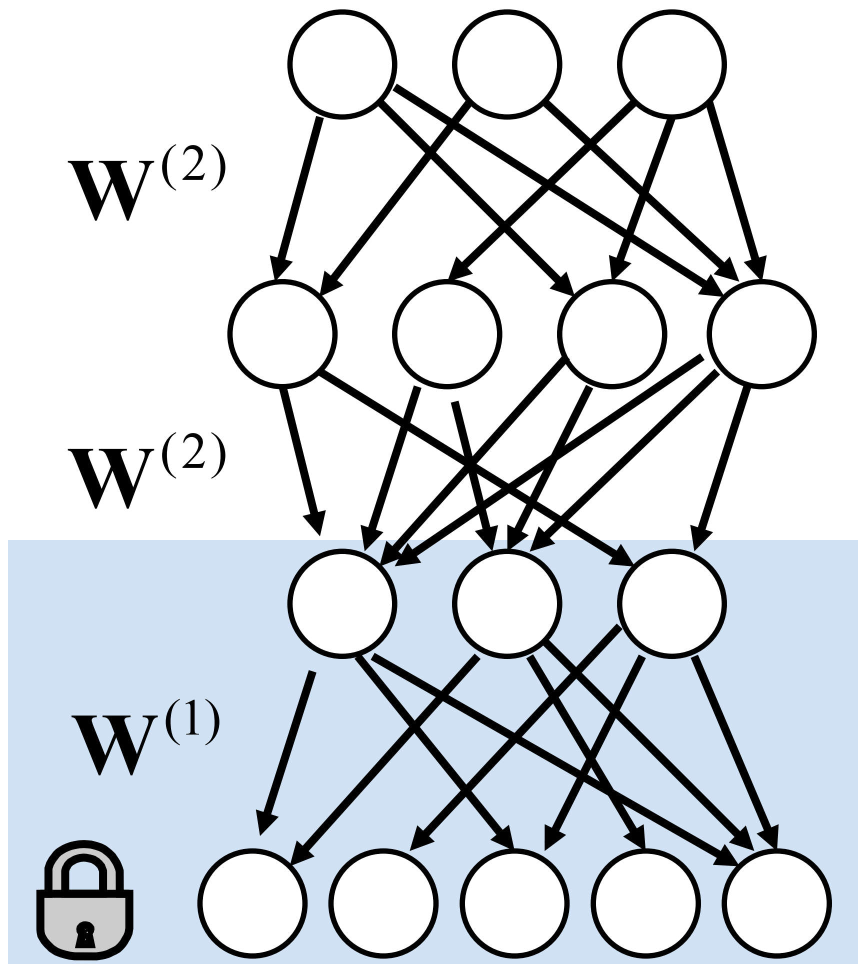
Greedy learning

- Start assuming the weights are tied at all levels => the DBN is equivalent to an RBM
- We can train the DBN easily using the RBM training rule
- Fix the weights in the first layer, use the output of the first layer as the input for the next, and use the same trick



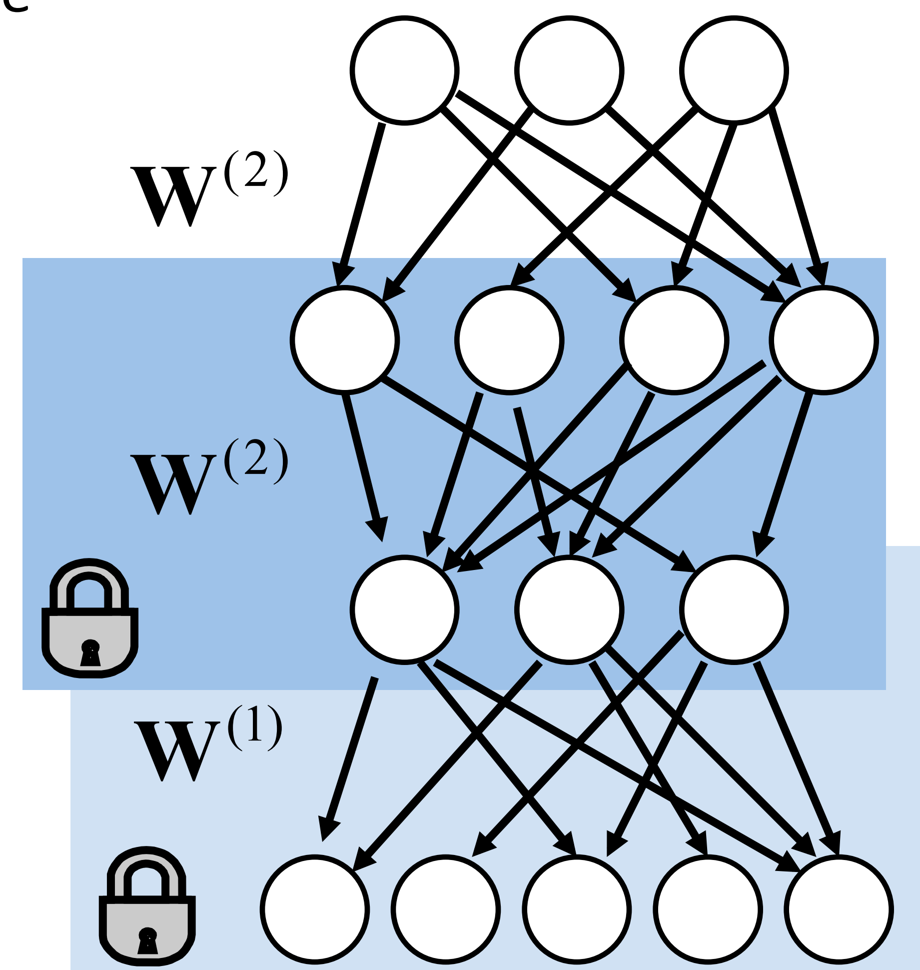
Greedy learning

- Start assuming the weights are tied at all levels => the DBN is equivalent to an RBM
- We can train the DBN easily using the RBM training rule
- Fix the weights in the first layer, use the output of the first layer as the input for the next, and use the same trick



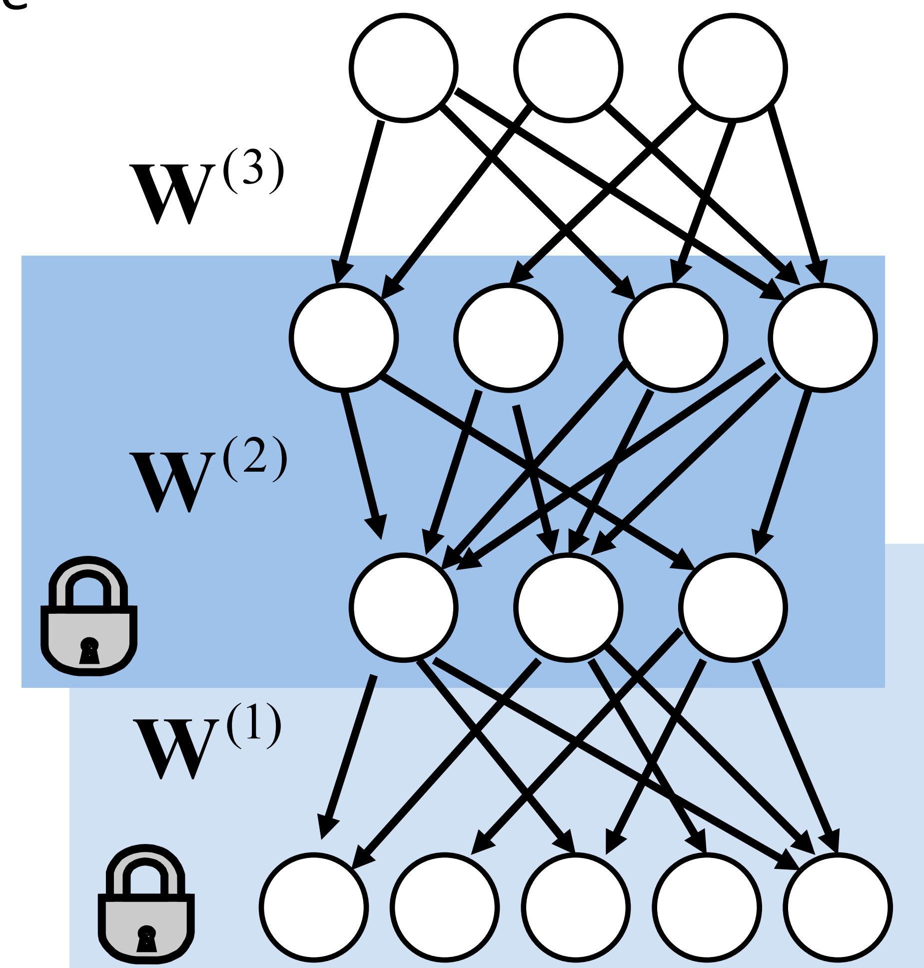
Greedy learning

- Start assuming the weights are tied at all levels => the DBN is equivalent to an RBM
- We can train the DBN easily using the RBM training rule
- Fix the weights in the first layer, use the output of the first layer as the input for the next, and use the same trick
- And so on …



Greedy learning

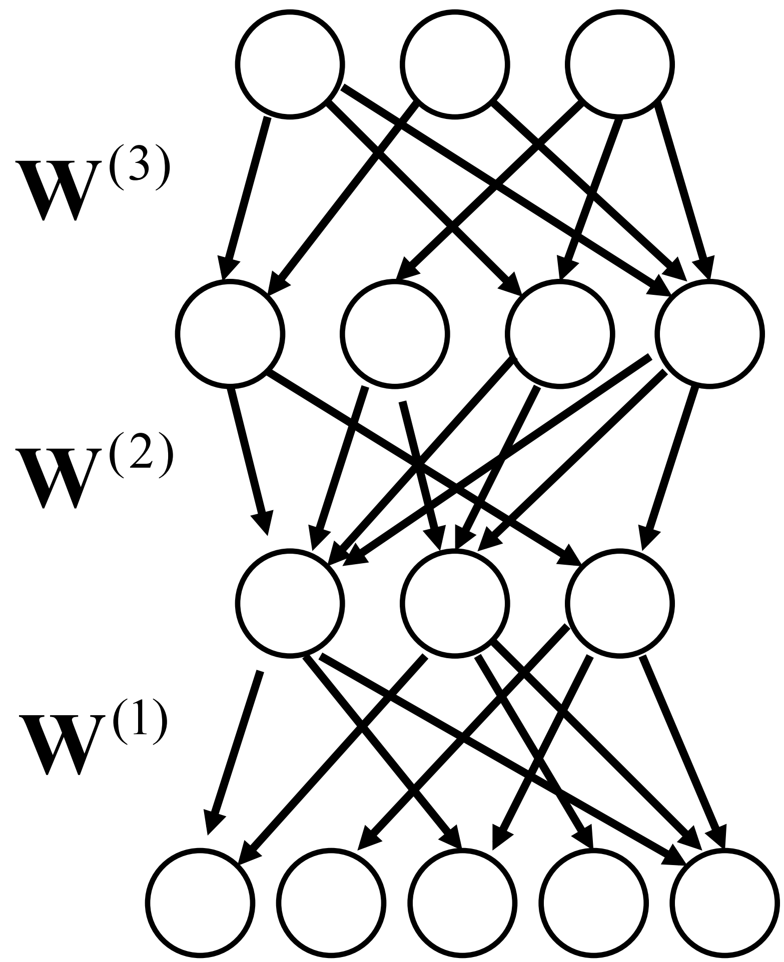
- Start assuming the weights are tied at all levels => the DBN is equivalent to an RBM
- We can train the DBN easily using the RBM training rule
- Fix the weights in the first layer, use the output of the first layer as the input for the next, and use the same trick
- And so on …



Greedy learning

- Greedy learning or “learning by re-representing”
- Guaranteed to improve the model only until the second layer
- Works fairly well in practice

- After this greedy phase, one should proceed to a global fine-tuning of the weights



Demos

- Generating walks
- Generating digits

Hands-on

- I'll show how to write a DBN model for hand-written digits
- Greedy phase only
- Ingredients: RBM, RBM with labels