Artificial Neural Networks
Lecture 9: Generative Models

Tudor Berariu
tudor.berariu@gmail.com

Faculty of Automatic Control and Computers
University Politehnica of Bucharest

Lecture : 16th of December, 2015
Last Updated: 16th of December, 2015

Course Progress

1. Introduction to Artificial Neural Networks
2. Linear Discriminants; The Perceptron Algorithm
3. Feedforward Neural Networks; Backpropagation
4. Optimization Algorithms
5. Convolutional Neural Networks
6. Radial Basis Function Networks
7. Reinforcement Learning; Recurrent Neural Networks
8. Networks with external memory
9. **Energy-based models (Generative models)**

Today's Outline

Today's Outline

1. Energy-based models
   - Hopfield Networks
   - Hopfield Nets with Stochastic Units
   - Hopfield Nets with Hidden Units
2. Boltzmann Machines
3. Sigmoid Belief Networks
4. Deep Belief Networks

Hopfield Networks

Definition

A Hopfield network is a recurrent fully-connected neural network with symmetric connections and no link from a neuron to itself.

Its units are McCulloch-Pitts bipolar neurons (states are -1 or 1) or binay neurons (states are 0 or 1).

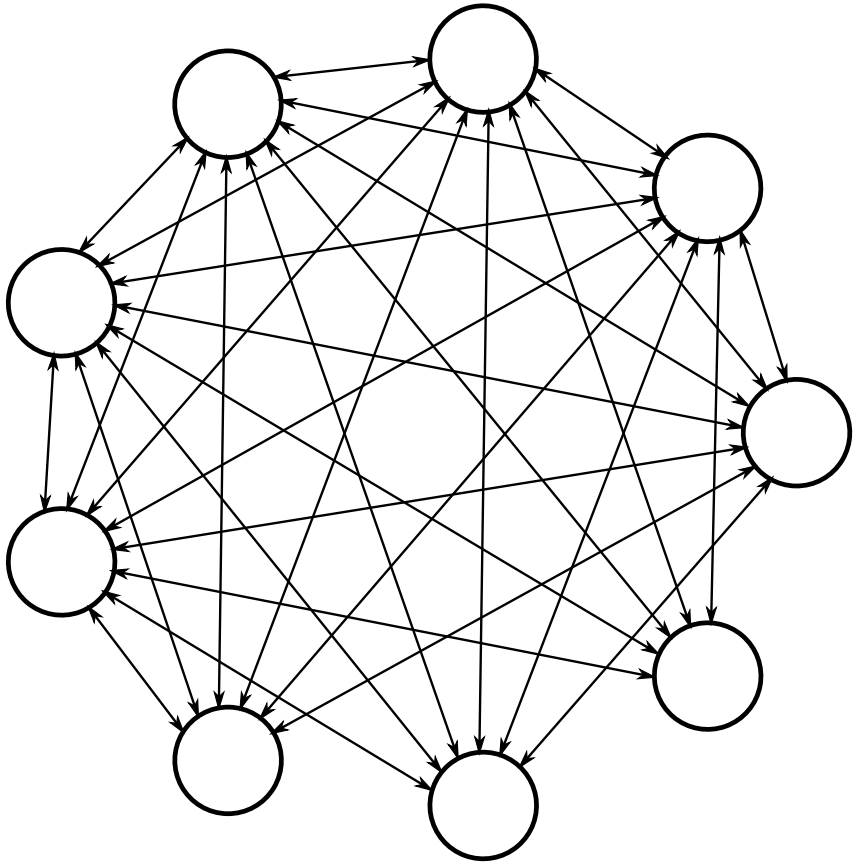


Figure: A Hopfield Network

Hopfield Networks

A Hopfield network is an energy-based model.

Each binary configuration has an energy:

$$E = -\frac{1}{2}\sum_{i}\sum_{j\neq i} x_i w_{ij} x_j$$

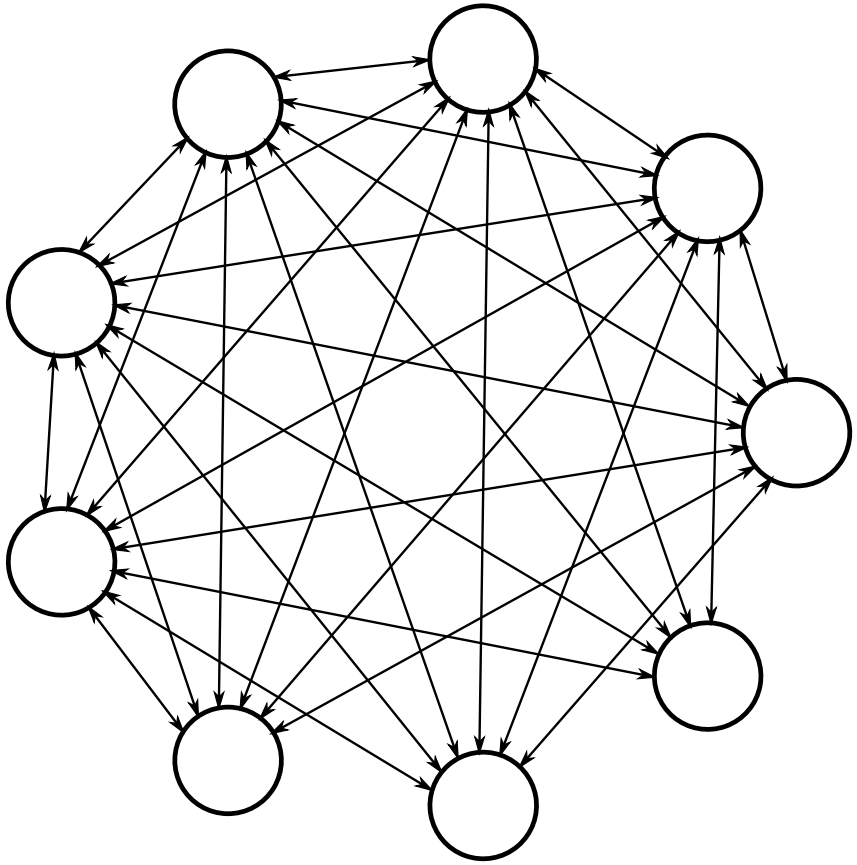


Figure: A Hopfield Network

Hopfield Networks

Sequentially applying the binary threshold decision rule settles the network to an energy minimum.

$$x_i \longleftarrow \begin{cases} 1 & \text{if } \sum_{j \neq i} x_j w_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}$$

(each neuron settles to the state which gives a lower energy)

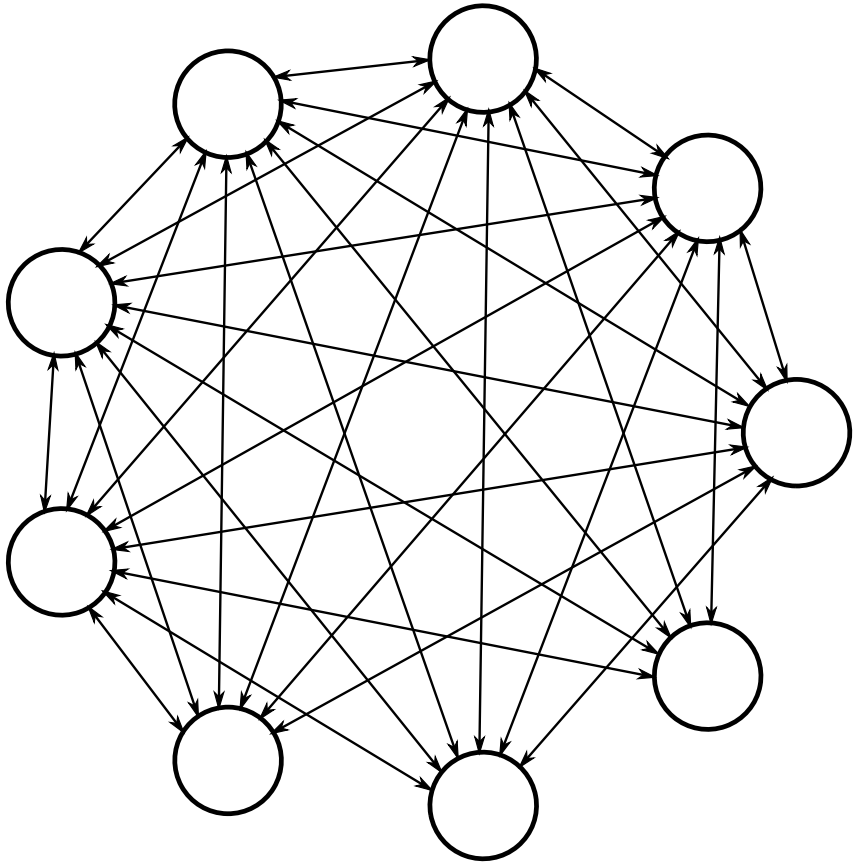


Figure: A Hopfield Network

Hopfield Networks

Proof:

Remember the update rule:

$$x_i \longleftarrow \begin{cases} 1 & \text{if } \sum_{j \neq i} x_j w_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Considering all $x_j, j \neq i$ fixed:

$$\begin{aligned} \Delta E_{x_i:0\to1} &= E_{x_i=1} - E_{x_i=0} \\ &= -\frac{1}{2} \sum_{j \neq i} w_{ij} x_j \end{aligned}$$

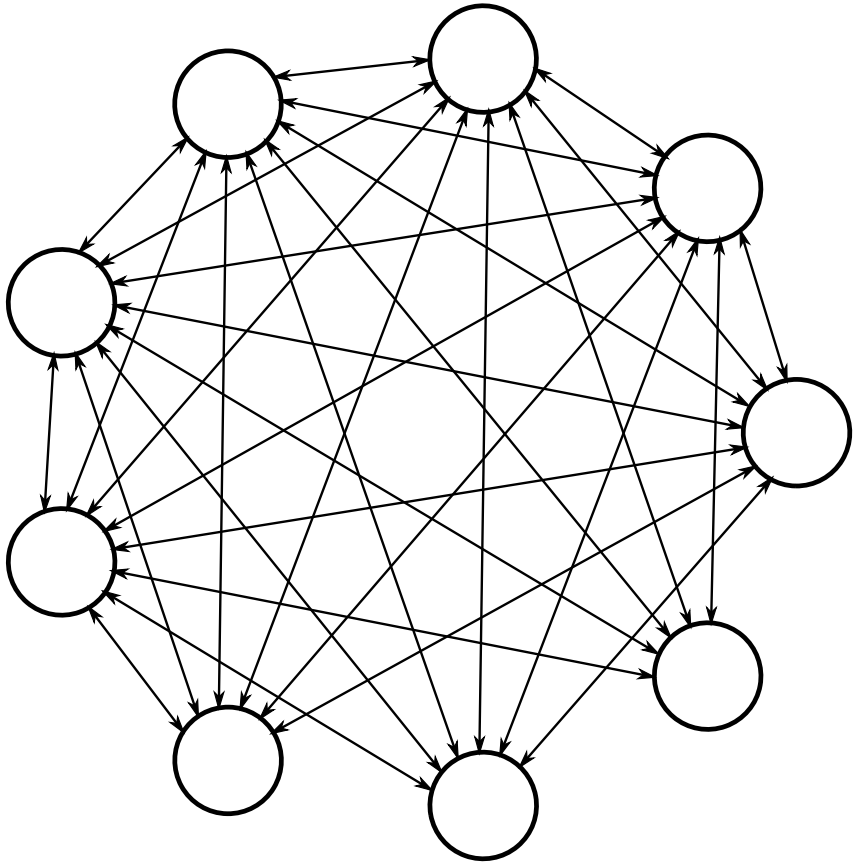


Figure: A Hopfield Network

Hopfield Networks

Proof:
Remember the update rule:

$$x_i \longleftarrow \begin{cases} 1 & \text{if } \sum_{j \neq i} x_j w_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Considering all $x_j, j \neq i$ fixed:

$$\begin{aligned} \Delta E_{x_i:0\to1} &= E_{x_i=1} - E_{x_i=0} \\ &= -\frac{1}{2} \sum_{j \neq i} w_{ij} x_j \end{aligned}$$

Each stochastic update lowers the global energy.

There are N neurons, hence 2^N possible configurations.

Conclusion: A Hopfield network with N units and asynchronous dynamics, which starts from any given network state, eventually reaches a stable state at a local minimum of the energy function.

Hopfield Networks

Patterns can be stored in local minima using Hebbian learning

$$\Delta w_{ij} = x_i x_j$$

Pattern example:

```
__xxxxxxxx__
__xxxxxxxxx_
_______xxx_
_______xxx_
___xxxxxxx__
___xxxxxxx__
_______xxx_
_______xxx_
__xxxxxxxxx_
__xxxxxxxx__
```

Hopfield Networks

Patterns can be stored in local minima using Hebbian learning

$$\Delta w_{ij} = x_i x_j$$

Given some noisy input, the network will reconstruct the original pattern by settling to a local minimum.

Hopfield networks are autoassociative memories.

Correct	back to:
x_xxxxx_xx__	__xxxxxxxx__
__x__xxxxxx_	__xxxxxxxxx_
x___x__xxxx_	_____xxx_
_____xxx_	_____xxx_
___xxxxxxx__	___xxxxxxx__
x__xxxx_xx__	___xxxxxxx__
_____xxx_	_____xxx_
__x_xx__xxx_	_____xxx_
__xxxxxxxxx_	__xxxxxxxxx_
__xxxxxx____	__xxxxxxxx__

Learning algorithm for Hopfield

Given M patterns $\mathbf{s}_1, \ldots, \mathbf{s}_M$ of size N:

$$\mathbf{W} = \sum_{i=1}^{M} \mathbf{s}^i \cdot (\mathbf{s}^i)^T \; - \; M \cdot \mathbf{I} \tag{1}$$

Important: $w_{ii} = 0, \forall i \in \{1 \ldots N\}$.

Recovering patterns

Given some noisy pattern $\hat{\mathbf{s}}$:

1. $\mathbf{x} \longleftarrow \hat{\mathbf{s}}$
2. While $\mathbf{x}$ not in a local minimum:
   - Choose i at random from $\{1, \ldots, N\}$
   - Update x_i:

$$x_i \longleftarrow \begin{cases} 1 & \text{if } \sum_{j \neq i} x_j w_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}$$

3. return $\mathbf{x}$

Spurious minima

- The capacity of a Hopfield network is about $0.15N$ memories where N is the number of neurons.
- If M exceeds $0.15N$, the network converges to a local minimum different from the learned patterns (called spurious minima).
- Improvement: **unlearning** [HFP83]

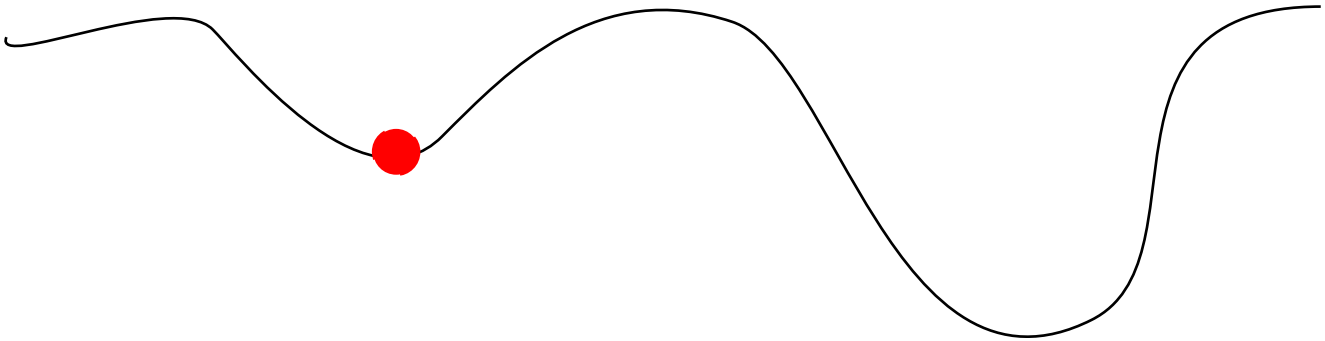
$$\Delta w_{ij} = -s_i s_j$$

where $\mathbf{s}$ is a spurious minima.

Today's Outline

Stochastic units

- A Hopfield network cannot escape local minima.
- Stochastic units might jump over energy hills.



Stochastic units

- A unit i will go to state 0 with probability $p_i(0)$ and to state 1 with probabilit $p_i(1)$

$$\frac{p_i(1)}{p_i(0)} = \frac{e^{-\frac{E_i(1)}{T}}}{e^{-\frac{E_i(0)}{T}}}$$

- If $\Delta E_i = E_i(1) - E_i(0)$:

$$p_i(1) = \frac{1}{1 + e^{\frac{\Delta E_i}{T}}}$$

where T is the *temperature*.

Influence of temperature

$$p_i(1) = \frac{1}{1 + e^{\frac{\Delta E_i}{T}}}$$

- the system will usually go to lowest energy states
- sometimes it will jump to higher energy states
- the temperature controls this behavior

Influence of temperature

$$p_i(1) = \frac{1}{1 + e^{\frac{\Delta E_i}{T}}}$$

- if $T = 0$, the neuron becomes deterministic:
  - $p_i(1) = 1$ if $\Delta E_i < 0$ $(E_i(1) < E_i(0))$
  - $p_i(1) = 0$ if $\Delta E_i > 0$

Influence of temperature

$$p_i(1) = \frac{1}{1 + e^{\frac{\Delta E_i}{T}}}$$

- at high temperatures probabilities tend to get closer to 0.5
- at small temperatures probabilities tend to get closer to 0 or 1

T	$1/4$	$1/2$	1	2	4	8	16	32
$p_i(1)$	1.0	0.999	0.993	0.924	0.777	0.651	0.577	0.538

Table: Transition probabilities for $\Delta E = 5$

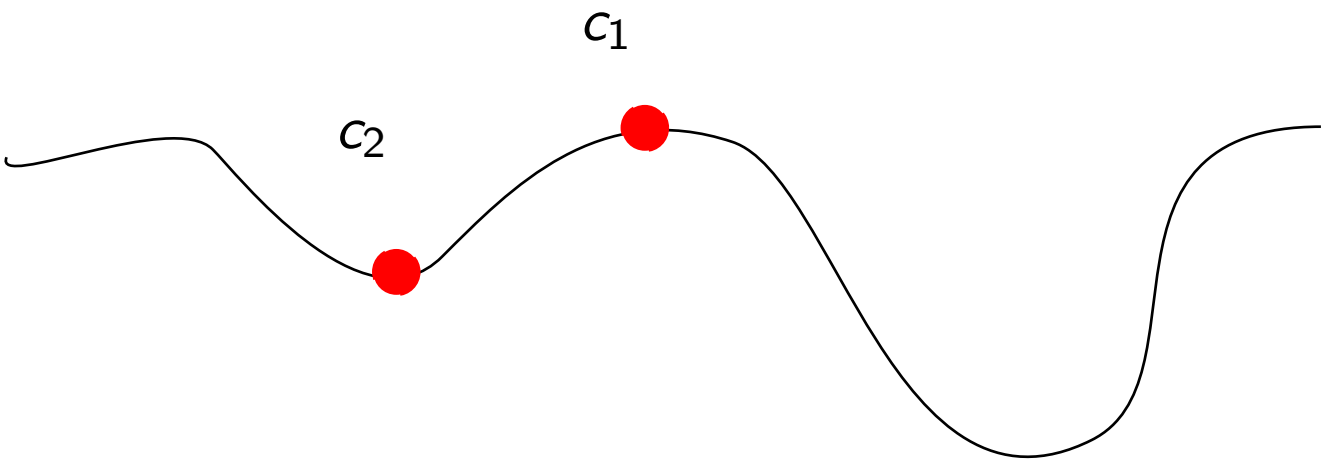
Influence of temperature

$$p(c_1 \to c_2) = \frac{1}{1 + e^{\frac{E(c_2) - E(c_1)}{T}}}$$

- transition from configuration c_1 to configuration c_2

T	$1/4$	$1/2$	1	2	4	8	16	32
$p(c_1 \to c_2)$	1.0	0.999	0.993	0.924	0.777	0.651	0.577	0.538
$p(c_2 \to c_1)$	2.06e-9	4.5e-5	0.006	0.0758	0.222	0.348	0.422	0.461

Table: Transition probabilities for $\Delta E = E(c_1) - E(c_2) = -5$



Simulated annealing

- using a high temperature will make the network evolve over a large set of sets
- using a low temperature will need a large amount of time to escape local minima
- **simulated annealing:** start with a high temperature and decrease it as the network settles to a minimum

Thermal Equilibrium at a fixed temperature

Definition

A network reaches **thermal equilibrium** when the average distribution of configuration over time doesn't change anymore.
(stationary distribution)

- reaching thermal equilibrium doesn't mean the configuration doesn't change anymore

Today's Outline

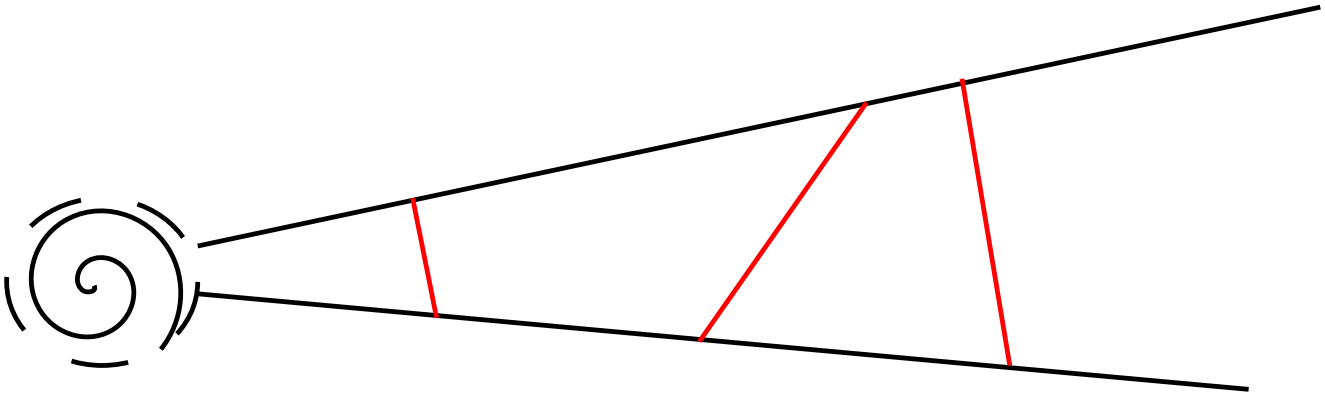
Why more units?

- as the learning rule involve only $x_i x_j$ products, the model is not able to capture relations higher than second order
- adding some units might extract higher order features

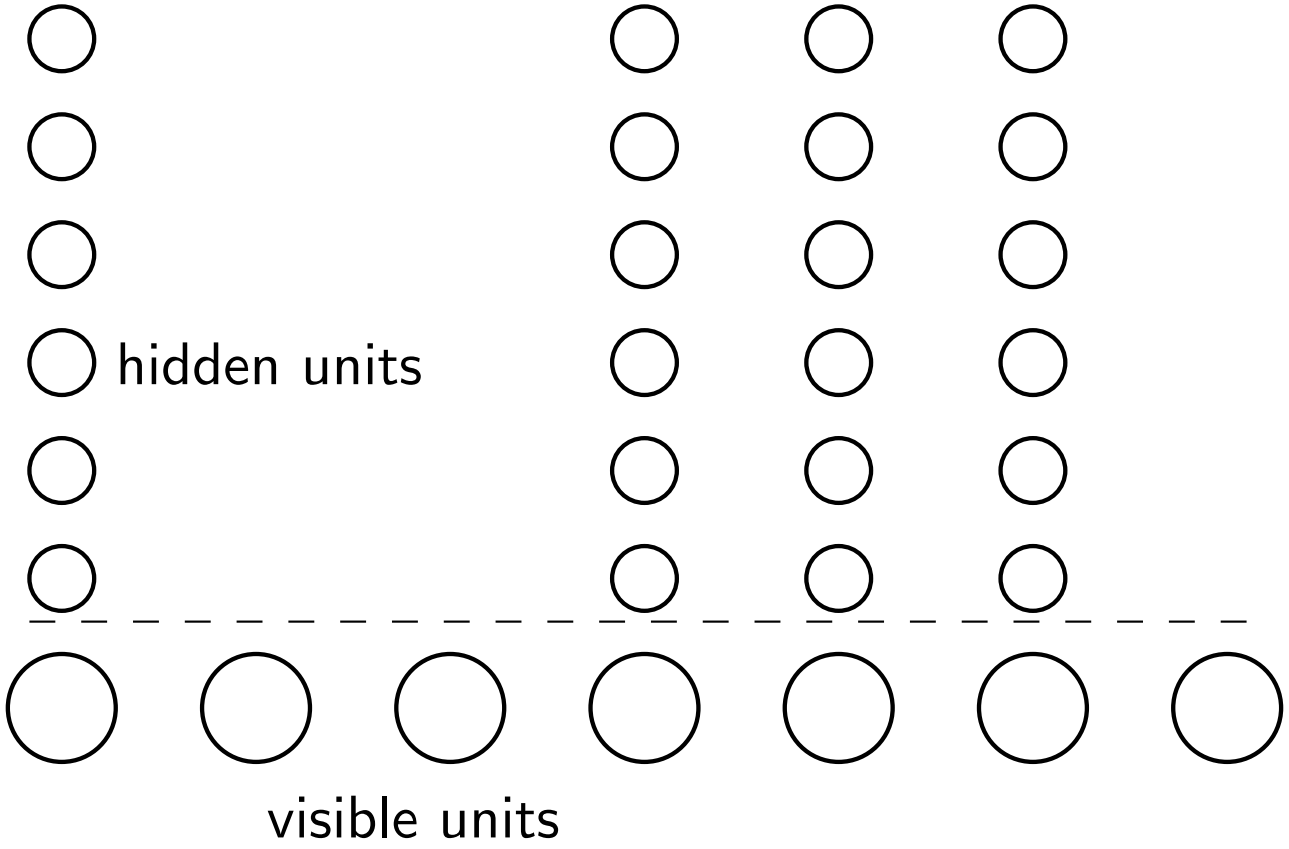
Building interpretations of the visual input

- hidden units might represent higher order interpretations of the visual inputs (G. Hinton)
- weights represent constraints between interpretations
- energy represents *badness* of the interpretation

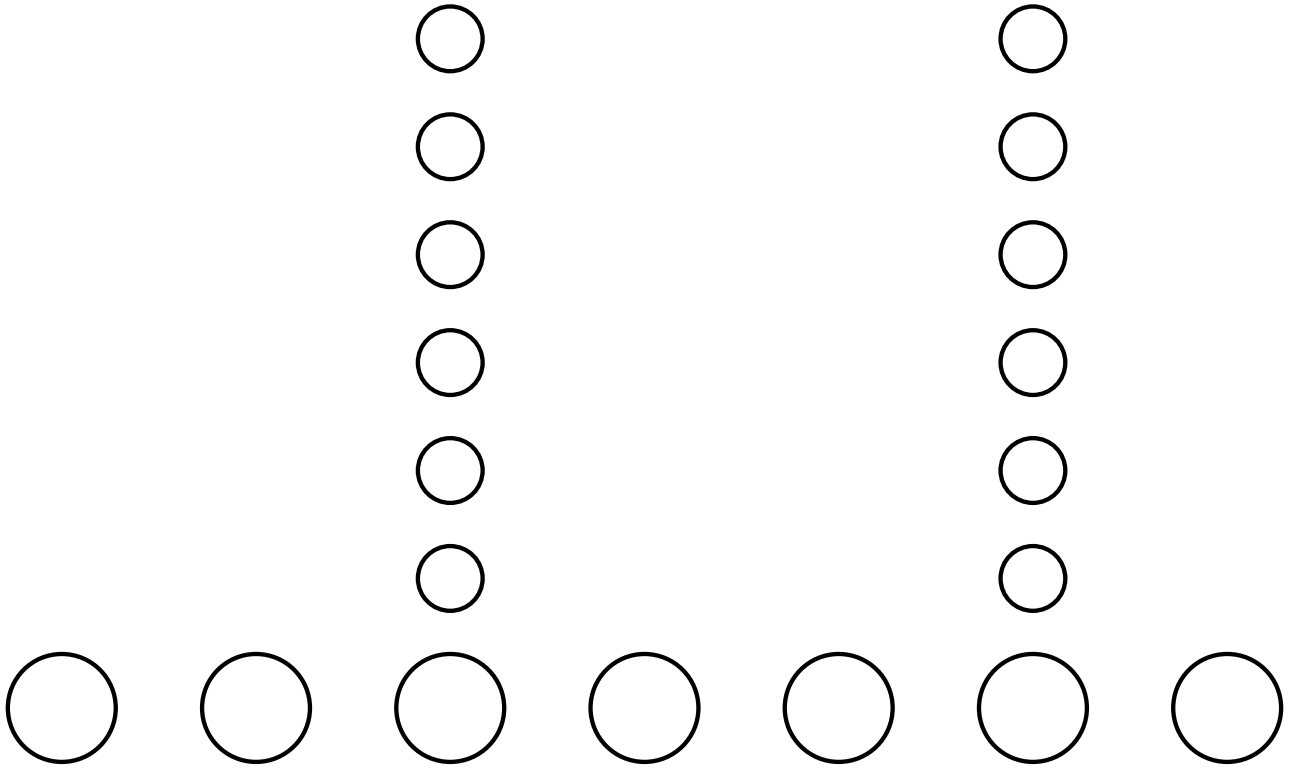
Lost information in vision



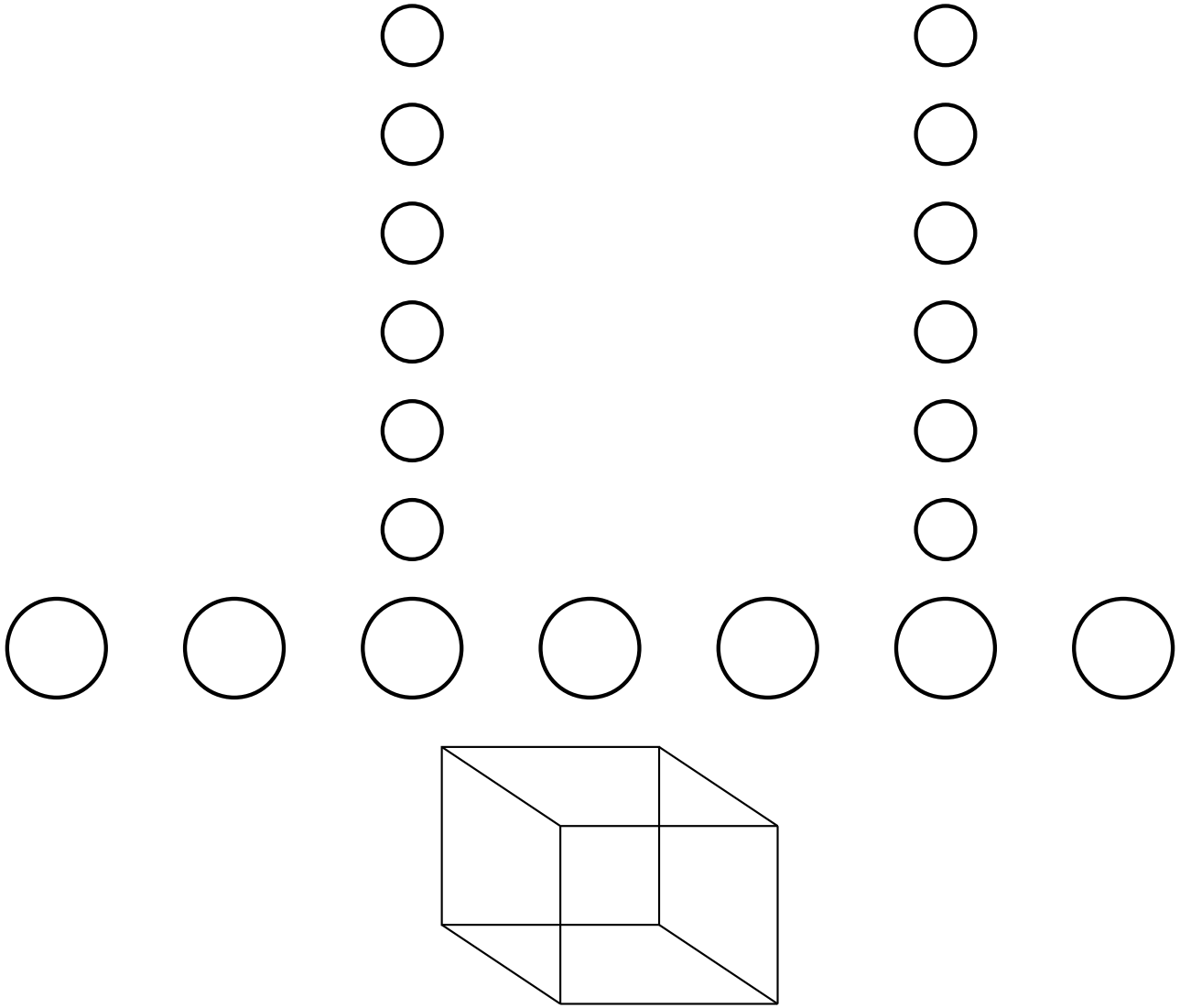
Hidden Units as Interpretations (J. Hinton)



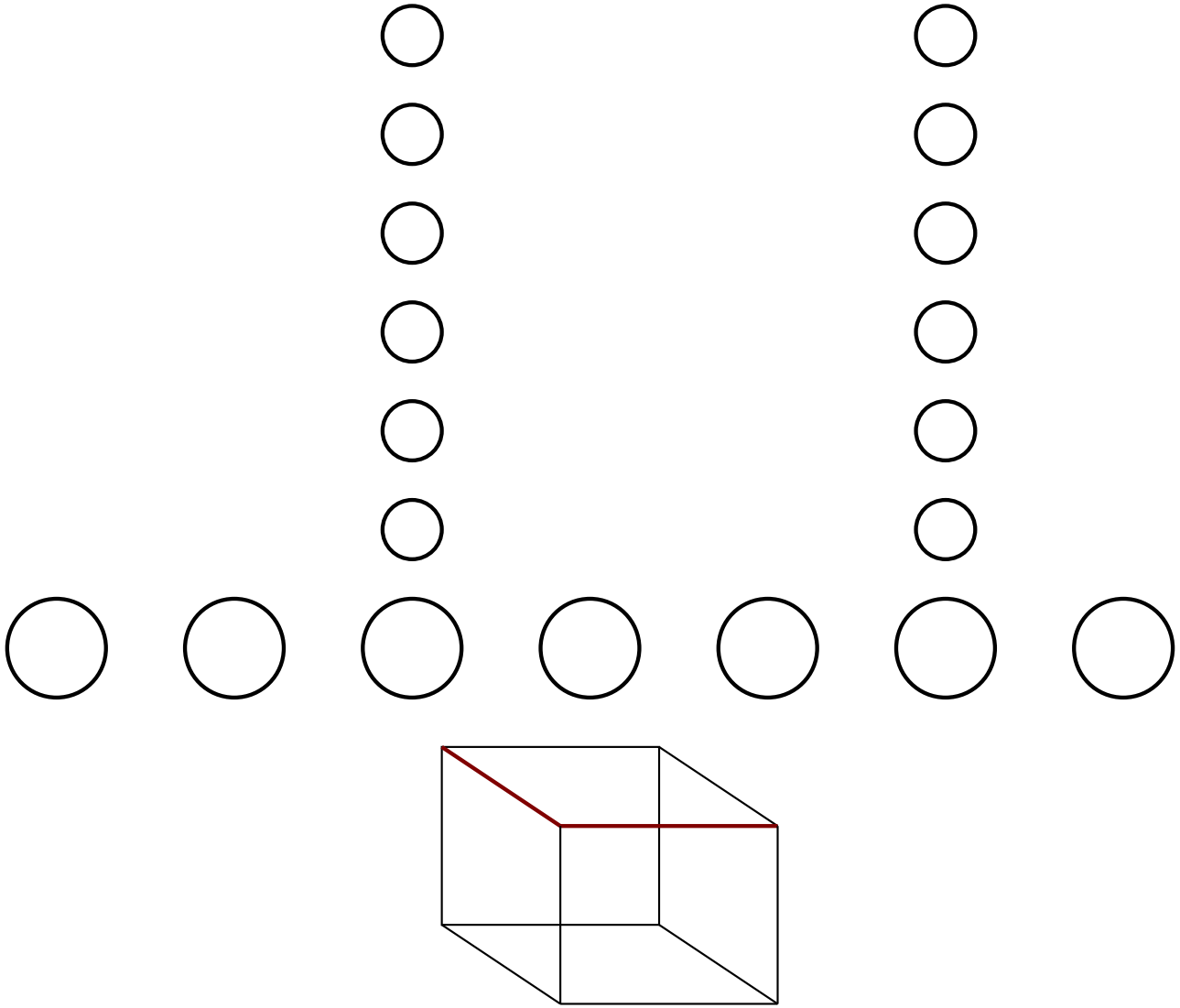
Hidden Units as Interpretations (J. Hinton)



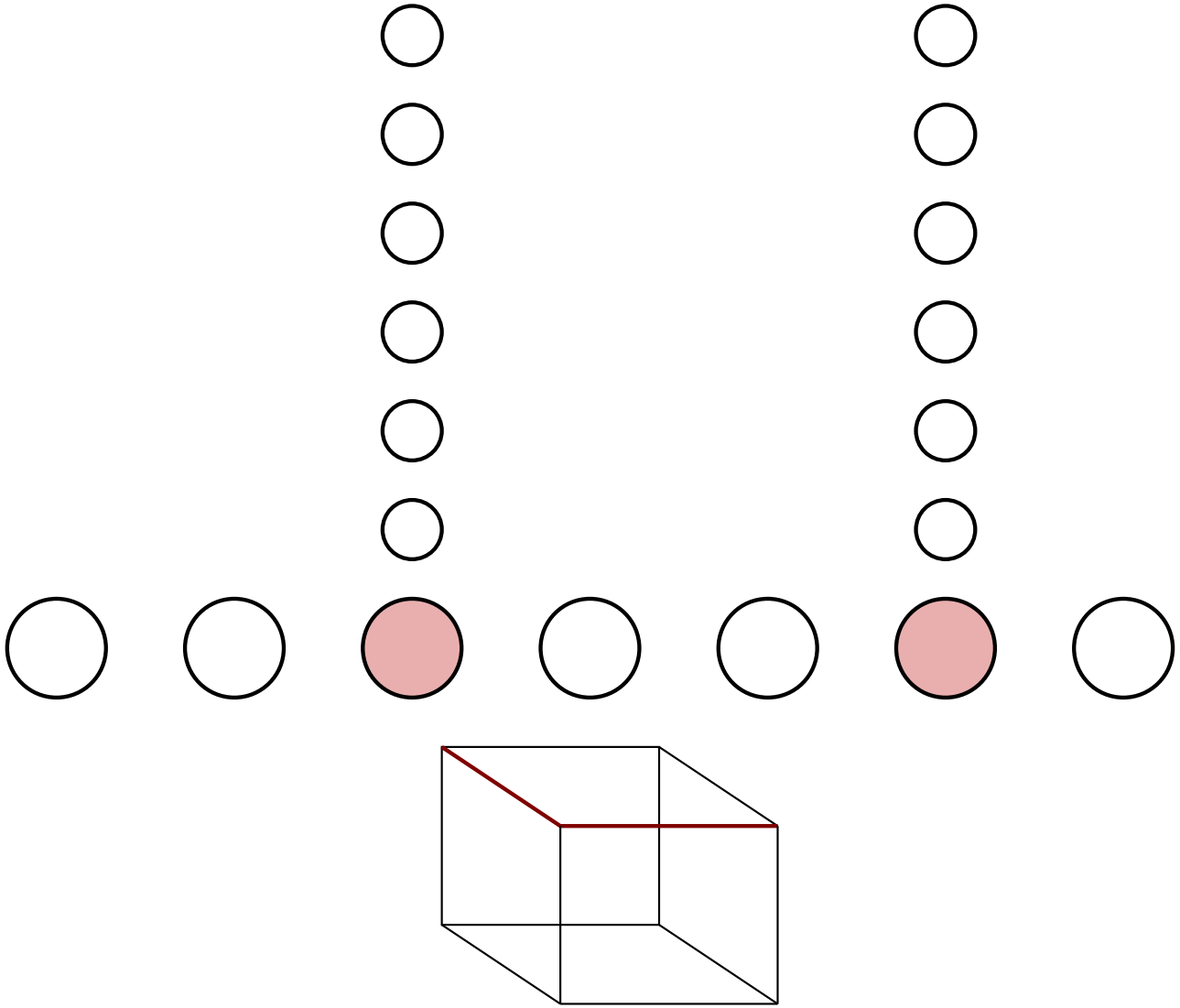
Hidden Units as Interpretations (J. Hinton)



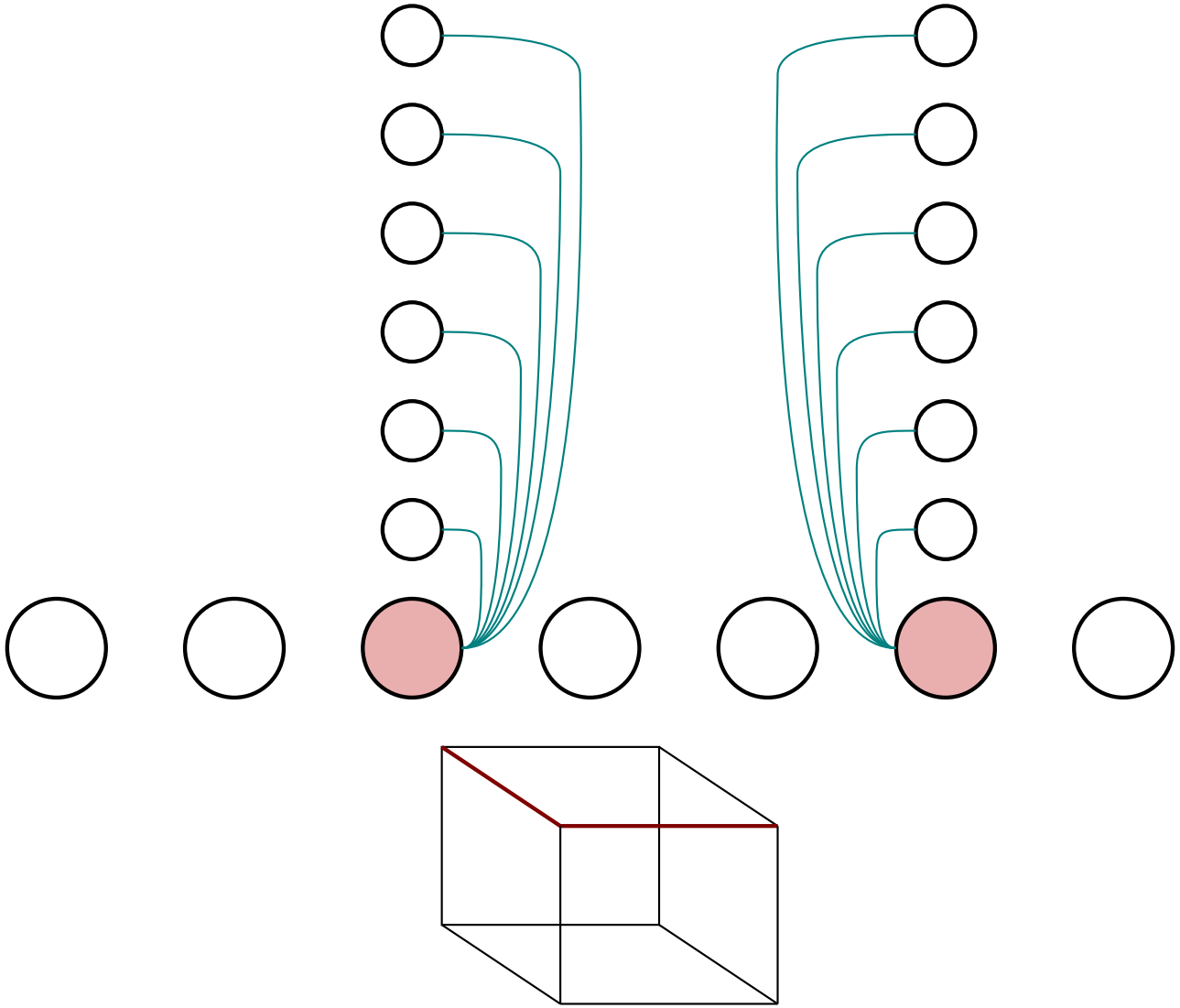
Hidden Units as Interpretations (J. Hinton)



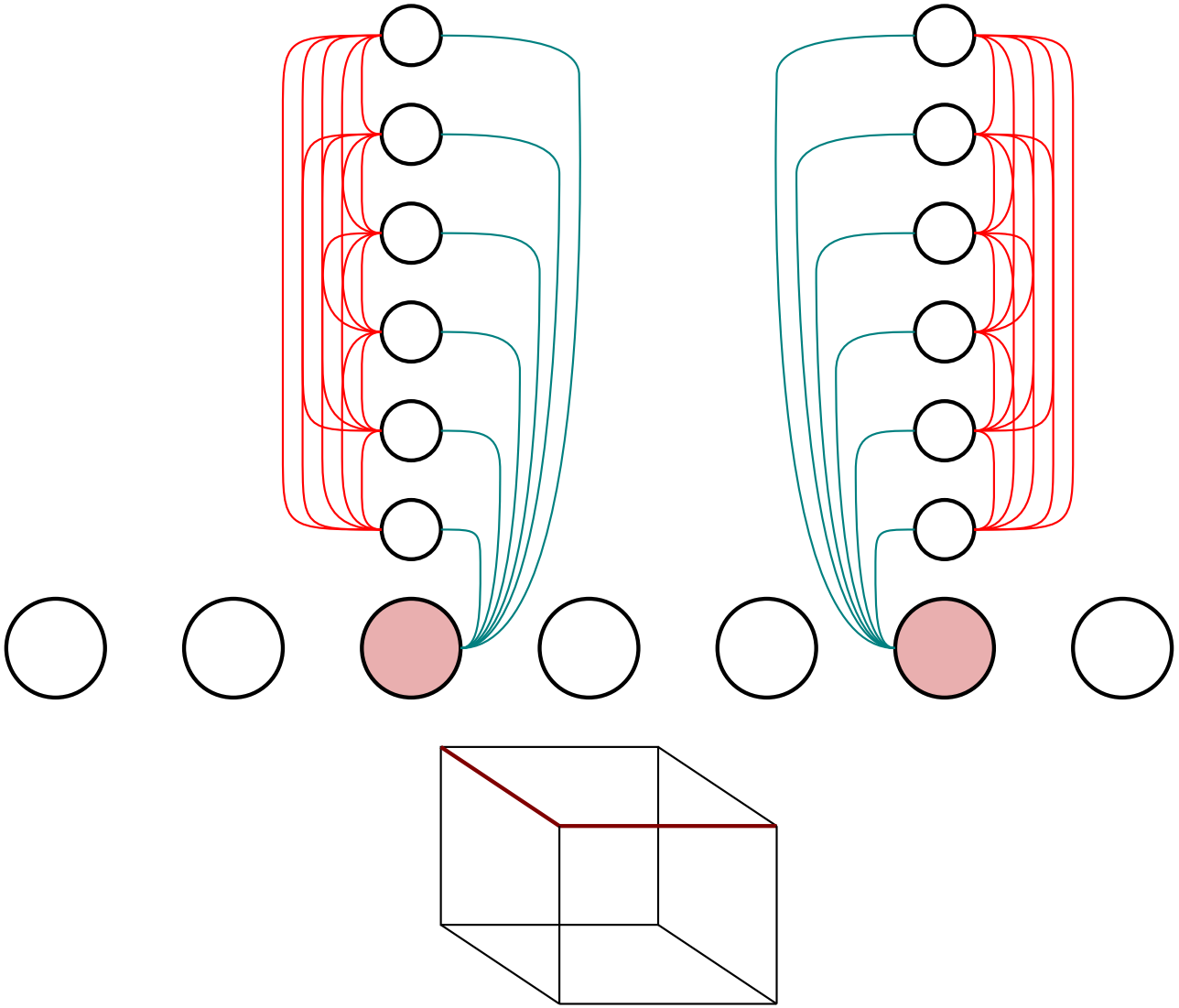
Hidden Units as Interpretations (J. Hinton)



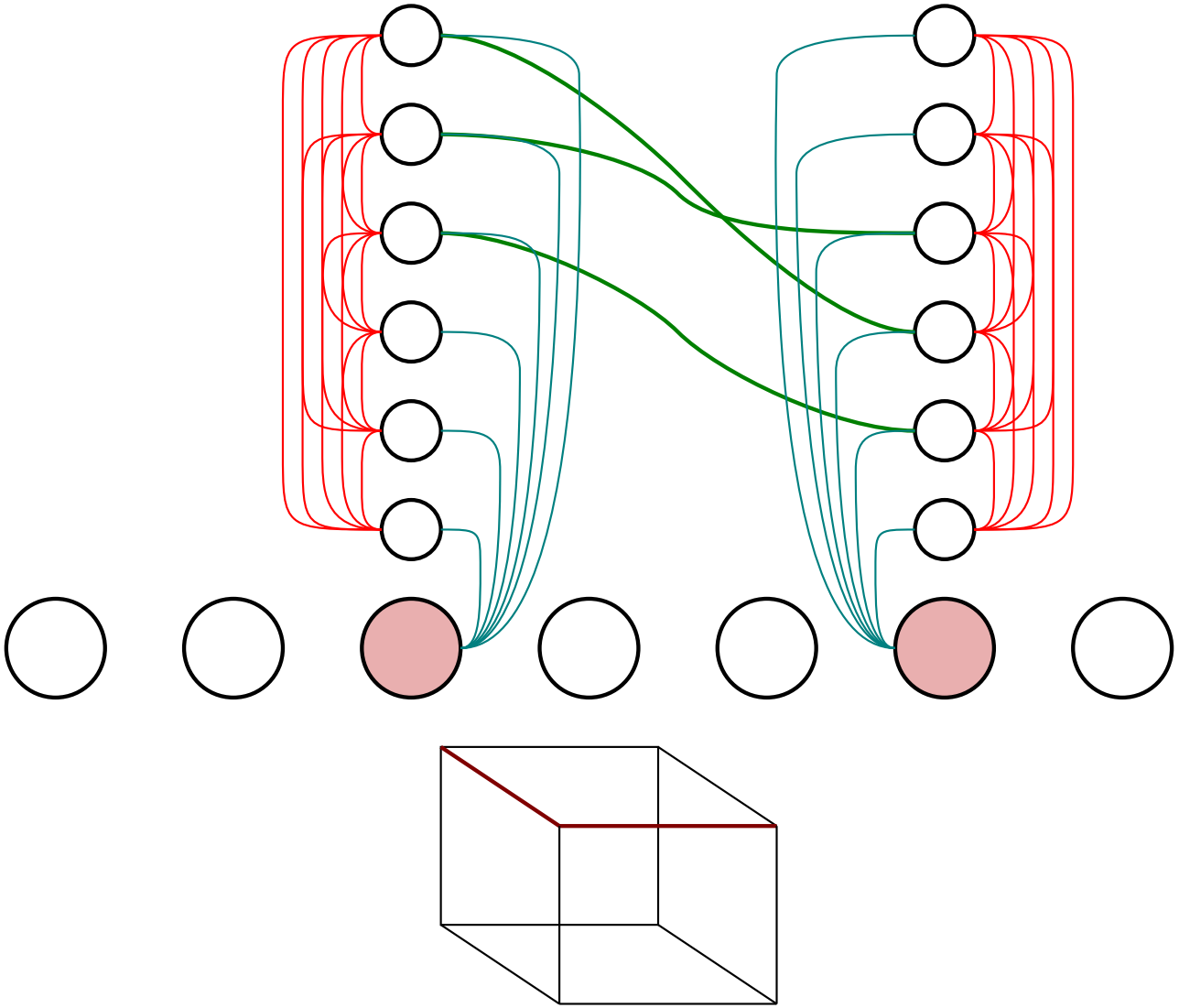
Hidden Units as Interpretations (J. Hinton)



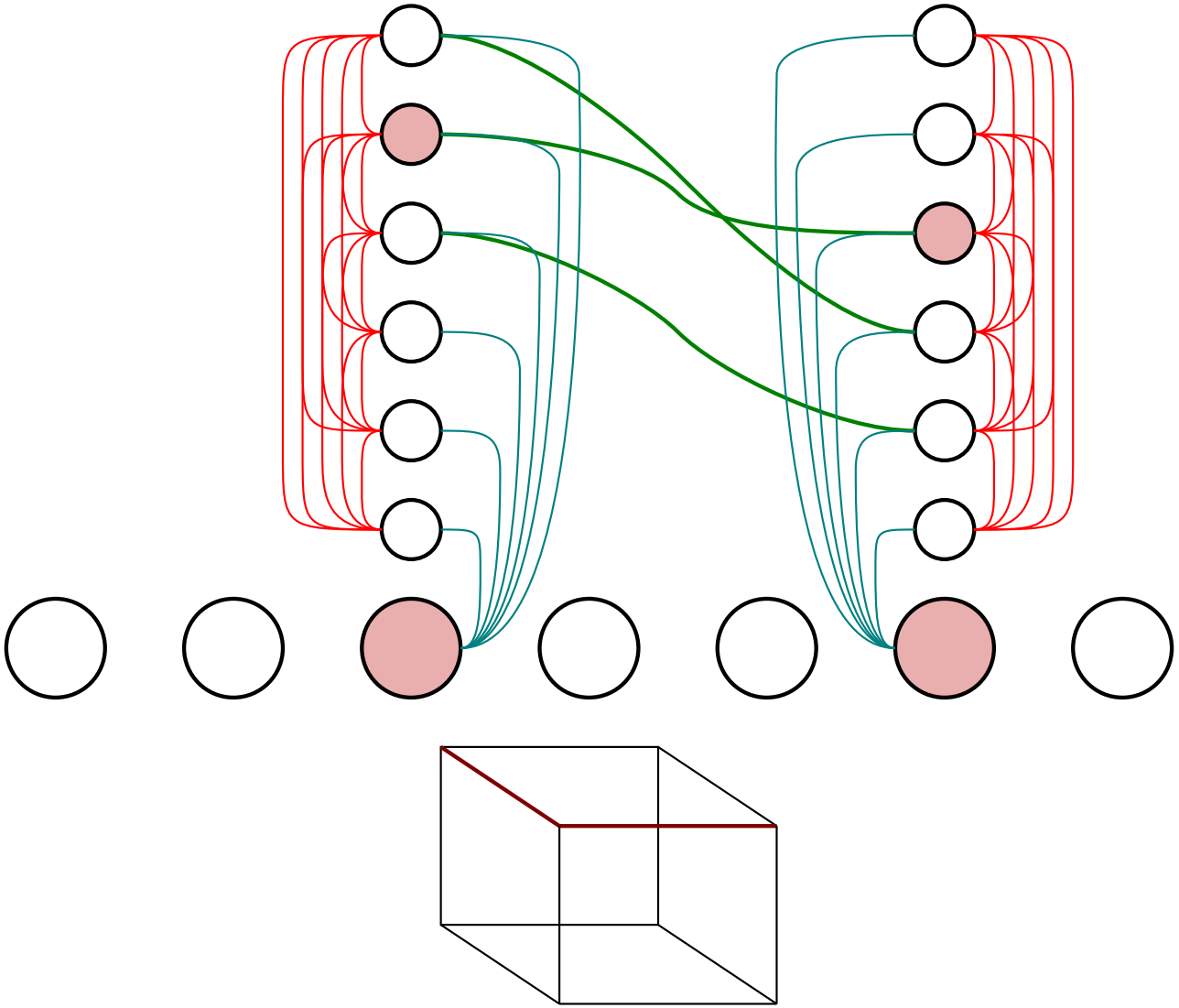
Hidden Units as Interpretations (J. Hinton)



Hidden Units as Interpretations (J. Hinton)



Hidden Units as Interpretations (J. Hinton)



Learning weights

- this is an unsupervised learning task
- learning the weights between the hidden units and the visible units seems quite hard

Today's Outline

Today's Outline

1. Energy-based models
2. Boltzmann Machines
   - Definition
   - Boltzmann Machine learning algorithm
   - Restricted Boltzmann Machines
3. Sigmoid Belief Networks
4. Deep Belief Networks

Goal

- Build a model of some data set that consists of binary vectors with two goals in mind:
  1. the model should be able to *generate* binary vectors from the same distribution as the training data set;
  2. the model should be able to assign a probability to any given binary vector.

Boltzmann Machines

Definition

Boltzmann Machines are stochastic Hopfield networks with hidden units.

Boltzmann Machines are energy-based models, so they define the probability distribution of their states through a energy function:

$$P(\mathbf{x}) = \frac{e^{-E(\mathbf{x})}}{Z} \tag{2}$$

Boltzmann Machines are universal approximators of probability mass functions.

How to get probabilities?

- the probability of a configuration is given by the energy of the joint configuration (visible and hidden units):

$$p(\mathbf{v}, \mathbf{h}) \propto e^{-E(\mathbf{v},\mathbf{h})}$$

- the probability of a configuration is given by the probability to find the network in that state $(\mathbf{v}, \mathbf{h})$ after reaching thermal equilibrium

Energy of a joint configuration

$$
\begin{aligned}
E(\mathbf{v}, \mathbf{h}) \quad = \quad & -\sum_{i \in \mathcal{V}} b_i^{(\mathcal{V})} v_i - \frac{1}{2} \sum_{i,j \in \mathcal{V}} v_i v_j w_{ij}^{(\mathcal{V})} \\
& -\sum_{k \in \mathcal{H}} b_k^{(\mathcal{H})} h_k - \frac{1}{2} \sum_{k,l \in \mathcal{H}} h_k h_l w_{kl}^{(\mathcal{H})} \\
& -\frac{1}{2} \sum_{i \in \mathcal{V}, k \in \mathcal{H}} v_i h_k w_{ik}
\end{aligned}
$$

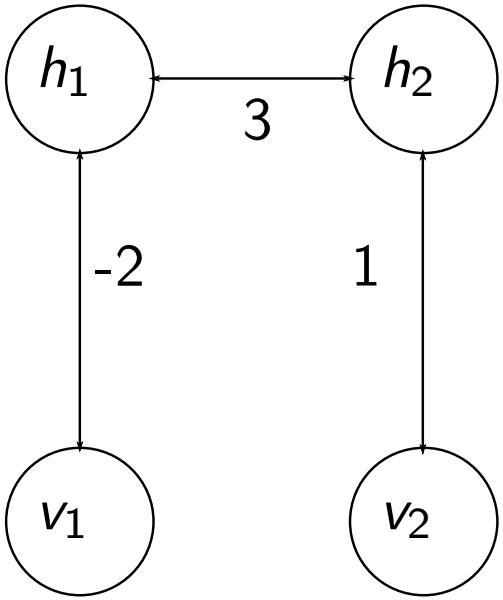
Probabilities

$$p(\mathbf{v}, \mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{w},\mathbf{k}} e^{-E(\mathbf{w},\mathbf{k})}}$$

$$p(\mathbf{v}) = \frac{\sum_{\mathbf{k}} e^{-E(\mathbf{v},\mathbf{k})}}{\sum_{\mathbf{w},\mathbf{k}} e^{-E(\mathbf{w},\mathbf{k})}}$$

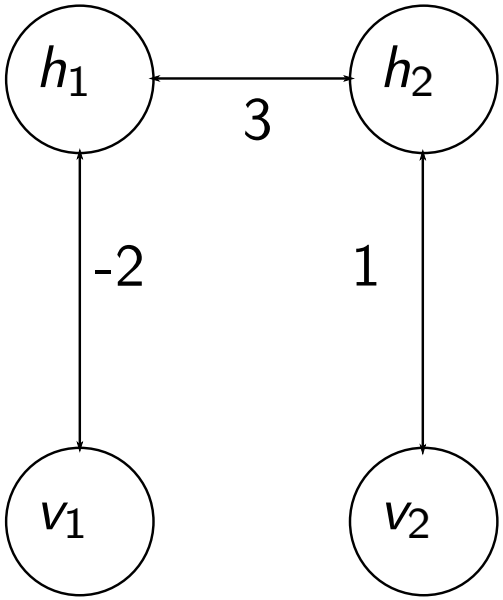
An example

v_1	v_2	h_1	h_2	E	e^{-E}	$p(\mathbf{v}, \mathbf{h})$	$p(\mathbf{v})$
1	1						
1	1						
1	1						
1	1						
1	0						
1	0						
1	0						
1	0						
0	1						
0	1						
0	1						
0	1						
0	0						
0	0						
0	0						
0	0						



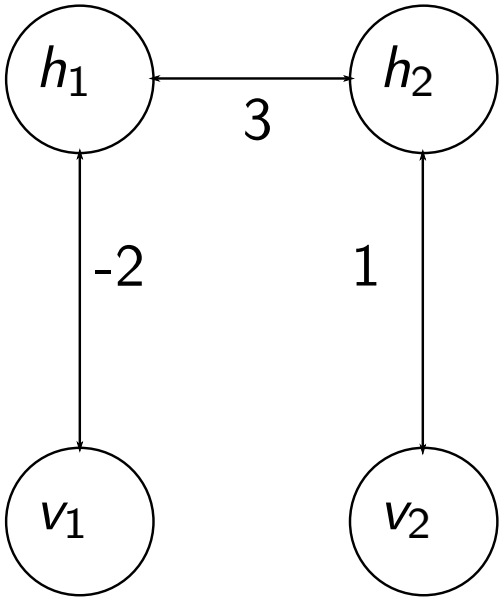
An example

v_1	v_2	h_1	h_2	E	e^{-E}	$p(\mathbf{v}, \mathbf{h})$	$p(\mathbf{v})$
1	1	1	1				
1	1	1	0				
1	1	0	1				
1	1	0	0				
1	0	1	1				
1	0	1	0				
1	0	0	1				
1	0	0	0				
0	1	1	1				
0	1	1	0				
0	1	0	1				
0	1	0	0				
0	0	1	1				
0	0	1	0				
0	0	0	1				
0	0	0	0				



An example

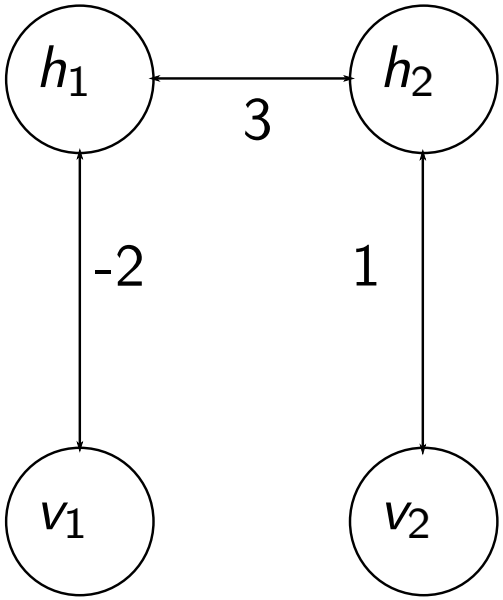
v_1	v_2	h_1	h_2	E	e^{-E}	$p(\mathbf{v},\mathbf{h})$	$p(\mathbf{v})$
1	1	1	1	-2			
1	1	1	0				
1	1	0	1				
1	1	0	0				
1	0	1	1				
1	0	1	0				
1	0	0	1				
1	0	0	0				
0	1	1	1				
0	1	1	0				
0	1	0	1				
0	1	0	0				
0	0	1	1				
0	0	1	0				
0	0	0	1				
0	0	0	0				



$$
\begin{aligned}
E(1,1) &= -v_1 h_1 * -2 \\
&\quad -h_1 h_2 * 3 \\
&\quad -v_2 h_2 * 1 \\
&= 2 - 3 - 1 \\
&= 1
\end{aligned}
$$

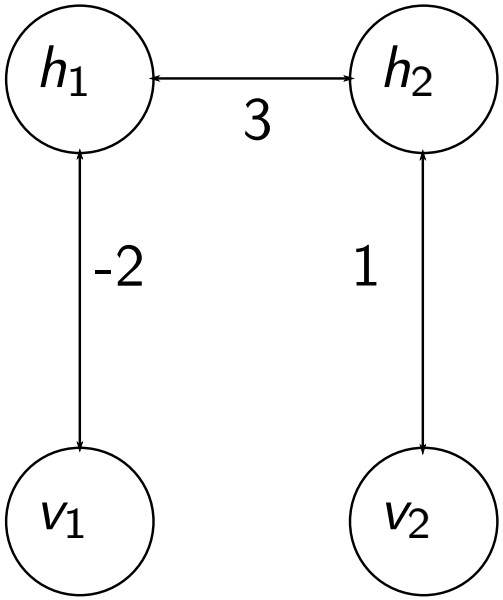
An example

v_1	v_2	h_1	h_2	E	e^{-E}	$p(\mathbf{v},\mathbf{h})$	$p(\mathbf{v})$
1	1	1	1	-2			
1	1	1	0	2			
1	1	0	1	-1			
1	1	0	0	0			
1	0	1	1	-1			
1	0	1	0	2			
1	0	0	1	0			
1	0	0	0	0			
0	1	1	1	-4			
0	1	1	0	0			
0	1	0	1	-1			
0	1	0	0	0			
0	0	1	1	-3			
0	0	1	0	0			
0	0	0	1	0			
0	0	0	0	0			



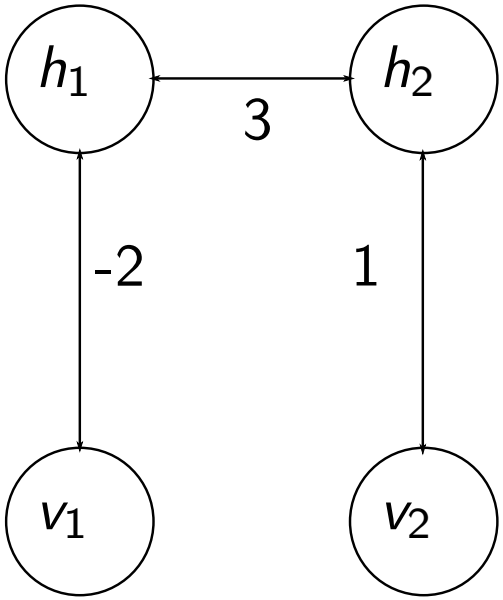
An example

v_1	v_2	h_1	h_2	E	e^{-E}	$p(\mathbf{v},\mathbf{h})$	$p(\mathbf{v})$
1	1	1	1	-2	7.389		
1	1	1	0	2	0.135		
1	1	0	1	-1	2.718		
1	1	0	0	0	1		
1	0	1	1	-1	2.718		
1	0	1	0	2	0.135		
1	0	0	1	0	1		
1	0	0	0	0	1		
0	1	1	1	-4	54.598		
0	1	1	0	0	1		
0	1	0	1	-1	2.718		
0	1	0	0	0	1		
0	0	1	1	-3	20.085		
0	0	1	0	0	1		
0	0	0	1	0	1		
0	0	0	0	0	1		



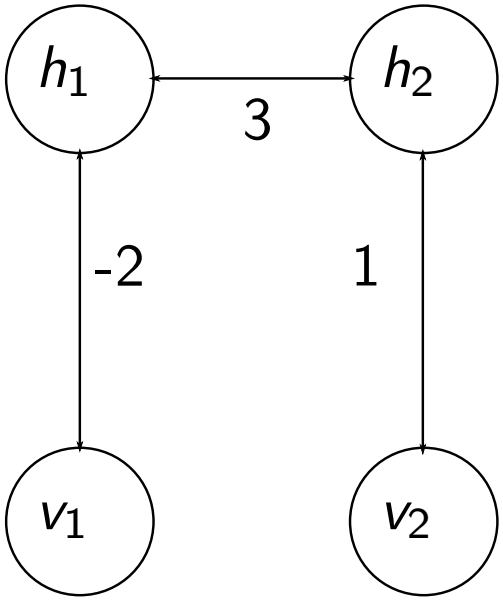
An example

v_1	v_2	h_1	h_2	E	e^{-E}	$p(\mathbf{v},\mathbf{h})$	$p(\mathbf{v})$
1	1	1	1	-2	7.389	0.075	
1	1	1	0	2	0.135	0.001	
1	1	0	1	-1	2.718	0.027	
1	1	0	0	0	1	0.010	
1	0	1	1	-1	2.718	0.027	
1	0	1	0	2	0.135	0.0013	
1	0	0	1	0	1	0.010	
1	0	0	0	0	1	0.010	
0	1	1	1	-4	54.598	0.554	
0	1	1	0	0	1	0.010	
0	1	0	1	-1	2.718	0.0275	
0	1	0	0	0	1	0.010	
0	0	1	1	-3	20.085	0.203	
0	0	1	0	0	1	0.010	
0	0	0	1	0	1	0.010	
0	0	0	0	0	1	0.010	



An example

v_1	v_2	h_1	h_2	E	e^{-E}	$p(\mathbf{v},\mathbf{h})$	$p(\mathbf{v})$
1	1	1	1	-2	7.389	0.075	0.1141
1	1	1	0	2	0.135	0.001	
1	1	0	1	-1	2.718	0.027	
1	1	0	0	0	1	0.010	
1	0	1	1	-1	2.718	0.027	0.0492
1	0	1	0	2	0.135	0.0013	
1	0	0	1	0	1	0.010	
1	0	0	0	0	1	0.010	
0	1	1	1	-4	54.598	0.554	0.6022
0	1	1	0	0	1	0.010	
0	1	0	1	-1	2.718	0.0275	
0	1	0	0	0	1	0.010	
0	0	1	1	-3	20.085	0.203	0.2343
0	0	1	0	0	1	0.010	
0	0	0	1	0	1	0.010	
0	0	0	0	0	1	0.010	



Computing probabilities

- the method is not feasible for large networks as the number of possible configurations grows exponentially
- probabilities can be obtained with Monte Carlo Markov Chain
  - let the network start from a random configuration and then reach thermal equilibrium $\longrightarrow p(\mathbf{v}, \mathbf{h})$
  - let the network start from a random configuration with the visible units *clamped* to $\mathbf{v}$ and then reach thermal equilibrium $\longrightarrow p(\mathbf{h}|\mathbf{v})$

Today's Outline

Learning in Boltzmann Machines

- Goal: learn a model of a set of input vectors.
- Maximize the product of probabilities the Boltzmann machine assigns to the vectors in the training set.

$$\text{maximize} \prod_{x^{(i)} \in S} p(\mathbf{x}^{(i)}) \equiv \text{maximize} \sum_{x^{(i)} \in S} \log p(\mathbf{x}^{(i)})$$

Learning Algorithm

- $p(\mathbf{v})$ at thermal equilibrium is proportional to e^{-E}
- $\log p(\mathbf{v})$ will be linear to E which is linear to the weights

Learning Algorithm

- $p(\mathbf{v})$ at thermal equilibrium is proportional to e^{-E}
- $\log p(\mathbf{v})$ will be linear to E which is linear to the weights

$$\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = \langle x_i x_j \rangle_{\mathbf{v}} - \langle x_i x_j \rangle_{model}$$

Learning Algorithm

$$\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = \langle x_i x_j \rangle_{\mathbf{v}} - \langle x_i x_j \rangle_{model}$$

where:

- $\langle x_i x_j \rangle_{\mathbf{v}}$ is the expected value of $x_i x_j$ at thermal equilibrium when visible units are *clamped* to $\mathbf{v}$

Learning Algorithm

$$\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = \langle x_i x_j \rangle_{\mathbf{v}} - \langle x_i x_j \rangle_{model}$$

where:

- $\langle x_i x_j \rangle_{\mathbf{v}}$ is the expected value of $x_i x_j$ at thermal equilibrium when visible units are *clamped* to $\mathbf{v}$
- $\langle x_i x_j \rangle_{model}$ is the expected value of $x_i x_j$ at thermal equilibrium

Learning rule

$$\Delta w_{ij} \propto \langle x_i x_j \rangle_{\mathbf{v}} - \langle x_i x_j \rangle_{model}$$

- $\langle x_i x_j \rangle_{\mathbf{v}}$ corresponds to the **positive phase** of learning
- $\langle x_i x_j \rangle_{model}$ corresponds to the **negative phase** of learning

Learning rule

$$\Delta w_{ij} \propto \langle x_i x_j \rangle_{\mathbf{v}} - \langle x_i x_j \rangle_{model}$$

- $\langle x_i x_j \rangle_{\mathbf{v}}$ corresponds to **Hebbian learning**
- $\langle x_i x_j \rangle_{model}$ corresponds to **unlearning**

Learning rule

$$\Delta w_{ij} \propto \langle x_i x_j \rangle_{\mathbf{v}} - \langle x_i x_j \rangle_{model}$$

Remember the expression for the probability of a vector **v**

$$p(\mathbf{v}) = \frac{\sum_{\mathbf{k}} e^{-E(\mathbf{v},\mathbf{k})}}{\sum_{\mathbf{w},\mathbf{k}} e^{-E(\mathbf{w},\mathbf{k})}}$$

- positive phase increases the numerator: configurations **k** that work well with **v** get a decreased energy
- negative phase decreases the denominator: lowers energy for probable configurations

What we need to know next?

- How to compute statistics, i.e. the exepected values of $x_i x_j$ in both phases.

What we need to know next?

- How to compute statistics, i.e. the exepected values of $x_i x_j$ in both phases.
- Inefficient answer (using two MCMC chains):
  - positive phase
    - start with random states for hidden units and update with the visible units clamped them until thermal equilibrium is reached
    - repeat for all training vectors
  - negative phase:
    - start with random states for all units and update them until thermal equilibrium is reached

Learning algorithms

- the previous algorithm is really slow
  - that's why researchers gave up BMs in the 80s and focuse on Backpropagation [Ben09]
- it's hard to detect when thermal equilibrium has been reached
- improvements to this algorithm using mean field approximation

Today's Outline

1. Energy-based models
2. Boltzmann Machines
   - Definition
   - Boltzmann Machine learning algorithm
   - Restricted Boltzmann Machines
3. Sigmoid Belief Networks
4. Deep Belief Networks

Restricted Boltzmann Machines

Definition

Restricted Boltzmann Machines [Smo86] are Boltzmann Machines with only one hidden layer and with no intra-layer connections (no connections between hidden units and no connections between visible units).

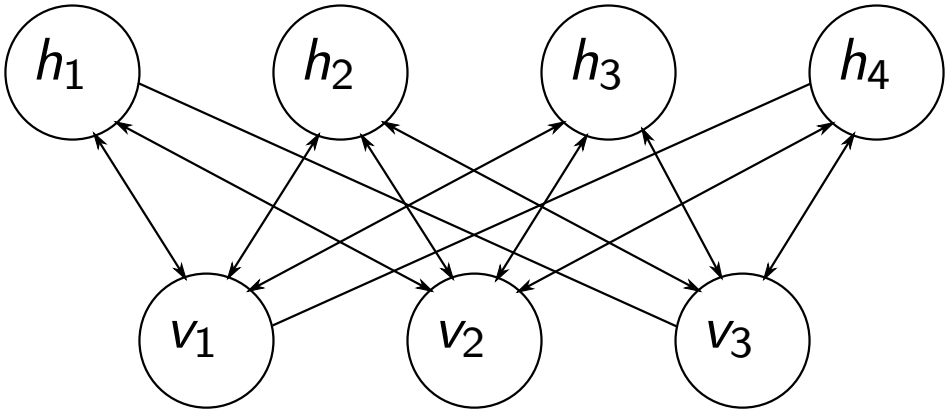


Figure: A Restricted Boltzmann Machine

Advantages of **RBMs**

- $P(\mathbf{h}|\mathbf{v})$ and $P(\mathbf{v}|\mathbf{h})$ are tractable because they factorize
- unless the RBM already perfectly models the input distribution, adding a hidden unit always improves the log-likelihood [LRB08]

Advantages of RBMs

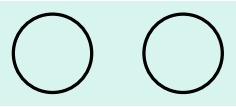
- It takes one step to reach thermal equilibrium when visible units are clamped.

$$p(h_j = 1) = \frac{1}{1 + e^{-(b + \sum_{i \in \mathcal{V}} v_i w_{ji})}}$$

- Above probabilities can be computed in parallel
  $\Rightarrow$ Gibbs sampling is faster

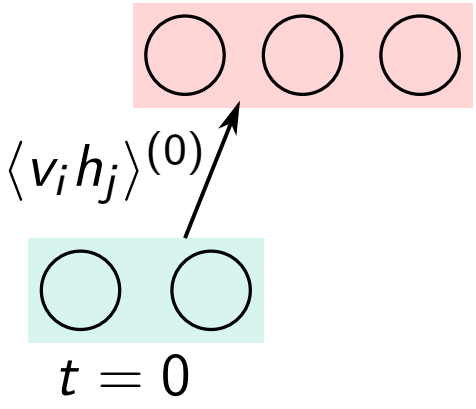
Alternating Updates to units

some training vector

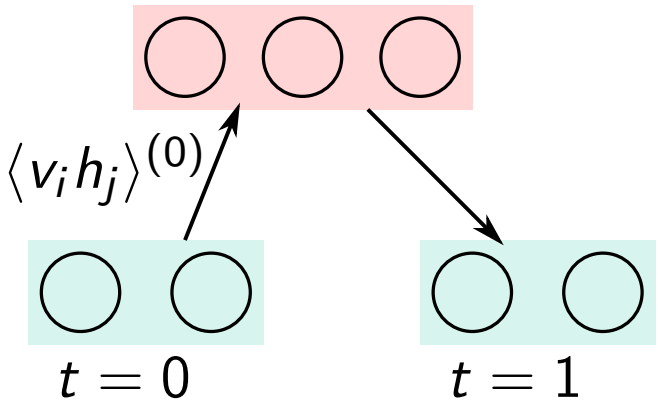


$t = 0$

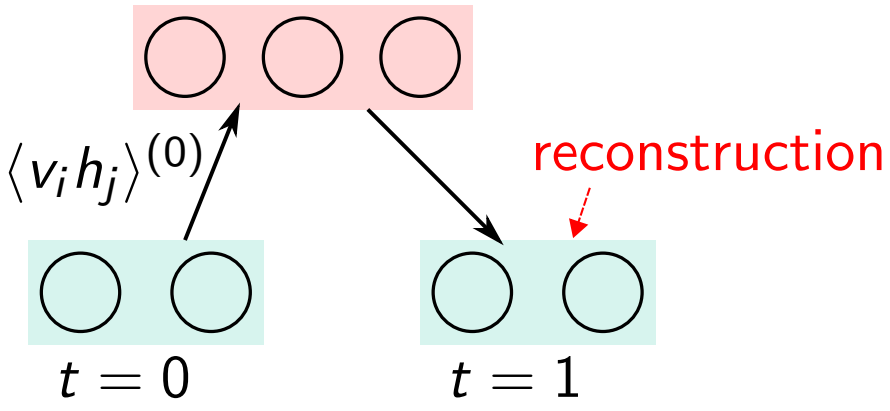
Alternating Updates to units



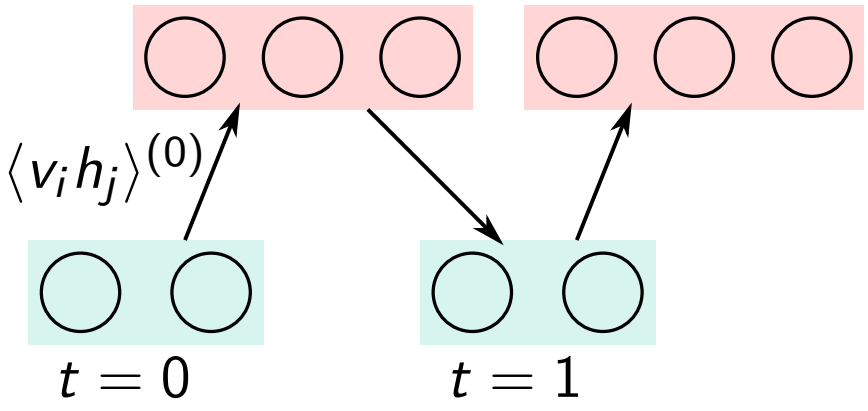
Alternating Updates to units



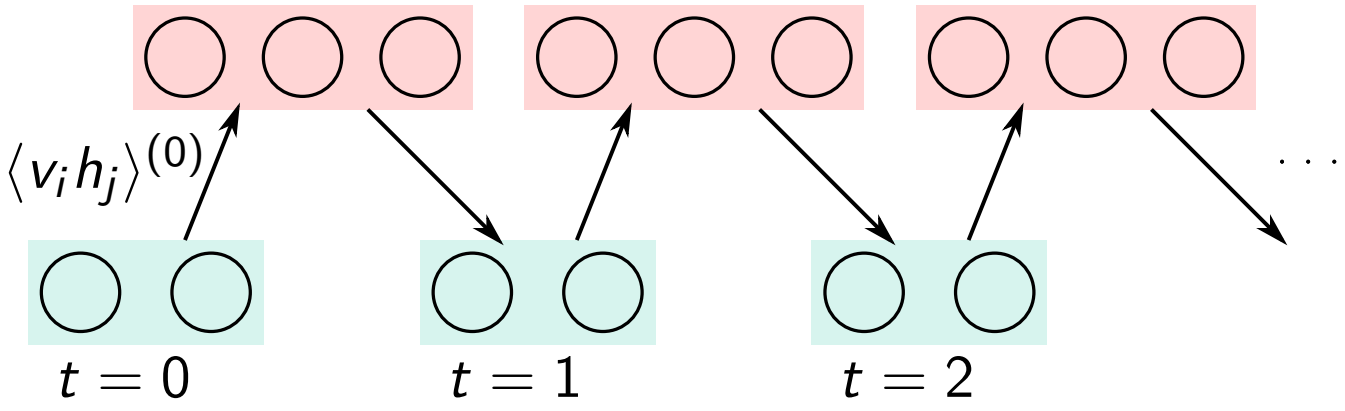
Alternating Updates to units



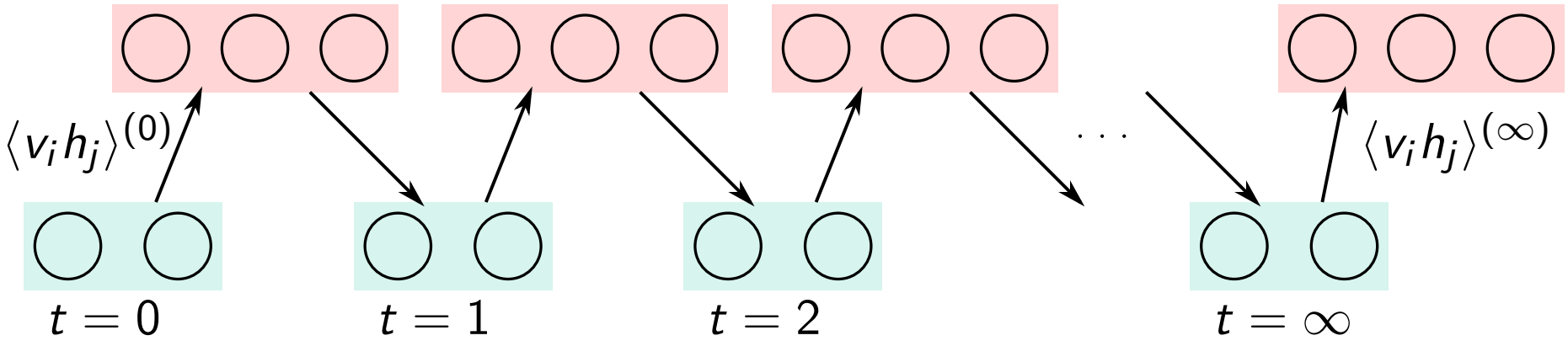
Alternating Updates to units



Alternating Updates to units



Alternating Updates to units

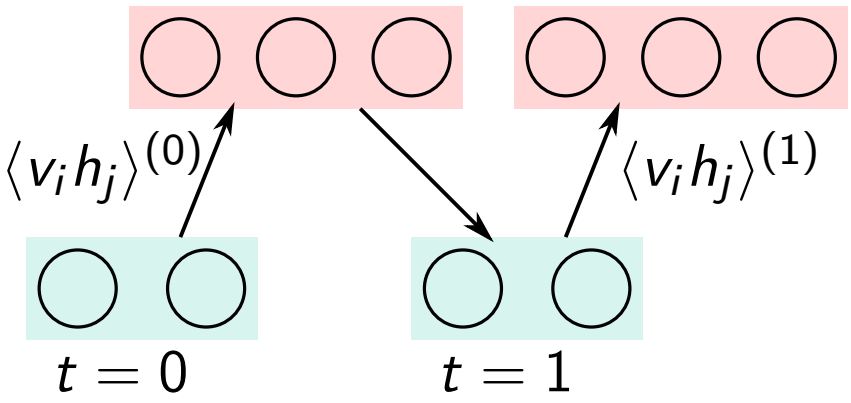


$$\Delta w_{ij} = \eta(\langle v_i h_j \rangle^{(0)} - \langle v_i h_j \rangle^{(\infty)})$$

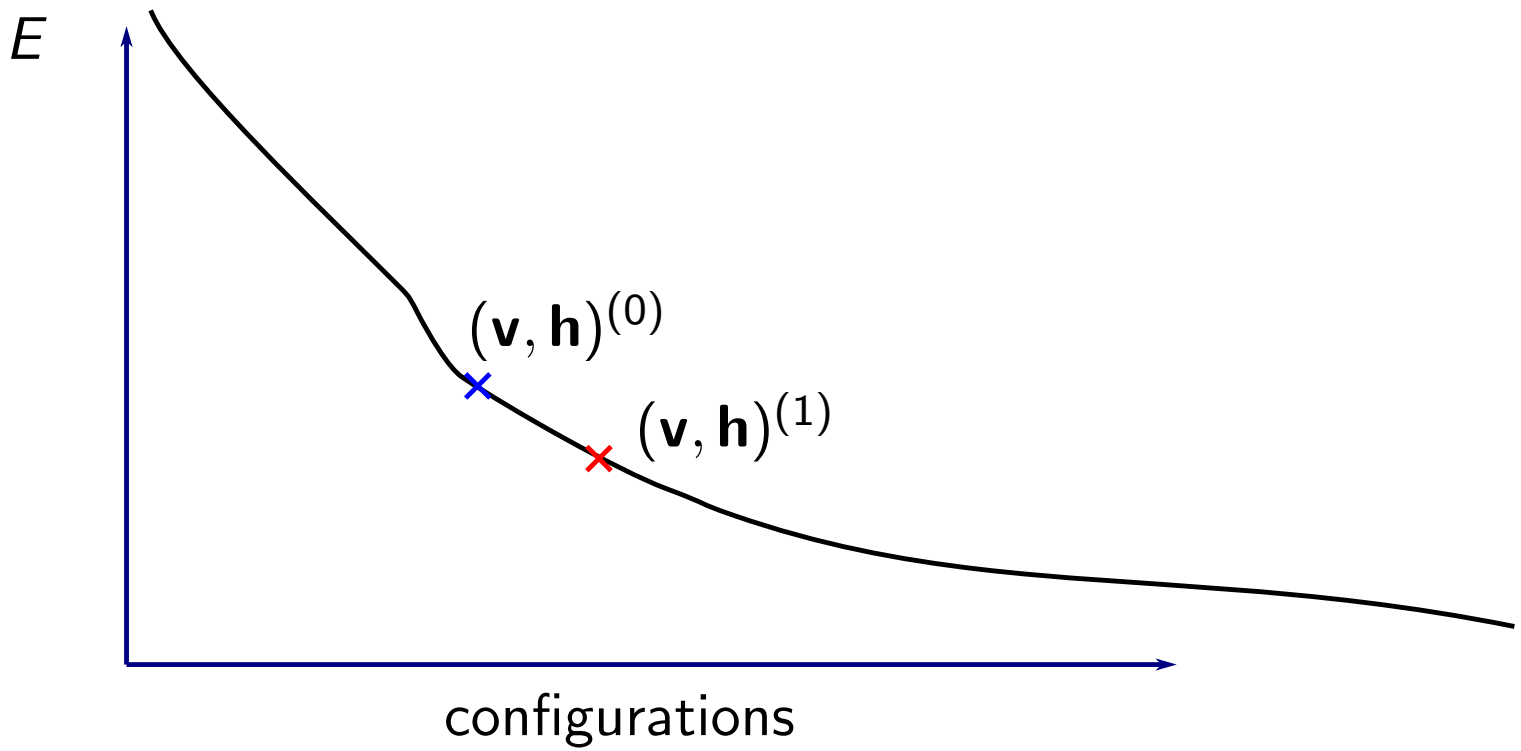
Contrastive Divergence

In practice, three updates are enough. [CPH05]

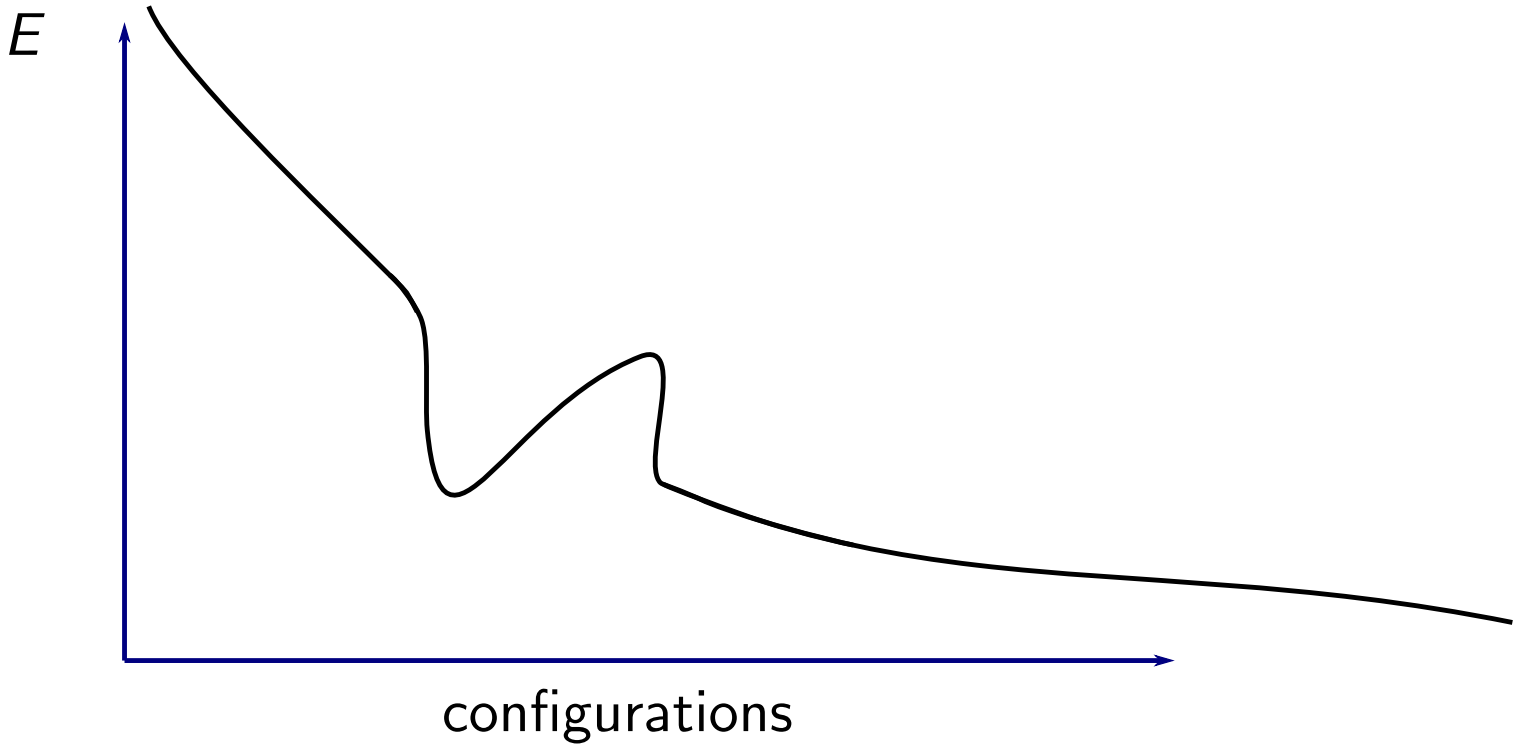
$$\Delta w_{ij} = \eta(\langle v_i h_j \rangle^{(0)} - \langle v_i h_j \rangle^{(1)})$$



What happens?



What happens?



Learning Algorithm using Contrastive Divergence

Algorithm 1 Learning using Contrastive Divergence

procedure $\text{RBM-LEARN}(\mathbf{x}, \eta, \mathbf{W}, \mathbf{b}^{(\mathcal{V})}, \mathbf{b}^{(\mathcal{H})})$
  for all hidden units i **do**
    sample $h_i^{(1)}$ from $P(h_i^{(1)} = 1|\mathbf{x}) = \frac{1}{1+e^{-b_i^{(\mathcal{H})} - \sum_j x_j w_{ij}}}$
  for all visible units j **do**
    sample $v_j^{(2)}$ from $P(v_j^{(2)} = 1|\mathbf{h}^{(1)}) = \frac{1}{1+e^{-b_j^{(\mathcal{V})} - \sum_i h_i^{(1)} w_{ij}}}$
  for all hidden units i **do**
    compute $P(h_i^{(2)} = 1|\mathbf{x}^{(2)}) = \frac{1}{1+e^{-b_i^{(\mathcal{H})} - \sum_j v_j^{(2)} w_{ij}}}$
  $\mathbf{W} \longleftarrow \mathbf{W} + \eta(\mathbf{h}^{(1)}\mathbf{x}^{\mathsf{T}} - \mathbf{h}^{(2)}\mathbf{v}^{(2)})$
  $\mathbf{b}^{\mathcal{V}} \longleftarrow \mathbf{b}^{\mathcal{V}} - \eta(\mathbf{x} - \mathbf{v}^{(2)})$
  $\mathbf{b}^{\mathcal{H}} \longleftarrow \mathbf{b}^{\mathcal{H}}$

Persistent Contrastive Divergence

- fast, mini-batch algorithm
- proposed by Tielemen in 2008 [Tie08]

PCD algorithm

1. **Positive phase**:
   - clamp visible units to a training vector
   - compute exact value of $\langle v_i h_j \rangle$
   - average $\langle v_i h_j \rangle$ over all examples in the mini-batch
2. **Negative phase**:
   - keep a set of persistent *fantasy particles* (global configurations)
   - make a few alternating parallel updates
   - average $\langle v_i h_j \rangle$ over all fantasy particles

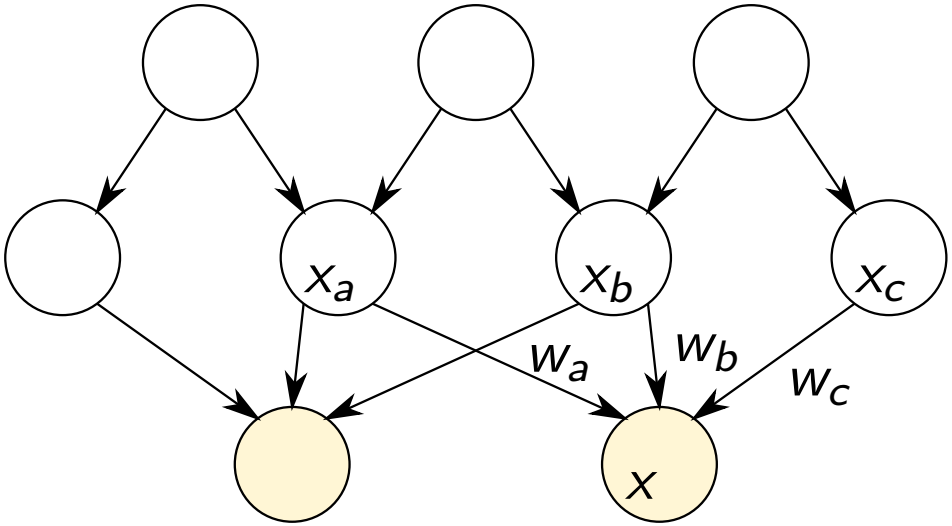
Today's Outline

Sigmoid Belief Networks

- **causal**, **generative** models introduced by Neal [Nea92]
- networks represented as directed acyclic graphs
- all units are stochastic neurons

$$P(x_i = 1 | \mathbf{A}_{x_i}) = \frac{1}{1 + e^{-b - \sum_{x_j \in \mathbf{A}_{x_i}} w_{ji} X_j}}$$

Sigmoid Belief Networks



$$p(x = 1) = \sigma(b + w_a x_a + w_b x_b + w_c x_c)$$

Today's Outline

1. Energy-based models
2. Boltzmann Machines
3. Sigmoid Belief Networks
   - Why learning is hard in SBN?
   - The wake-sleep algorithm
4. Deep Belief Networks

Learning in SBN

- although it's easy to generate samples of all the units, it's hard to infer the posterior distribution over possible configurations of the hidden units given the visible ones

$$P(\mathbf{h}|\mathbf{v})$$

- even one sample from the posterior is hard to get

Learning rule

- maximum likelihood learning

$$\Delta w_{\vec{ji}} \propto x_j(x_i - p_i)$$

Maximum likelihood

- The probability of a specific network configuration $\mathbf{x}$:

$$P(\mathbf{x}) = P(x_i|\mathbf{Par}_{x_i})P(\mathbf{Par}_{x_i}|Rest)P(Rest)$$

Maximum likelihood

- The probability of a specific network configuration $\mathbf{x}$:

$$
\begin{aligned}
P(\mathbf{x}) &= P(x_i|\mathbf{Par}_{x_i})P(\mathbf{Par}_{x_i}|Rest)P(Rest) \\
\log(P(\mathbf{x})) &= \log(P(x_i|\mathbf{Par}_{x_i})) + \log(P(\mathbf{Par}_{x_i}|Rest)) + \\
&\quad + \log(P(Rest))
\end{aligned}
$$

Maximum likelihood

- The probability of a specific network configuration $\mathbf{x}$:

$$
\begin{aligned}
P(\mathbf{x}) &= P(x_i|\mathbf{Par}_{x_i})P(\mathbf{Par}_{x_i}|Rest)P(Rest) \\
\log(P(\mathbf{x})) &= \log(P(x_i|\mathbf{Par}_{x_i})) + \log(P(\mathbf{Par}_{x_i}|Rest)) + \\
&\quad + \log(P(Rest)) \\
\frac{\partial \log(P(\mathbf{x}))}{\partial w_{ji}} &= \frac{\partial}{\partial w_{ji}} \log(P(x_i|\mathbf{Par}_{x_i}))
\end{aligned}
$$

Maximum likelihood

- The probability of a specific network configuration $\mathbf{x}$:

$$
\begin{aligned}
P(\mathbf{x}) &= P(x_i|\mathbf{Par}_{x_i})P(\mathbf{Par}_{x_i}|Rest)P(Rest) \\
\log(P(\mathbf{x})) &= \log(P(x_i|\mathbf{Par}_{x_i})) + \log(P(\mathbf{Par}_{x_i}|Rest)) + \\
&\quad + \log(P(Rest)) \\
\frac{\partial \log(P(\mathbf{x}))}{\partial w_{ji}} &= \frac{\partial}{\partial w_{ji}} \log(P(x_i|\mathbf{Par}_{x_i})) \\
\frac{\partial \log(P(\mathbf{x}))}{\partial w_{ji}} &= \frac{\partial}{\partial w_{ji}} \Bigg(x_i \log\left(\sigma\Big(\sum_{x_k \in \mathbf{Par}_{x_i}} x_j w_{ki}\Big)\right) + \\
&\quad (1 - x_i)\Bigg(1 - \log\left(\sigma\Big(\sum_{x_k \in \mathbf{Par}_{x_i}} x_j w_{ki}\Big)\right)\Bigg)
\end{aligned}
$$

Maximum likelihood

$$\frac{\partial \log(P(\mathbf{x}))}{\partial w_{ji}} = x_j(x_i - p_i)$$

Why learning is hard?

- the posterior is not a factorial

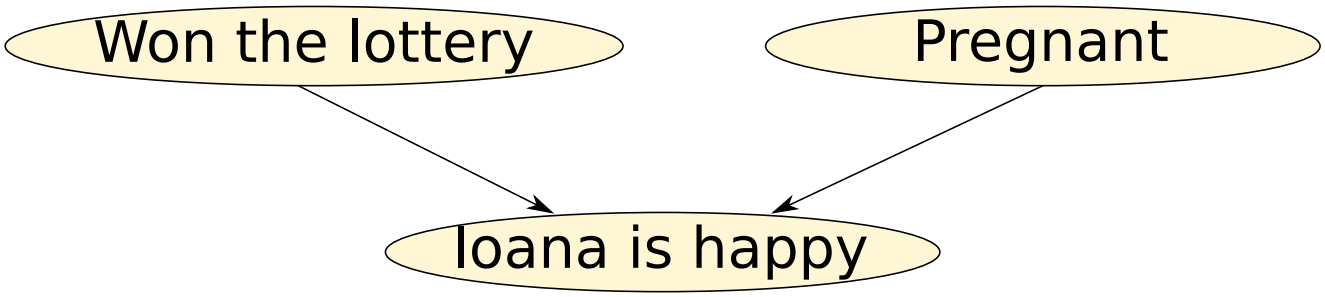


Figure: Example of *explaining away*

- every layer depends on the one above

Monte Carlo Markov Chain

- MCMC can be used to get samples from the posterior distribution [Nea92]
- method:
  1. clamp the visible units to a training example
  2. run Gibbs sampling simulations until thermal equilibrium is reached
  3. change the weights (batch mode)
- problems:
  - thermal equilibrium must be reached
  - the method does not scale

Today's Outline

Overview of the algorithm

- a *variational* method
  1. use an approximation of the posterior distribution
     - ignore the *explaining away*
  2. do maximum likelihood learning

Overview of the algorithm

- proposed by G. Hinton [HDFN95]
- as the name suggests, the algorithm alternates two phases

Overview of the algorithm

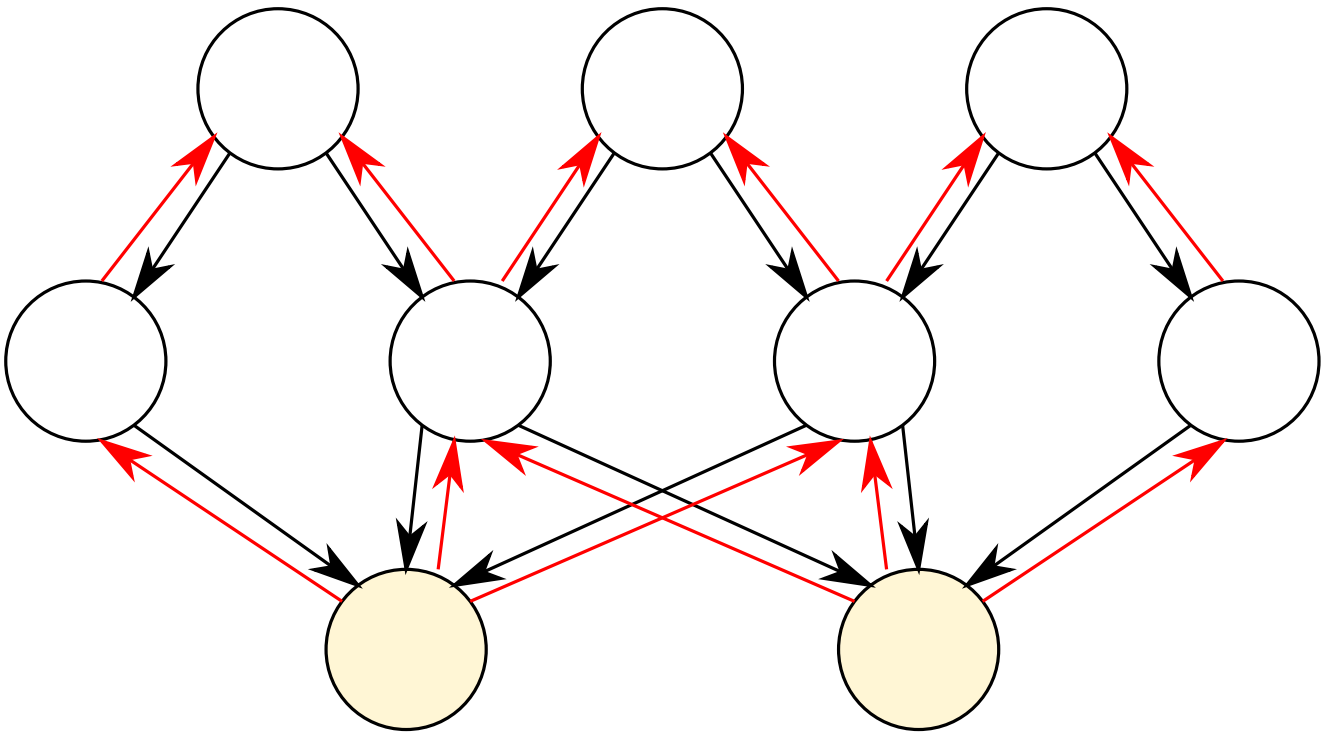
- proposed by G. Hinton [HDFN95]
- as the name suggests, the algorithm alternates two phases
- two sets of weights are used:
  - *recognition* connections
  - *generative* connections

Overview of the algorithm

- proposed by G. Hinton [HDFN95]
- as the name suggests, the algorithm alternates two phases
- two sets of weights are used:
  - *recognition* connections
  - *generative* connections
- in the **wake** phase, recognition connections are used to adjust generative connections
- in the **sleep** phase, generative connections are used to adjust recognition connections

The wake-sleep algorithm

- start with random weights
- alternate wake and sleep phases



Wake phase

- put an example from the training set on the visible units
- make a bottom-up computation using the *recognition* weights
- each units makes a stochastic decision about its state

Wake phase

- put an example from the training set on the visible units
- make a bottom-up computation using the *recognition* weights
- each units makes a stochastic decision about its state
- the final configuration will be used as if it were a true sample from the posterior distribution
- the generative weights are learnt using maximum likelihood

Sleep phase

- put a random vector on the top layer of hidden units
- make a top-down computation using the *generative* weights
- each units makes a stochastic decision about its state

Sleep phase

- put a random vector on the top layer of hidden units
- make a top-down computation using the *generative* weights
- each units makes a stochastic decision about its state
- the *recognition* weights are learnt using maximum likelihood

Problems with the algorithm

- it doesn't optimize a well defined objective function
- there is no guarantee that the algorithm converges

- for a better and more recent approach see [MG14]

Today's Outline

Today's Outline

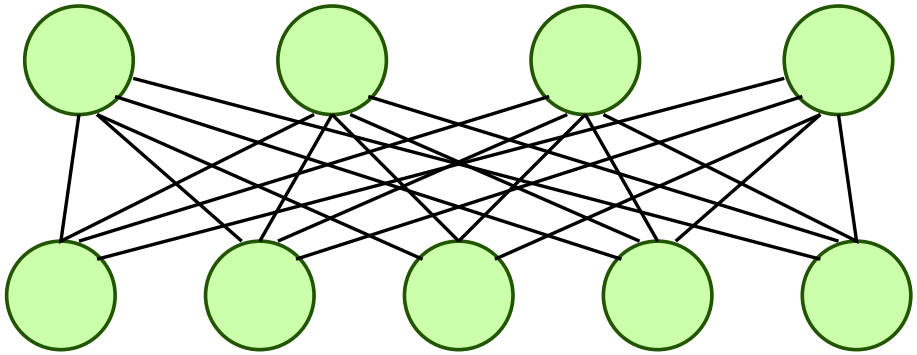
1. Energy-based models
2. Boltzmann Machines
3. Sigmoid Belief Networks
4. Deep Belief Networks
   - Definition
   - Greedy Learning
   - Fine-tuning algorithm
   - Pre-training supervised predictors

Deeb Belief Networks

- **DBNs** [HOT06] where the first successful *deep* non-convolutional architectures [BGC15]
- **DBNs** exploit the unsupervised learning algorithm for **RBMs** for each layer
- after **DBNs**, other algorithms for auto-encoders were proposed
- **RBMs**and **DBNs** recently used to initialize weights for FFNs [HOT06] as studies show that this technique leads to better results [BLP+07] [EMB+09] [LBLL09]

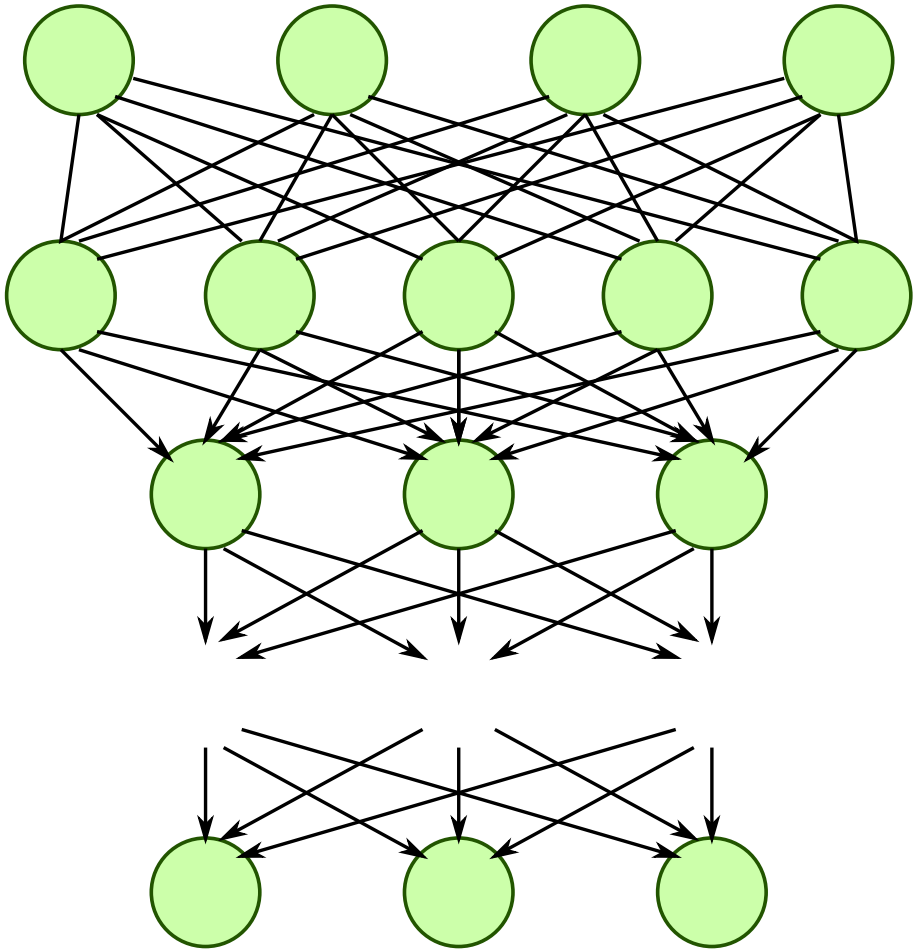
Deep Belief Networks

- considered model for deep belief networks [HOT06]:
  - two top layers that form an associative memory



Deep Belief Networks

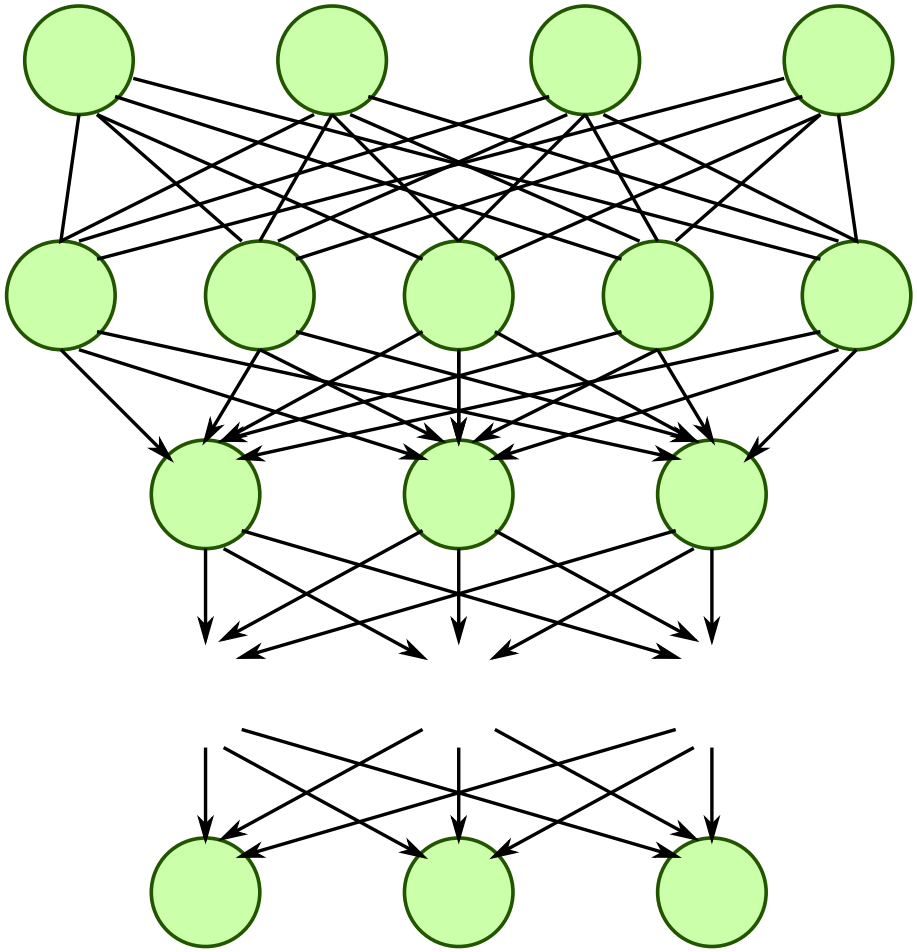
- considered model for deep belief networks [HOT06]:
  - two top layers that form an associative memory
  - any additional number of layers that form a directed acyclic graph (Sigmoid Belief Networks)



The problem with inference

remember from previous section:

- **inference** is hard because of the **explaining away** phenomenon
- no easy access to $p(\mathbf{h}|\mathbf{v})$
  - MCMC - needs a huge amount of time
  - the *wake-sleep* algorithm offers a poor aproximation for deep networks

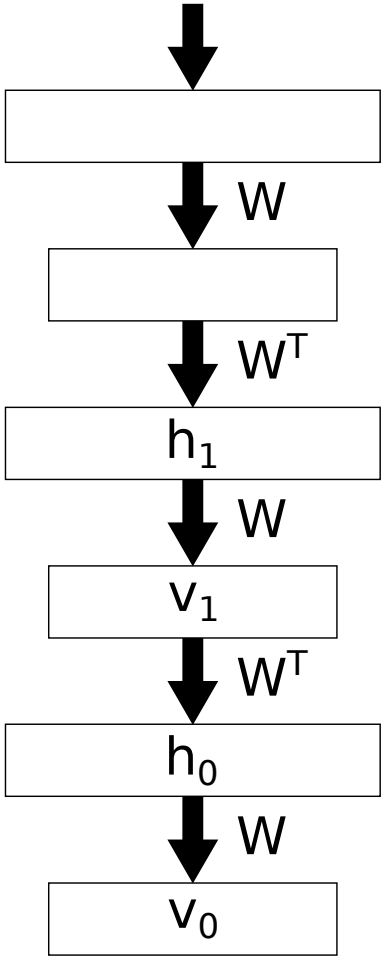


Complementary priors

- let the extra (above) hidden layers construct a complementary prior that cancels explaining away

Complementary priors

- let the extra (above) hidden layers construct a complementary prior that cancels explaining away
- example of such a network is a DBN with tied weights (mathematical treatment in [HOT06])



Advantage of using complementary priors

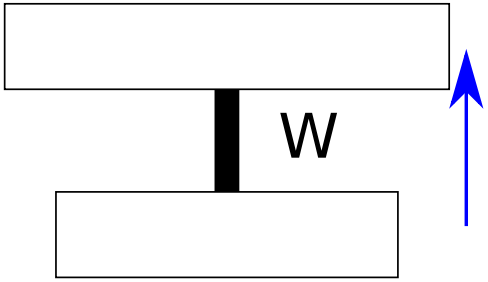
- **generating** data is simple:
  1. start with a random configuration in the top layer
  2. do a top-down pass with stochastic decisions
- **inference** becomes simple due to complementary priors:

$$\mathbf{h}^{(\tau)} = \mathbf{W}^{\mathsf{T}}\mathbf{v}^{(\tau)} \tag{3}$$

Relation between DBN and RBM

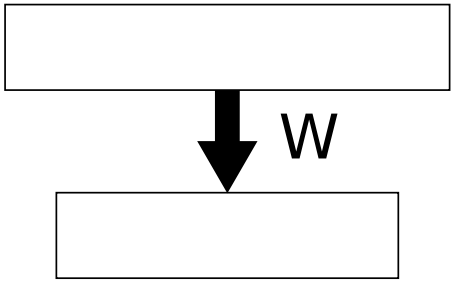
- a Restricted Boltzmann Machine is the same with an infinitely Deep Belief Network with shared weights
- the distribution of a RBM at thermal equilibrium is the same with the one generated by the infinite directed belief network

RBMs as infinitely deep BN

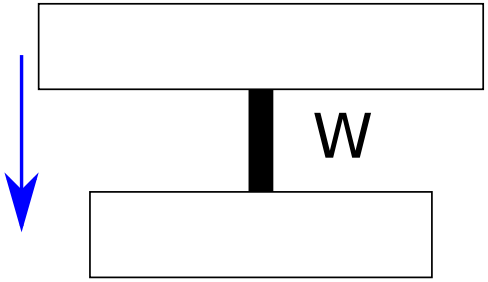


Generating data

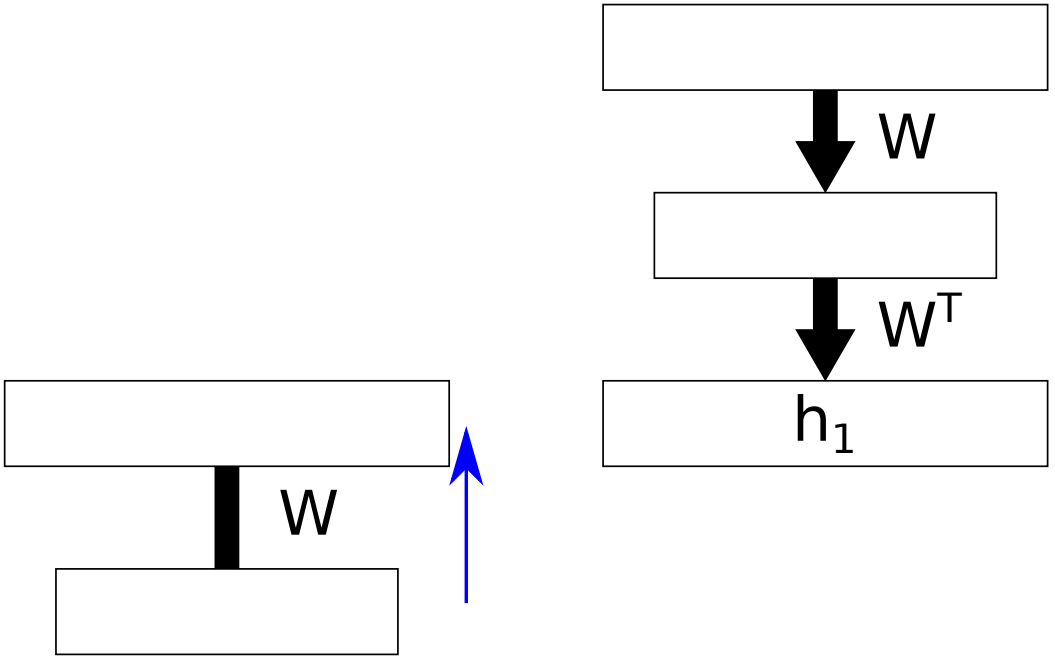
RBMs as infinitely deep BN



Generating data

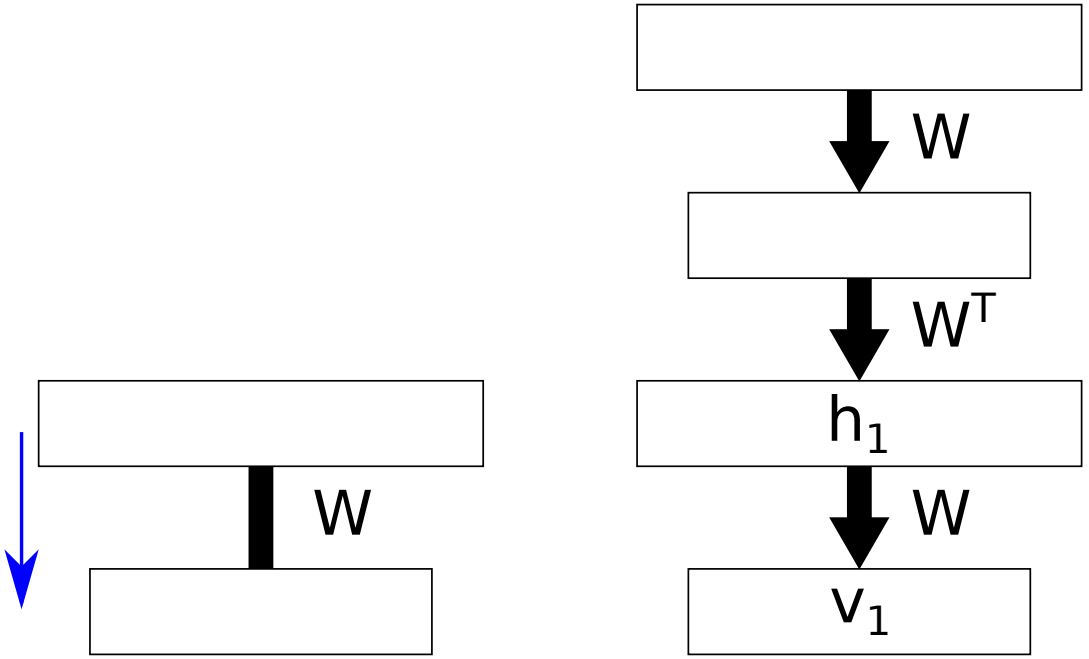


RBMs as infinitely deep BN



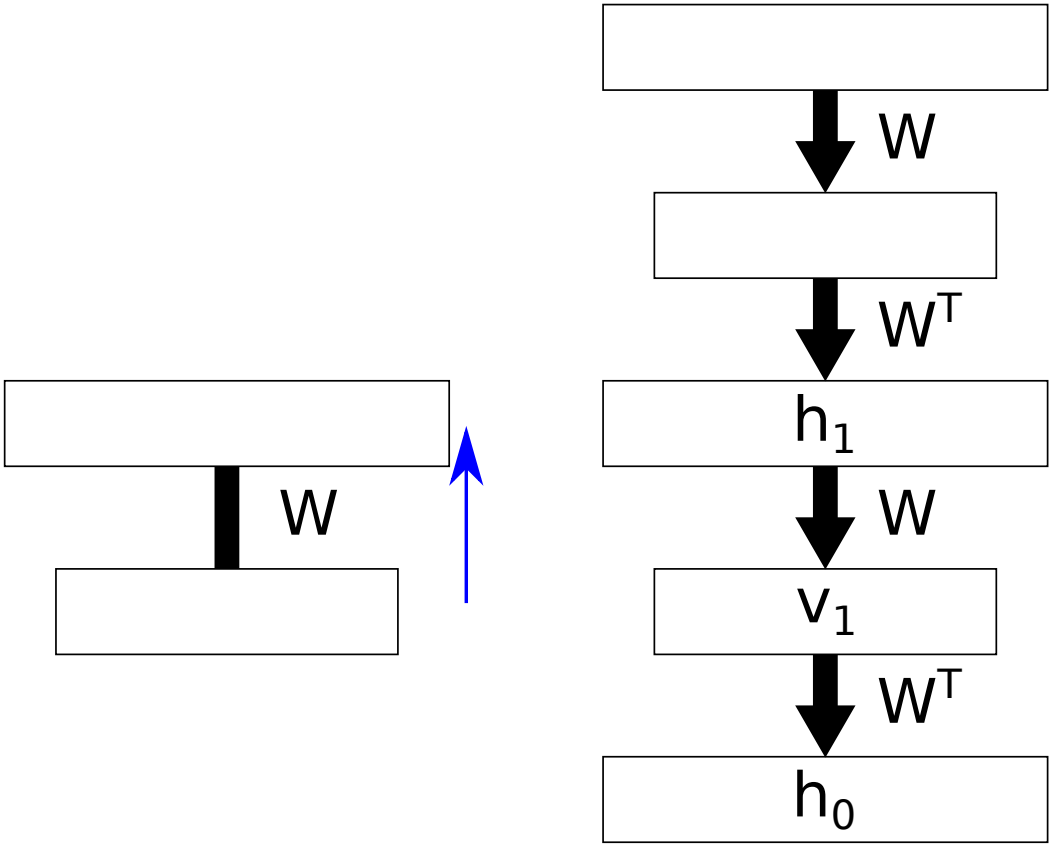
Generating data

RBMs as infinitely deep BN



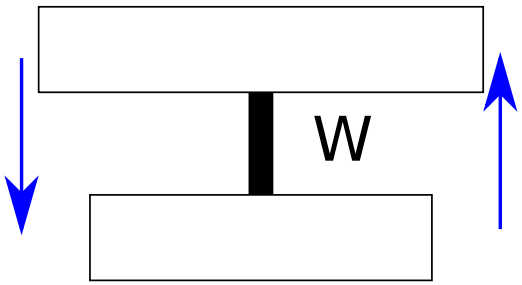
Generating data

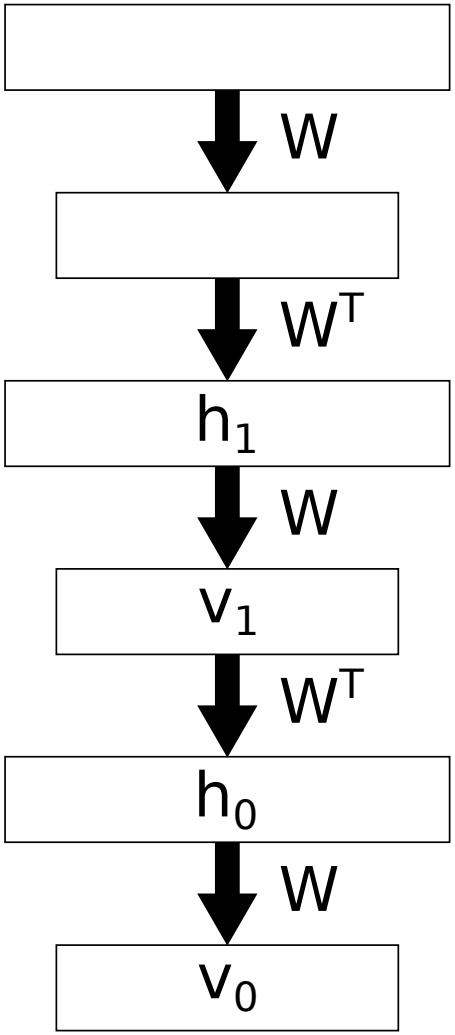
RBMs as infinitely deep BN



Generating data

RBMs as infinitely deep BN

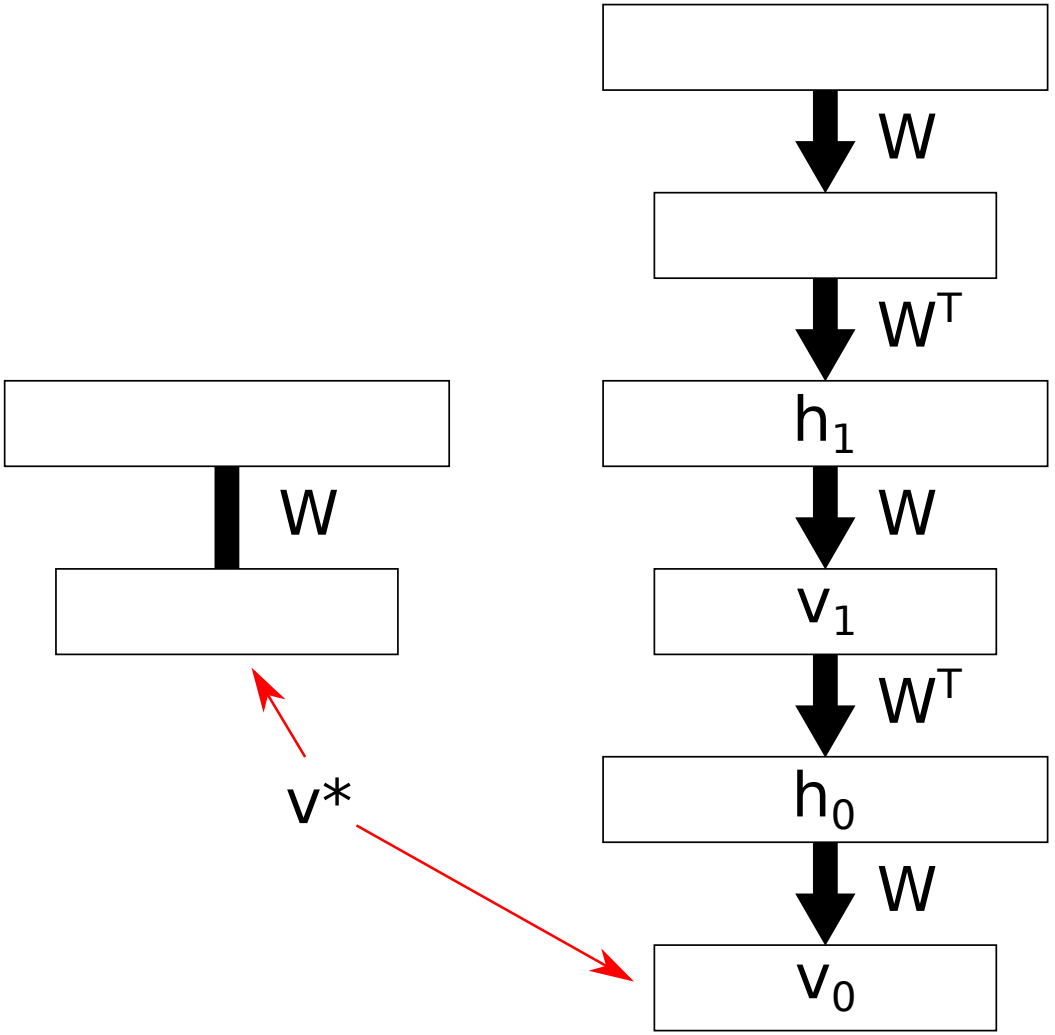




Generating data

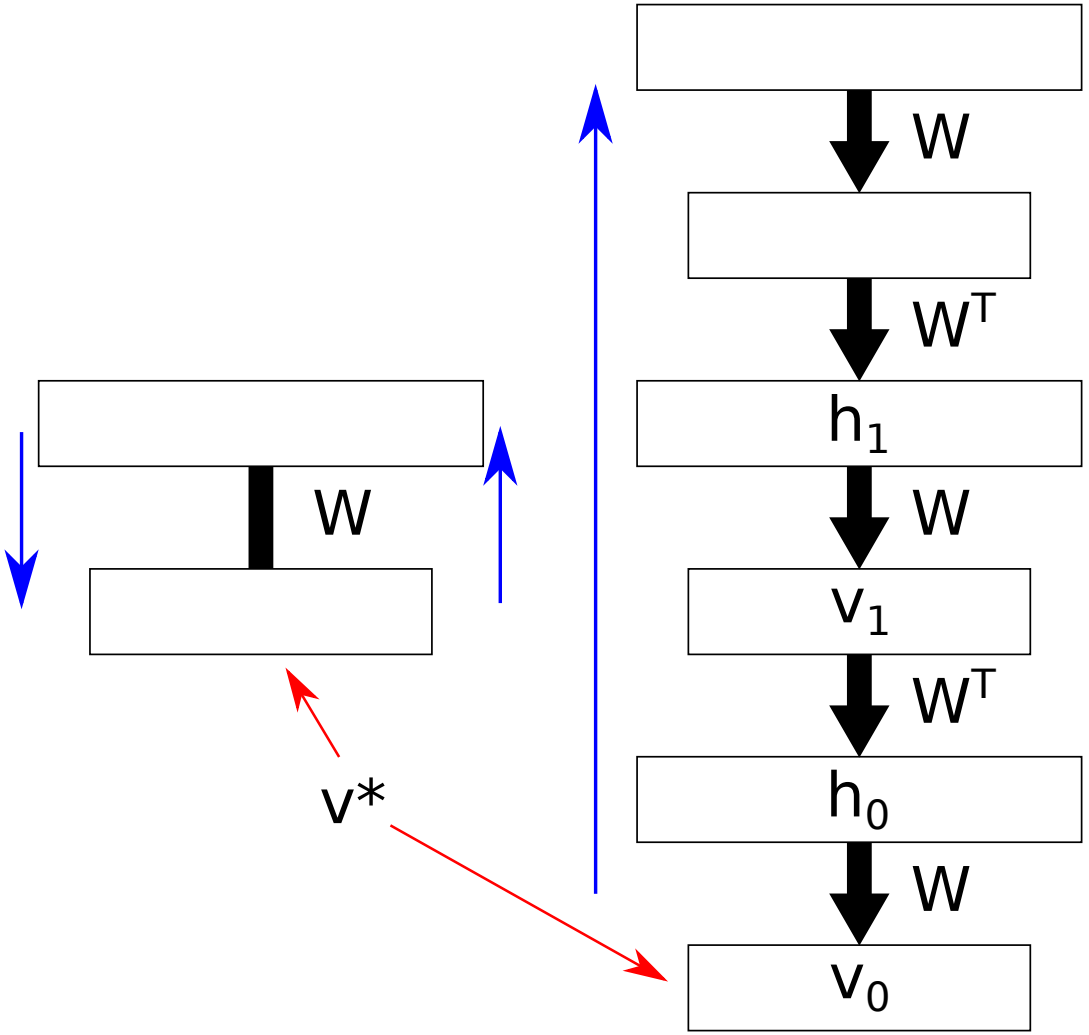
The sample one gets in $\mathbf{v}_0$ in the infinitely deep belief network is a sample at thermal equilibrium from the RBM.

Inference



Inference

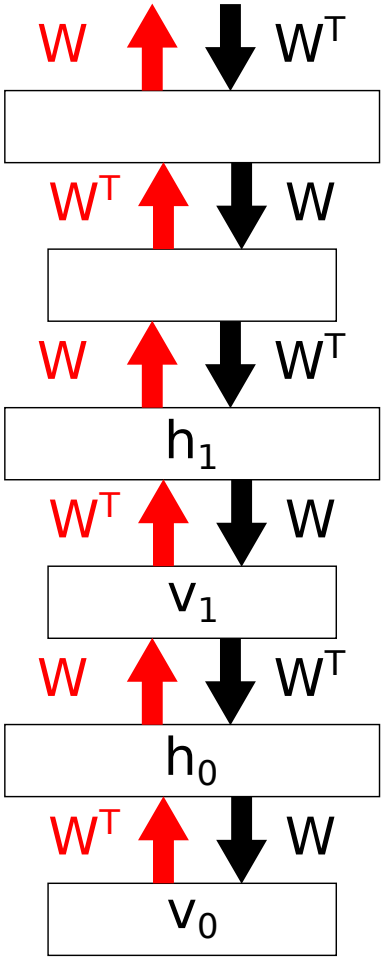
Inference



Inference

Inference in DBNs is the same with settling the RBM to thermal equilibrium starting from data.

Learning



- Consider the general learning rule:

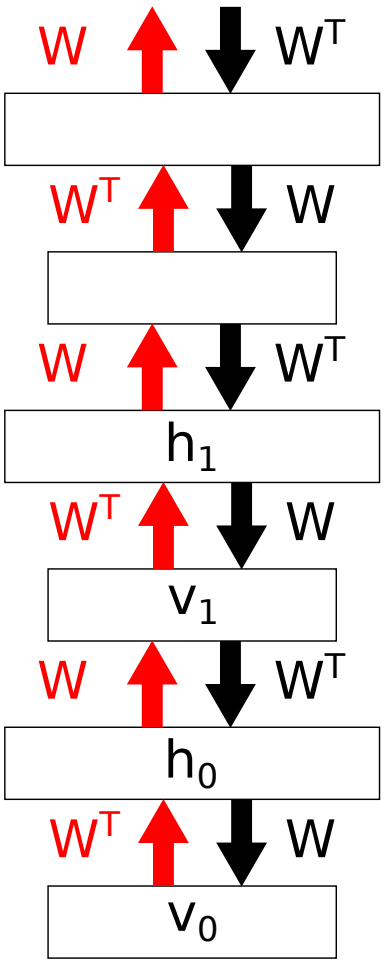
$$\Delta w_{ij} \propto x_j(x_i - p_i)$$

- For the weights between $\mathbf{v}^{(0)}$ and $\mathbf{h}^{(0)}$:

$$\Delta w_{ij} \propto h_j^{(0)}(v_i^{(0)} - p_i)$$

- But... $v_i^{(1)}$ is an unbiased sample from $p_i^{(0)}$

Learning

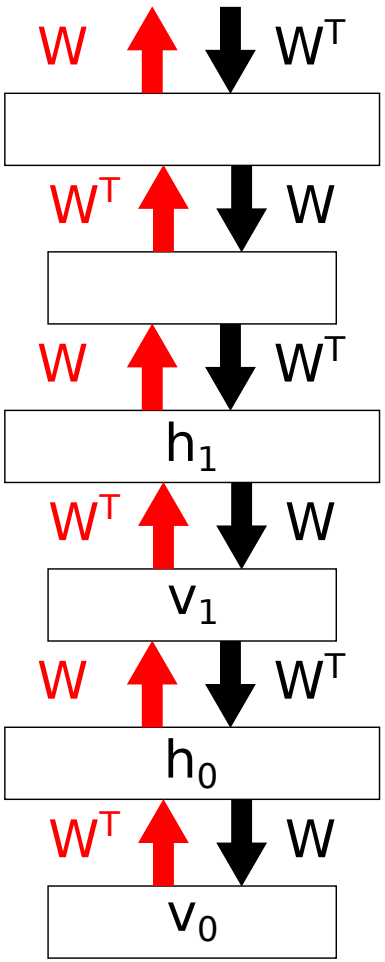


- Learning rule considering all layers:

$$\Delta w_{ji} \quad \propto \quad h_j^{(0)}(v_i^{(0)} - v_i^{(1)})$$

everything averaged over all considered examples

Learning

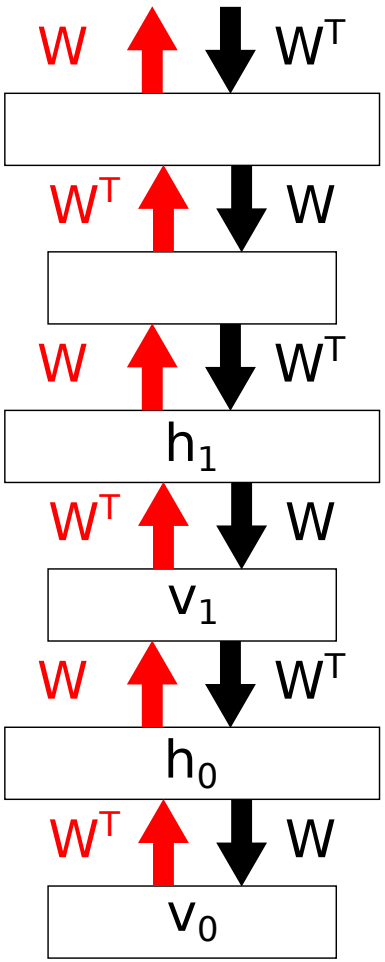


- Learning rule considering all layers:

$$\Delta w_{ji} \propto h_j^{(0)}(v_i^{(0)} - v_i^{(1)}) + v_i^{(1)}(h_j^{(0)} - h_j^{(1)})$$

everything averaged over all considered examples

Learning

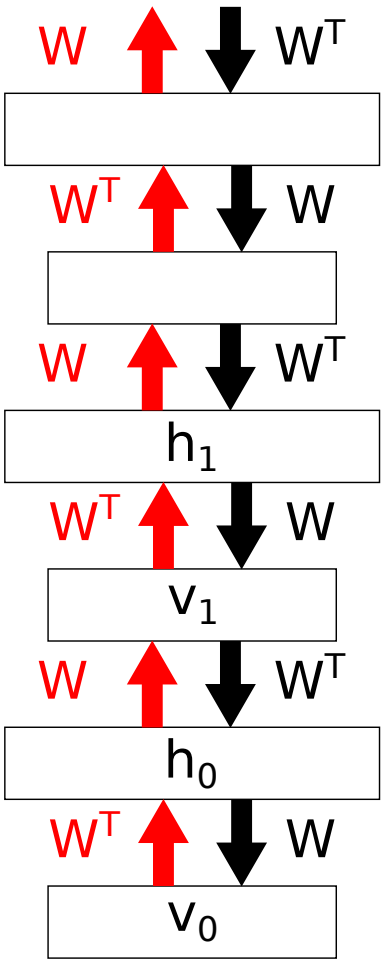


- Learning rule considering all layers:

$$\begin{aligned}\Delta w_{ji} \quad \propto \quad & h_j^{(0)}(v_i^{(0)} - v_i^{(1)}) \\ & + v_i^{(1)}(h_j^{(0)} - h_j^{(1)}) \\ & + h_j^{(1)}(v_i^{(1)} - v_i^{(2)})\end{aligned}$$

everything averaged over all considered examples

Learning

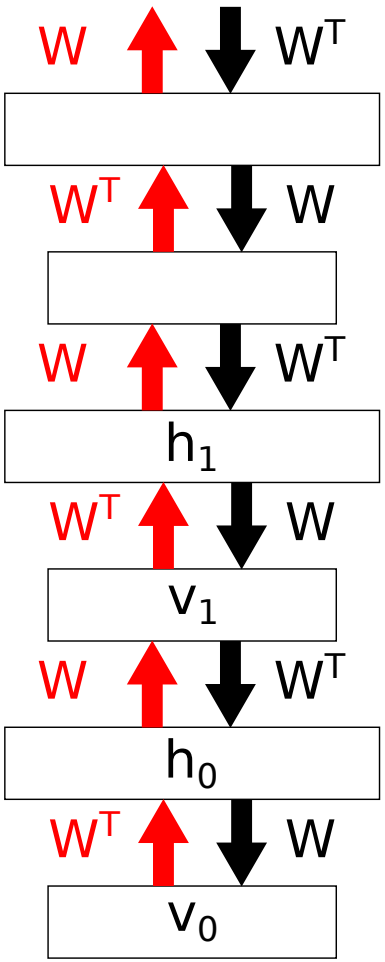


- Learning rule considering all layers:

$$
\begin{aligned}
\Delta w_{ji} \quad \propto \quad & h_j^{(0)}(v_i^{(0)} - v_i^{(1)}) \\
& + v_i^{(1)}(h_j^{(0)} - h_j^{(1)}) \\
& + h_j^{(1)}(v_i^{(1)} - v_i^{(2)}) \\
& + v_j^{(2)}(h_j^{(1)} - h_j^{(2)}) + \dots
\end{aligned}
$$

everything averaged over all considered examples

Learning



- Learning rule considering all layers:

$$\begin{aligned}\Delta w_{ji} \quad \propto \quad & h_j^{(0)}(v_i^{(0)} - v_i^{(1)}) \\ & + v_i^{(1)}(h_j^{(0)} - h_j^{(1)}) \\ & + h_j^{(1)}(v_i^{(1)} - v_i^{(2)}) \\ & + v_j^{(2)}(h_j^{(1)} - h_j^{(2)}) + \dots\end{aligned}$$

- This leads to the learning rule for RBM:

$$\Delta w_{ji} \quad \propto \quad h^{(0)} v^{(0)} - h^{(\infty)} v^{(\infty)}$$

everything averaged over all considered examples

Learning

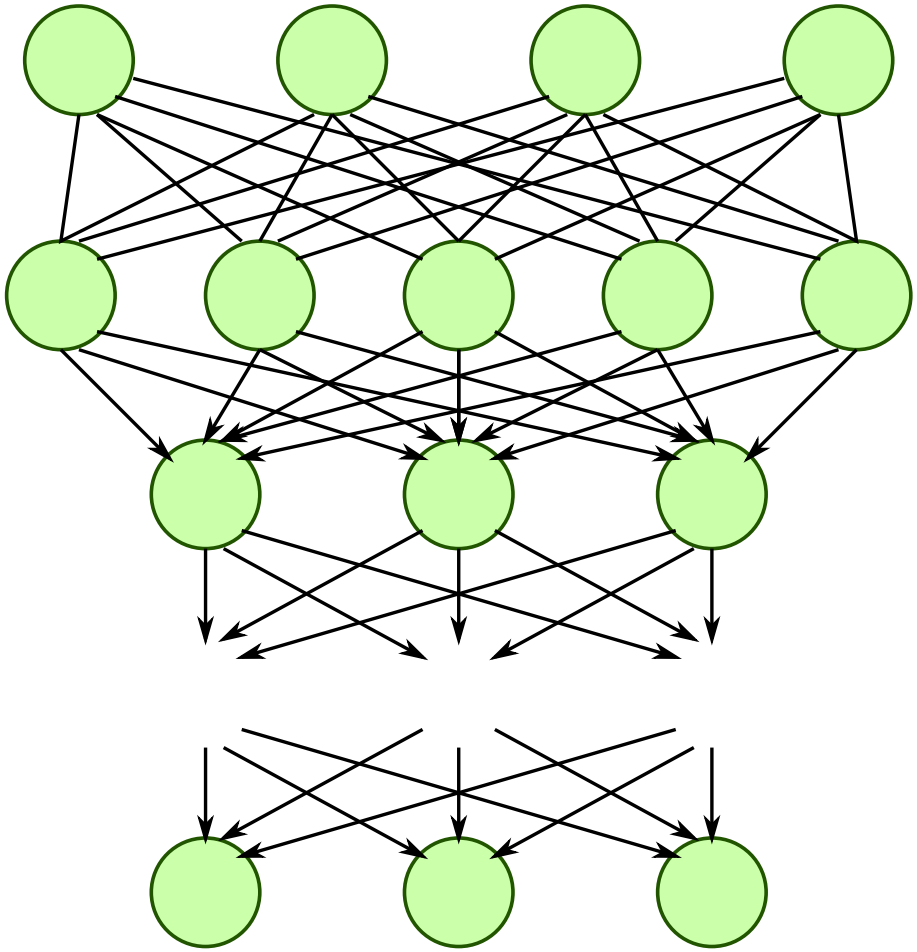
- constrastive divergence is efficient for deep belief nets with tied weights [Hin02]
- the learning rule cannot work for DBNs without tied weights

Today's Outline

1. Energy-based models
2. Boltzmann Machines
3. Sigmoid Belief Networks
4. Deep Belief Networks
   - Definition
   - Greedy Learning
   - Fine-tuning algorithm
   - Pre-training supervised predictors

The used model

- back to the considered model for deep belief networks [HOT06]:
  - two top layers that form an associative memory
  - any additional number of layers that form a directed acyclic graph (Sigmoid Belief Networks)

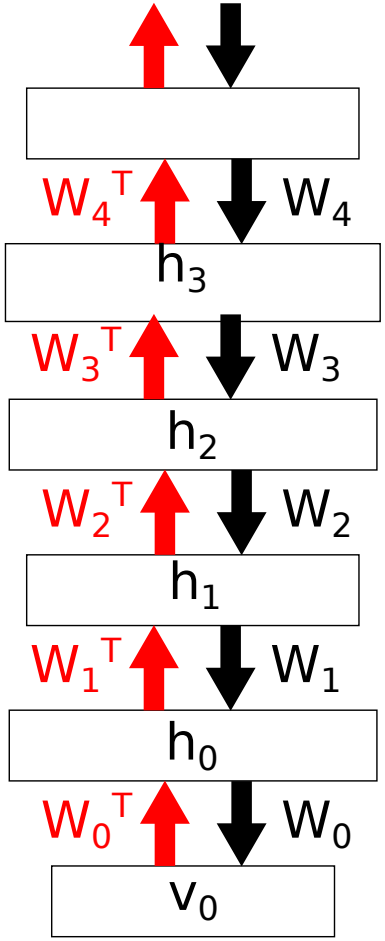


Fast, greedy algorithm for learning

- stack several Boltzmann Machines one over the other
- learn weights for layers bottom-up

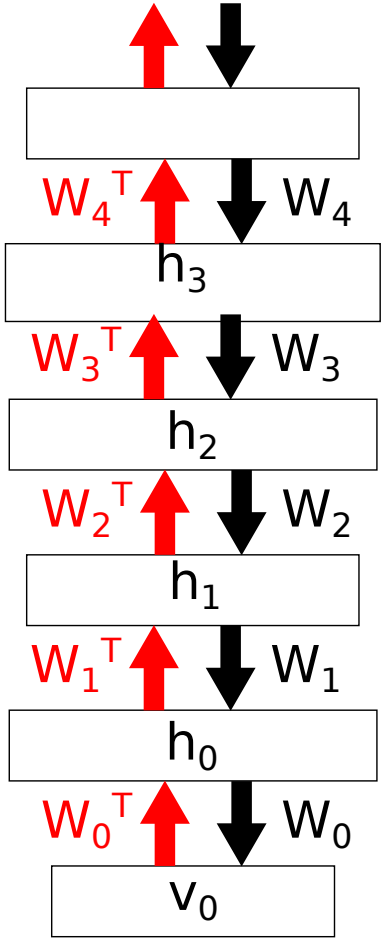
Fast, greedy learning

- learn $\mathbf{W}_0$ assuming that above layers construct a complementary prior



Fast, greedy learning

- learn $\mathbf{W}_0$ assuming that above layers construct a complementary prior
- in fact, the assumption is that all the layers above are constrained to have the same weights
  $\longrightarrow$ we have to learn a RBM



The algorithm

- for all hidden layers $l = 0 \ldots$:
  1. learn W_l assuming all the weighs above are tied
  2. freeze W_l and infer posterior distributions over h_{l+1} using W_l^{T}

The algorithm

- for all hidden layers $l = 0 \ldots$:
  1. learn W_l assuming all the weighs above are tied
  2. freeze W_l and infer posterior distributions over h_{l+1} using W_l^{T}

Observations

- if full Boltzmann Machine likelihood learning would be used, the algorithm is guaranteed to never decrease the log probability of the data under the full generative model
- using contrastive divergence minimization still gives good results
- adding more layers trained for enough time can only improve the performance

Today's Outline

1. Energy-based models
2. Boltzmann Machines
3. Sigmoid Belief Networks
4. Deep Belief Networks
   - Definition
   - Greedy Learning
   - Fine-tuning algorithm
   - Pre-training supervised predictors

How to tune the model?

- Backpropagation
- Contrastive version of the wake-sleep algorithm [HOT06]

General Idea

- after using greedy learning, back-fitting of the weights is needed
- *recognition* weights are untied from *generative* weights
- a version of the *wake-sleep* algorithm is used
- in the *up* phase, generative weights are adjusted
- in the *down* phase, recognition weights are adjusted
- *contrastive wake-sleep*: no need to settle top associative layer to thermal equilibrium

Today's Outline

1. Energy-based models
2. Boltzmann Machines
3. Sigmoid Belief Networks
4. Deep Belief Networks
   - Definition
   - Greedy Learning
   - Fine-tuning algorithm
   - Pre-training supervised predictors

Importance of using pre-training

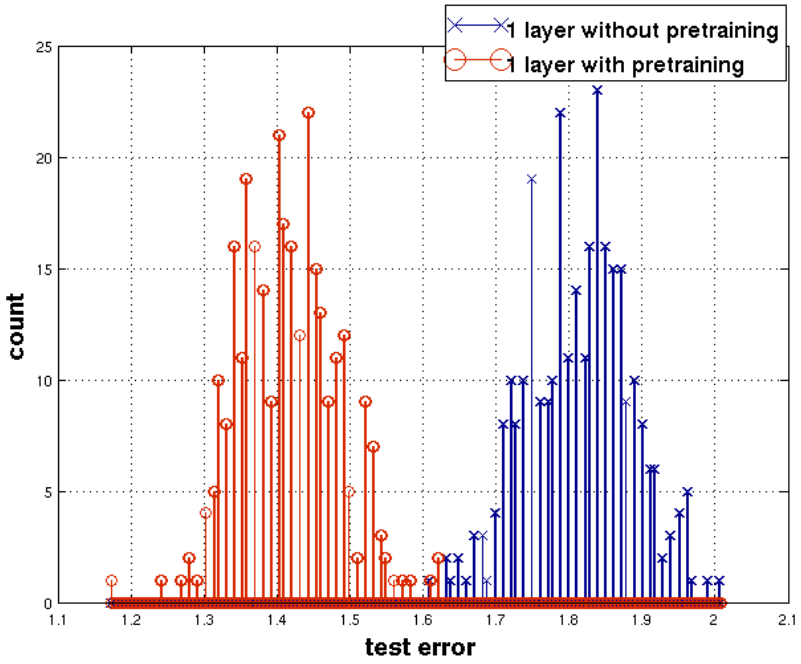


Figure: Histograms presenting the test errors obtained on MNIST using models trained with or without pre-training (400 different initializations each) - 1 hidden layer (from [EMB+09])

Importance of using pre-training

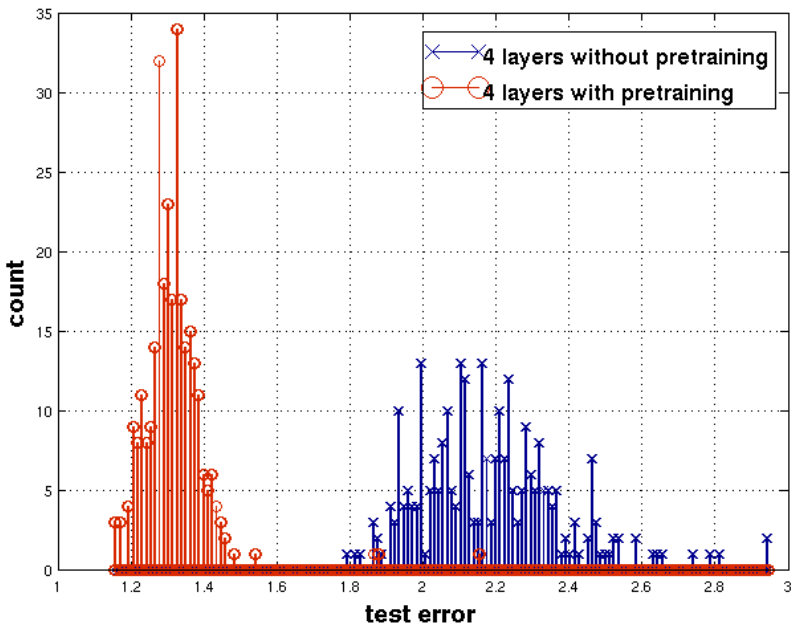


Figure: Histograms presenting the test errors obtained on MNIST using models trained with or without pre-training (400 different initializations each) - 3 hidden layers (from [EMB+09])

Importance of using pre-training

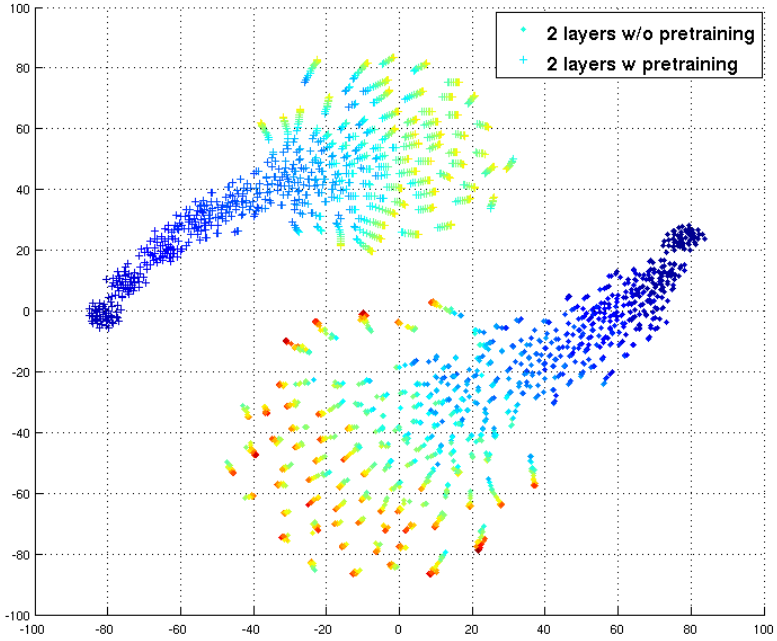


Figure: 2D visualization with tSNE of the functions represented by 50 networks with and 50 networks without pretraining, as supervised training proceeds over MNIST. Color from dark blue to yellow and red indicates a progression in training iterations (training is longer without pre-training). The plot shows models with 2 hidden layers but results are similar with other depths. (from [EMB+09])

Importance of using pre-training

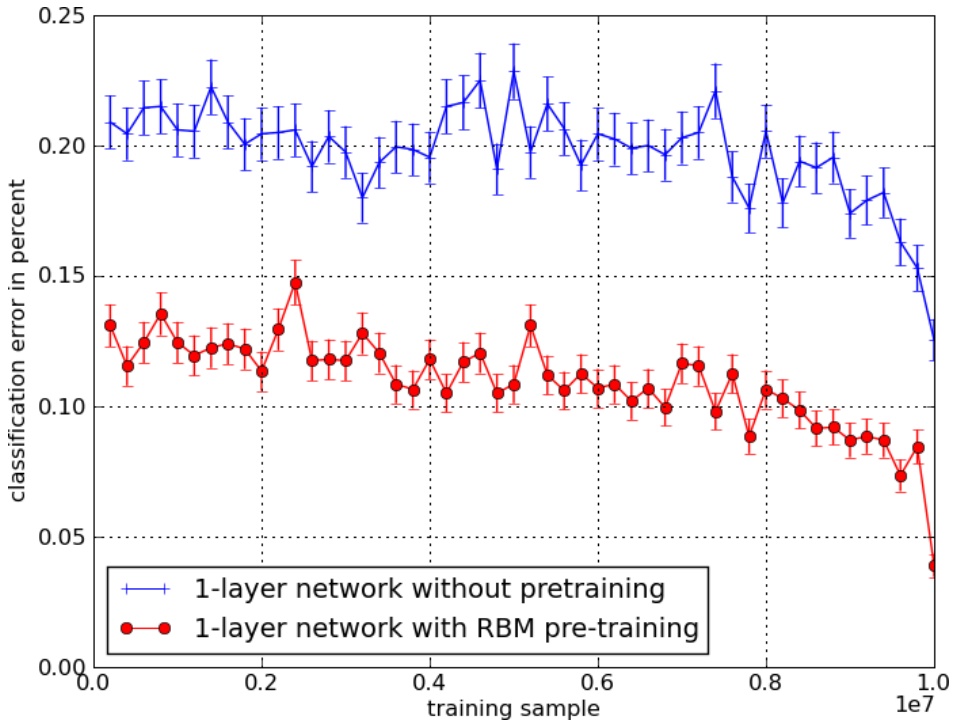


Figure: Error of 1-layer network with RBM pre-training and without, on the 10 million examples from InfiniteMNIST, used for training it. The errors are calculated in the same order as the examples were presented during training. (from [EBC+10])

Today's Outline

Summary

- Energy-based models can be used as memories by putting learned patterns in local minimas of the energy function.
- If nonrestricted Boltzmann Machines are hard to train, there are learning algorithms that work for **RBMs**
- Pre-training the parameters of a deep feed-forward architecture with an unsupervised method (e.g. **RBMs**) usually helps both generalization and test error.
- Besides pre-training, the unsupervised models can be used in a generative manner.

Today's Outline

References I

- Yoshua Bengio, *Learning deep architectures for ai*, Foundations and trends® in Machine Learning **2** (2009), no. 1, 1–127.
- Yoshua Bengio, Ian J. Goodfellow, and Aaron Courville, *Deep learning*, Book in preparation for MIT Press, 2015.
- Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, et al., *Greedy layer-wise training of deep networks*, Advances in neural information processing systems **19** (2007), 153.
- Miguel A Carreira-Perpinan and Geoffrey E Hinton, *On contrastive divergence learning*, Proceedings of the tenth international workshop on artificial intelligence and statistics, Citeseer, 2005, pp. 33–40.

References II

- Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio, *Why does unsupervised pre-training help deep learning?*, The Journal of Machine Learning Research **11** (2010), 625–660.
- Dumitru Erhan, Pierre-Antoine Manzagol, Yoshua Bengio, Samy Bengio, and Pascal Vincent, *The difficulty of training deep architectures and the effect of unsupervised pre-training*, International Conference on Artificial Intelligence and Statistics, 2009, pp. 153–160.
- Geoffrey E Hinton, Peter Dayan, Brendan J Frey, and Radford M Neal, *The" wake-sleep" algorithm for unsupervised neural networks*, Science **268** (1995), no. 5214, 1158–1161.
- John J Hopfield, DI Feinstein, and RG Palmer, *'unlearning'has a stabilizing effect in collective memories*.

References III

- Geoffrey E Hinton, *Training products of experts by minimizing contrastive divergence*, Neural computation **14** (2002), no. 8, 1771–1800.
- Geoffrey Hinton, Simon Osindero, and Yee-Whye Teh, *A fast learning algorithm for deep belief nets*, Neural computation **18** (2006), no. 7, 1527–1554.
- Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, and Pascal Lamblin, *Exploring strategies for training deep neural networks*, The Journal of Machine Learning Research **10** (2009), 1–40.
- Nicolas Le Roux and Yoshua Bengio, *Representational power of restricted boltzmann machines and deep belief networks*, Neural Computation **20** (2008), no. 6, 1631–1649.

References IV

- Andriy Mnih and Karol Gregor, *Neural variational inference and learning in belief networks*, arXiv preprint arXiv:1402.0030 (2014).
- Radford M. Neal, *Connectionist learning of belief networks*, Artificial Intelligence **56** (1992), no. 1, 71 – 113.
- Paul Smolensky, *Information processing in dynamical systems: Foundations of harmony theory*.
- Tijmen Tieleman, *Training restricted boltzmann machines using approximations to the likelihood gradient*, Proceedings of the 25th international conference on Machine learning, ACM, 2008, pp. 1064–1071.