



# Boltzmann Machines

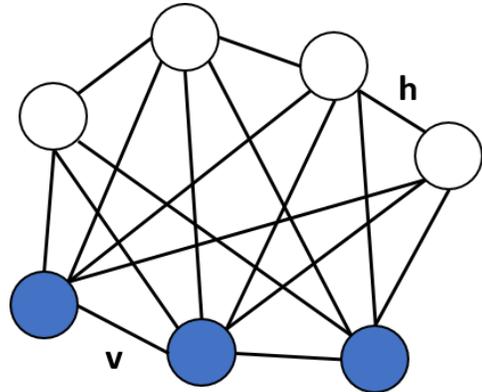
---

INTELLIGENT SYSTEMS FOR PATTERN RECOGNITION (ISPR)

DAVIDE BACCIU – DIPARTIMENTO DI INFORMATICA - UNIVERSITA' DI PISA

DAVIDE.BACCIU@UNIFI.IT

# Boltzmann Machines



An example of **Markov Random Field**

- Visible RV  $\mathbf{v} \in \{0, 1\}$
- Latent RV  $\mathbf{h} \in \{0, 1\}$
- $\mathbf{s} = [\mathbf{v}\mathbf{h}]$

- A linear energy function

$$E(\mathbf{s}) = -\frac{1}{2} \sum_{ij} M_{ij} s_i s_j - \sum_j b_j s_j = -\frac{1}{2} \mathbf{s}^T \mathbf{M} \mathbf{s} - \mathbf{b}^T \mathbf{s}$$

with **symmetric and no self-recurrent** connectivity

- Model parameters  $\theta = \{\mathbf{M}, \mathbf{b}\}$  **encode the interactions** between the variables (observable and not)

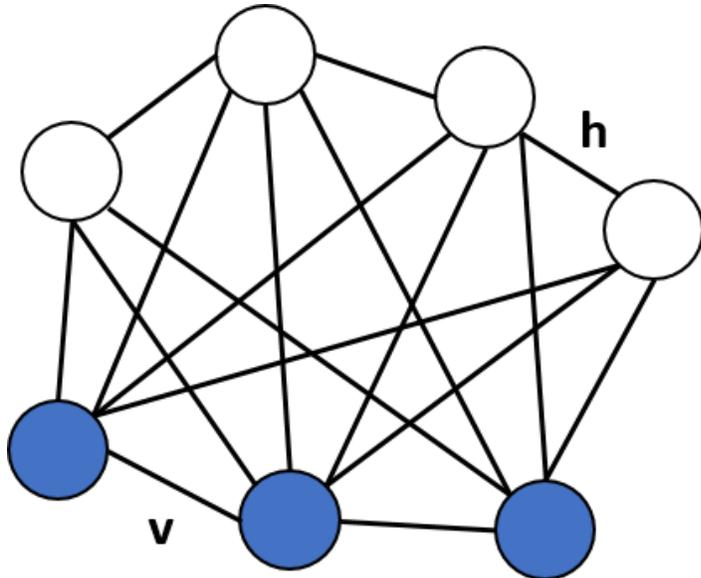
**Boltzmann machines are a type of Recurrent Neural Network**



UNIVERSITÀ DI PISA

# Boltzmann Machines ad Stochastic Networks

- A neural network of units whose activation is determined by a **stochastic function**
  - The state of a unit at a given timestep is **sampled** from a given **probability distribution**
  - The network learns a probability distribution  $P(\mathbf{V})$  from the training patterns



- Network includes both **visible  $v$**  and **hidden  $h$**  units
- Network activity is a sample from **posterior probability given inputs** (visible data)



# Stochastic Binary Neurons

- Spiking point neuron with **binary output**  $s_j$
- Typically, **discrete time** model with time into small  $\Delta t$  intervals
- At each time interval ( $t + 1 \equiv t + \Delta t$ ), the neuron can **emit a spike with probability**  $p_j^{(t)}$

$$s_j^{(t)} = \begin{cases} 1, & \text{with probability } p_j^{(t)} \\ 0, & \text{with probability } 1 - p_j^{(t)} \end{cases}$$

The key is in the definition of the spiking probability (needs to be a function of **local potential**  $x_j$ )

$$p_j^{(t)} \approx \sigma(x_j^{(t)})$$



# General Sigmoidal Stochastic Binary Network

---

Network of  $N$  neurons with binary activation  $s_j$

- Weight matrix  $\mathbf{M} = [M_{ij}]_{i,j} \in \{1, \dots, N\}$
- Bias vector  $\mathbf{b} = [b_j]_j \in \{1, \dots, N\}$

Local neuron **potential**  $x_j$  defined as usual

$$x_j^{(t+1)} = \sum_{i=1}^N M_{ij} s_i^{(t)} + b_j$$

A chosen neuron fires with spiking probability

$$p_j^{(t+1)} = P(s_j^{(t+1)} = 1 | \mathbf{s}^t) = \sigma(x_j^{(t+1)}) = \frac{1}{1 + e^{-x_j^{(t+1)}}}$$

Formulation highlights **Markovian dynamics**



# Update Dynamics

How does the model state (activation of all neurons) evolve in time?

Assume RV to be updated in parallel every  $\Delta t$  (**Parallel dynamics**)

$$P(\mathbf{s}^{(t+1)} | \mathbf{s}^{(t)}) = \prod_{j=1}^N P(\mathbf{s}_j^{(t+1)} | \mathbf{s}^t) = T(\mathbf{s}^{(t+1)} | \mathbf{s}^{(t)})$$

Yielding a **Markov process** for state update

$$P(\mathbf{s}^{(t+1)} = \mathbf{s}' | \mathbf{s}^{(t)} = \mathbf{s}) = \sum_{\mathbf{s}} T(\mathbf{s}' | \mathbf{s}) P(\mathbf{s}^{(t)} = \mathbf{s})$$



# The Boltzmann-Gibbs Distribution

---

Undirected (i.e. symmetric) connectivity enforces **detailed balance condition**

$$P(\mathbf{s})T(\mathbf{s}'|\mathbf{s}) = P(\mathbf{s}')T(\mathbf{s}|\mathbf{s}')$$

Ensures reversible transitions guaranteeing existence of equilibrium (**Boltzmann-Gibbs**) distribution

$$P_{\infty}(\mathbf{s}) = \frac{e^{-E(\mathbf{s})}}{Z}$$

where

- $E(\mathbf{s})$  is the **energy** function
- $Z = \sum_{\mathbf{s}} e^{-E(\mathbf{s})}$  is the **partition** function



# Learning

Ackley, Hinton and Sejnowski (1985)

Boltzmann machines can be trained so that the equilibrium distribution tends towards **any arbitrary distribution across binary vectors** given samples from that distribution

A couple of simplifications to start with

- Bias  $\mathbf{b}$  absorbed into weight matrix  $\mathbf{M}$
- Consider **only visible RV**  $\mathbf{s} = \mathbf{v}$

Use probabilistic learning techniques to fit the parameters, i.e. **maximizing the log-likelihood**

$$\mathcal{L}(\mathbf{M}) = \frac{1}{L} \sum_{l=1}^L \log P(\mathbf{v}^l | \mathbf{M})$$

given the  $P$  visible training patterns  $\mathbf{v}^l$



# Gradient Approach

---

- First, consider the gradient for a single pattern

$$\frac{\partial P(\mathbf{v}|\mathbf{M})}{\partial M_{ij}} = -\langle v_i v_j \rangle + v_i v_j$$

with **free expectations**  $\langle v_i v_j \rangle = \sum_{\mathbf{v}} P(\mathbf{v}) v_i v_j$

- Then, the log-likelihood gradient

$$\frac{\partial \mathcal{L}}{\partial M_{ij}} = -\langle v_i v_j \rangle + \langle v_i v_j \rangle_c$$

with **clamped expectations**  $\langle v_i v_j \rangle_c = \frac{1}{L} \sum_{l=1}^p v_i^l v_j^l$



# A Neural Interpretation, Once Again!

It is **Hebbian learning**!

$$\underbrace{\langle v_i v_j \rangle}_c - \underbrace{\langle v_i v_j \rangle}_{dream}$$

- **wake** part is the standard Hebb rule applied to the empirical distribution of data that the machine sees coming in from the outside world
- **dream** part is an **anti-hebbian term** concerning correlation between units when **generated by the internal dynamics** of the machine

Can only capture quadratic correlation!



# Learning with Hidden Variables

---

- To efficiently capture higher-order correlations we need to introduce hidden RV  $\mathbf{h}$

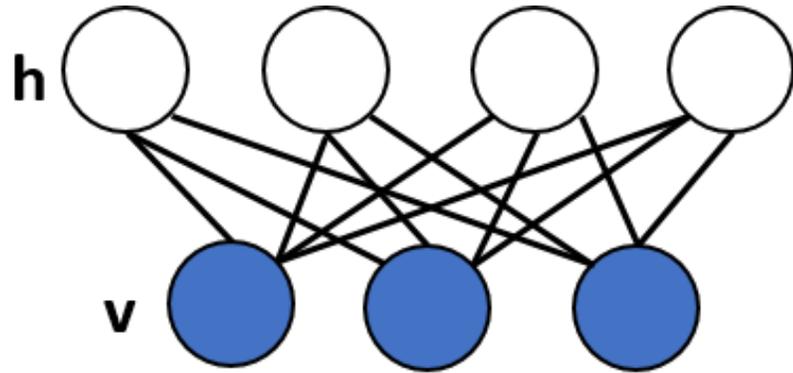
- Again log-likelihood gradient ascent ( $\mathbf{s} = [\mathbf{v}\mathbf{h}]$ )

$$\frac{\partial P(\mathbf{v}|\mathbf{M})}{\partial M_{ij}} = \sum_{\mathbf{h}} s_i s_j P(\mathbf{h}|\mathbf{v}) - \sum_{\mathbf{s}} s_i s_j P(\mathbf{s}) = \langle s_i s_j \rangle_c - \langle s_i s_j \rangle$$

- Expectations again become intractable due to the partition function  $Z$



# Restricted Boltzmann Machines (RBM)



A special Boltzmann machine

- **Bipartite** graph
- Connections only between **hidden and visible units**

- **Energy** function, highlighting bipartition in hidden (***h***) and visible (***v***) units

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{v}^T \mathbf{M} \mathbf{h} - \mathbf{b}^T \mathbf{v} - \mathbf{c}^T \mathbf{h}$$

- Learning (and inference) **becomes tractable** due to graph bipartition which **factorizes distribution**



# The RBM Catch

---

Hidden units are **conditionally independent** given visible units, and **viceversa**

$$P(h_j|\mathbf{v}) = \sigma\left(\sum_i M_{ij}v_i + c_j\right)$$

$$P(v_i|\mathbf{h}) = \sigma\left(\sum_j M_{ij}h_j + b_i\right)$$

They can be updated in batch!



# Training Restricted Boltzmann Machines

---

Again by likelihood maximization, yields

$$\frac{\partial \mathcal{L}}{\partial M_{ij}} = \underbrace{\langle v_i h_j \rangle_c}_{data} - \underbrace{\langle v_i h_j \rangle}_{model}$$

A **Gibbs sampling** approach

Wake

- Clamp data on  $v$
- Sample  $v_i h_j$  for all pairs of connected units
- Repeat for all elements of dataset

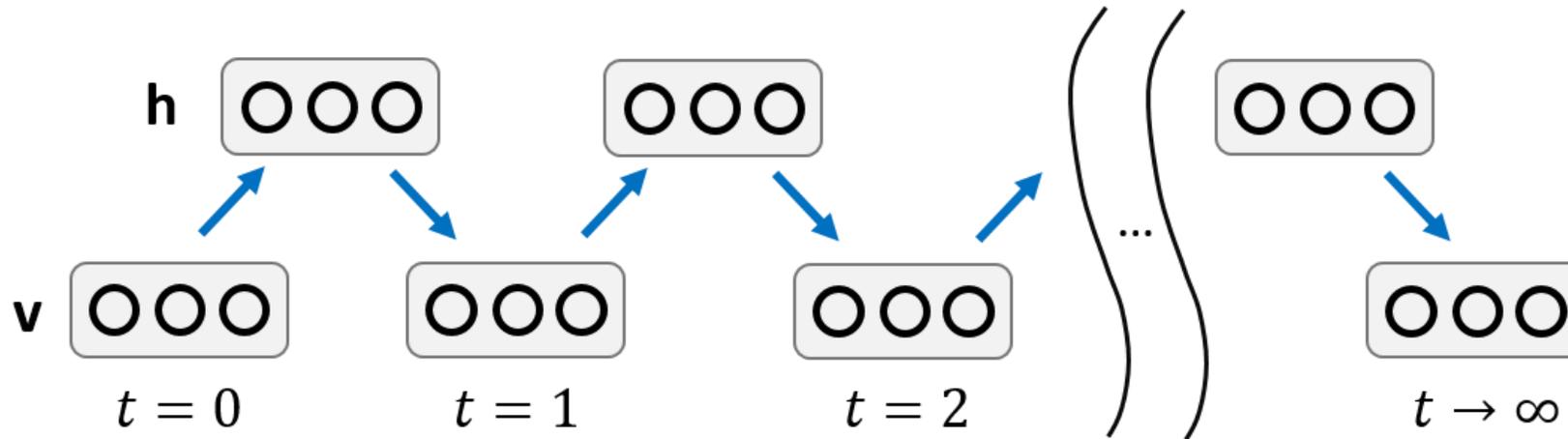
Dream

- Don't clamp units
- Let network reach equilibrium
- Sample  $v_i h_j$  for all pairs of connected units
- Repeat many times to get a good estimate



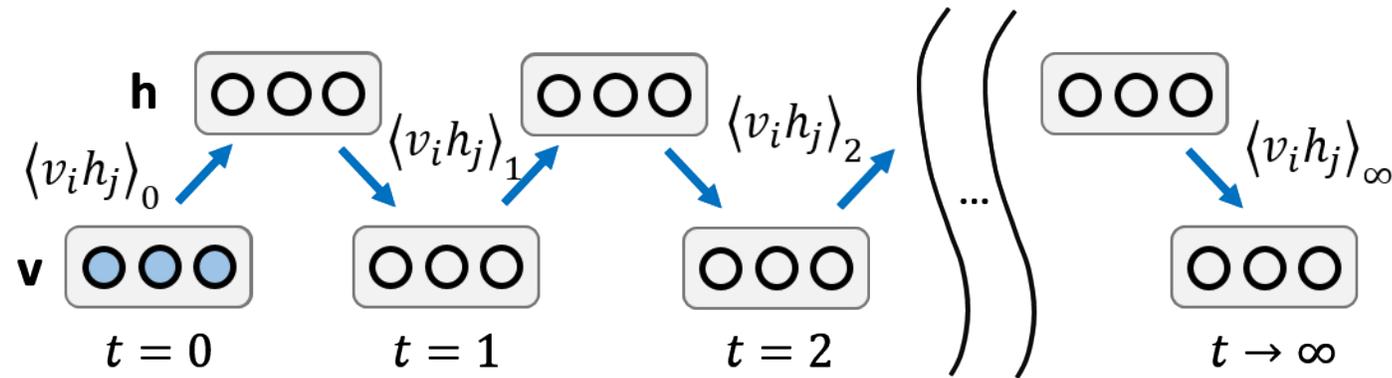
# Gibbs-Sampling RBM

$$\frac{\partial \mathcal{L}}{\partial M_{ij}} = \underbrace{\langle v_i h_j \rangle_c}_{data} - \underbrace{\langle v_i h_j \rangle}_{model}$$



It is difficult to obtain an unbiased sample of the second term

# Gibbs-Sampling RBM: Plugging-in Data

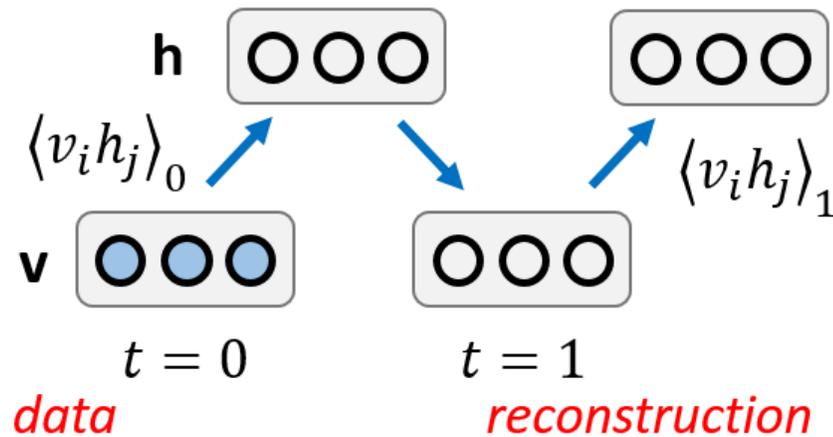


1. Start with a **training vector on the visible** units
2. **Alternate** between updating all the hidden units in parallel and updating all the visible units in parallel (**iterate**)

$$\frac{\partial \mathcal{L}}{\partial M_{ij}} = \underbrace{\langle v_i h_j \rangle_0}_{data} - \underbrace{\langle v_i h_j \rangle_\infty}_{model}$$

# Contrastive-Divergence Learning

Gibbs sampling can be painfully slow to converge (**high variance**)



1. Clamp a training vector  $v^l$  on **visible units**
2. Update **all hidden** units in parallel
3. Update all the visible units in parallel to get a **reconstruction**
4. Update the hidden units again

$$\underbrace{\langle v_i h_j \rangle_0}_{\text{data}} - \underbrace{\langle v_i h_j \rangle_1}_{\text{reconstruction}}$$



# What does Contrastive Divergence Learn?

---

- A very **crude approximation** of the gradient of the **log-likelihood**
  - It does not even follow the gradient closely
- **More closely approximating** the gradient of an objective function called the **Contrastive Divergence**
  - It ignores one tricky term in this objective function, so it is not even following that gradient
- Sutskever and Tieleman (2010) have shown that it is **not following the gradient of any function**



# RBM-CD in Code

```
for epoch = 1: maxepoch

%--- Compute wake part
data = dataOr > rand ( size ( data ) ); %Stochastic clamped input
poshidP = 1 ./ ( 1 + exp ( -data*W - bh ) ); %Hidden activation probability
wake = data ' * poshidP ;
% Alternatively : wake = data ' * ( poshidP > rand (size(poshidP)) ) ;

%---Compute dream part
poshidS = poshidP > rand ( size ( poshidP ) ); %Stochastic hidden activation
reconDataP = 1 ./ ( 1 + exp ( -poshidS*W' - bv ) ); %Data reconstruction probability
reconData = reconDataP > rand ( size ( data ) ); % Stochastic reconstructed data
neghidP = 1 ./ ( 1 + exp ( -reconData*W - bh ) );
dream = reconData ' * neghidP ;
% Alternatively : dream = reconData '*( neghidP > rand (size( neghidP))) ;

% Reconstruction error
err = sum ( sum ( ( data - negdata ) .^ 2 ) );

%---CD_1 Update
deltaW = (wake - dream) / numcases ;
deltaBh = (sum ( poshidP ) - sum ( neghidP )) / numcases ;
deltaBv = (sum ( data ) - sum ( reconData )) / numcases ;
...
end
```

# Boltzmann Machines in Python

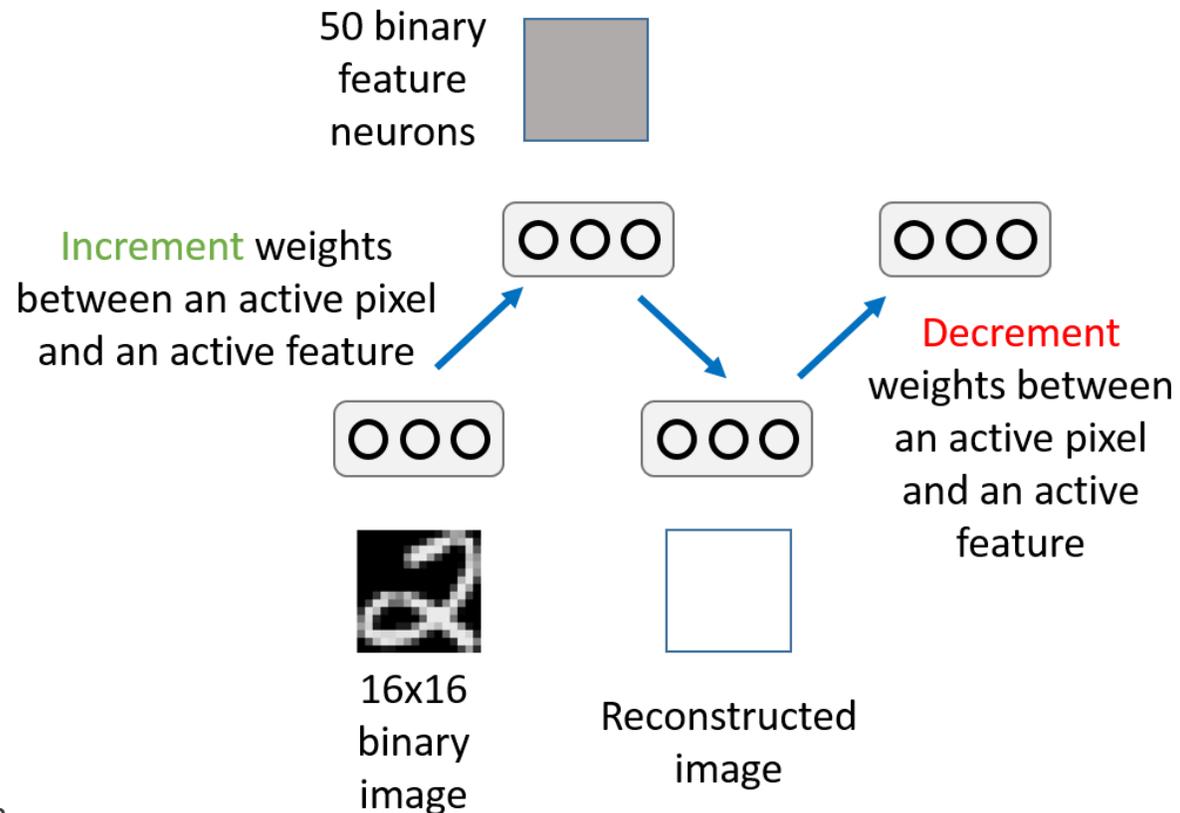
---

- Boltzmann machines implementations are available in all major deep learning libraries: Theano, Torch, Tensorflow, ...
- [sklearn.neural\\_network](#) contains an implementation of a binary RBM
- Little support in Python libraries for generative and graphical models
- Plenty of personal implementations on Github



# Character Recognition

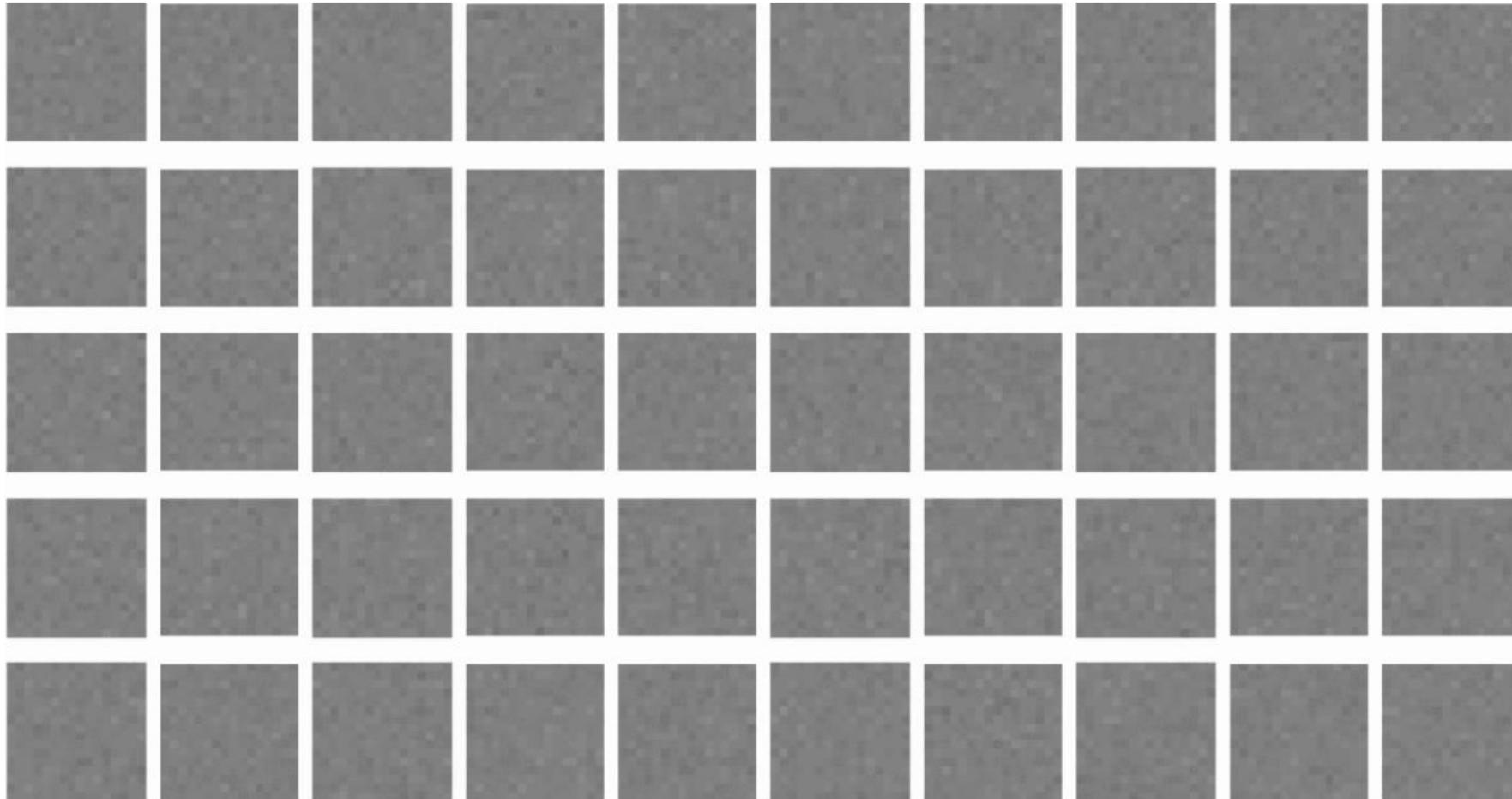
Learning good features for reconstructing images of number 2 handwriting



Slide credit goes to G. Hinton

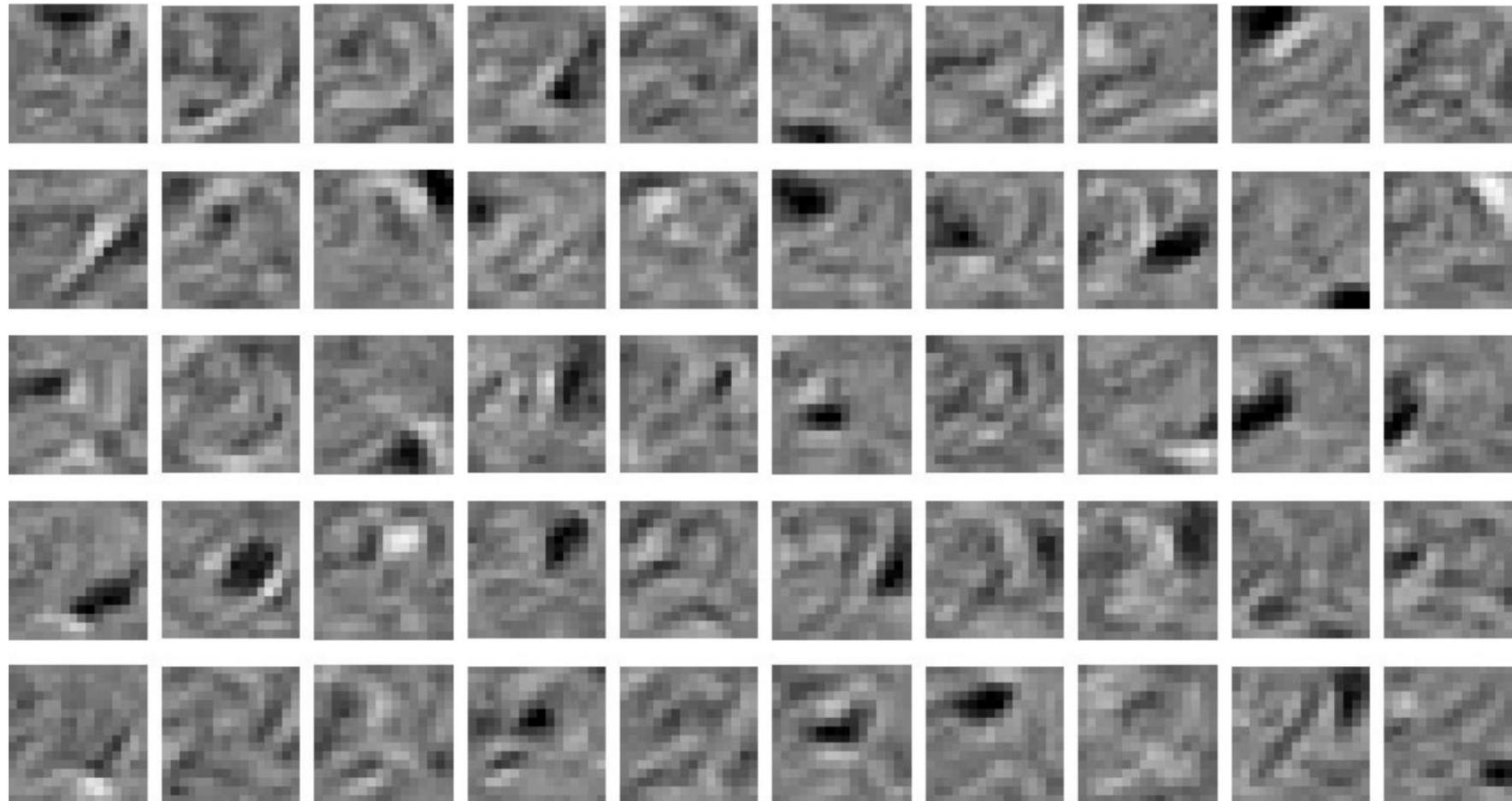
# Weight Learning

---

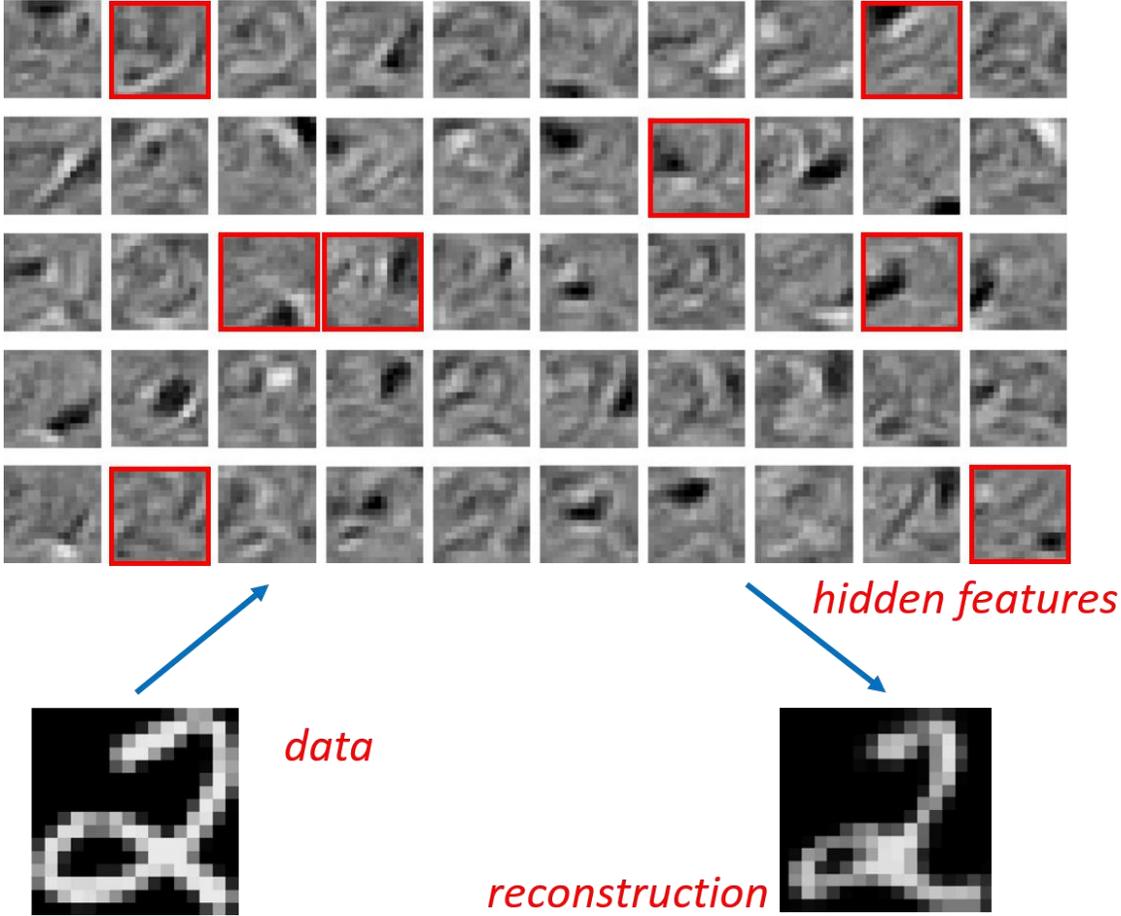


# Final Weights

---

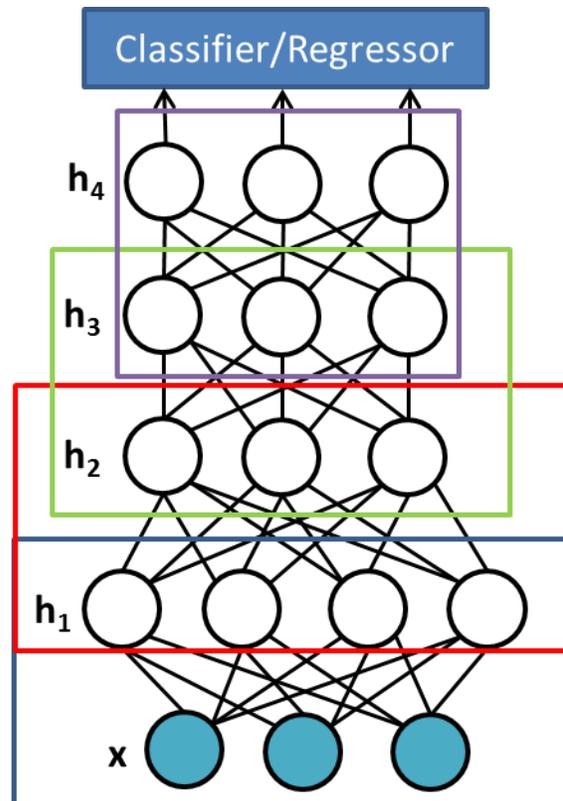


# Digit Reconstruction



# One Last Final Reason for Introducing RBM

## Deep Belief Network



The fundamental building block for popular deep learning architectures (Deep RBM as well)

A network of **stacked RBM** trained layer-wise by **Contrastive Divergence** plus a supervised read-out layer



# Take Home Messages

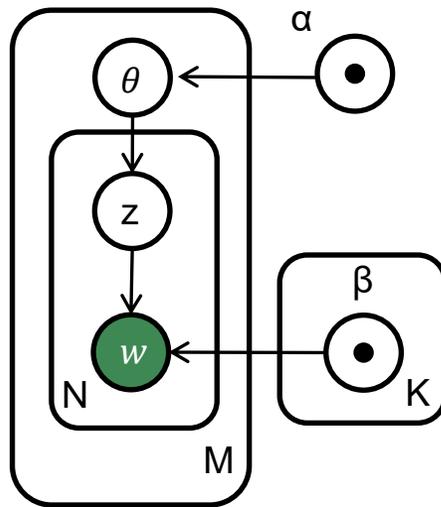
---

- Boltzmann Machines
  - A first bridge between (undirected) generative models and (recurrent) neural networks
  - **Neural activity** regulated by stochastic behavior
  - Training has both a ML and an **Hebbian** interpretation
  - Require approximations for computational tractability
- Restricted Boltzmann Machines
  - Tractable model thanks to **bipartite connectivity**
  - Trained by a very short Gibbs sampling (**contrastive divergence**)
  - Can be very powerful if **stacked** (deep learning)



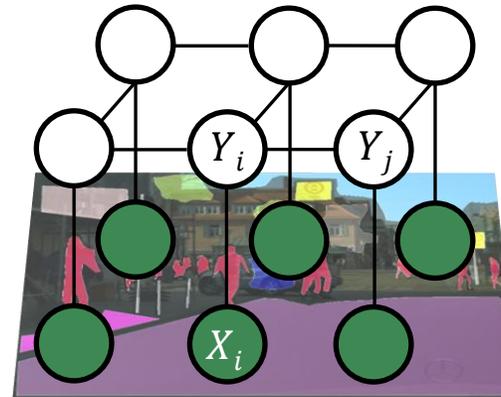
# Generative Graphical Models – Wrap up

## Bayesian Models/Nets



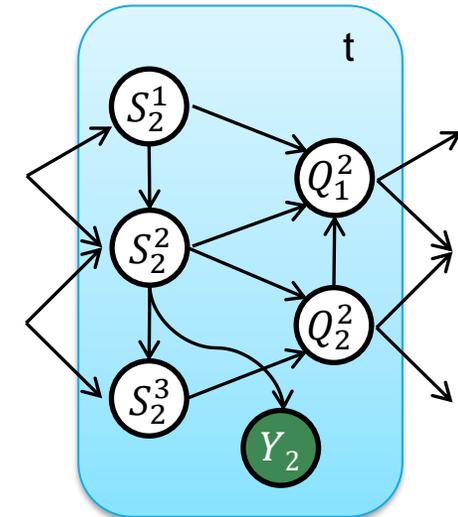
- Unsupervised data understanding
- Interpretability
- Weak on supervised performance

## Markov Random Fields



- Knowledge and constraints through feature function
- CRF: the supervised way to generative
- Computationally heavy

## Dynamic Models



- Topology unfolds on data structure
- Structured data processing
- Complex causal relationships



# Tractability

---

- Consider **using generative models** when
  - Need interpretability
  - Need to incorporate prior knowledge
  - Unsupervised learning or learning with partially observable supervision
  - Need reusable/portable learned knowledge
- Consider **avoiding generative models** when
  - Having tight computational constraints
  - Dealing with raw, noisy low-level data
- **Variational** inference and **sampling**
  - Efficient ways to learn an approximation to intractable distributions
- Neural networks can be **used as variational functions** or to **implement sampling** processes



# Next Lecture

---

Introduction to the deep learning module

Deep Autoencoders

- Autoencoders and dimensionality reduction
- Neural autoencoders (sparse, denoising contractive)
- Deep neural autoencoders and pretraining
- Deep generative-based autoencoders
- Visualization and multi-modal data fusion with autoencoders

