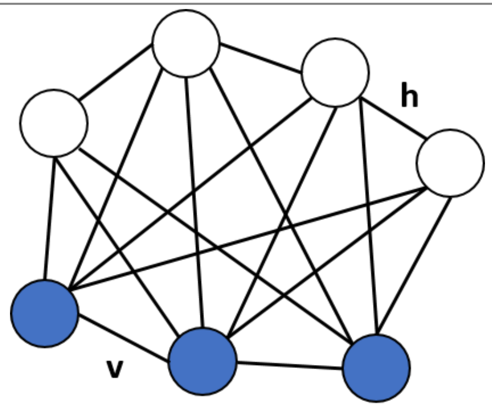
Boltzmann Machines

INTELLIGENT SYSTEMS FOR PATTERN RECOGNITION (ISPR)

DAVIDE BACCIU – DIPARTIMENTO DI INFORMATICA - UNIVERSITA’ DI PISA

DAVIDE.BACCIU@UNIPI.IT

Boltzmann Machines



An example of Markov Random Field

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

- A linear energy function

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

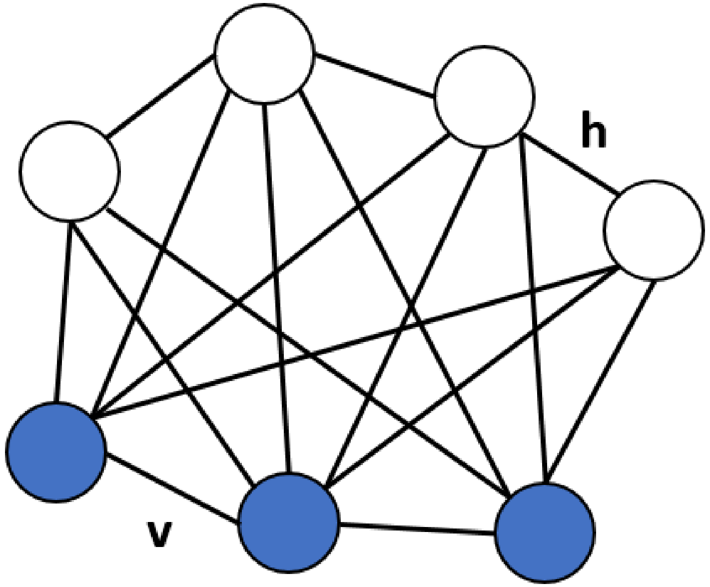
with symmetric and no self-recurrent connectivity

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

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

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 $P(\boldsymbol{V})$ from the training patterns



- Network includes both visible $\boldsymbol{v}$ and hidden $\boldsymbol{h}$ 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 Δt 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)

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

UNIVERSITÀ DI PISA

General Sigmoidal Stochastic Binary Network

Network of N neurons with binary activation s_j

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

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

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

Formulation highlights Markovian dynamics

The Boltzmann-Gibbs Distribution

Undirected connectivity ensures reversible transitions guaranteeing existence of equilibrium (Boltzmann-Gibbs) distribution

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

where

- $E(\boldsymbol{s})$ is the energy function
- $Z = \sum_{\boldsymbol{s}} e^{-E(\boldsymbol{s})}$ 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 $\boldsymbol{b}$ absorbed into weight matrix $\boldsymbol{M}$
- Consider only visible RV $\boldsymbol{s} = \boldsymbol{v}$

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

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

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

UNIVERSITÀ DI PISA

Gradient Approach

- First, consider the gradient for a single pattern

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

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

- 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!

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

- 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

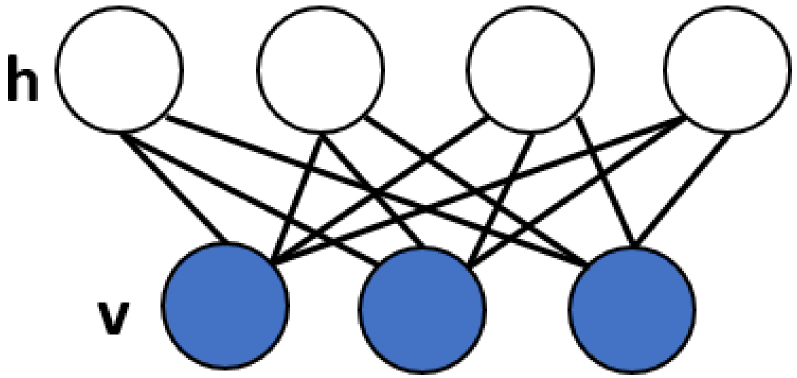
- To efficiently capture higher-order correlations we need to introduce hidden RV $\boldsymbol{h}$
- Again log-likelihood gradient ascent ($\boldsymbol{s} = [\boldsymbol{vh}]$)

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

- Expectations again become intractable due to the partition function Z

UNIVERSITÀ DI PISA

Restricted Boltzmann Machines (RBM)



A special Boltzmann machine
- Bipartite graph
- Connections only between hidden and visible units

- Energy function, highlighting bipartition in hidden ($\boldsymbol{h}$) and visible ($\boldsymbol{v}$) units

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

- 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

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

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

They can be updated in batch!

Training Restricted Boltzmann Machines

Again by likelihood maximization, yields

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

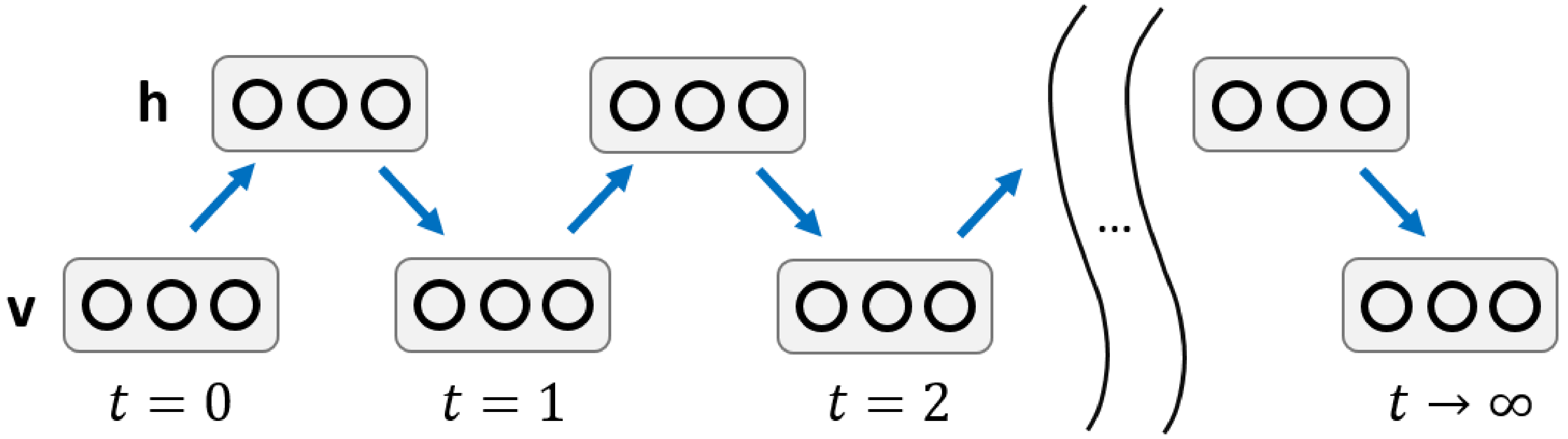
A Gibbs sampling approach

Wake
- Clamp data on $\boldsymbol{v}$
- 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

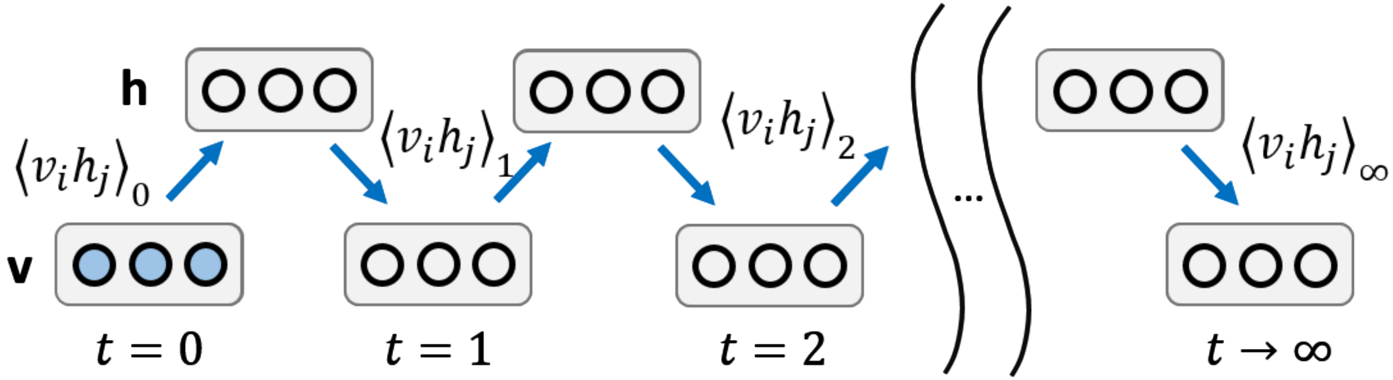
Gibbs-Sampling RBM

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



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

Gibbs-Sampling RBM: Plugging-in Data

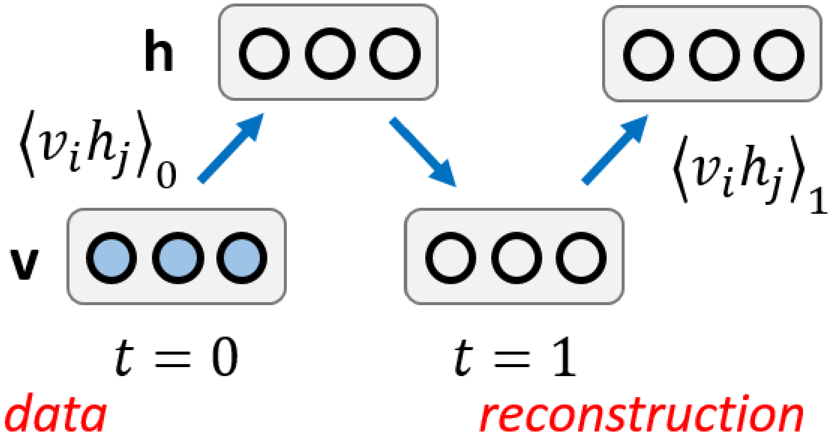


1. Start with a training vector on the visible units
2. Alternate between updating all the hidden units in parallel and updating all the visible units in parallel (iterate)

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

Contrastive-Divergence Learning

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



1. Clamp a training vector $\boldsymbol{v}^l$ 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

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

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

So Why Using it?

Because **He** says so!

It works well enough in many significant applications

RBM-CD in Code

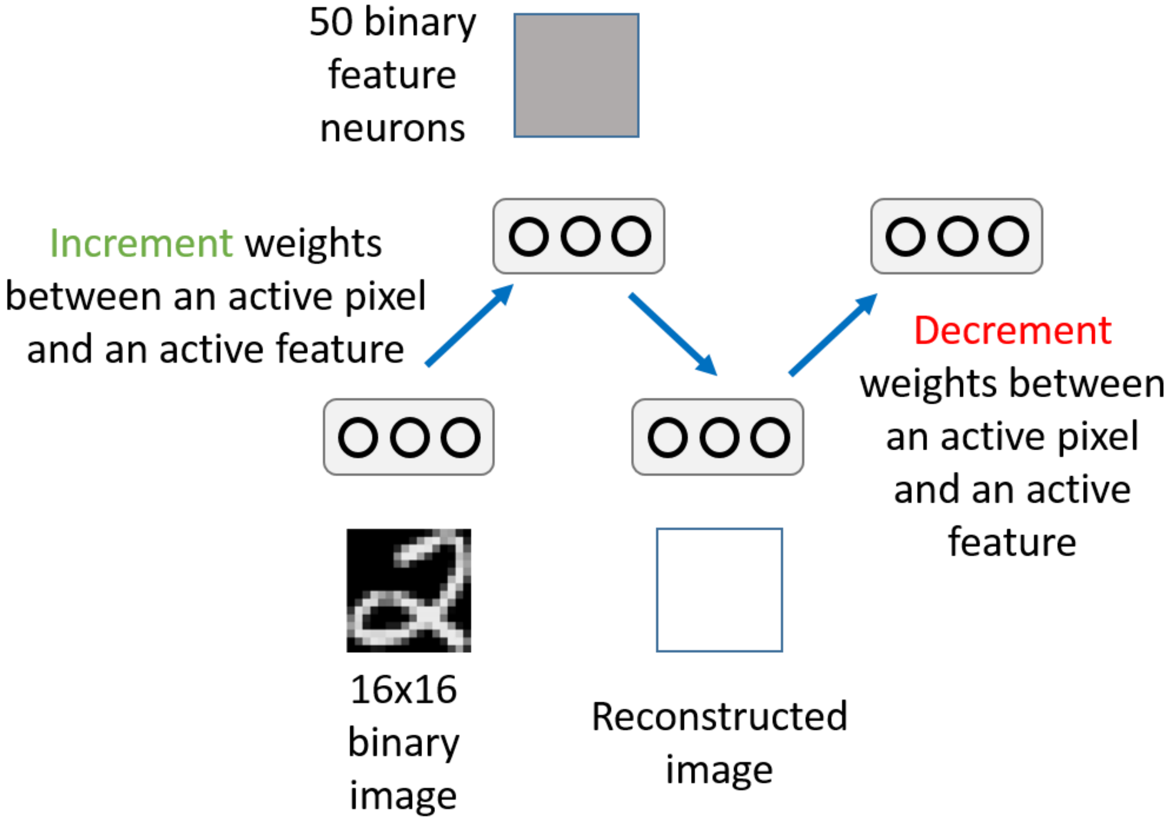
```
for epoch = 1: maxepoch

%−−− 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 ( s i z e ( 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 ( s i z e ( neghidP ) ) ) ;

% Reconstruction error
e r r = sum (sum (  ( data−negdata ) . ^ 2 ) ) ;

%−−−CD_1 Update
deltaW = (wake − dream ) / numcases ;
deltaBh = (sum ( poshidP ) − sum ( neghidP ) ) / numcases ;
deltaB v = (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

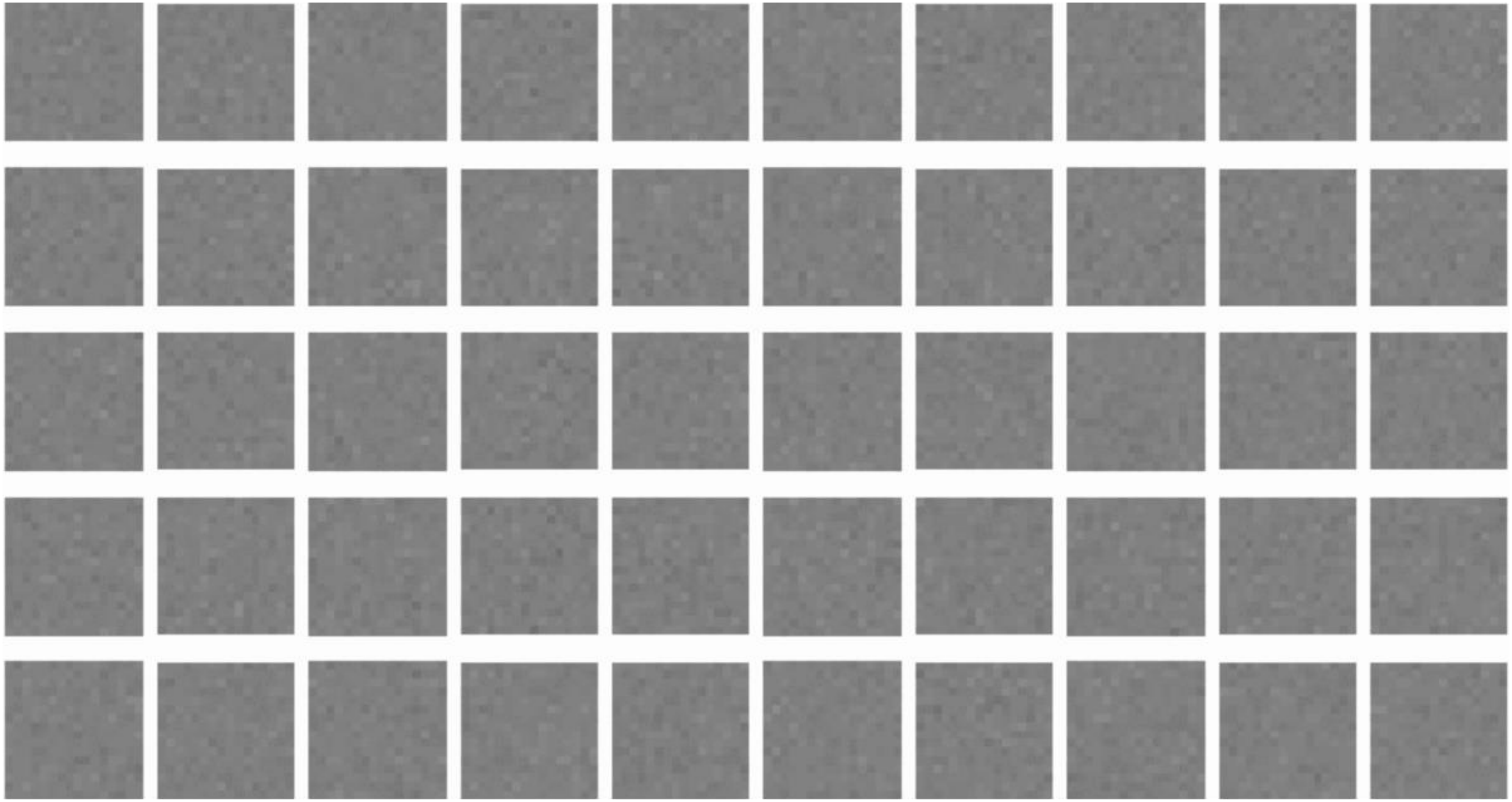
Character Recognition

Learning good features for reconstructing images of number 2 handwriting

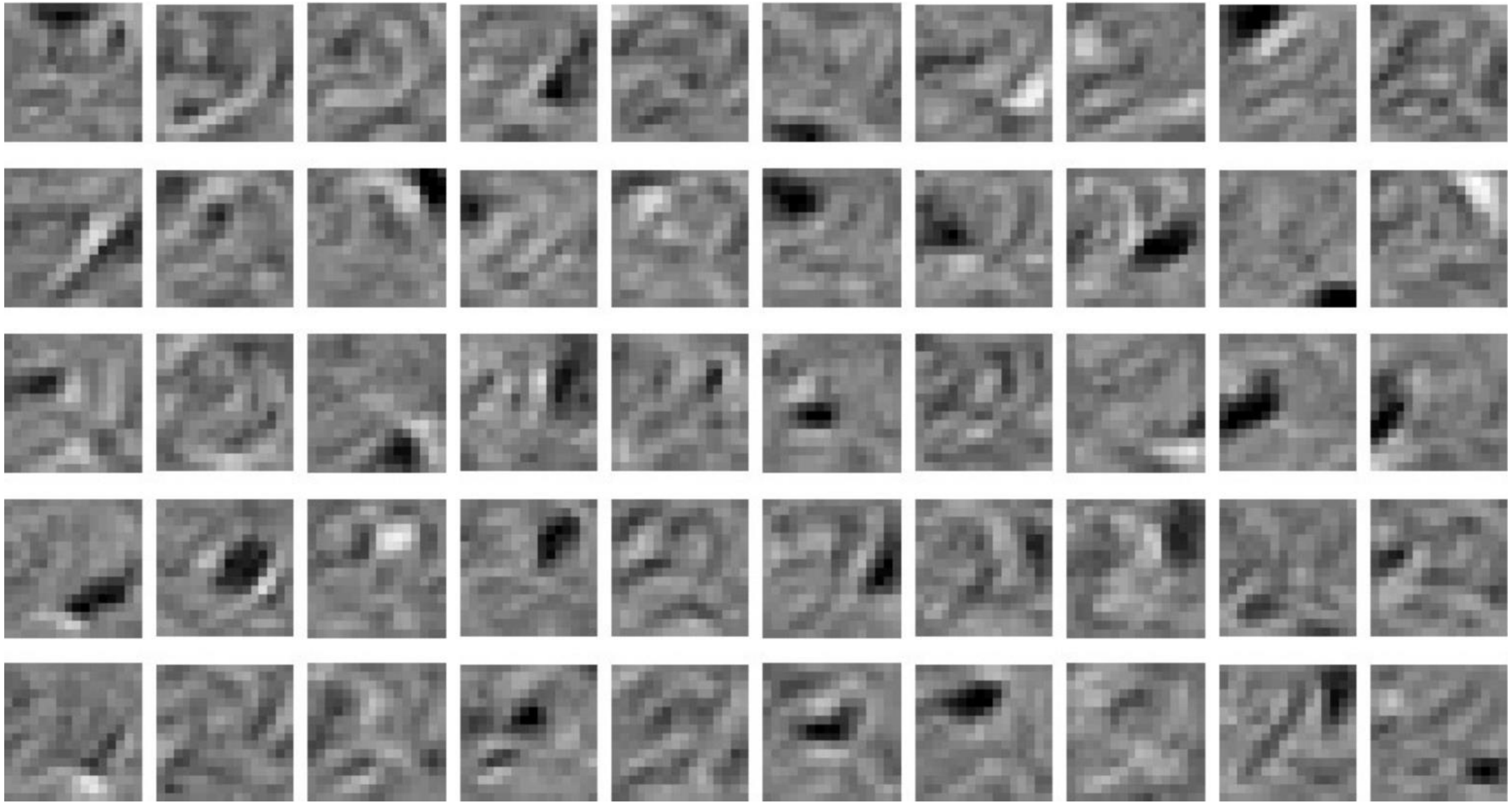


Slide credit goes to G. Hinton

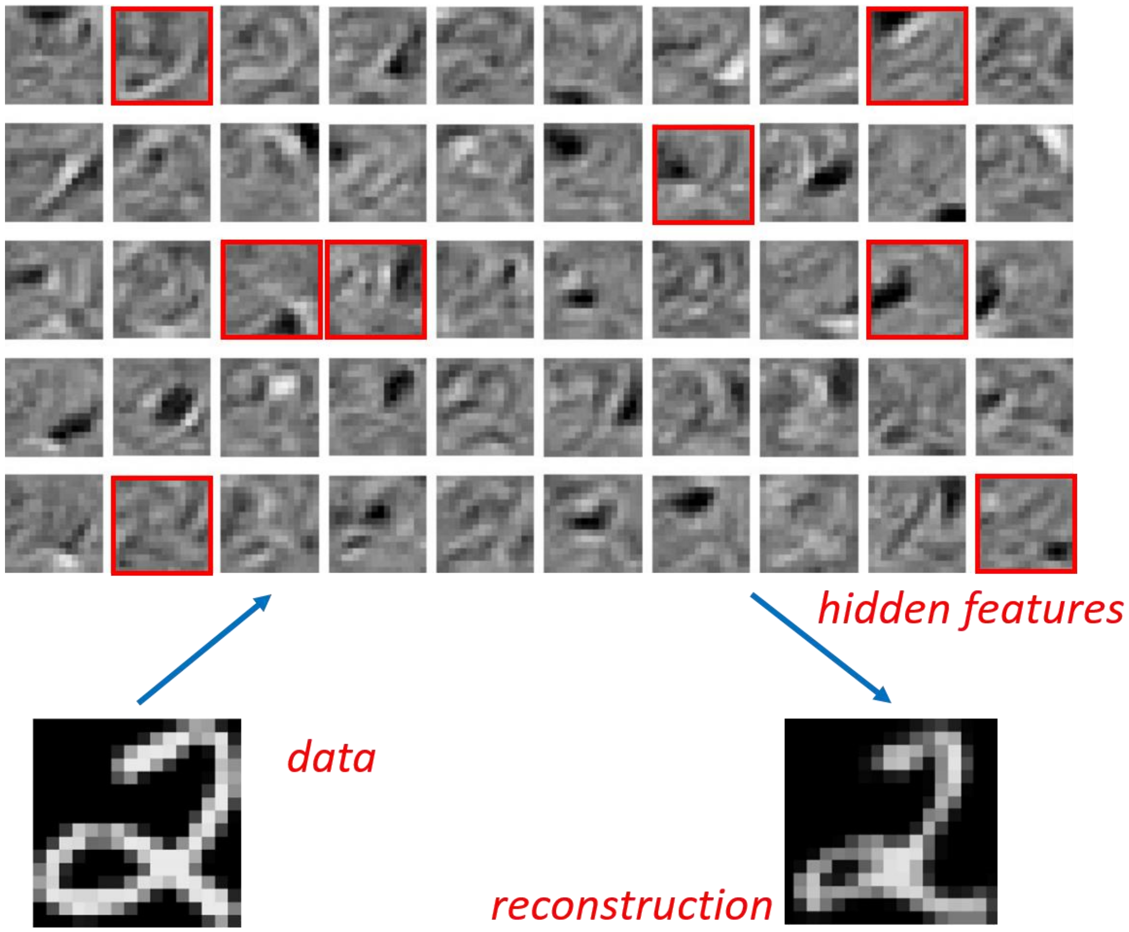
Weight Learning



Final Weights



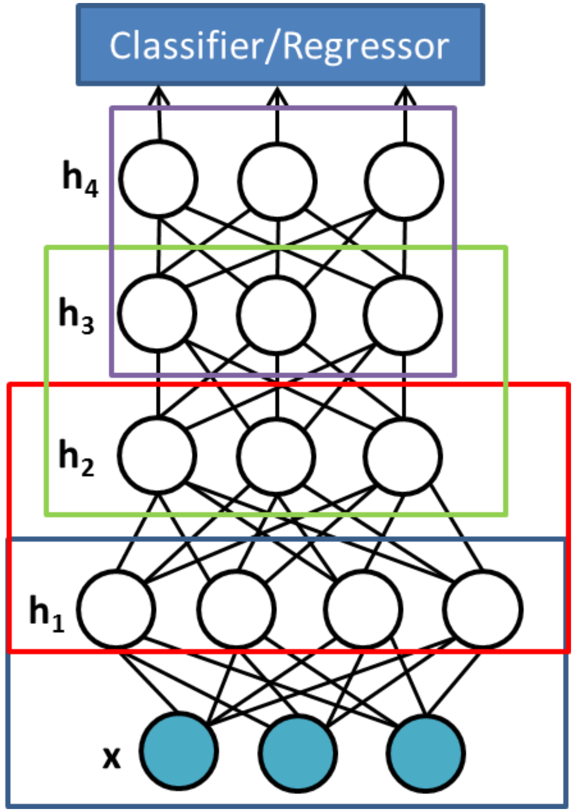
Digit Reconstruction



Università di Pisa

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)

IN SUPREMÆ DIGNITATIS 1343 UNIVERSITÀ DI PISA

Next Two Lectures

Convolutional Neural Networks

- Introduction to the Deep Learning module
- Basic components of a convolutional neural network
  - Convolutions, striding, pooling, ReLu, batchnorm layers
- Notable architectures
  - From AlexNet to ResNets and MobileNets
- CNN besides simple object recognition
  - Semantic segmentation, sequence processing, dilated convolutions