Undirected Graphical Models

Generative and Deep Learning (GDL)

Davide Bacciu (davide.bacciu@unipi.it)

UNIVERSITÀ DI PISA

Lecture Outline

Markov Random Fields

- Undirected graphical models
- Express constraints between RV without needing to use probabilities
- Conditional MRF learning discriminative posteriors

MRF with tractable inference

- Linear Conditional MRF
- Exact posterior inference with Sum-Product

MRF with approximated inference

- (Restricted) Boltzmann Machines
- Learning by Gibbs sampling (contrastive divergence)

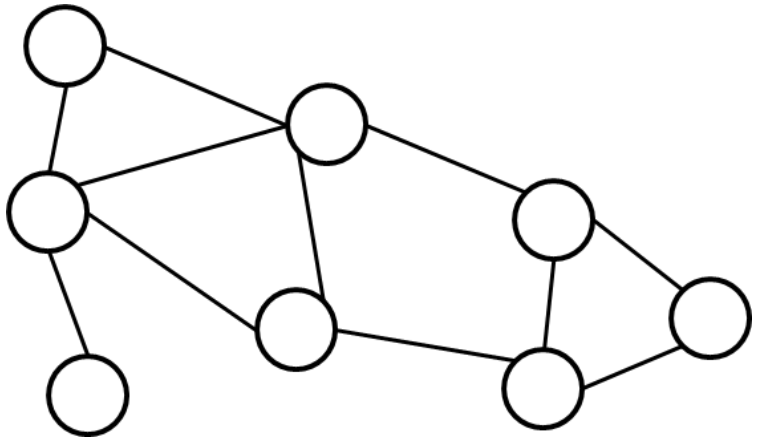
Module wrap-up

Markov Random Fields

Markov Random Fields (MFRs)

Repetita

- Undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ (a.k.a. Markov Networks)
- Nodes $v \in \mathcal{V}$ represent random variables X_v
  - Shaded ⇒ observed
  - Empty ⇒ un-observed
- Edges $e \in \mathcal{E}$ describe bi-directional dependencies between variables (constraints)



Likelihood & Potential Functions

Define $\boldsymbol{X} = X_1, \ldots, X_N$ as the RVs associated to the N nodes in the undirected graph $\mathcal{G}$

$$P(\boldsymbol{X}) = \frac{1}{Z} \prod_C \psi_C(\boldsymbol{X}_C)$$

- $\boldsymbol{X}_C \rightarrow$ RV associated with nodes in the maximal clique C
- $\psi_C(\boldsymbol{X}_C) \rightarrow$ (positive) potential function for clique C
  - They are not probabilities!
  - Express which configurations of the local variables are preferred
- $Z \rightarrow$ partition function ensuring normalization

$$Z = \sum_{\boldsymbol{X}} \prod_C \psi_C(\boldsymbol{X}_C)$$

Definition (Boltzmann distribution)

A convenient and widely used strictly positive representation of the potential functions is

$$\psi_C(\boldsymbol{X}_C) = \exp\{-E(\boldsymbol{X}_C)\}$$

where $E(\boldsymbol{X}_C)$ is the energy function

Factorizing Potential Functions

In general, we will assume to work with MRF where the partition functions factorize as

$$\psi_C(\boldsymbol{X}_C) = \exp\left(\sum_k \theta_{Ck} f_{Ck}(\boldsymbol{X}_C)\right)$$

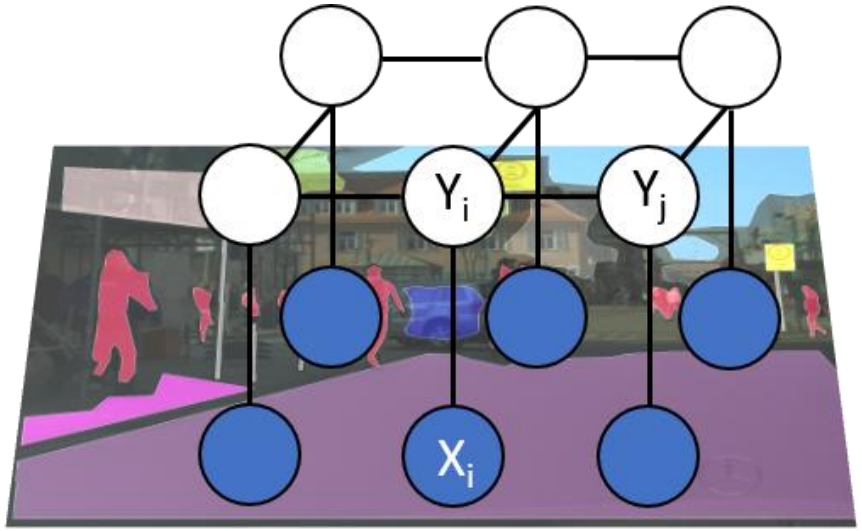
where

- f_{Ck} (or f_k) are feature functions or sufficient statistics to compute the potential of clique C
- $\theta_{Ck} \in \mathbb{R}$ are parameters
- k indexes over the available feature functions

Feature functions

What does a feature function $f_k(\boldsymbol{X}_k, \boldsymbol{Y}_k)$ do?

- Represent couplings or constraints between random variables
- Often very simple, such as linear functions



- Make noisy binary pixel X_i and its clean version Y_i have same sign

$$f_i(X_i, Y_i) = X_i Y_i$$

- Constrain nearby interpretations to be similar

$$f_{ij}(Y_i, Y_j) = Y_i Y_j$$

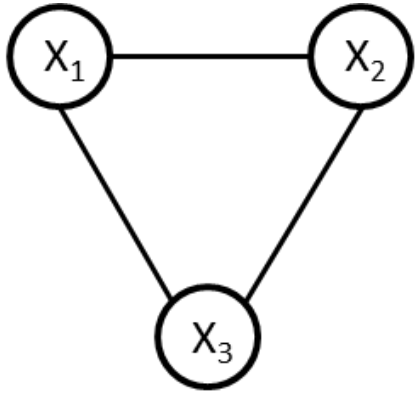
Background Segmentation

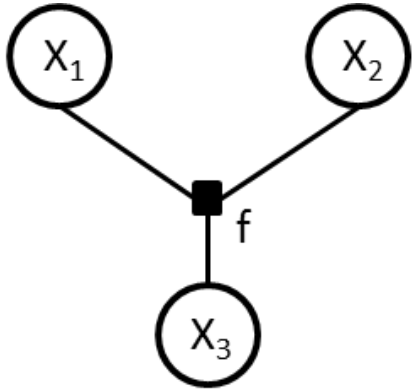
Background Segmentation

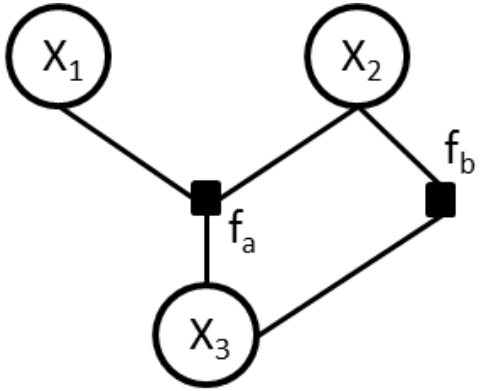
Factor Graphs

Undirected graphical models do not express the factorization of potentials into feature functions ⇒ factor graphs

- RV are again circular nodes
- Factors f_{Ck} are denoted as square nodes
- Edges connect a factor to the RV







$$\psi(X_1, X_2, X_3) = f(X_1, X_2, X_3)$$
$$\psi(X_1, X_2, X_3) = f_a(X_1, X_2, X_3) f_b(X_2, X_3)$$

Conditional Random Fields (CRF)

Restricting to Conditional Probabilities

In ML a part of the random variables can be assumed to be always observable ⇒ input data

- $\boldsymbol{X}_k$ - observable inputs in the factor k
- $\boldsymbol{Y}_k$ - hidden (or partly observable) RV
- $f_k(\boldsymbol{X}_k, \boldsymbol{Y}_k)$ - factor feature function

Under this assumption we can directly model the conditional distribution

$$P(\boldsymbol{Y}|\boldsymbol{X}) = \frac{1}{Z(\boldsymbol{X})} \prod_k \exp\{\theta_k f_k(\boldsymbol{X}_k, \boldsymbol{Y}_k)\}$$

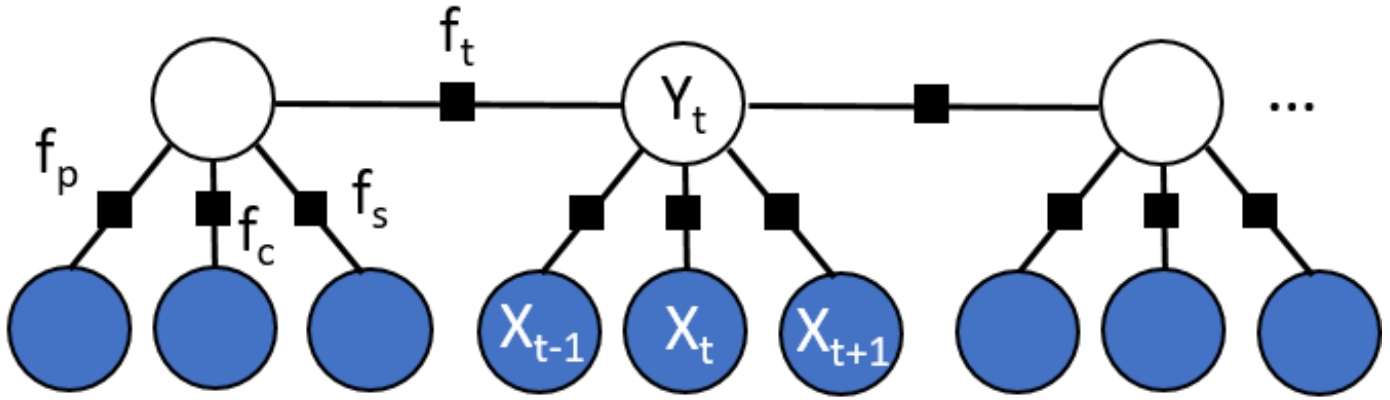
where $\boldsymbol{X}$ is the joint input that is always available

$$Z(\boldsymbol{X}) = \sum_{\boldsymbol{y}} \prod_k \exp\{\theta_k f_k(\boldsymbol{X}_k, \boldsymbol{Y}_k = \boldsymbol{y}_k)\}$$

Exact Inference – Linear CRF

CRF for Sequential Data

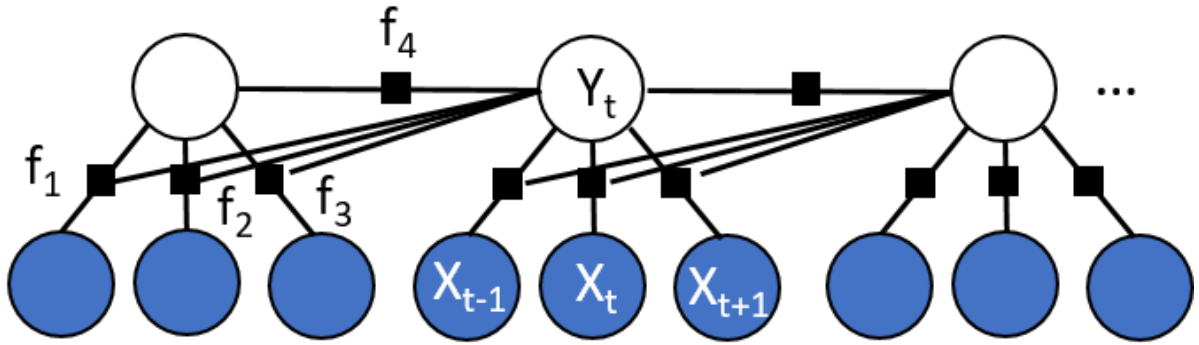
Modeling relative influence of suffix and prefix symbols



$$P(\boldsymbol{Y}|\boldsymbol{X},\theta) = \frac{1}{Z(\boldsymbol{X})}\prod_t \exp\{\theta_p f_p(X_{t-1},Y_t) + \theta_c f_c(X_t,Y_t) + \theta_s f_s(X_{t+1},Y_t) + \theta_t f_t(Y_{t-1},Y_t)\}$$

Generic LCRF Formulation

Modeling explicitly input influence on transition



General Linear CRF Likelihood:

$$P(\boldsymbol{Y}|\boldsymbol{X},\theta) = \frac{1}{Z(\boldsymbol{X})}\prod_t\prod_k \exp(\theta_k f_k(Y_t, Y_{t-1}, \boldsymbol{X}_t))$$

Examples of Linear CRF Feature Functions

Linear CRF have found wide applications

- Text processing: POS-tagging, semantic role identification
- Bioinformatics: sequence alignment, protein structure prediction

Feature functions have often the form $f_k(\boldsymbol{X}_k, \boldsymbol{Y}_k) = \mathbb{1}_{\boldsymbol{y}_k = \hat{\boldsymbol{y}}_k} q(\boldsymbol{X}_c)$

- f_k is non-zero only for a specific output configuration $\hat{\boldsymbol{y}}_k$
- f_k then depends only on $\boldsymbol{X}_k$ (i.e. features are not shared by classes)

Observation functions $q(\boldsymbol{X}_c)$: word begins with capital, ends with -ing, ...

Posterior Inference in LCRF

Is there an equivalent of the smoothing problem in LCRF? Yes: $P(Y_t, Y_{t-1}|\boldsymbol{X})$

- Solved by (exact) forward-backward inference
- Sum-product message passing on the LCRF factor graph

$$P(Y_t, Y_{t-1}|\boldsymbol{X}) \propto \alpha_{t-1}(Y_{t-1})\psi_t(Y_t, Y_{t-1}, X_t)\beta_t(Y_t)$$

Clique weighting

$$\psi_t(Y_t, Y_{t-1}, X_t) = \exp\{\theta_e f_e(X_t, Y_t) + \theta_t f_t(Y_{t-1}, Y_t)\}$$

Forward Message

$$\alpha_t(i) = \sum_j \psi_t(i, j, X_t)\alpha_{t-1}(j)$$

Backward Message

$$\beta_t(j) = \sum_i \psi_{t+1}(i, j, X_{t+1})\beta_{t+1}(i)$$

Training LCRF

Maximum (conditional) log-likelihood

$$\max_{\theta} \mathcal{L}(\theta) = \max_{\theta} \sum_{n=1}^{n} \log P(\boldsymbol{y}^n | \boldsymbol{x}^n, \theta)$$

Substituting LCRF conditional formulation

$$\mathcal{L}(\theta) = \sum_{n} \sum_{t} \sum_{k} \theta_k f_k(Y_t^n, Y_{t-1}^n, \boldsymbol{X}_t^n) - \sum_{n} \log Z(\boldsymbol{X}^n)$$

Training LCRF

Maximum (conditional) log-likelihood

$$\max_{\theta} \mathcal{L}(\theta) = \max_{\theta} \sum_{n=1}^{n} \log P(\boldsymbol{y}^n | \boldsymbol{x}^n, \theta)$$

Substituting LCRF conditional formulation

$$\mathcal{L}(\theta) = \sum_{n} \sum_{t} \sum_{k} \theta_k f_k(Y_t^n, Y_{t-1}^n, \boldsymbol{X}_t^n) - \sum_{n} \log Z(\boldsymbol{X}^n) - \sum_{k} \frac{\theta_k^2}{2\sigma^2}$$

Penalized with a regularization term, e.g. based on $\|\theta\|^2$

Optimizing the Likelihood

- Typically $\mathcal{L}(\theta)$ cannot be maximized in closed form
- Use partial derivatives

$$\frac{\partial \mathcal{L}(\theta)}{\partial \theta_k} = \sum_{n,t} f_k(Y_t^n, Y_{t-1}^n, \boldsymbol{X}_t^n) - \sum_{n,\ t} \sum_{y,y'} f_k(y, y', \boldsymbol{X}_t^n) P(y, y' | \boldsymbol{X}^n) - \frac{\theta_k}{\sigma^2}$$

- First term is $\mathbb{E}[f_k]$ under the empirical distribution (i.e. with y, y' clamped)
- Second term is the $\mathbb{E}[f_k]$ under model distribution
- When gradient is zero these are equal (apart for regularization)

Stochastic Gradient Descent

In practice we can learn the θ parameters by SGD (or variants)

$$\theta^m = \theta^{m-1} - \nu_m \nabla \mathcal{L}_n(\theta^{m-1})$$

where

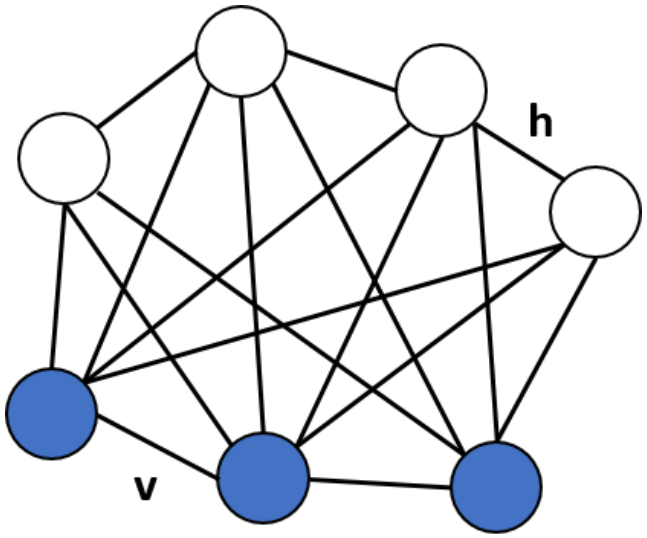
$$\nabla \mathcal{L}_{nk}(\theta) = \sum_t f_k(Y_t^n, Y_{t-1}^n, \boldsymbol{X}_t^n) - \sum_t \sum_{y,y'} f_k(y, y', \boldsymbol{X}_t^n) P(y, y' | \boldsymbol{X}^n) - \frac{\theta_k}{N\sigma^2}$$

and $P(y, y' | \boldsymbol{X}^n)$ is estimated by sum-product inference

Approximated Inference - Boltzmann Machines

Boltzmann Machines

An example of specialized case of Markov Random Field



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

Usual 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.

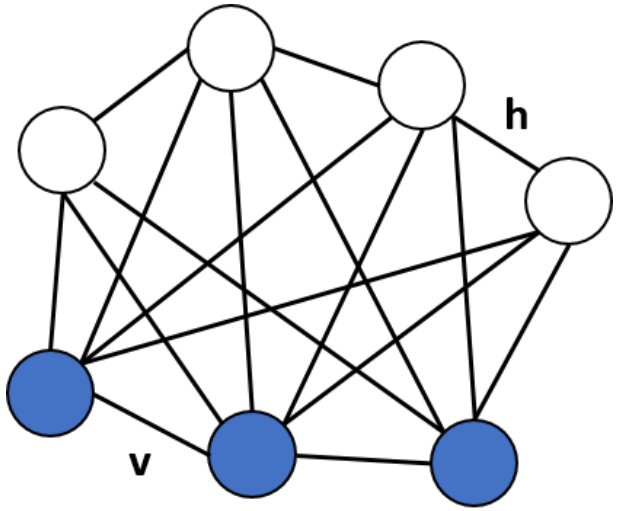
Model parameters $\theta = \{\boldsymbol{M}, \boldsymbol{b}\}$ encode the interactions between the variables

Boltzmann machines are also 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 $P(\mathbf{s})$ over binary neurons



- At each time t, a neuron can emit an output $s_j \in \{0,1\}$ 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 defining output probability as a function of **local potential** $x_j^{(t)}$

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

Sigmoidal Neurons and the Boltzmann-Gibbs Distribution

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

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

A chosen neuron fires with spiking probability

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

Formulation highlights Markovian dynamics

Undirected connectivity and the lack of self loops ensures existence of an equilibrium Boltzmann-Gibbs distribution

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

where

- $P(\boldsymbol{s})$ is the marginal probability of neurons having output $\boldsymbol{s}$
- $E(\boldsymbol{s})$ 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

- Let us simplify notation and absorb bias $\boldsymbol{b}$ into weight matrix $\boldsymbol{M}$
- Use probabilistic learning techniques to fit the parameters, i.e. maximizing the log-likelihood

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

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

- Since in $\boldsymbol{s} = [\boldsymbol{v}\boldsymbol{h}]$ we also have the hidden variables $\boldsymbol{h}$ we will have to introduce them in the likelihood as usual (marginalization)

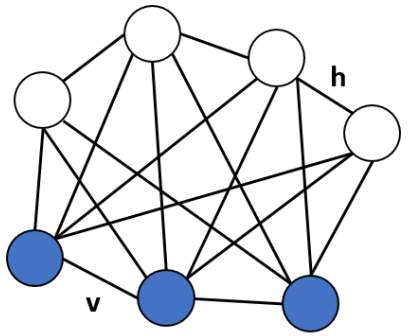
Learning with Hidden Variables

- Learning proceeds as usual by log-likelihood maximization via gradient ascent
- Looking at the gradient for a single pattern $\boldsymbol{s} = [\boldsymbol{vh}]$

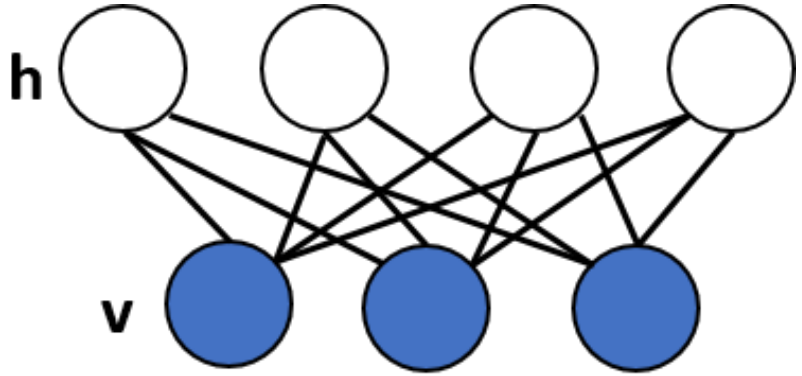
$$\frac{\partial P(\boldsymbol{v}|\boldsymbol{M})}{\partial M_{ij}} = \underbrace{\sum_{\boldsymbol{h}} s_i s_j P(\boldsymbol{h}|\boldsymbol{v})}_{\text{Clamped expectations}} - \underbrace{\sum_{\boldsymbol{s}} s_i s_j P(\boldsymbol{s})}_{\text{Free (model) expectations}} = \langle s_i s_j \rangle_c - \langle s_i s_j \rangle$$

They become intractable due to the partition function Z so we will need to approximate them. How?

By sampling, of course! Ideally, we would use Gibbs sampling but unless we impose some restrictions in the ancestors' structure, it will be too slow!



Restricted Boltzmann Machines (RBM)



A special Boltzmann machine

- Bipartite graph
- Connections only between hidden and visible units

- Learning (inference) becomes tractable due to graph bipartition which factorizes distributions and makes Gibbs sampling highly parallel
- Hidden units are conditionally independent given visible units, and viceversa

$$P(h_j|\boldsymbol{v}) = \sigma\left(\sum_i M_{ij} v_i + c_j\right) \quad \text{and} \quad P(v_i|\boldsymbol{h}) = \sigma\left(\sum_j M_{ij} h_j + b_i\right)$$

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

Sampling for Data Expectation

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

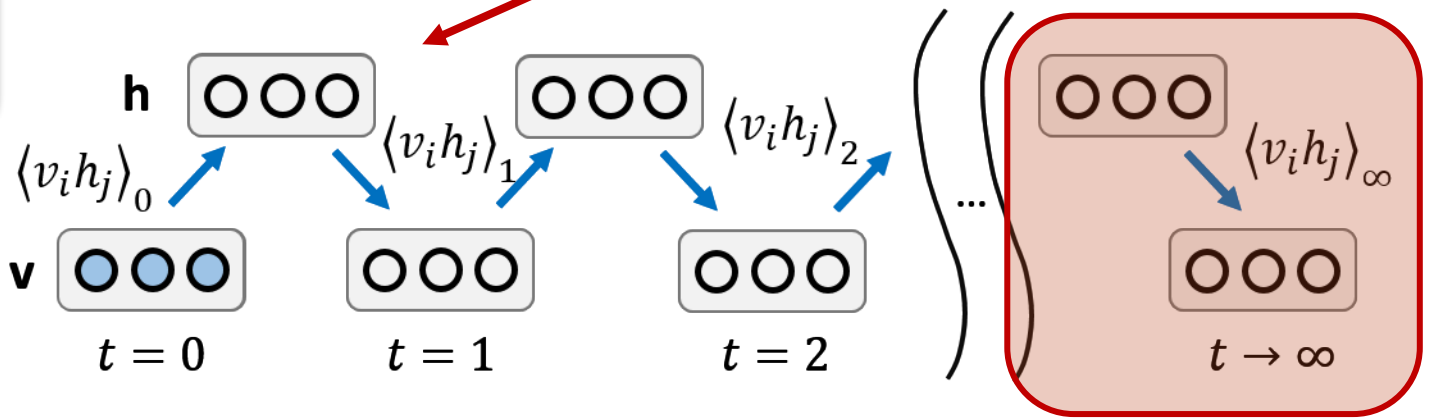
Sampling for Model Expectation

- Random sample $\boldsymbol{v}$
- Let network reach equilibrium (∞)
- Sample $\mathrm{v_i h_j}$ at equilibrium for all pairs of connected units
- Repeat many times to get a good sample

Gibbs-Sampling RBM

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

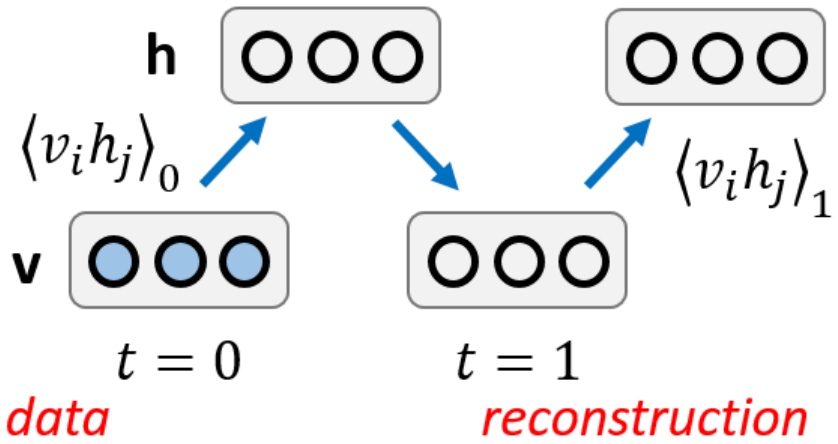
$$\frac{\partial \mathcal{L}}{\partial M_{ij}} = \underbrace{\langle v_i h_j \rangle_c}_{data} - \underbrace{\langle v_i h_j \rangle}_{model} = \underbrace{\langle v_i h_j \rangle_0}_{data} - \underbrace{\langle v_i h_j \rangle_\infty}_{model}$$



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

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

$$\frac{\partial \mathcal{L}}{\partial M_{ij}} = \underbrace{\langle v_i h_j \rangle_0}_{data} - \underbrace{\langle v_i h_j \rangle_1}_{reconstruction}$$

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!

And it works well enough

RBM for Digit Recognition

Learning good features for reconstructing images of handwritten digits (MNIST)

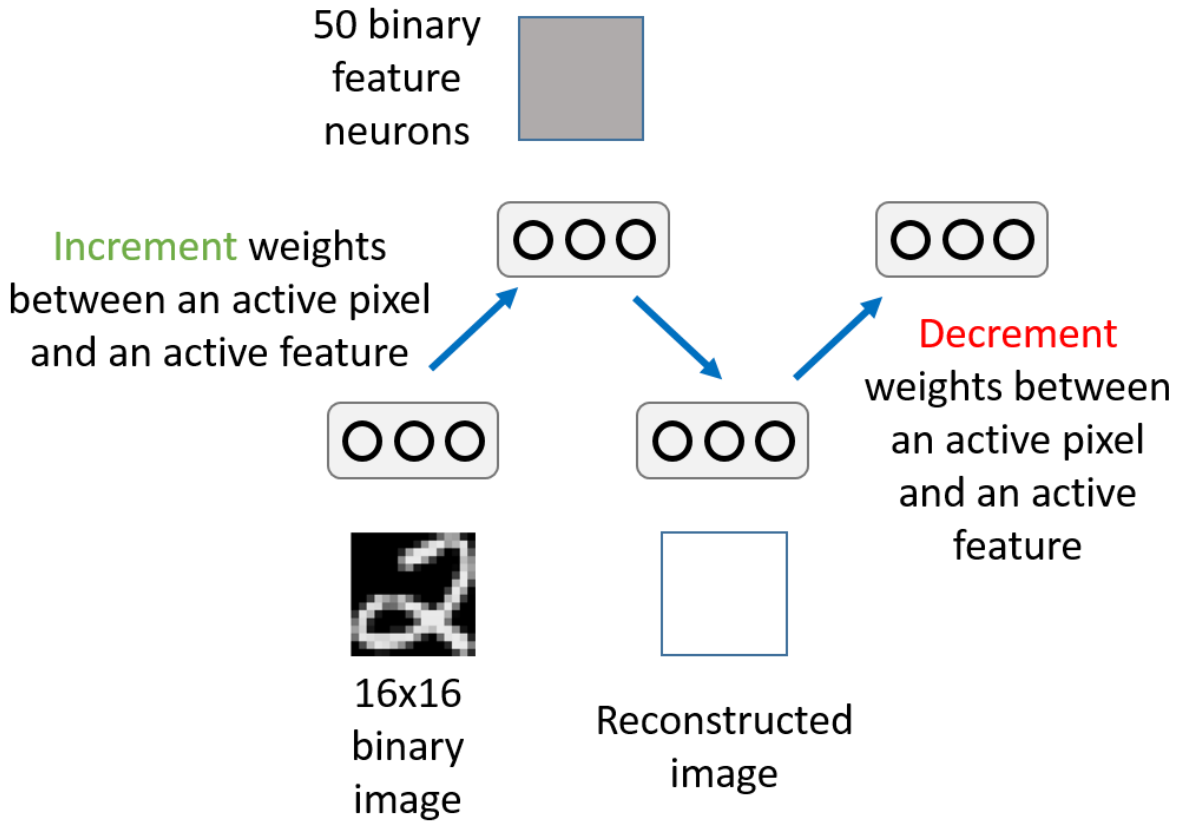
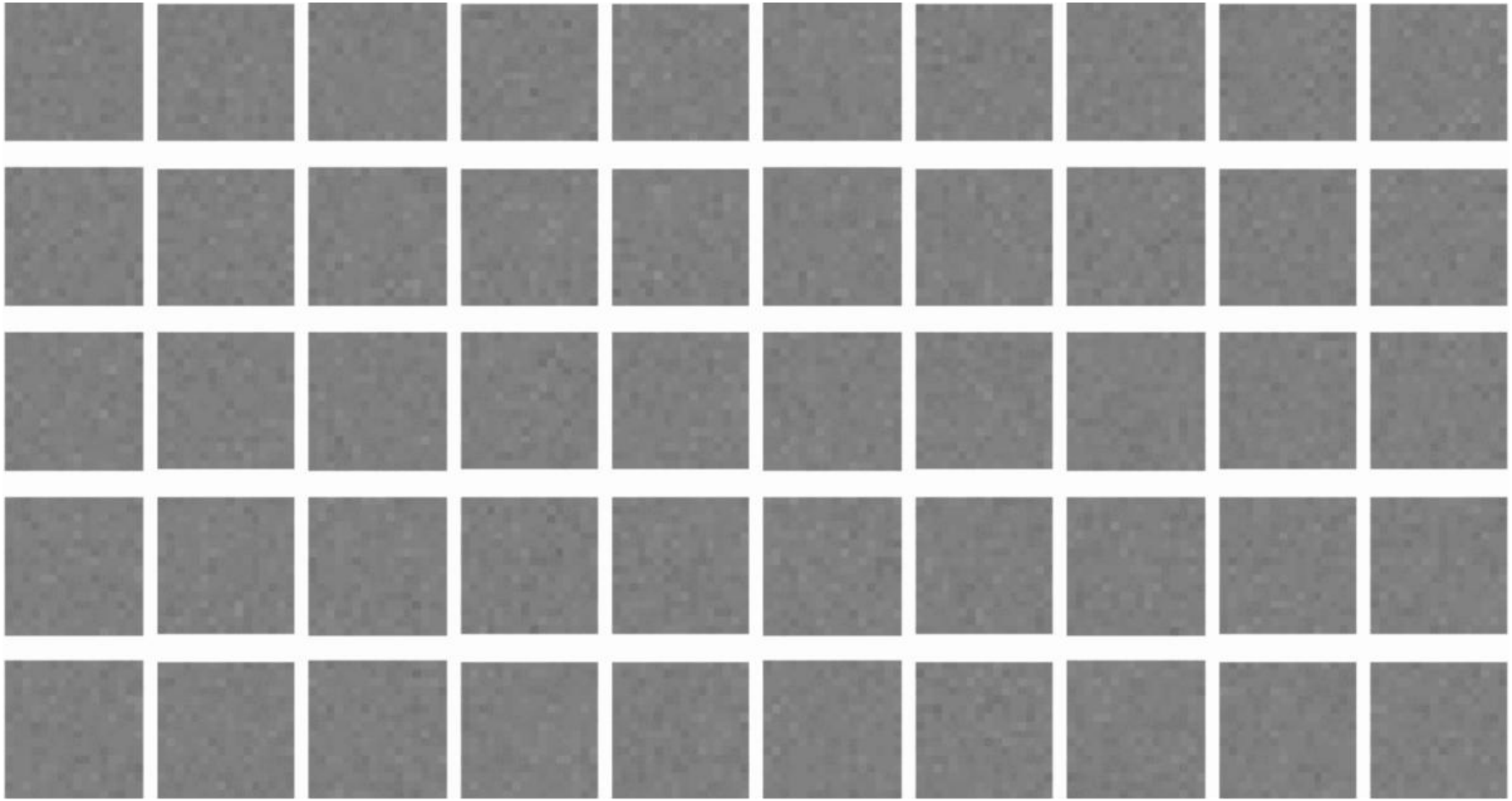
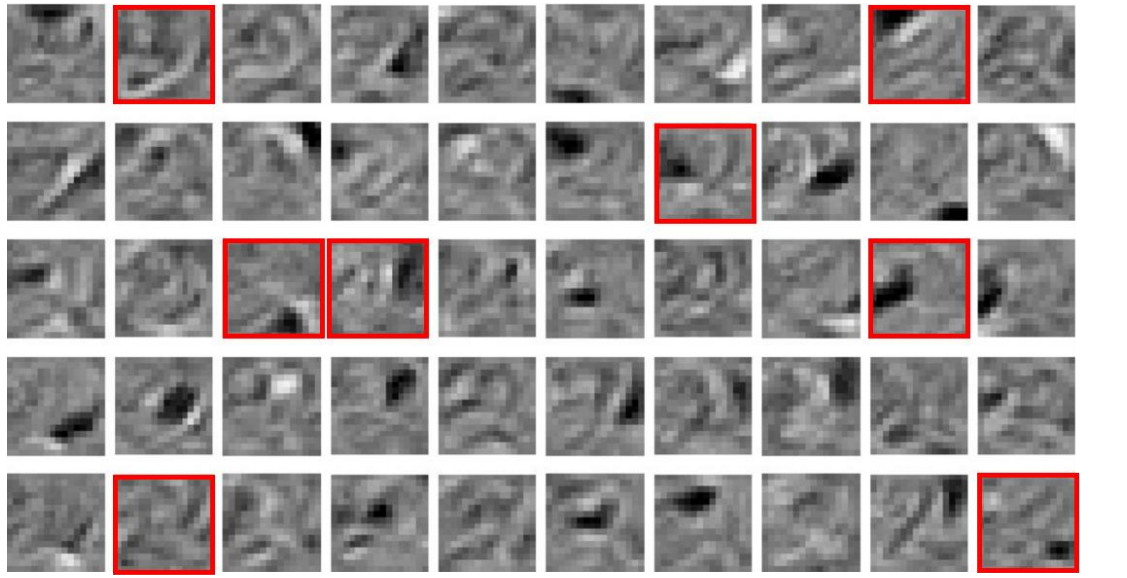


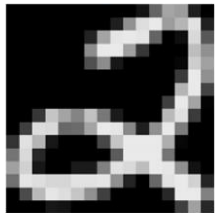
Figure credit: G. Hinton

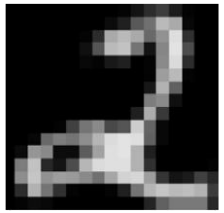
RBM weight matrix evolution during learning



Digit Reconstruction







hidden features

data

reconstruction

Wrap-up

MRF Software

- CRFsuite - Fast implementation of linear/chain CRFs for NLP applications (native C++; Scikit-like package python-crfsuite)
- PyStruct - Python CRF package including 2D lattices, graph structures and several inference algorithms
- pgmpy - Python library for graphical models (includes CRF, MRF and more)
- Pyro - Ubers' own PyTorch provide an implementation of Deep CRF
- UGM - Matlab library for Markov Random Fields
- CRF implementations (in particular linear) are present in major DL libraries (e.g. Tensorflow, PyTorch)

MRF - A Python Example

```
from pgmpy . models import MarkovModel
from pgmpy . factors . discrete import DiscreteFactor
import numpy as np
from pgmpy . inference import BeliefPropagation
. . .
MM=MarkovModel ( ) ;
# Add edges ( and nodes if not existent )
MM. add_edges_from ( [ ( 'f1', 'f2' ), ( 'f2', 'f3' ), ( 'o1', 'f1' ), ( 'o2', 'f2' ), ( 'o3', 'f3' ) ] )

#Generate transition feature
transition =np . array ( [ 10, 90, 90, 10 ] ) ;
#Generate corresponding factor
factorH1 = DiscreteFactor ( [ 'f1', 'f2' ], cardinality = [ 2, 2 ], values = transition )
#Add it to the model
MM. add _factors ( factorH1 )

#Solve smoothing by belief propagation ( i.e. estimate hidden RV)
belief_propagation = BeliefPropagation (MM)
ymax = belief_propagation . map_query ( variables = [ 'f1', 'f2', 'f3' ] , \
evidence = { 'o1' : toVal ( 'class1' ), 'o2' : toVal ( 'class1' ), 'o3' : toVal ( 'class2' ) } )
...
```

RBM-CD in (Matlab) Code

```
for epoch = 1: maxepoch
    %--- Compute data  part (wake)
    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 model  part (dream)
    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))) ;
    %---CD_1 Update
    deltaW = (wake - dream ) / numcases ;
...
end
```

Take Home Messages

Markov Random Fields - Undirected graphical models

- Allow to express constraints between RV without needing to use probabilities
  - Feature functions are often simple hand-coded feature detectors
  - Parameters linearly combine features
- Conditional Random Fields constrain MRF learning discriminative posteriors
- CRF/MRF are often used as final refinement and output layer in NNs

Boltzmann Machines

- A first bridge between (undirected) generative models and (recurrent) neural networks
- Neural activity regulated by stochastic behavior

Restricted Boltzmann Machines

- Tractable model thanks to bipartite connectivity
- Trained by a very short Gibbs sampling (contrastive divergence)

A network of **stacked RBMs** trained layer-wise by **Contrastive Divergence** has initiated deep learning

Final considerations on probabilistic models

- Consider using probabilistic models when
  - Need interpretability
  - Need to incorporate prior knowledge
  - Unsupervised learning or learning with partially observable supervision
  - Need reusable/portable learned knowledge
- Consider avoiding probabilistic 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 (2) Lectures

- 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
- Solving tasks with CNNs