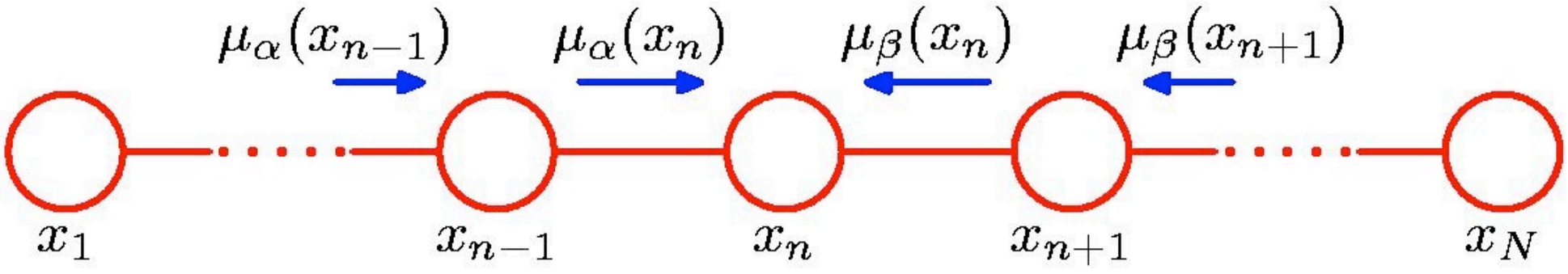
4a. Inference in Graphical Models

Inference on a Chain (Rep.)



- The first values of μ_α and μ_β are:

$$\mu_\alpha(x_2) = \sum_{x_1} \psi_{1,2}(x_1, x_2) \qquad \mu_\beta(x_{N-1}) = \sum_{x_N} \psi_{N-1,N}(x_{N-1}, x_N)$$

- The partition function can be computed at any node:

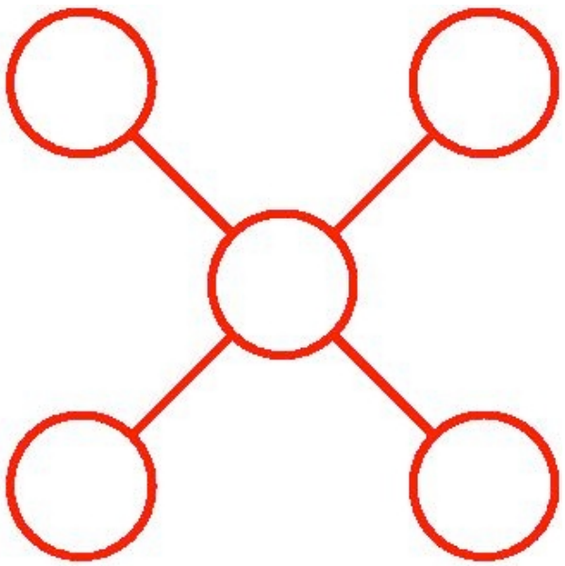
$$Z = \sum_{x_n} \mu_\alpha(x_n)\mu_\beta(x_n)$$

- Overall, we have $O(NK^2)$ operations to compute the marginal $p(x_n)$

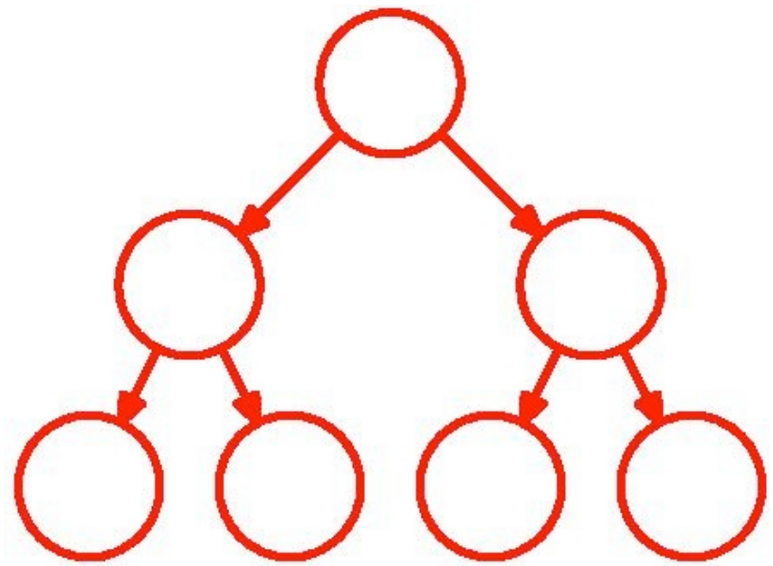
More General Graphs

The message-passing algorithm can be extended to more general graphs:

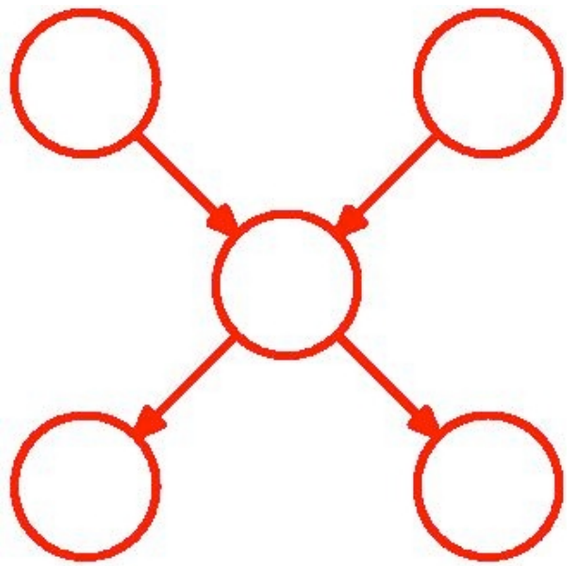
Undirected Tree



Directed Tree



Polytree



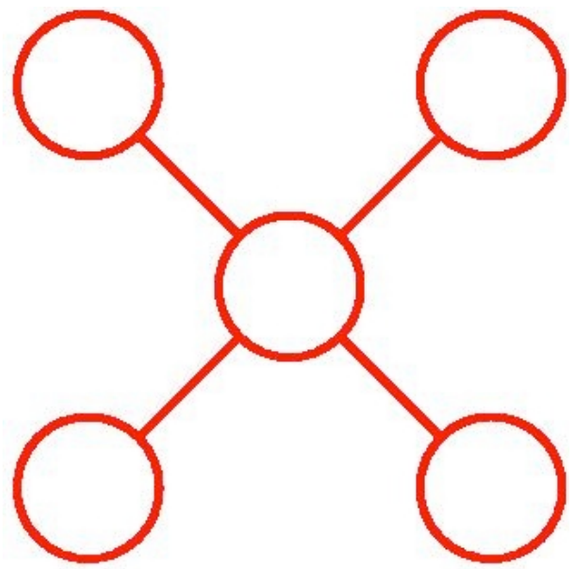
It is then known as the **sum-product algorithm.**

A special case of this is **belief propagation**.

More General Graphs

The message-passing algorithm can be extended to more general graphs:

Undirected Tree



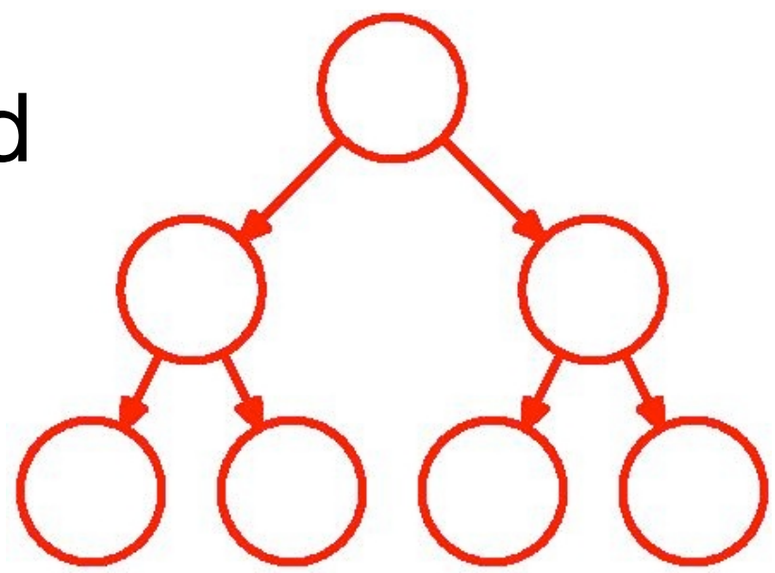
An **undirected tree** is defined as a graph that has exactly one path between any two nodes

More General Graphs

The message-passing algorithm can be extended to more general graphs:

A **directed tree** has only one node without parents and all other nodes have exactly one parent

Directed Tree



Conversion from a directed to an undirected tree is no problem, because no links are inserted

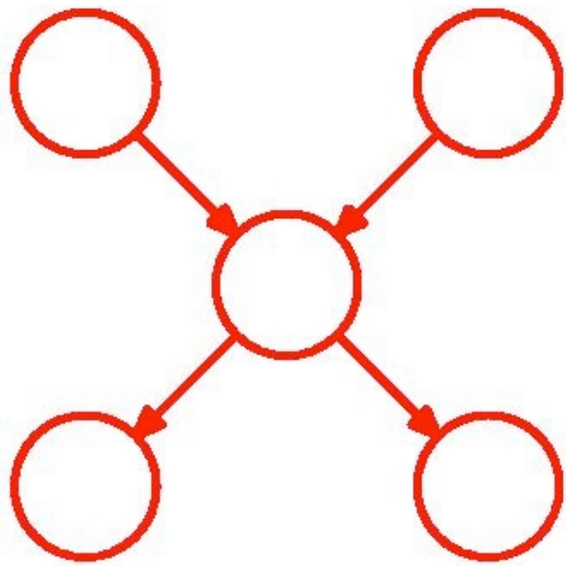
The same is true for the conversion back to a directed tree

More General Graphs

The message-passing algorithm can be extended to more general graphs:

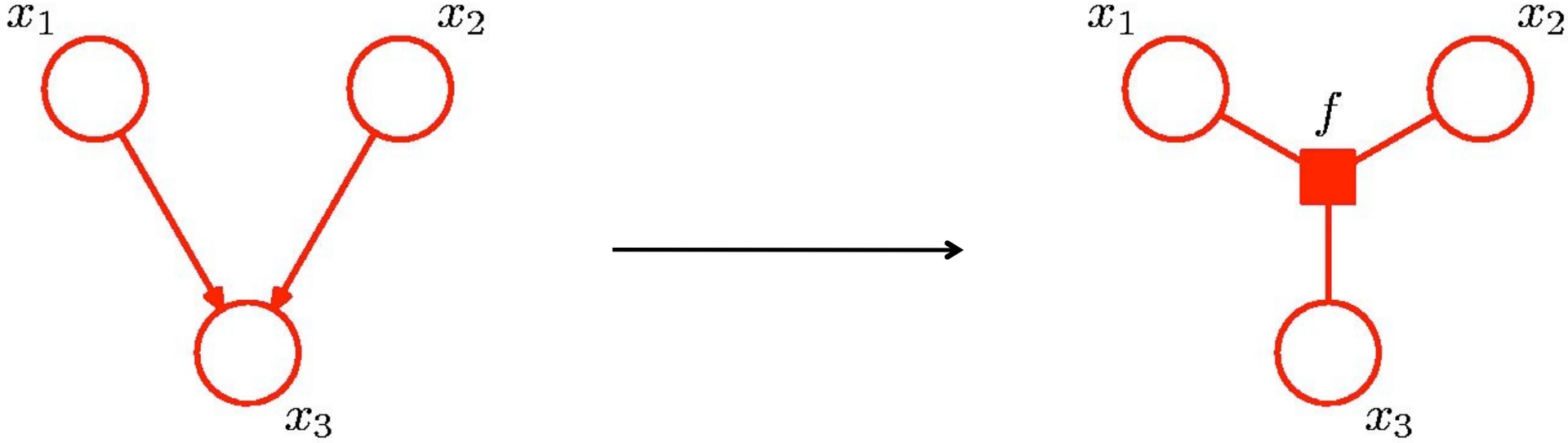
Polytrees can contain nodes with several parents, therefore moralization can remove independence relations

Polytree



Factor Graphs

- The Sum-product algorithm can be used to do inference on undirected and directed graphs.
- A representation that generalizes directed and undirected models is the **factor graph**.



$$p(\mathbf{x}) = p(x_1)p(x_2)p(x_3|x_1, x_2)$$

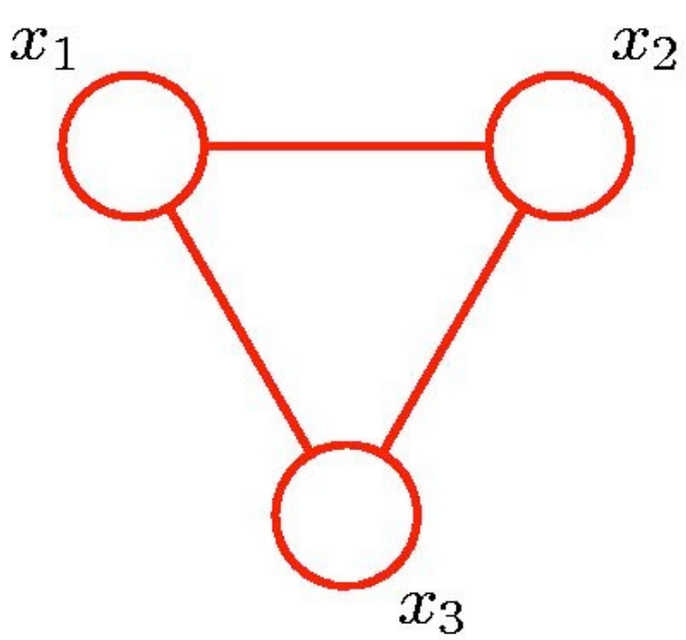
Directed graph

$$f(x_1, x_2, x_3) = p(x_1)p(x_2)p(x_3 \mid x_1, x_2)$$

Factor graph

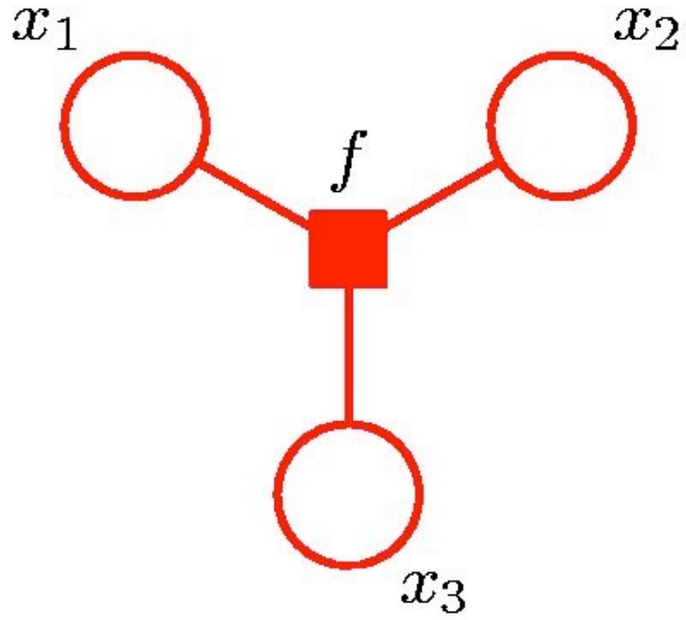
Factor Graphs

- The Sum-product algorithm can be used to do inference on undirected and directed graphs.
- A representation that generalizes directed and undirected models is the **factor graph**.



$$\psi(x_1, x_2, x_3)$$

Undirected graph



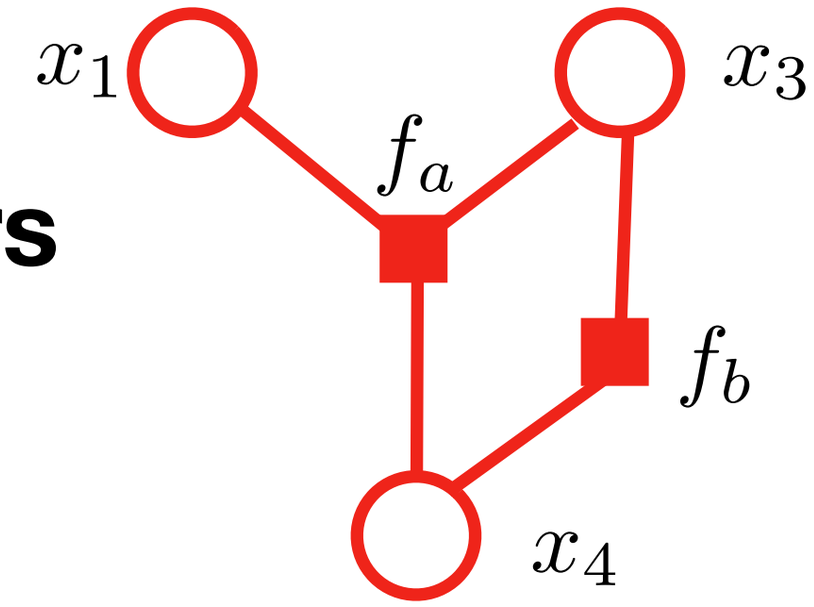
$$f(x_1, x_2, x_3) = \psi(x_1, x_2, x_3)$$

Factor graph

Factor Graphs

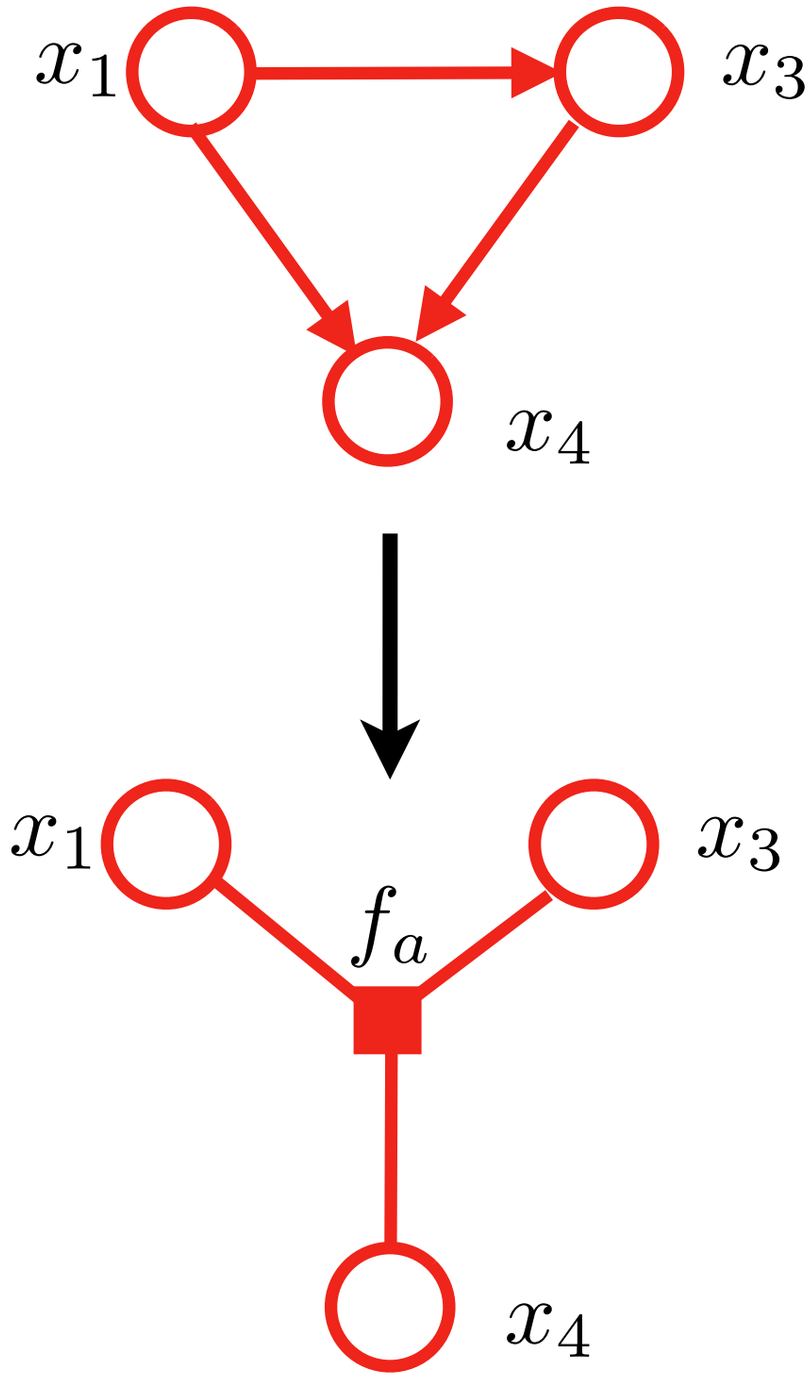
Factor graphs

- can contain **multiple factors** for the same nodes
- are more general than undirected graphs
- are **bipartite**, i.e. they consist of two kinds of nodes and all edges connect nodes of different kind



Factor Graphs

- Directed trees convert to tree-structured factor graphs
- The same holds for undirected trees
- Also: directed polytrees convert to tree-structured factor graphs
- And: Local cycles in a directed graph can be removed by converting to a factor graph



The Sum-Product Algorithm

Assumptions:

- all variables are discrete
- the factor graph has a tree structure

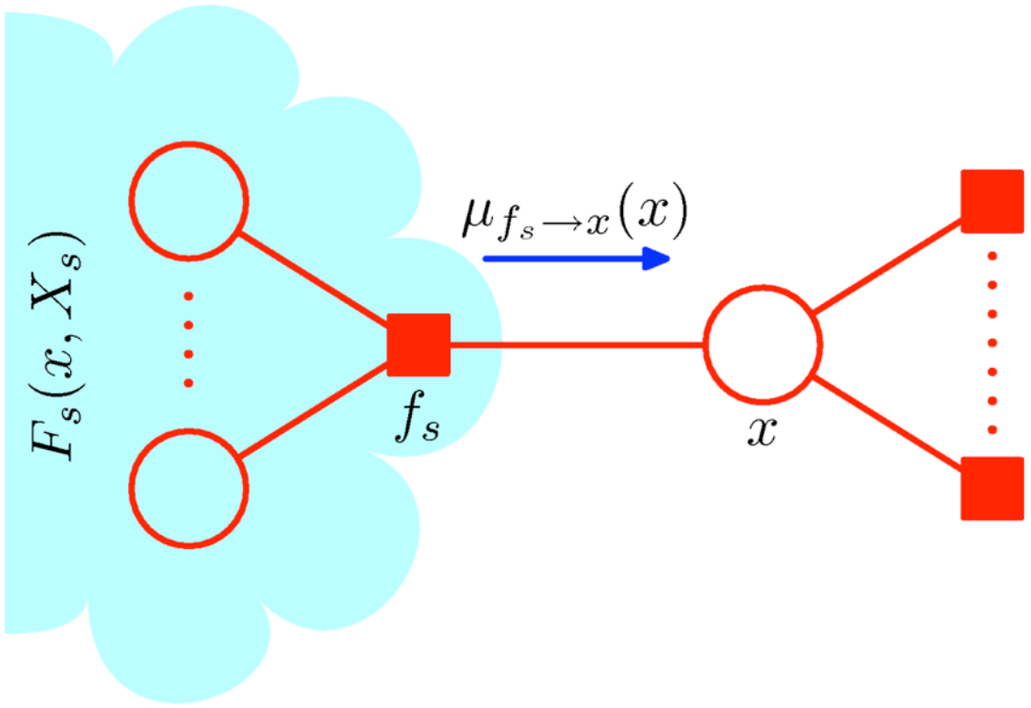
The factor graph represents the joint distribution as a product of factor nodes:

$$p(\mathbf{x}) = \prod_s f_s(\mathbf{x}_s)$$

The marginal distribution at a given node x is

$$p(x) = \sum_{\mathbf{x} \setminus x} p(\mathbf{x})$$

The Sum-Product Algorithm



For a given node x the joint can be written as

$$p(\mathbf{x}) = \prod_{s \in \mathrm{ne}(x)} F_s(x, X_s)$$

Product of all factors associated with f_s

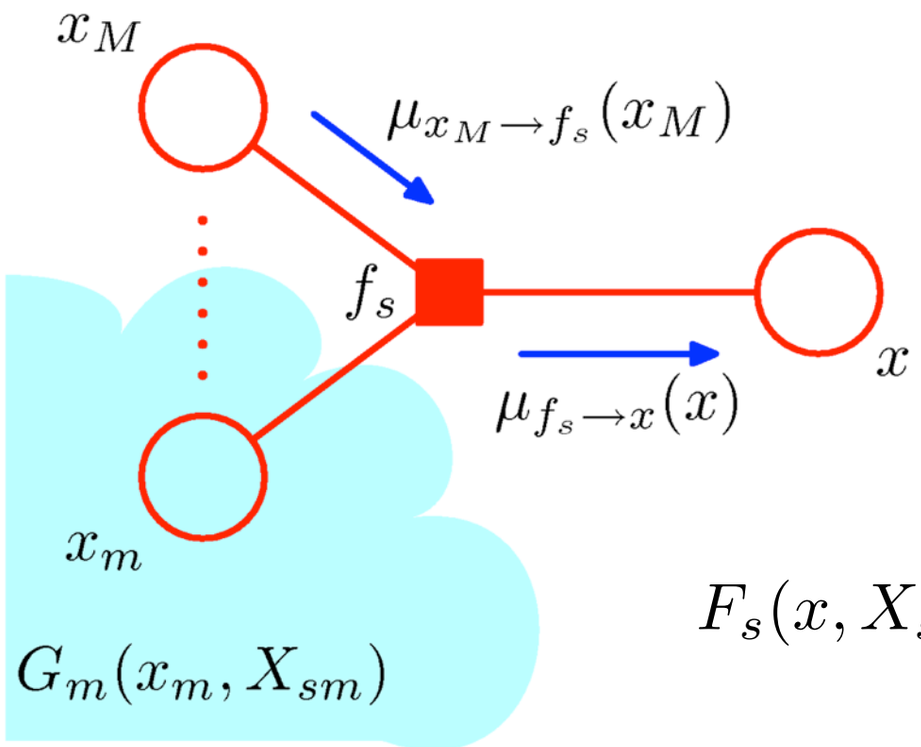
Thus, we have $p(x) = \sum_{\mathbf{x} \setminus x} \prod_{s \in \mathrm{ne}(x)} F_s(x, X_s)$

Key insight: Sum and product can be exchanged!

$$p(x) = \prod_{s \in \mathrm{ne}(x)} \sum_{X_s} F_s(x, X_s) = \prod_{s \in \mathrm{ne}(x)} \mu_{f_s \to x}(x)$$

"Messages from factors to node x"

The Sum-Product Algorithm



The factors in the messages can be factorized further:

$$F_s(x, X_s) = f_s(x, x_1, \ldots, x_M) G_1(x_1, X_{s_1}) \ldots G_M(x_M, X_{s_M})$$

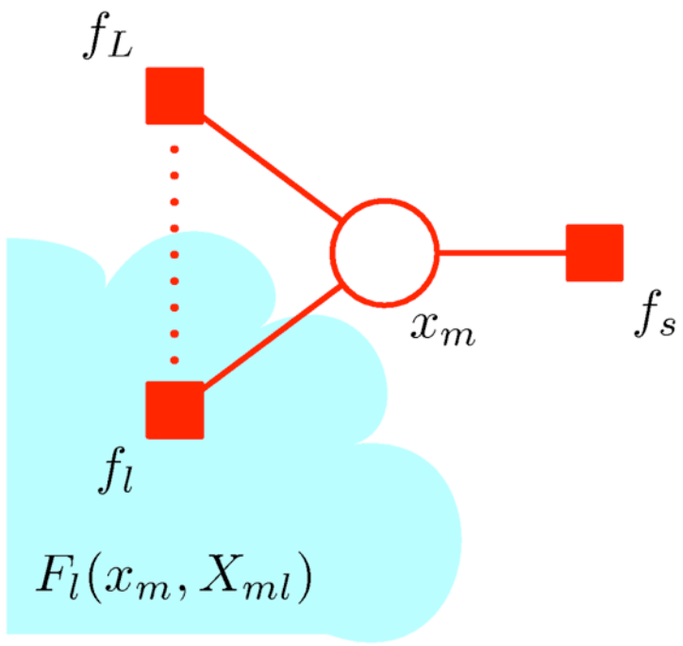
The messages can then be computed as

$$\mu_{f_s \to x}(x) = \sum_{x_1} \cdots \sum_{x_M} f_s(x, x_1, \ldots, x_M) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \sum_{X_{s_m}} G_m(x_m, X_{s_m})$$

$$= \sum_{x_1} \cdots \sum_{x_M} f_s(x, x_1, \ldots, x_M) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \mu_{x_m \to f_s}(x_m)$$

"Messages from nodes to factors"

The Sum-Product Algorithm



The factors G of the neighboring nodes can again be factorized further:

$$G_M(x_m, X_{s_m}) = \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} F_l(x_m, X_{m_l})$$

This results in the exact same situation as above!
We can now recursively apply the derived rules:

$$\mu_{x_m \to f_s}(x_m) = \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} \sum_{X_{m_l}} F_l(x_m, X_{m_l})$$

$$= \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} \mu_{f_l \to x_m}(x_m)$$

The Sum-Product Algorithm

Summary marginalization:

1. Consider the node x as a root note
2. Initialize the recursion at the leaf nodes as:

$$\mu_{f \to x}(x) = 1 \quad \text{(var) or} \quad \mu_{x \to f}(x) = f(x) \quad \text{(fac)}$$

3. Propagate the messages from the leaves to the root x
4. Propagate the messages back from the root to the leaves
5. We can get the marginals at every node in the graph by multiplying all incoming messages

The Max-Sum Algorithm

Sum-product is used to find the marginal distributions at every node, but:

How can we find the setting of all variables that **maximizes** the joint probability? And what is the value of that maximal probability?

Idea: use sum-product to find all marginals and then report the value for each node x that maximizes the marginal $p(x)$

However: this does not give the **overall** maximum of the joint probability

The Max-Sum Algorithm

Observation: the max-operator is distributive, just like the multiplication used in sum-product:

$$\max(ab, ac) = a \max(b, c) \qquad \text{if} \qquad a \geq 0$$

Idea: use max instead of sum as above and exchange it with the product

Chain example:

$$\max_{\mathbf{x}} p(\mathbf{x}) = \frac{1}{Z} \max_{x_1} \ldots \max[\psi_{1,2}(x_1, x_2) \ldots \psi_{N-1,N}(x_{N-1}, x_N)]$$

$$= \frac{1}{Z} \max_{x_1} [\psi_{1,2}(x_1, x_2) [\ldots \max \psi_{N-1,N}(x_{N-1}, x_N)]]$$

Message passing can be used as above!

The Max-Sum Algorithm

To find the maximum value of $p(\mathbf{x})$, we start again at the leaves and propagate to the root.

Two problems:

- no summation, but many multiplications; this leads to numerical instability (very small values)
- when propagating back, multiple configurations of $\mathbf{x}$ can maximize $p(\mathbf{x})$, leading to wrong assignments of the overall maximum

Solution to the first:

Transform everything into log-space and use sums

The Max-Sum Algorithm

Solution to the second problem:

Keep track of the arg max in the forward step, i.e. store at each node which value was responsible for the maximum:

$$\phi(x_n) = \arg\max_{x_{n-1}} [\ln f_{n-1,n}(x_{n-1}, x_n) + \mu_{x_{n-1} \to f_{n-1,n}}(x_n)]$$

Then, in the back-tracking step we can recover the arg max by recursive substitution of:

$$x_{n-1}^{\max} = \phi(x_n^{\max})$$

Other Inference Algorithms

Junction Tree Algorithm:

- Provides exact inference on general graphs.
- Works by turning the initial graph into a **junction tree** and then running a sum-product-like algorithm
- A junction tree is obtained from an undirected model by **triangulation** and mapping cliques to nodes and connections of cliques to edges
- It is the maximal spanning tree of cliques

Problem: Intractable on graphs with large cliques.

Cost grows exponentially with the number of variables in the largest clique (“tree width”).

Other Inference Algorithms

Loopy Belief Propagation:

- Performs Sum-Product on general graphs, particularly when they have loops
- Propagation has to be done several times, until a convergence criterion is met
- No guarantee of convergence and no global optimum
- Messages have to be **scheduled**
- Initially, unit messages passed across all edges
- Approximate, but tractable for large graphs

Conditional Random Fields

- Another kind of undirected graphical model is known as Conditional Random Field (CRF).
- CRFs are used for classification where labels are represented as discrete random variables $\mathbf{y}$ and features as continuous random variables $\mathbf{x}$
- A CRF represents the conditional probability

$$p(\mathbf{y} \mid \mathbf{x}, \mathbf{w}) = \frac{\prod_C \phi_C(\mathbf{x}_C, \mathbf{y}_C; \mathbf{w})}{\sum_{\mathbf{y}'} \prod_C \phi_C(\mathbf{x}_C, \mathbf{y}'_C; \mathbf{w})}$$

  where $\mathbf{w}$ are parameters learned from training data.
- CRFs are **discriminative** and MRFs are **generative**

Conditional Random Fields

Derivation of the formula for CRFs:

$$p(\mathbf{y} \mid \mathbf{x}, \mathbf{w}) = \frac{p(\mathbf{y}, \mathbf{x} \mid \mathbf{w})}{p(\mathbf{x} \mid \mathbf{w})} = \frac{p(\mathbf{y}, \mathbf{x} \mid \mathbf{w})}{\sum_{y'} p(\mathbf{y}', \mathbf{x} \mid \mathbf{w})} = \frac{\prod_C \phi_C(\mathbf{x}_C, \mathbf{y}_C; \mathbf{w}) \quad Z}{Z \quad \sum_{\mathbf{y}'} \prod_C \phi_C(\mathbf{x}_C, \mathbf{y}'_C; \mathbf{w})}$$

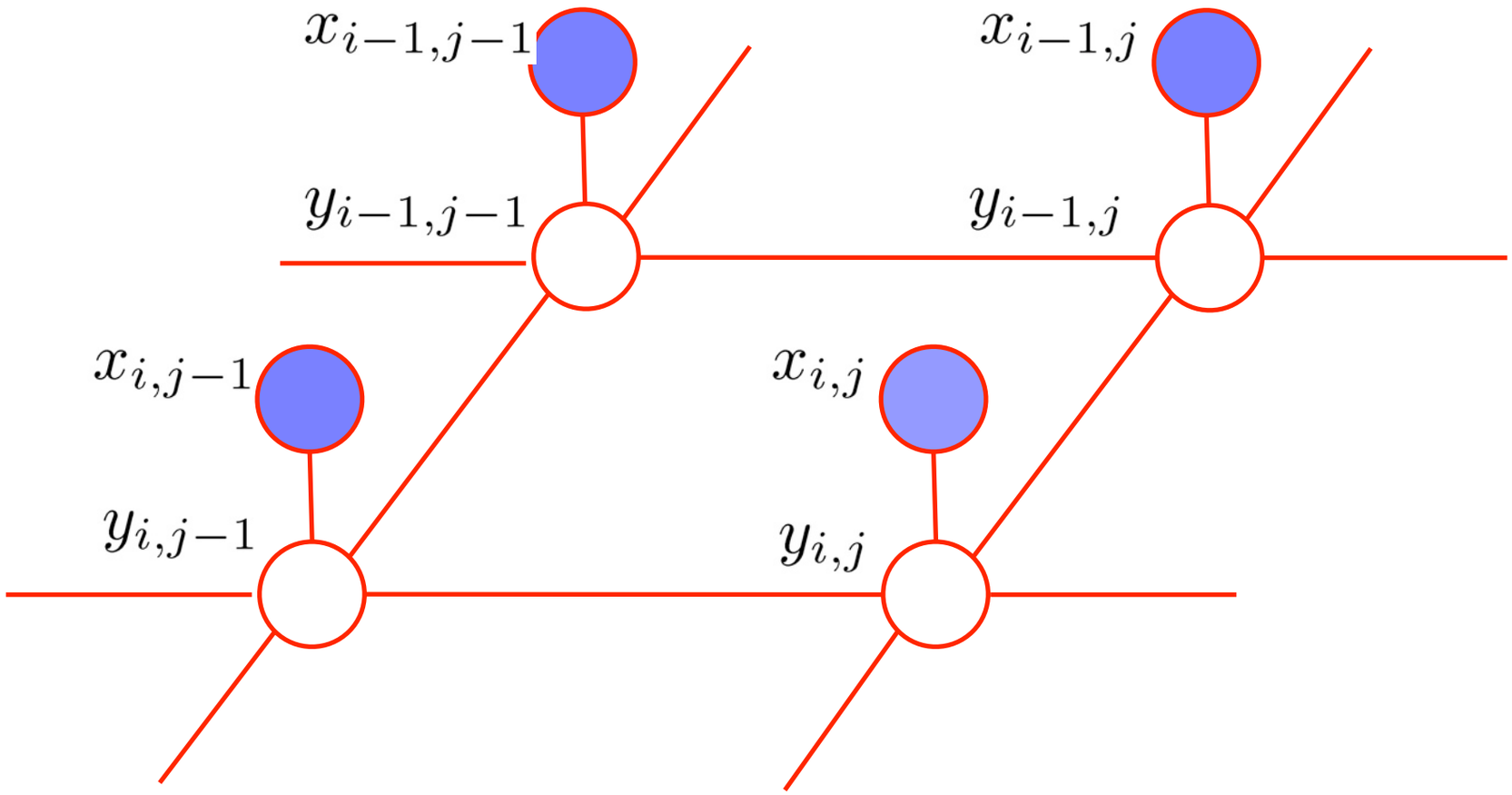
In the training phase, we compute parameters $\mathbf{w}$ that maximize the posterior:

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} p(\mathbf{w} \mid \mathbf{x}^*, \mathbf{y}^*) \propto p(\mathbf{y}^* \mid \mathbf{x}^*, \mathbf{w}) p(\mathbf{w})$$

where (x^*,y^*) is the training data and $p(w)$ is a Gaussian prior. In the inference phase we maximize

$$\arg\max_{\mathbf{y}} p(\mathbf{y} \mid \mathbf{x}, \mathbf{w}^*)$$

Conditional Random Fields



Typical example: **observed** variables $x_{i,j}$ are intensity values of pixels in an image and **hidden** variables $y_{i,j}$ are object labels

Note: the definition of $x_{i,j}$ and $y_{i,j}$ is different from the one in C.M. Bishop (pg.389)!

CRF Training

We minimize the negative log-posterior:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \{-\ln p(\mathbf{w} \mid \mathbf{x}^*, \mathbf{y}^*)\} = \arg\min_{\mathbf{w}} \{-\ln p(\mathbf{y}^* \mid \mathbf{x}^*, \mathbf{w}) - \ln p(\mathbf{w})\}$$

Computing the likelihood is intractable, as we have to compute the partition function for each $\mathbf{w}$. We can approximate the likelihood using **pseudo-likelihood**:

where

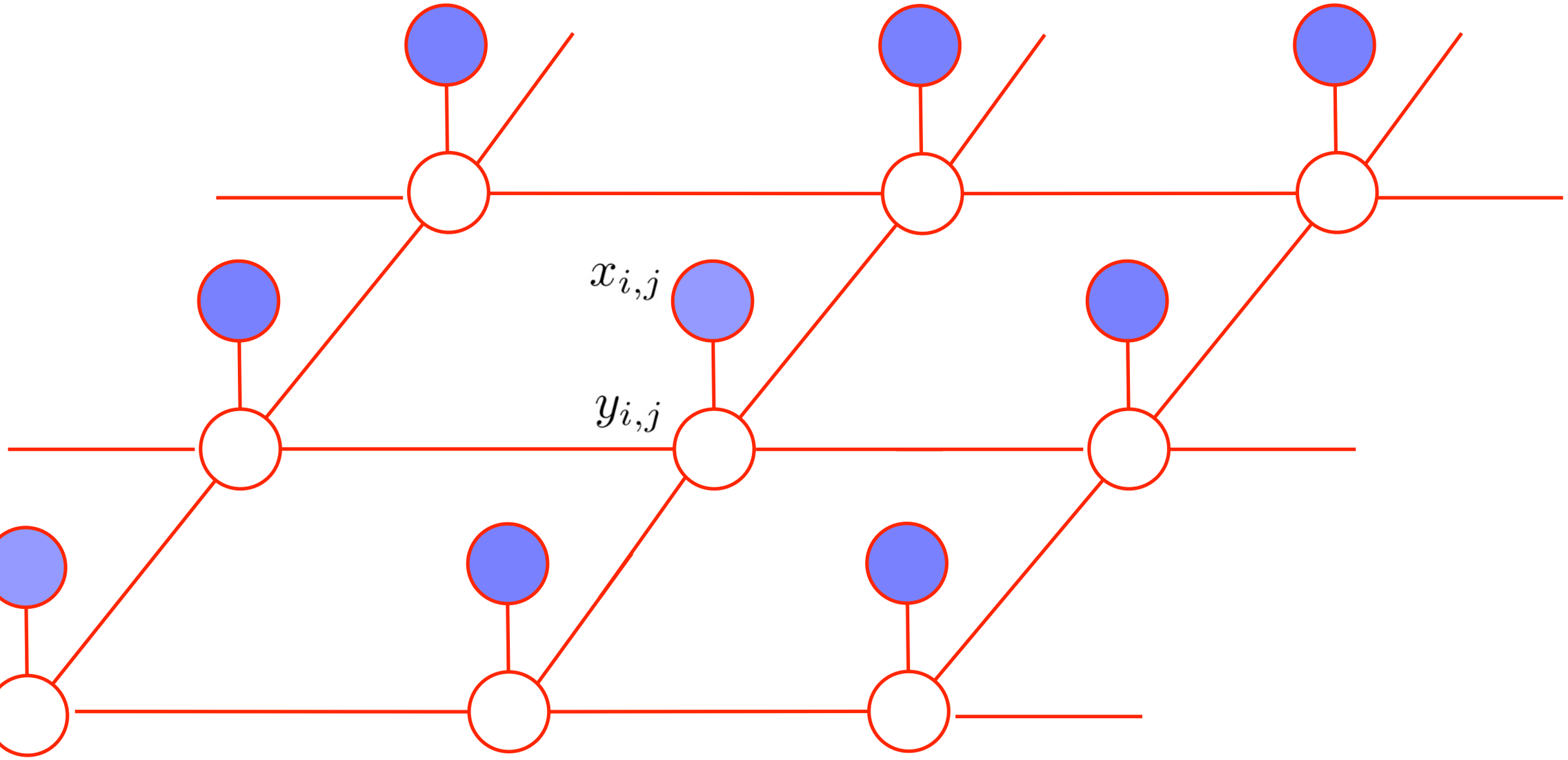
$$p(\mathbf{y}^* \mid \mathbf{x}^*, \mathbf{w}) \approx \prod_i p(y_i^* \mid \mathcal{M}(y_i^*), \mathbf{x}^*, \mathbf{w})$$

Markov blanket

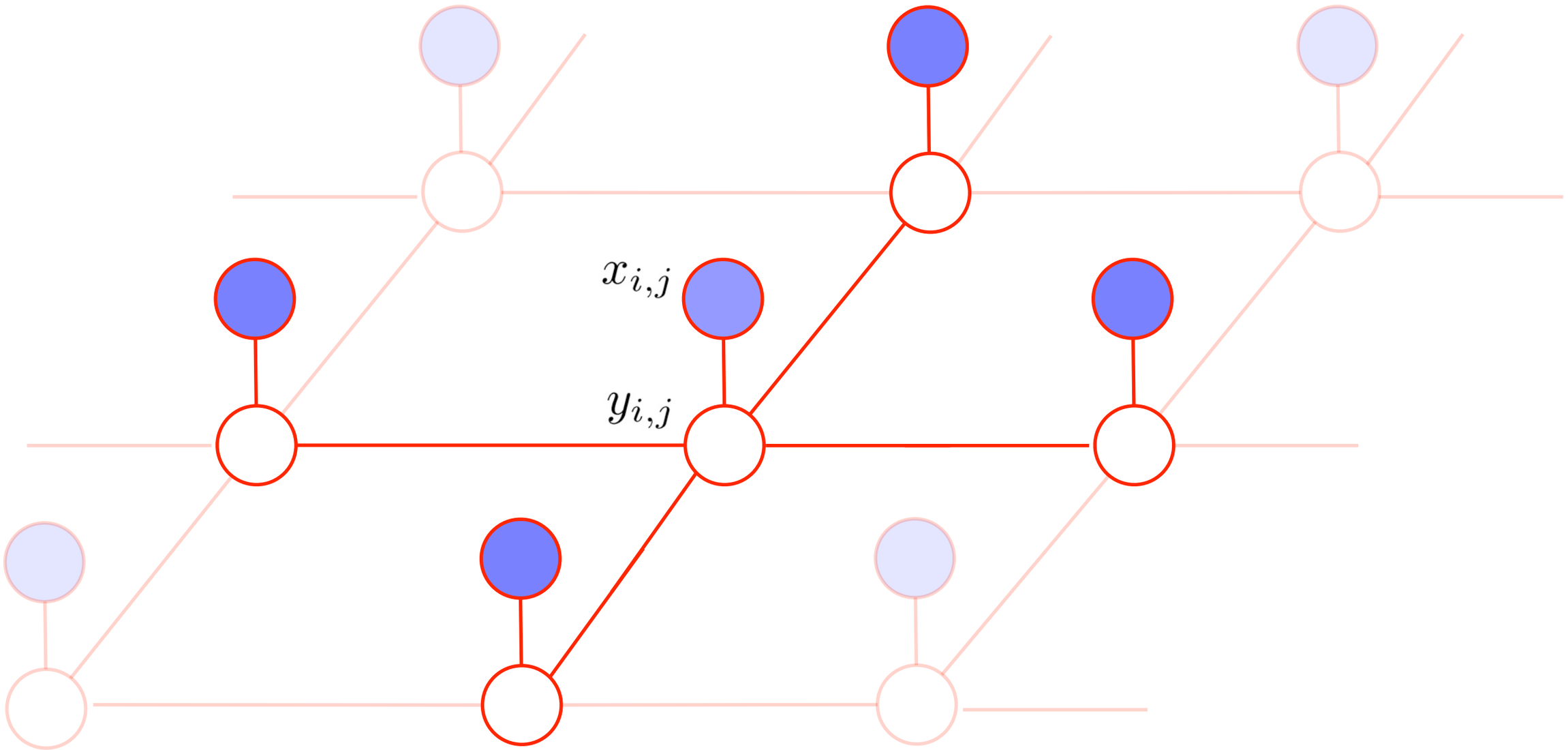
C_i: All cliques containing y_i

$$p(y_i^* \mid \mathcal{M}(y_i^*), \mathbf{x}^*, \mathbf{w}) = \frac{\prod_{C_i} \phi_{C_i}(\mathbf{x}_{C_i}^*, y_i^*, \mathbf{y}_{C_i}^*; \mathbf{w})}{\sum_{y_i'} \prod_{C_i} \phi_C(\mathbf{x}_{C_i}^*, y_i', \mathbf{y}_{C_i}^*; \mathbf{w})}$$

Pseudo Likelihood



Pseudo Likelihood



Pseudo-likelihood is computed only on the Markov blanket of y_i and its corresp. feature nodes.

Potential Functions

- The only requirement for the potential functions is that they are positive. We achieve that with:

$$\phi_C(\mathbf{x}_C, \mathbf{y}_C, \mathbf{w}) := \exp(\mathbf{w}^T f(\mathbf{x}_C, \mathbf{y}_C))$$

where f is a compatibility function that is large if the labels $\mathbf{y}_C$ fit well to the features $\mathbf{x}_C$.

- This is called the **log-linear model.**
- The function f can be, e.g. a local classifier

CRF Training and Inference

Training:

- Using pseudo-likelihood, training is efficient. We have to minimize:

$$L(\mathbf{w}) = -lpl(\mathbf{y}^* \mid \mathbf{x}^*, \mathbf{w}) + \frac{1}{2\sigma^2}\mathbf{w}^T\mathbf{w}$$

Log-pseudo-likelihood

Gaussian prior

- This is a convex function that can be minimized using gradient descent

Inference:

- Only approximatively, e.g. using loopy belief propagation

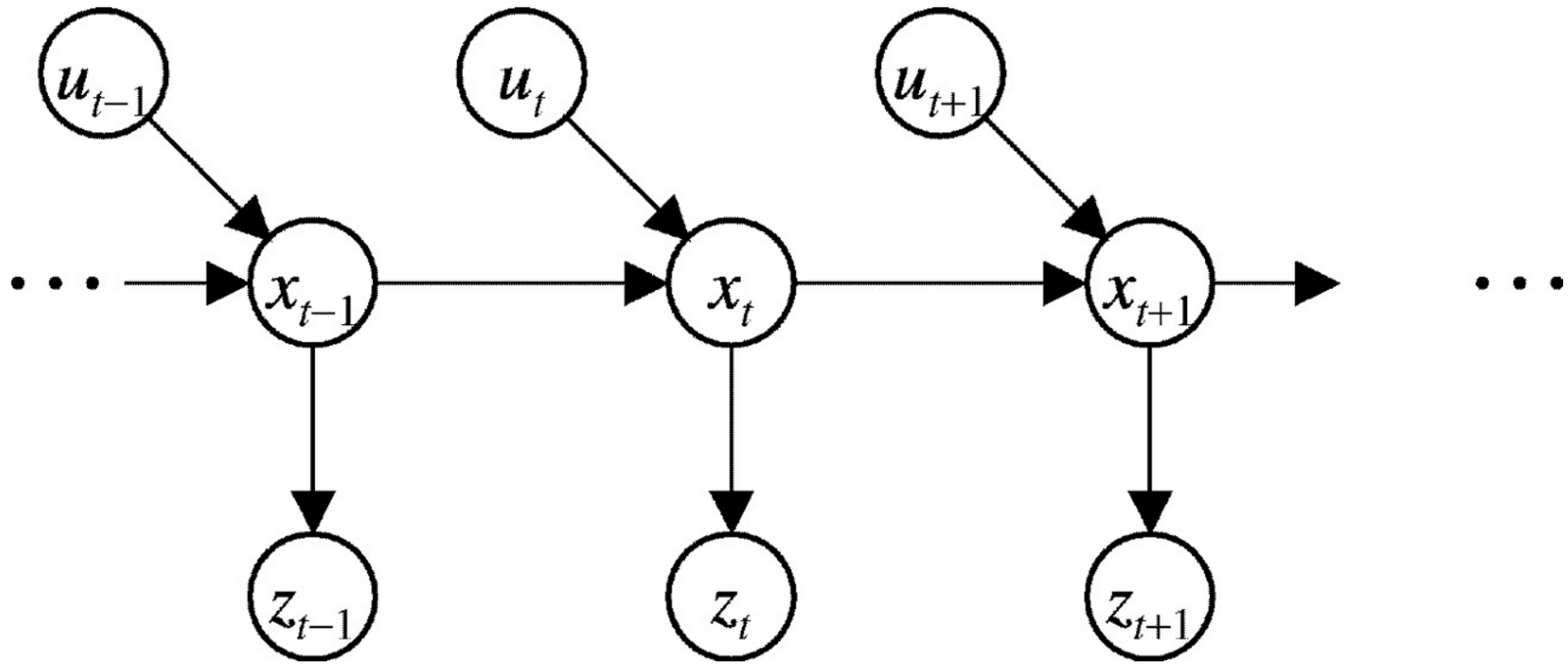
Summary

- Undirected Graphical Models represent conditional independence more intuitively using graph separation
- Their factorization is done based on potential functions The normalizer is called the partition function, which in general is intractable to compute
- Inference in graphical models can be done efficiently using the sum-product algorithm (message passing).
- Another inference algorithm is loopy belief propagation, which is approximate, but tractable
- Conditional Random Fields are a special kind of MRFs and can be used for classification

5. Hidden Markov Models

Graphical Representation (Rep.)

We can describe the overall process using a ***Dynamic Bayes Network***:



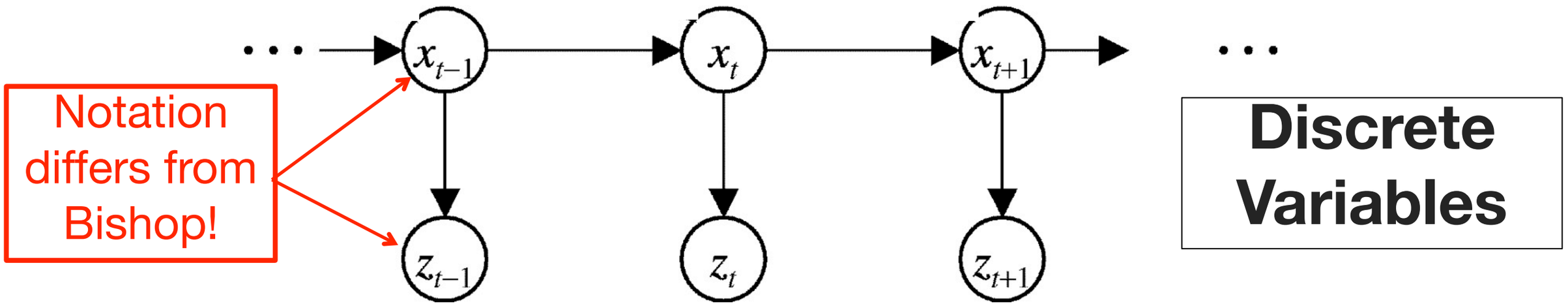
- This incorporates the following Markov assumptions:

$$p(z_t \mid x_{0:t}, u_{1:t}, z_{1:t}) = p(z_t \mid x_t) \quad \text{(measurement)}$$

$$p(x_t \mid x_{0:t-1}, u_{1:t}, z_{1:t}) = p(x_t \mid x_{t-1}, u_t) \quad \text{(state)}$$

Graphical Representation

We can describe the overall process using a ***Markov chain of latent variables***:



- This incorporates the following Markov assumptions:

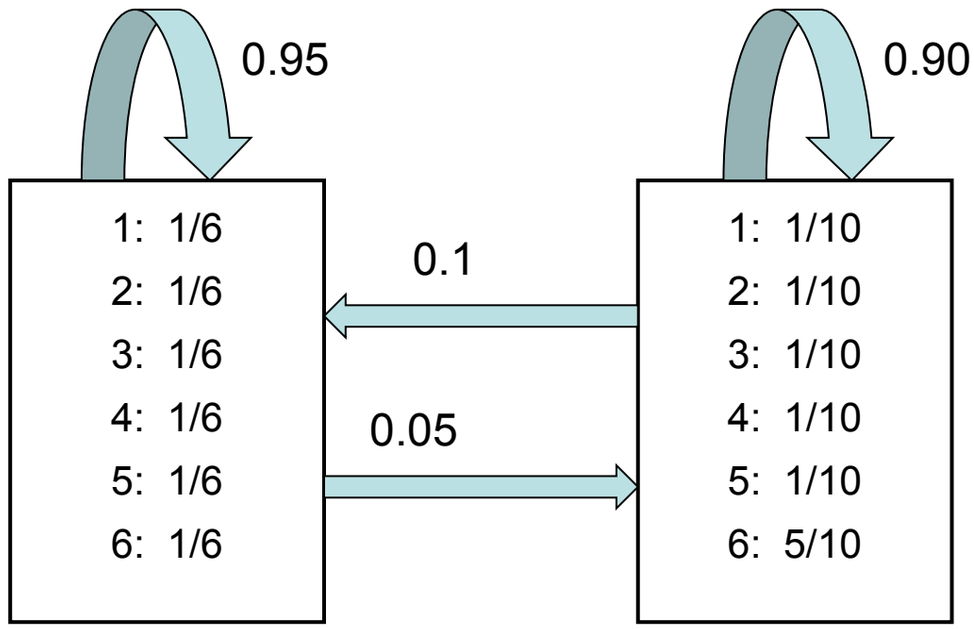
$$p(z_t \mid x_{0:t}, z_{1:t}) = p(z_t \mid x_t) \quad \text{(measurement)}$$

$$p(x_t \mid x_{0:t-1}, z_{1:t}) = p(x_t \mid x_{t-1}) \quad \text{(state)}$$

Example

"Occasionally dishonest casino":

- observations: faces of a die $z_t \in \{1, 2, \ldots, 6\}$
- hidden states: two different dice, one fair, one loaded

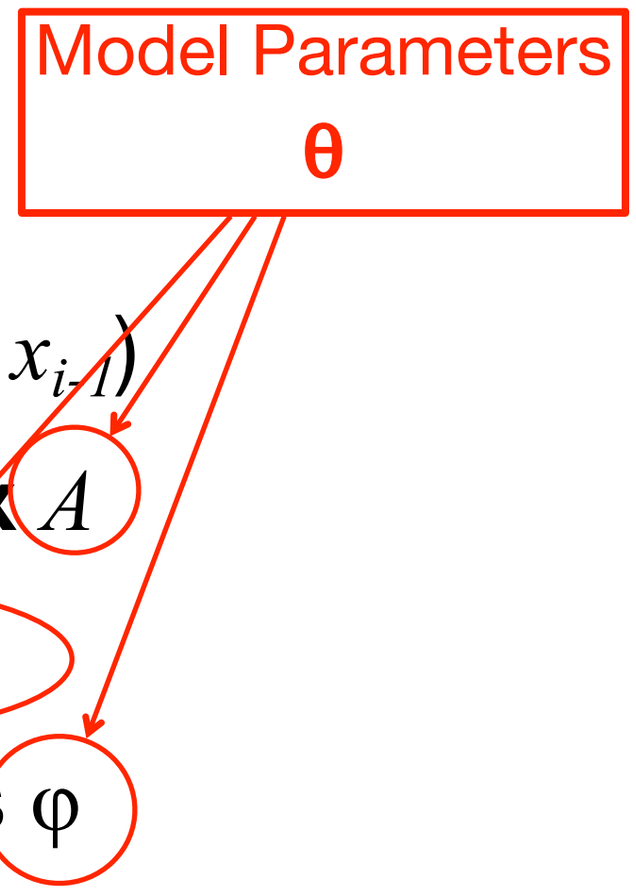


Rolls: 6641532161621152346532143566342616552342323151424641566632 46

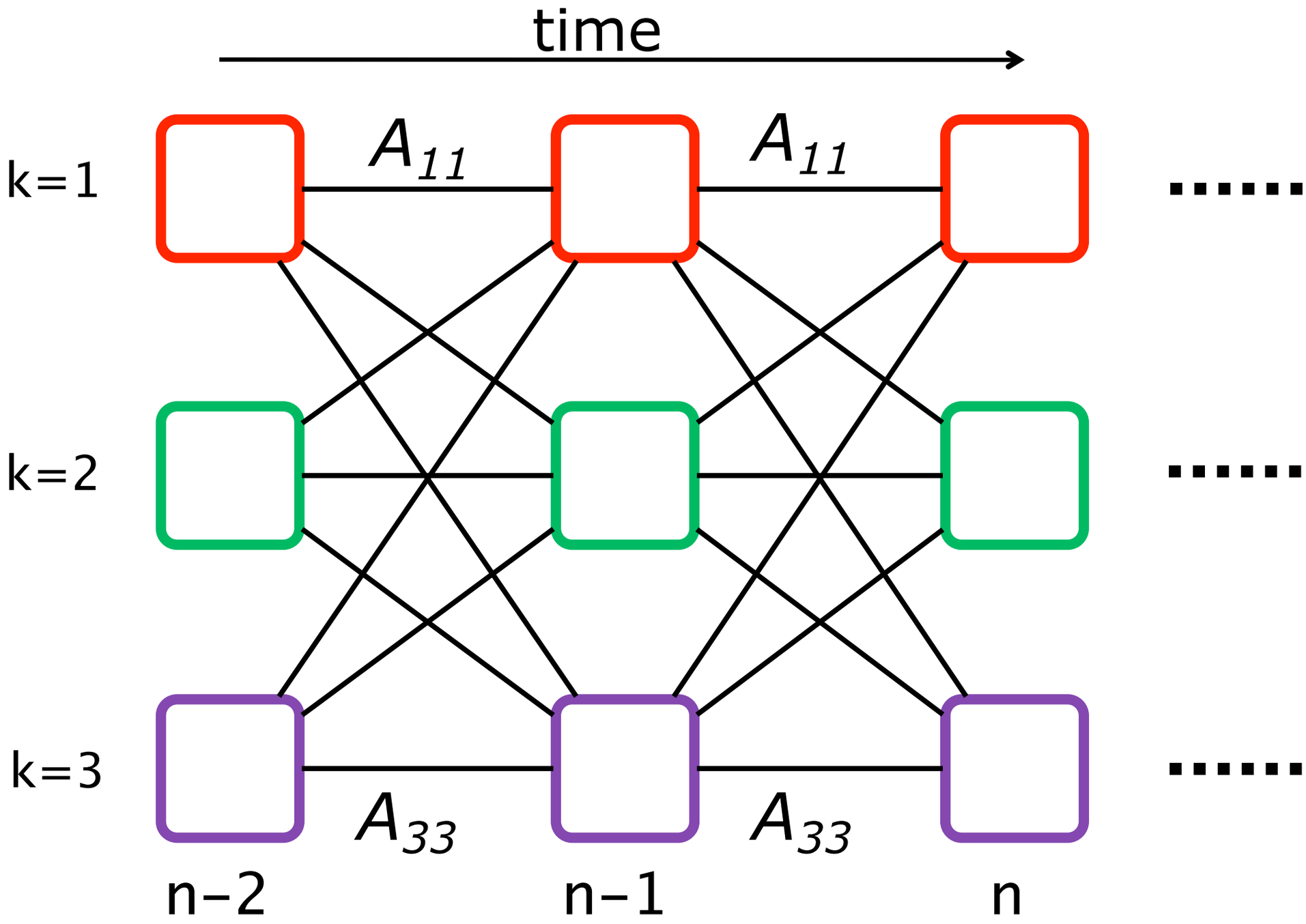
Die: LLLLLLLLLLLLLLFFFFFFLLLLLLLLLLLLLLFFFFFFFFFFFFFFFFLLLLLLLL

Formulation as HMM

1. Discrete random variables
   - Observation variables: $\{z_n\}$, $n = 1..N$
   - State variables (unobservable): $\{x_n\}$, $n = 1..N$
   - Number of states K: $x_n \epsilon \{1..K\}$
2. Transition model $p(x_i \mid x_{i-1})$
   - Markov assumption (x_i only depends on x_{i-1})
   - Represented as a $K \times K$ **transition matrix** A
   - Initial probability: $p(x_0)$ repr. as π_1, π_2, π_3
3. Observation model $p(z_i \mid x_i)$ with parameters φ
   - Observation only depends on the current state
   - Example: output of a "local" place classifier



The Trellis Representation



Application Example (1)

- Given an observation sequence $z_1, z_2, z_3 \ldots$
- Assume that the model parameters $\theta = (A, \pi, \varphi)$ are known
- What is the probability that the given observation sequence is actually observed under this model, i.e. $p(Z|\ \theta)$?
- If we are given several different models, we can choose the one with highest probability
- Expressed as a **supervised learning problem**, this can be interpreted as the inference step (classification step)

Application Example (2)

- Given an observation sequence $z_1, z_2, z_3 \ldots$
- Assume that the model parameters $\theta = (A, \pi, \varphi)$ are known
- What is the state sequence $x_1, x_2, x_3 \ldots$ that explains best the given observation sequence?
- In the case of place recognition: which is the sequence of truly visited places that explains best the sequence of obtained place labels (classifications)?

Application Example (3)

- Given an observation sequence $z_1, z_2, z_3 \ldots$
- What are the optimal model parameters $\theta = (A, \pi, \varphi)$?
- This can be interpreted as the **training step**
- It is in general the most difficult problem

Summary: 3 Operations on HMMs

1. Compute data likelihood $p(Z|\theta)$ from a known model
   - Can be computed with the **forward-backward** algorithm

2. Compute optimal state sequence with a known model
   - Can be computed with the **Viterbi**-Algorithm

3. Learn model parameters for an observation sequence
   - Can be computed using **Expectation-Maximization** (or Baum-Welch)

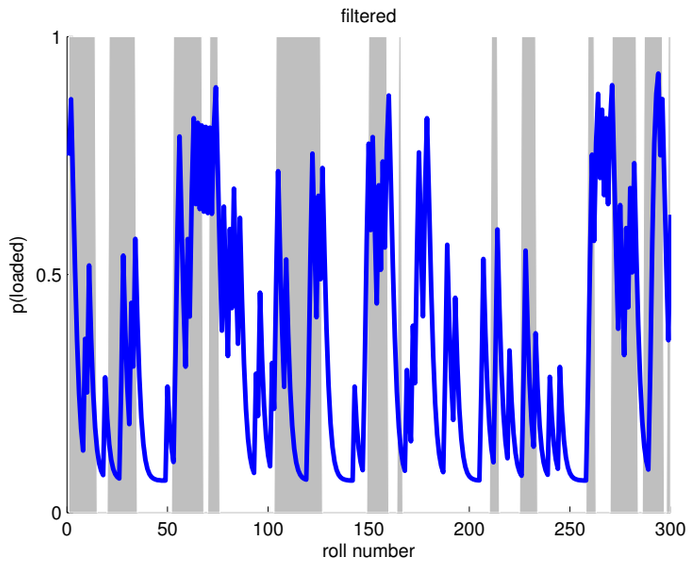
1. Computing the Data Likelihood

- Assume: given a state sequence $x_1, x_2, x_3 \ldots$

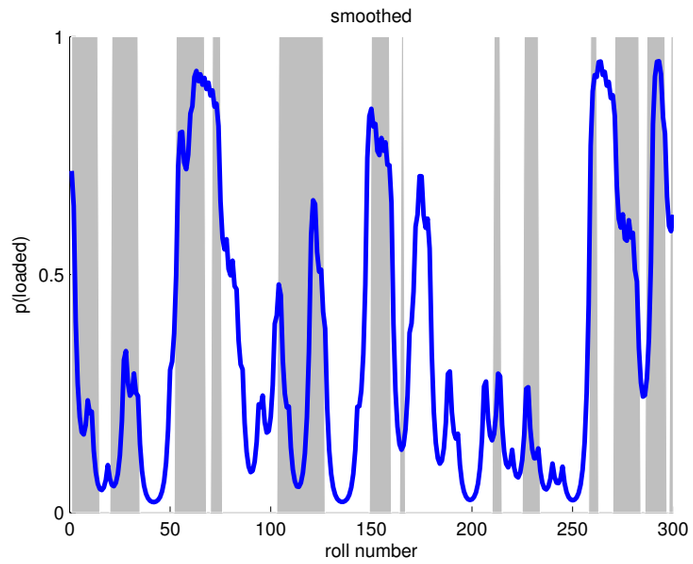
Two possible operations:

- **Filtering:** computes $p(x_t \mid \mathbf{z}_{1:t})$, i.e. state probability only based on previous observations
- **Smoothing:** computes $p(x_t \mid \mathbf{z}_{1:T})$, state probability based on **all** observations (including those from the future)

Filtering



Smoothing



The Forward Algorithm

- First, we compute the prediction from the last time step:

$$p(x_t = j \mid \mathbf{z}_{1:t-1}) = \sum_i p(x_t = j \mid x_{t-1} = i) p(x_{t-1} = i \mid \mathbf{z}_{1:t-1})$$

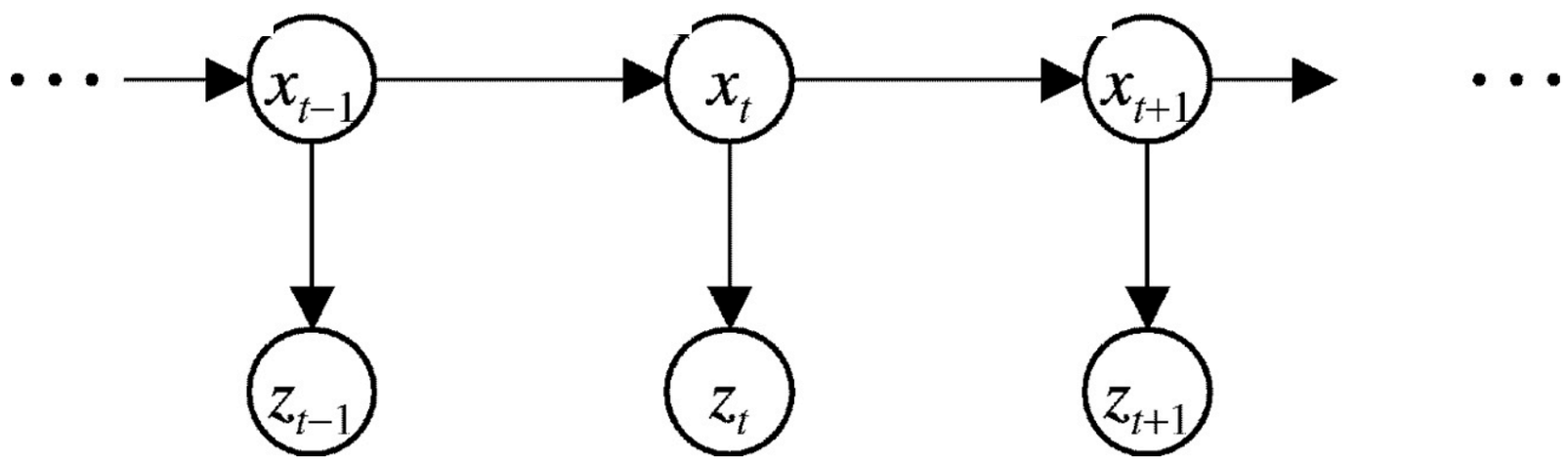
- Then, we do the update using Bayes rule:

$$\begin{aligned}\alpha_t(j) &:= p(x_t = j \mid \mathbf{z}_{1:t}) = p(x_t = j \mid z_t, \mathbf{z}_{1:t-1}) \\ &= \frac{1}{Z_t} p(\mathbf{z}_t \mid x_t = j, \cancel{\mathbf{z}_{1:t-1}}) p(x_t = j \mid \mathbf{z}_{1:t-1})\end{aligned}$$

- This is exactly the same as the Bayes filter from the first lecture!

The Forward-Backward Algorithm

- As before we set $\alpha_t(j) := p(x_t = j \mid \mathbf{z}_{1:t})$
- We also define $\beta_t(j) := p(\mathbf{z}_{t+1:T} \mid x_t = j)$



The Forward-Backward Algorithm

- As before we set $\alpha_t(j) := p(x_t = j \mid \mathbf{z}_{1:t})$
- We also define $\beta_t(j) := p(\mathbf{z}_{t+1:T} \mid x_t = j)$
- This can be recursively computed (backwards):

$$\beta_{t-1}(i) = p(\mathbf{z}_{t:T} \mid x_{t-1} = i)$$

$$= \sum_j p(x_t = j, z_t, \mathbf{z}_{t+1:T} \mid x_{t-1} = i)$$

$$= \sum_j p(\mathbf{z}_{t+1:T} \mid x_t = j, \cancel{x_{t-1} = i}, \cancel{z_t}) p(x_t = j, z_t \mid x_{t-1} = i)$$

$$= \sum_j p(\mathbf{z}_{t+1:T} \mid x_t = j) p(z_t \mid x_t = j, \cancel{x_{t-1} = i}) p(x_t = j \mid x_{t-1} = i)$$

$$= \sum_j \beta_t(j) p(z_t \mid x_t = j) p(x_t = j \mid x_{t-1} = i)$$

The Forward-Backward Algorithm

- As before we set $\alpha_t(j) := p(x_t = j \mid \mathbf{z}_{1:t})$
- We also define $\beta_t(j) := p(\mathbf{z}_{t+1:T} \mid x_t = j)$
- This can be recursively computed (backwards):

$$\begin{aligned}\beta_{t-1}(i) &= p(\mathbf{z}_{t:T} \mid x_{t-1} = i) \\ &= \sum_j \beta_t(j) p(z_t \mid x_t = j) p(x_t = j \mid x_{t-1} = i)\end{aligned}$$

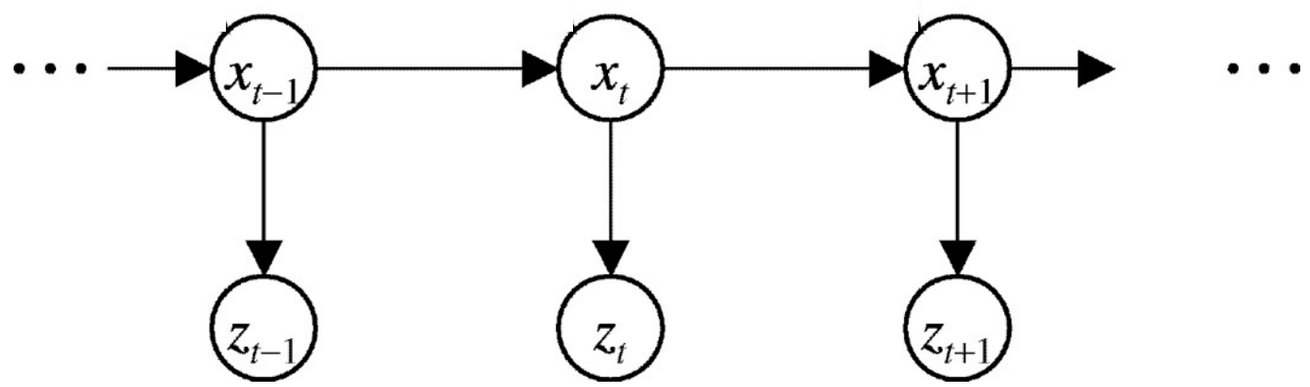
- This is exactly the same as the message-passing algorithm (sum-product)!
  - forward messages α_t (vector of length K)
  - backward messages β_t (vector of length K)

2. Computing the Most Likely States

- Goal: find a state sequence $x_1, x_2, x_3 \ldots$ that maximizes the probability $p(X, Z|\theta)$

- Define $\delta_t(j) := \max_{x_1,\ldots,x_{t-1}} p(\mathbf{x}_{1:t-1}, x_t = j \mid \mathbf{z}_{1:t})$

This is the probability of state j by taking the most probable path.



2. Computing the Most Likely States

- Goal: find a state sequence $x_1, x_2, x_3 \ldots$ that maximizes the probability $p(X, Z|\theta)$

- Define $\delta_t(j) := \max_{x_1,\ldots,x_{t-1}} p(\mathbf{x}_{1:t-1}, x_t = j \mid \mathbf{z}_{1:t})$

This can be computed recursively:

$$\delta_t(j) := \max_i \delta_{t-1}(i) p(x_t \mid x_{t-1}) p(z_t \mid x_t)$$

we also have to compute the argmax:

$$a_t(j) := \arg\max_i \delta_{t-1}(i) p(x_t \mid x_{t-1}) p(z_t \mid x_t)$$

The Viterbi algorithm

- Initialize:
  - $\delta(x_0) = p(x_0)\, p(z_0 \mid x_0)$
  - $\psi(x_0) = 0$
- Compute recursively for $n=1 \ldots N$:
  - $\delta(x_n) = p(z_n|x_n) \max_{x_{n-1}} [\delta(x_{n-1})\, p(x_n|x_{n-1})]$
  - $a(x_n) = \operatorname{argmax}_{x_{n-1}} [\delta(x_{n-1})\, p(x_n|x_{n-1})]$
- On termination:
  - $p(Z,X|\theta) = \max_{x_N} \delta(x_N)$
  - $x_N^* = \operatorname{argmax}_{x_N} \delta(x_N)$
- Backtracking:
  - $x_n^* = a(x_{n+1})$

3. Learning the Model Parameters

- Given an observation sequence $z_1, z_2, z_3 \ldots$
- Find optimal model parameters θ
- We need to maximize the likelihood $p(Z|\theta)$
- Can not be solved in closed form
- Iterative algorithm:
  Expectation Maximization (EM) or for the case of HMMs: Baum-Welch algorithm

Expectation Maximisation

- Objective: Find the model parameters knowing the observations: π,A,ϕ
- Result:
  - Train the HMM to recognize sequences of input
  - Train the HMM to generate sequences of input
- Technique: Expectation Maximisation
  - E: Find the best state sequence given the parameters
  - M: Find the parameters using the state sequence
  - Maximisation of the log-likelihood:

$$\arg\max_{pi,A,\phi} -\log\left(P\left(\{Z_i\}\ddot{}\pi, A,\Phi\phi\right)\right)$$

The Baum-Welsh algorithm

- E-Step (assuming we know π,A,ϕ, i.e. θ^{old})
- Define the posterior probability of being in state i at step k:
- Define $\gamma(x_n) = p(x_n|Z)$

The Baum-Welsh algorithm

- E-Step (assuming we know π,A,ϕ, i.e. θ^{old})
- Define the posterior probability of being in state i at step k:
- Define $\gamma(x_n) = p(x_n|Z)$
- It follows that $\gamma(x_n) = \alpha(x_n)\,\beta(x_n) / p(Z)$

The Baum-Welsh algorithm

- E-Step (assuming we know π,A,ϕ, i.e. θ^{old})
- Define the posterior probability of being in state i at step k:
- Define $\gamma(x_n) = p(x_n|Z)$
- It follows that $\gamma(x_n) = \alpha(x_n)\,\beta(x_n) / p(Z)$
- Define $\xi(x_{n-1}, x_n) = p(x_{n-1}, x_n|Z)$
- It follows that

$$\xi(x_{n-1}, x_n) = \alpha(x_{n-1})p(z_n|x_n)p(x_n|x_{n-1})\beta(x_n) / p(Z)$$

- We need to compute:

$$Q(\theta, \theta^{old}) = \sum_X p(X|Z, \theta^{old}) \log p(Z, X|\theta)$$

Expected complete data log-likelihood

The Baum-Welsh algorithm

- Maximizing Q also maximizes the likelihood: $p(Z|\theta) \geq p(Z|\theta^{old})$
- M-Step:
  - $$\pi_k = \frac{\sum_{\mathbf{x}} \gamma(\mathbf{x}) x_{1k}}{\sum_{j=1} \sum_{\mathbf{x}} \gamma(\mathbf{x}) x_{1j}}$$

here, we need forward and backward step!

  - $$A_{jk} = \frac{\sum_{t=2}^{T} \xi(x_{t-1,j}, x_{tk})}{\sum_{l=1}^{K} \sum_{t=2}^{T} \xi(x_{t-1,j}, x_{tl})}$$
- With these new values, Q is recomputed
- This is done until the likelihood does not increase anymore (convergence)

The Baum-Welsh algorithm - summary

- Start with an initial estimate of $\theta=(\pi,A,\varphi)$
  e.g. uniformly and k-means for φ
- Compute $Q(\theta,\theta^{old})$ (E-Step)
- Maximize Q (M-step)
- Iterate E and M until convergence
- In each iteration one full application of the forward-backward algorithm is performed
- Result gives a **local** optimum
- For other local optima, the algorithm needs to be started again with new initialization

The Scaling problem

- Probability of sequences

$$\prod_i p(x_i \mid ...) << 1$$

  - Probabilities are very small
  - The product of the terms soon is very small

- Usually: converting to log-space works
- But: we have sums of products!
- Solution: Rescale/Normalise the probability during the computation, e.g.:

$$\hat{\alpha}(x_n) = \alpha(x_n) \,/\, p(z_1, z_2, \ldots, z_n)$$

Summary

- HMMs are a way to model sequential data
- They assume discrete states
- Three possible operations can be performed with HMMs:
  - Data likelihood, given a model and an observation
  - Most likely state sequence, given a model and an observation
  - Optimal Model parameters, given an observation
- Appropriate scaling solves numerical problems
- HMMs are widely used, e.g. in speech recognition