EE613

Machine Learning for Engineers

Generative models. Introduction to Graphical models

jean-marc odobez

Fall 2019

overview

- Graphical models fundamentals
  - bayesian networks, representations
  - probability factorization
  - conditional independence
  - undirected graphical models
- Learning
  - Maximum Likelihood, Bayesian learning, Maximum a Posteriori (MAP)
  - the EM algorithm, latent variable models (GMM)
- Continuous Latent variable
  - Principle Component Analysis (PCA)
  - Probabilistic PCA
- Inference algorithms

Graphical models : inference

- given a factorized form of the distribution
  - directed graphical model (product of probability of each node given its parents)
  - undirected graphical model (product over cliques)

$$p(\mathbf{x}) = \prod_{k=1}^{L} p(x_k|\mathrm{pa}_k)$$

$$p(\mathbf{x}) = \frac{1}{Z}\prod_{C} \psi_C(\mathbf{x}_C)$$

- **inference**: given a learned model, compute posterior of one or more subset of the nodes given other nodes
- approach
  - use factorized form to do this efficiently
  - inference will appear as
    - propagation of local messages
    - local updates
- remaining of lecture :
  - exact inference
    inference on chain + derive general algorithm for trees
  - approximate inference: sampling methods

Inference on a chain : example

$$p(\mathbf{x}) = \frac{1}{Z}\psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3)\psi_{3,4}(x_3, x_4)$$

- How to infer the marginal $p(x_3)$?
- We consider discrete variables, each with K states
  - each potential $\psi_{i-1,i}(x_{i-1}, x_i)$ can be presented by a K x K table
- Note : we focus on undirected models: directed chains can be easily transformed in undirected versions

Inference on a chain : example

$$p(\mathbf{x}) = \frac{1}{Z}\psi_{1,2}(x_1,x_2)\psi_{2,3}(x_2,x_3)\psi_{3,4}(x_3,x_4)$$

$$p(x_3) = \sum_{x_1}\sum_{x_2}\sum_{x_4} p(\mathbf{x}) = \frac{1}{Z}\sum_{x_1}\sum_{x_2}\sum_{x_4}\psi_{1,2}(x_1,x_2)\psi_{2,3}(x_2,x_3)\psi_{3,4}(x_3,x_4)$$

$$p(x_3) = \frac{1}{Z}\left(\sum_{x_2}\psi_{2,3}(x_2,x_3)\underline{\sum_{x_1}\psi_{1,2}(x_1,x_2)}\right)\underline{\left(\sum_{x_4}\psi_{3,4}(x_3,x_4)\right)}$$

$$p(x_3) = \frac{1}{Z}\underline{\left(\sum_{x_2}\psi_{2,3}(x_2,x_3)\mu_\alpha(x_2)\right)}\mu_\beta(x_3)$$

$$\boxed{\Rightarrow p(x_3) = \frac{1}{Z}\mu_\alpha(x_3)\mu_\beta(x_3)}$$

1) compute $\mu_\alpha(x_2) = \sum_{x_1}\psi_{1,2}(x_1,x_2)$ $\quad \mu_\beta(x_3) = \sum_{x_4}\psi_{3,4}(x_3,x_4)$

2) compute $\mu_\alpha(x_3) = \sum_{x_2}\psi_{2,3}(x_2,x_3)\mu_\alpha(x_2)$

- marginal as product of two "messages" coming from the left and the right

Inference on a chain : generalisation (1)

$$p(\mathbf{x}) = \frac{1}{Z}\psi_{1,2}(x_1,x_2)\psi_{2,3}(x_2,x_3)\cdots\psi_{N-1,N}(x_{N-1},x_N)$$

Similarly, infer marginal of one of the node

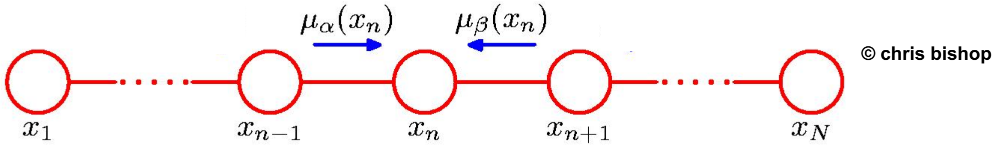
$$p(x_n) = \sum_{x_1}\cdots\sum_{x_{n-1}}\sum_{x_{n+1}}\cdots\sum_{x_N} p(\mathbf{x})$$

Brute force evaluation – complexity exponential in number of steps

- K^{N-1} summation
- K products per summation term

=> around K^N operations

Inference on a chain : generalisation (2)



$$p(x_n) = \sum_{x_1} \cdots \sum_{x_{n-1}} \sum_{x_{n+1}} \cdots \sum_{x_N} p(\mathbf{x}) \qquad p(\mathbf{x}) = \frac{1}{Z}\psi_{1,2}(x_1,x_2)\psi_{2,3}(x_2,x_3)\cdots\psi_{N-1,N}(x_{N-1},x_N)$$

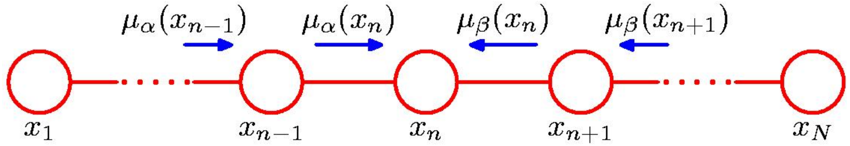
$$p(x_n) = \frac{1}{Z} \underbrace{\left[\sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1},x_n) \cdots \left[\sum_{x_1} \psi_{1,2}(x_1,x_2)\right] \cdots \right]}_{\mu_\alpha(x_n)}$$

$$\underbrace{\left[\sum_{x_{n+1}} \psi_{n,n+1}(x_n,x_{n+1}) \cdots \left[\sum_{x_N} \psi_{N-1,N}(x_{N-1},x_N)\right] \cdots \right]}_{\mu_\beta(x_n)}$$

The marginal can be interpreted as the product of local messages coming from

- the left summarizing the **marginalization** on the nodes on the left
- the right summarizing the **marginalization** involving the right nodes

Inference on a chain : generalisation (3)

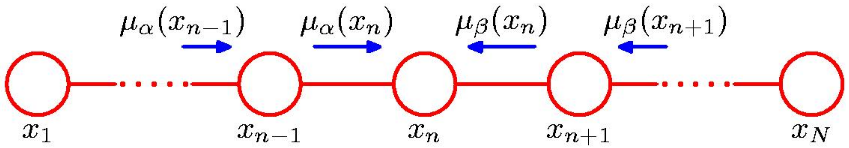


$$\begin{aligned}\mu_\alpha(x_n) &= \sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1},x_n)\left[\sum_{x_{n-2}} \cdots\right] \\ &= \sum_{x_{n-1}} \psi_{n-1,n}(x_{n-1},x_n)\mu_\alpha(x_{n-1}).\end{aligned}$$

$$\begin{aligned}\mu_\beta(x_n) &= \sum_{x_{n+1}} \psi_{n,n+1}(x_n,x_{n+1})\left[\sum_{x_{n+2}} \cdots\right] \\ &= \sum_{x_{n+1}} \psi_{n,n+1}(x_n,x_{n+1})\mu_\beta(x_{n+1}).\end{aligned}$$

The messages can be computed recursively

Inference on a chain : generalisation (4)



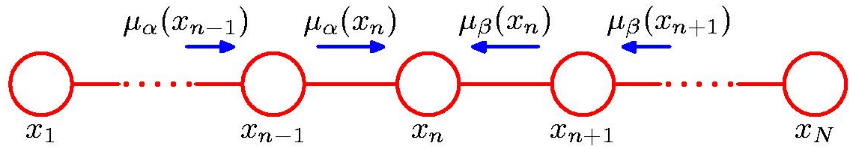
- Initial conditions

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

- Normalization constant given by $Z = \sum_{x_n} \mu_\alpha(x_n)\mu_\beta(x_n)$
- Summary: to compute local marginals
  - compute and store all forward messages $\mu_\alpha(x_n)$
  - compute and store all backward messages $\mu_\beta(x_n)$
  - compute Z at any node
  - for all variables required, compute $p(x_n) = \frac{1}{Z}\mu_\alpha(x_n)\mu_\beta(x_n)$
- Complexity
  - each recursion step K^2 operations
  - 2 recursions with N steps

  Complexity : N x K^2 - linear in the number of steps

Inference on a chain : generalisation (5)



- Computation of marginals given some observed nodes (evidence)
  - apply exactly the same procedure
  - sums over observed nodes 'disappear'

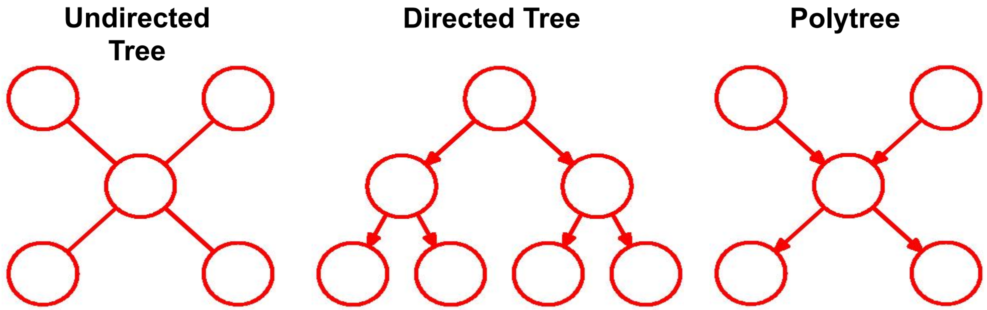
    => clamp all terms involving evidence to their values
- Example : compute $p(x_n | x_4 = a)$
  - assume n > 4 then, in alpha recursion

$$\mu_\alpha(x_4) = \sum_{x_3} \psi_{3,4}(x_3, x_4)\mu_\alpha(x_3)$$

=> needs to be computed only for $x_4 = a$

and $$\mu_\alpha(x_5) = \sum_{x_4 \in \{a\}} \psi_{4,5}(x_4, x_5)\mu_\alpha(x_4) = \psi_{4,5}(a, x_5)\mu_\alpha(x_4 = a)$$

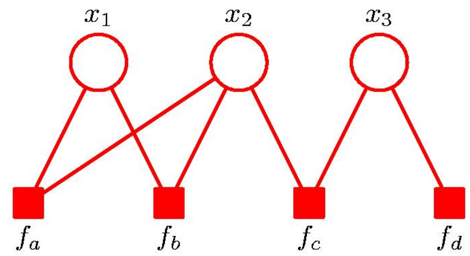
Generalization of principle to other graph structures



© chris bishop

- Undirected graphs : Trees = no loop (or cycles)
  - existence of a unique path from any node to any other node
- Directed graphs
  - Trees : one root, all other nodes have one parent at most
  - Polytrees :
    - unique path from any node to any other node (not taking into account arrow directions)

Factor graphs



© chris bishop

$$p(\mathbf{x}) = f_a(x_1, x_2) f_b(x_1, x_2) f_c(x_2, x_3) f_d(x_3)$$

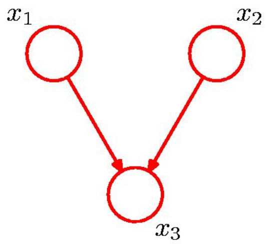
$$p(\mathbf{x}) = \prod_s f_s(\mathbf{x}_s)$$

- Probability distributions: product over factors
- Factor graphs
  - explicit representation of this fact
  - bipartite graph with variables as one node type; factors as the other nodes

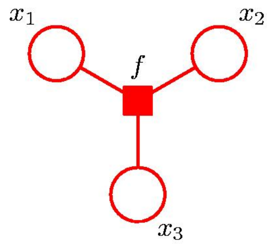
    edges: link between variable node x_i and factor node f_s iif x_i is a variable of f_s
  - useful to generalize message passing algorithms

Factor graphs from directed graphs

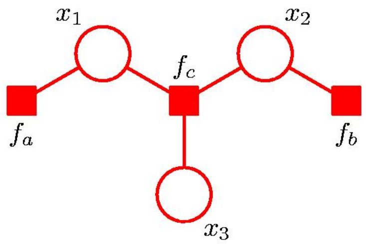
© chris bishop



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



$$f(x_1,x_2,x_3) = p(x_1)p(x_2)p(_3|x_1,x_2)$$

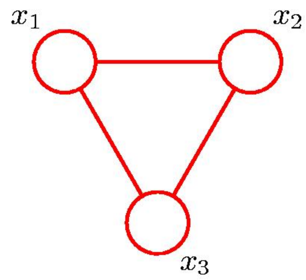


$$\begin{aligned} f_a(x_1) &= p(x_1) \\ f_b(x_2) &= p(x_2) \\ f_c(x_1,x_2,x_3) &= p(x_3|x_1,x_2) \end{aligned}$$

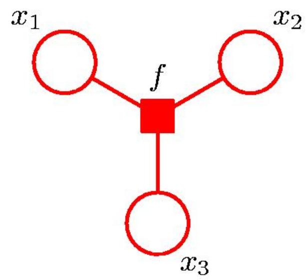
- Allows to be more explicit about the factorization (when transforming a directed graph to an undirected one)
- Note: above sample
  - polytree
  - resulting factor graph has a tree structure

Factor graphs from undirected graphs

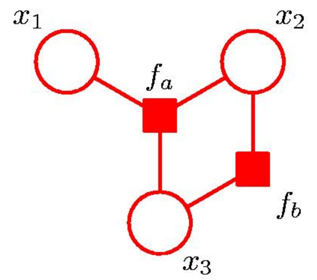
© chris bishop



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



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



$$f_a(x_1,x_2,x_3)f_b(x_2,x_3) = \psi(x_1,x_2,x_3)$$

example with the same clique structure but a different factorization

Exact inference on trees: sum product algorithm

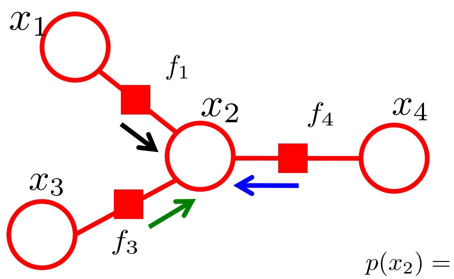
- Assumption: factor graph has a tree structure

  => can be applied to directed/undirected trees, directed polytrees
- Objectives
  - obtain an efficient, exact inference algorithm for finding marginals;
  - in situations where several marginals are required, allow computations to be shared efficiently
- Key principles
  - use factorized expression
  - distributive law : interchange summation and produc => ab+ac = a(b+c)

  => generalizes principle seen on chains to more 'branches'

- Note: belief propagation is a special case of sum-product algorithm

Sum product algorithm : example (1)



$$p(x_2) = \frac{1}{Z}\sum_{x_1}\sum_{x_3}\sum_{x_4} f_1(x_1,x_2) f_3(x_3,x_2) f_4(x_2,x_4)$$

$$p(x_2) = \frac{1}{Z}\underbrace{\left(\sum_{x_1} f_1(x_1,x_2)\right)}_{\mu_{f_1\to x_2}(x_2)}\underbrace{\left(\sum_{x_3} f_3(x_3,x_2)\right)}_{\mu_{f_3\to x_2}(x_2)}\underbrace{\left(\sum_{x_4} f_4(x_2,x_4)\right)}_{\mu_{f_4\to x_2}(x_2)}$$

$$p(x_2) = \frac{1}{Z}\mu_{f_1\to x_2}(x_2)\mu_{f_3\to x_2}(x_2)\mu_{f_4\to x_2}(x_2)$$

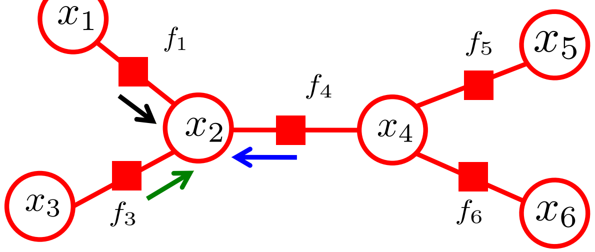
- Factor-to-variable messages $\mu_{f_j\to x_i}(x_i)$
  - summarizes **marginalization**

    in subtree associated with factor f_j

    not comprising node x_i

Sum product algorithm : example (2)

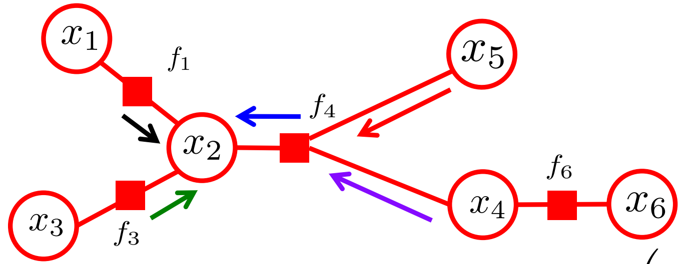
$$p(x_2) = \frac{1}{Z} \sum_{x_1} \sum_{x_3} \sum_{x_4} \sum_{x_5} \sum_{x_6} f_1(x_1, x_2) f_3(x_3, x_2) f_4(x_2, x_4) f_5(x_4, x_5) f_6(x_4, x_6)$$



$$p(x_2) = \frac{1}{Z} \left(\sum_{x_1} f_1(x_1, x_2) \right) \left(\sum_{x_3} f_3(x_3, x_2) \right) \left(\underbrace{\sum_{x_4} f_4(x_2, x_4) (\sum_{x_5} f_5(x_4, x_5)) (\sum_{x_5} f_6(x_4, x_6))}_{\mu_{f_4 \to x_2}(x_2)} \right)$$

- Only subtree where nodes were added is affected
  - message has to be computed recursively from subtree

Sum product algorithm : example (3)



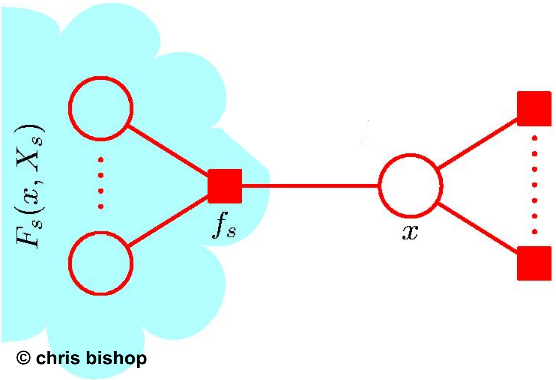
$$\underbrace{\mu_{f_4 \to x_2}(x_2)} = \sum_{x_4} \sum_{x_5} f_4(x_2, x_4, x_5) \underbrace{\left(\sum_{x_6} f_6(x_4, x_6) \right)}_{\mu_{x_4 \to f_4}(x_4)}$$

- At factor node f : to propagate message to a given node
  - combine using the factor term messages **coming from all the other subtrees** of the factor f => involves **variable-to-factor messages**
- **Variable-to-factor message** $\mu_{x_i \to f_j}(x_i)$
  - summarizes marginalization

    in subtree associated with variable x_i

    not comprising factor f_j and node x_i

Sum product algorithm : general case (1)

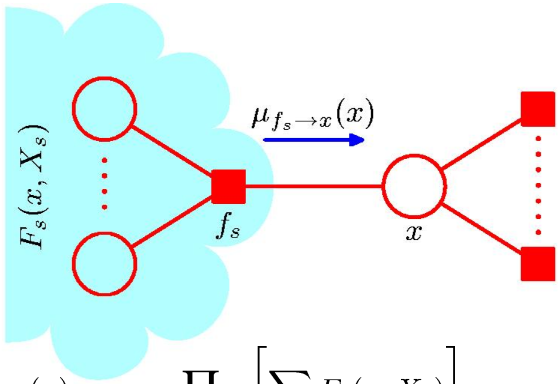


$$p(x) = \sum_{\mathbf{x} \setminus x} p(\mathbf{x})$$

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

- Tree structure : partition factors in different subgroups related to neighbors

Sum product algorithm : general case (2)



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

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

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

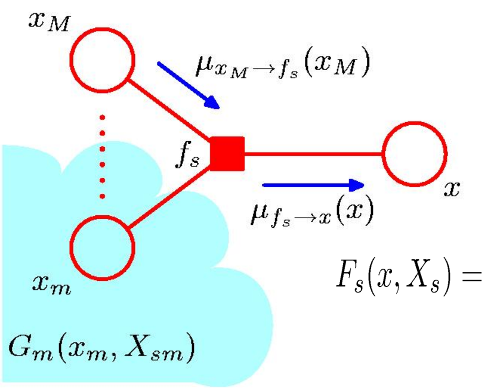
product of incoming messages from factors

$$\mu_{f_s \to x}(x) \equiv \sum_{X_s} F_s(x, X_s)$$

factor-to-variable message = marginalization over all variables in subtree linked to f_s that are not x

How to compute these messages recursively ?

Sum product algorithm : general case (3)



© chris bishop

$$\mu_{f_s \to x} = \sum_{X_s} F_s(x, X_s)$$

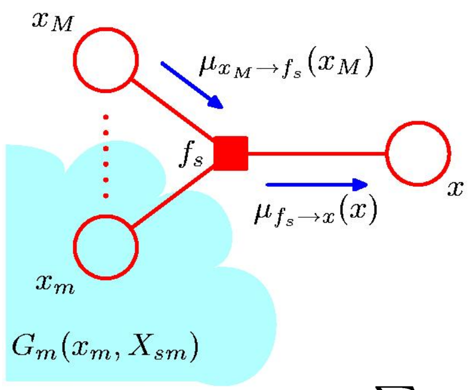
note $X_s = \{x_1, ..., x_m, X_{s1}, ..., X_{sM}\}$

$$F_s(x, X_s) = f_s(x, x_1, \ldots, x_M) \underbrace{G_1(x_1, X_{s1})}_{\text{product of factors involving nodes from subtree linked to } x_1} \ldots G_M(x_M, X_{sM})$$

thus

$$\begin{aligned}
\mu_{f_s \to x}(x) &= \sum_{x_1} \ldots \sum_{x_M} f_s(x, x_1, \ldots, x_M) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \left[\sum_{X_{sm}} G_m(x_m, X_{sm}) \right] \\
&= \sum_{x_1} \ldots \sum_{x_M} f_s(x, x_1, \ldots, x_M) \prod_{m \in \mathrm{ne}(f_s) \setminus x} \underbrace{\mu_{x_m \to f_s}(x_m)}_{\text{variable-to-factors messages}}
\end{aligned}$$

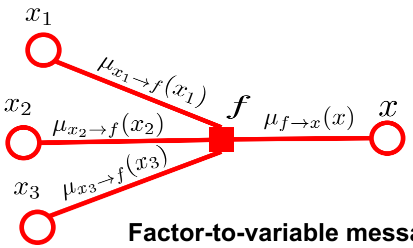
Sum-product algorithm : general case (4)



$$\begin{aligned}
\mu_{x_m \to f_s}(x_m) \equiv \sum_{X_{sm}} G_m(x_m, X_{sm}) &= \sum_{X_{sm}} \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} F_l(x_m, X_{ml}) \\
&= \prod_{l \in \mathrm{ne}(x_m) \setminus f_s} \mu_{f_l \to x_m}(x_m)
\end{aligned}$$

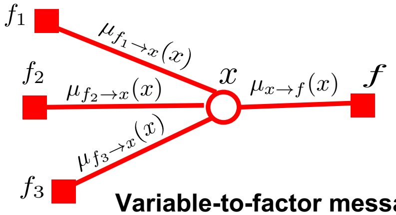
Variable-to-factor message = product of incoming factor-to-variable messages

Sum product algorithm : update summary



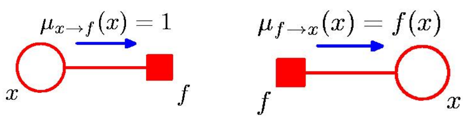
Factor-to-variable message

$$\mu_{f \to x}(x) = \sum_{x_i \in \mathrm{ne}(f) \setminus x} f(x, x_1, .., x_i, ..) \prod_{x_i} \mu_{x_i \to f}(x_i)$$

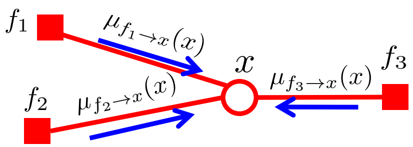


Variable-to-factor message

$$\mu_{x \to f}(x) = \prod_{f_i \in \mathrm{ne}(x) \setminus f} \mu_{f_i \to x}(x)$$



Initialisation



Marginals

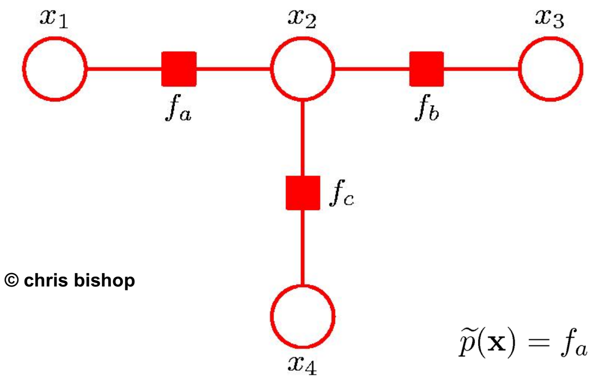
$$p(x) \propto \prod_{f \in \mathrm{ne}(x)} \mu_{f \to x}(x)$$

- **Message schedule** : a message can be sent from a node only if it has received all requested messages from its neighbours

Sum-product algorithm : general case (5)

- To compute all local marginals
  - (1) Pick an arbitrary node as root
  - (2) **Collect evidence** : compute and propagate messages from the leaf nodes to the root, storing received messages at every node
  - (3) **Distribute evidence:** Compute and propagate messages from the root to the leaf nodes, storing received messages at every node (Warning: this can include using messages from tep 2)
  - (4) Compute the product of received messages at each node for which the marginal is required, and normalize if necessary.
- Dealing with evidence/observations
  - as in the chain case, when nodes are observed, summation is not needed

Sum-product : worked out example (1)



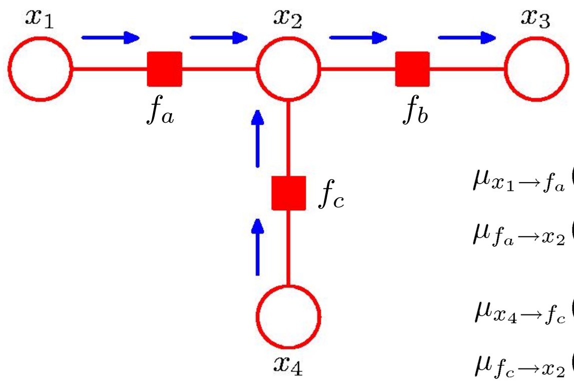
© chris bishop

$$\widetilde{p}(\mathbf{x}) = f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_2, x_4)$$

- First step

$$\text{select } x_3 \text{ as root node} \Rightarrow x_1 \text{ and } x_4 \text{ are leaves}$$

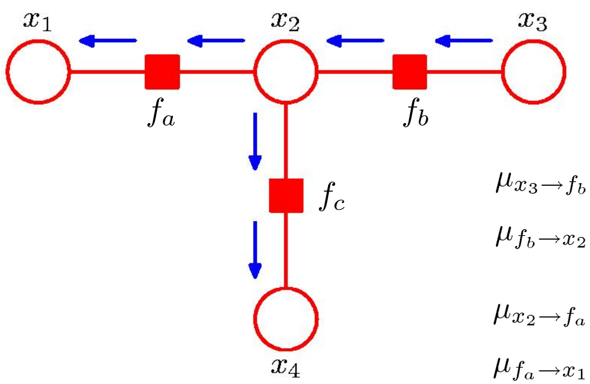
Sum-product : worked out example (2)



$$\begin{aligned}
\mu_{x_1 \to f_a}(x_1) &= 1 \\
\mu_{f_a \to x_2}(x_2) &= \sum_{x_1} f_a(x_1, x_2) \\
\mu_{x_4 \to f_c}(x_4) &= 1 \\
\mu_{f_c \to x_2}(x_2) &= \sum_{x_4} f_c(x_2, x_4) \\
\mu_{x_2 \to f_b}(x_2) &= \mu_{f_a \to x_2}(x_2) \mu_{f_c \to x_2}(x_2) \\
\mu_{f_b \to x_3}(x_3) &= \sum_{x_2} f_b(x_2, x_3) \mu_{x_2 \to f_b}(x_2)
\end{aligned}$$

- Forward sequence of messages (leaves to root)

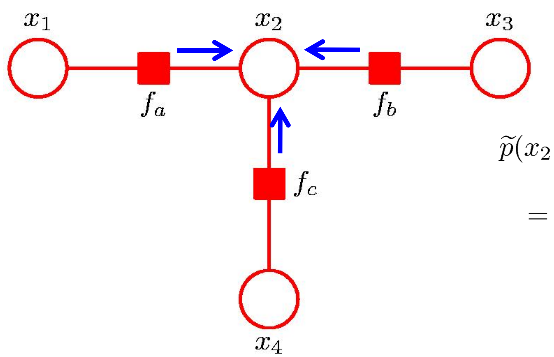
Sum-product : worked out example (3)



$$
\begin{aligned}
\mu_{x_3 \to f_b}(x_3) &= 1 \\
\mu_{f_b \to x_2}(x_2) &= \sum_{x_3} f_b(x_2, x_3) \\
\mu_{x_2 \to f_a}(x_2) &= \mu_{f_b \to x_2}(x_2)\mu_{f_c \to x_2}(x_2) \\
\mu_{f_a \to x_1}(x_1) &= \sum_{x_2} f_a(x_1, x_2)\mu_{x_2 \to f_a}(x_2) \\
\mu_{x_2 \to f_c}(x_2) &= \mu_{f_a \to x_2}(x_2)\mu_{f_b \to x_2}(x_2) \\
\mu_{f_c \to x_4}(x_4) &= \sum_{x_2} f_c(x_2, x_4)\mu_{x_2 \to f_c}(x_2)
\end{aligned}
$$

- Backward sequence of messages (root to leaves)

Sum-product : worked out example (4)



$$
\begin{aligned}
\widetilde{p}(x_2) &= \mu_{f_a \to x_2}(x_2)\mu_{f_b \to x_2}(x_2)\mu_{f_c \to x_2}(x_2) \\
&= \left[\sum_{x_1} f_a(x_1, x_2)\right]\left[\sum_{x_3} f_b(x_2, x_3)\right] \\
&\quad \left[\sum_{x_4} f_c(x_2, x_4)\right] \\
&= \sum_{x_1}\sum_{x_3}\sum_{x_4} f_a(x_1, x_2) f_b(x_2, x_3) f_c(x_2, x_4) \\
&= \sum_{x_1}\sum_{x_3}\sum_{x_4} \widetilde{p}(\mathbf{x})
\end{aligned}
$$

- Compute marginal (e.g. at node 2)
  - note : we need messages from both the forward and backward pass
- Verify that it corresponds to the definition

Remarks and discussions

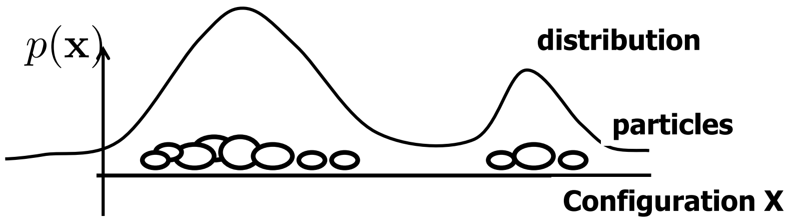
- Continuous variables
  - algorithm also works on Gaussian networks
- General graphs : junction tree
  - Message-passing algorithm can be generalized to do exact inference
  - Needs first to turn initial graph into a **Junction Tree,** and then run a sum-product like algorithm on it
  - Intractable on graphs with large cliques/loops
- Loopy belief propagation
  - apply sum-product algorithm on general graphs
  - initially, pass message across all links; then messages are passed around until convergence (not guaranteed)
  - **Approximate** but **tractable** for large graphs
  - sometimes works well, sometimes not at all

Graphical models : inference

- Exact inference
  - inference on chain
  - derive general algorithm for trees
- **Approximate inference: sampling methods**
  - **importance sampling**
  - Markov Chain Monte-Carlo (MCMC)

Sampling approaches

- **Intuition:**



approximate distribution using a set of M weighted samples

$$\{(\mathbf{x}^{(m)}, \pi^{(m)})\}_{m=1,..,M} \qquad \sum_m \pi^{(m)} = 1 \qquad p(\mathbf{x}) \simeq \sum_m \pi^{(m)} \delta(\mathbf{x} - \mathbf{x}^{(m)})$$

dirac distribution

- **Usage:**
  - compute expectation of function f

$$E_p[f] = \int f(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} \simeq \sum_m \pi^{(m)} f(\mathbf{x}^{(m)})$$

  - In particular, mean expectation of state (f(x)=x)

$$E_p[\mathbf{x}] \simeq \sum_m \pi^{(m)} \mathbf{x}^{(m)}$$

  - find max of distributions
- How do we get these samples ?

Perfect sampling

- Target distribution $p(\mathbf{x})$
- Draw M samples $\mathbf{x}^{(m)} \sim p(\mathbf{x}), \; m = 1 \cdots M$
- Approximation

$$p(\mathbf{x}) \approx \sum_{m=1}^{M} \boxed{\frac{1}{M}} \delta(\mathbf{x} - \mathbf{x}^{(m)})$$

weight of sample

- Expectation w.r.t. p

$$\mathbb{E}_p[f] = \int f(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} \longrightarrow I_M(f) = \frac{1}{M} \sum_{m=1}^{M} f(\mathbf{x}^{(m)})$$

- Approximation: unbiased, converges when M goes to infinity
- Usually (case of interest)
  - difficult to sample from p directly !
  - however, we assume that we can evaluate p(x) easily

Importance sampling

- Use a 'proposal' auxiliary function **q**
  - q : as close as possible to p (and supp(p) included in supp(q)) (i.e. $q(\mathbf{x}) = 0 \Rightarrow p(\mathbf{x}) = 0$)
  - Draw the samples from q instead of p

$$\mathbf{x}^{(m)} \sim q(\mathbf{x}),\ m = 1 \cdots M$$

$$\Rightarrow \mathbb{E}_p[f] = \int f(\mathbf{x}) \frac{p(\mathbf{x})}{q(\mathbf{x})} q(\mathbf{x}) \mathrm{d}\mathbf{x} \approx \frac{1}{M} \sum_{m=1}^{M} f(\mathbf{x}^{(m)}) \frac{p(\mathbf{x}^{(m)})}{q(\mathbf{x}^{(m)})}$$

- Importance weights $\boxed{\pi^{(m)} \propto \frac{p(\mathbf{x}^{(m)})}{q(\mathbf{x}^{(m)})} \qquad \sum_{m=1}^{M} \pi^{(m)} = 1}$

$\Rightarrow$ correction factor: samples were drawn from q instead of p

Importance sampling

- Importance weights $\boxed{\pi^{(m)} \propto \frac{p(\mathbf{x}^{(m)})}{q(\mathbf{x}^{(m)})} \qquad \sum_{m=1}^{M} \pi^{(m)} = 1}$

$\Rightarrow$ large weight if q is smaller than p

$\Rightarrow$ larger weights where q will simulate less samples than p would

samples generated from q(x), and reweighted

- Approximation of p $\quad p(\mathbf{x}) \approx \sum_{m=1}^{M} \pi^{(m)} \delta(\mathbf{x} - \mathbf{x}^{(m)})$

Comment

$$E_p[f] = \int f(\mathbf{x})p(\mathbf{x})d\mathbf{x} \simeq \sum_m \pi^{(m)} f(\mathbf{x}^{(m)}) \qquad \pi^{(m)} \propto \frac{p(\mathbf{x}^{(m)})}{q(\mathbf{x}^{(m)})}$$

- Expectation is well approximated if samples are drawn where
  - p(x) is large
  - f(x) is large
- Consequence
  - approximation works well if q(x) draws samples where p(x) is large, which indirectly means that q(x) should be as close as possible to p(x)
  - or if q(x) draws samples where f(x) is large

Example 1: likelihood weighted sampling

- For p(x) given by a Directed Acyclic Graph $\quad p(\mathbf{x}) = \prod_{k=1}^{L} p(x_k|\mathbf{x}_{\mathrm{pa}(k)})$
- Assume that there is no evidence (i.e. all x_k are unknown)
  - ⇒ just draw samples using ancestral sampling (i.e. draw them when parents are known, cf first course)
- Assume that some x_k are observed (k in Ev : evidence set)
  - use ancestral sampling for unobserved variables
  - use observed values for the others
  - ⇒ equivalent to draw sample from

$$q(\mathbf{x}) = \prod_{k \notin Ev} p(x_k|\mathbf{x}_{\mathrm{pa}(k)}) \prod_{k \in Ev} \delta(x_k - x_k^{obs})$$

  - easy to show that the weights are given by

$$\pi^{(m)} \propto \frac{p(\mathbf{x}^{(m)})}{q(\mathbf{x}^{(m)})} = \frac{\prod_{k \notin Ev} p(x_k^{(m)}|\mathbf{x}_{\mathrm{pa}(k)}^{(m)}) \prod_{k \in Ev} p(x_k^{(m)}|\mathbf{x}_{\mathrm{pa}(k)}^{(m)})}{\prod_{k \notin Ev} p(x_k^{(m)}|\mathbf{x}_{\mathrm{pa}(k)}^{(m)})} = \prod_{k \in Ev} p(x_k^{(m)}|\mathbf{x}_{\mathrm{pa}(k)}^{(m)})$$

Example 2: importance sampling and EM

- In EM algorithm, approximate the E step when E step can not be performed analytically or is costly (e.g. when latent space is large)
- Model: Z latent variables – X observed ones – parameters $\boldsymbol{\theta}$
- EM: we are optimizing the Expected Log-likelihood

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{old}) = \int p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{old}) \ln p(\mathbf{Z}, \mathbf{X}|\boldsymbol{\theta}) \mathrm{d}\mathbf{Z}$$

- Rather than computing the full posterior of latent variables, we can draw samples from it to approximate Q, which is then optimized in the usual way in the M step

$$Z^{(m)} \sim p(\mathbf{Z}|\mathbf{X}, \boldsymbol{\theta}^{old}) \qquad Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{old}) \simeq \frac{1}{M} \sum_{m=1}^{M} \ln p(\mathbf{Z}^{(m)}, \mathbf{X}|\boldsymbol{\theta})$$

- Special cases:
  - **stochastic EM**: draw only a single Z from the posterior (hard assignment)
  - extract the value of Z for which the posterior is maximum
    often used with HMM (when having enough training data),
    i.e. use Viterbi decoding to find optimal path and only keep this one for M step

Graphical models : inference

- Exact inference
  - inference on chain
  - derive general algorithm for trees
- **Approximate inference: sampling methods**
  - importance sampling
  - **Markov Chain Monte-Carlo (MCMC)**

Often used in
- undirected models (especially image processing)
- graphs with continuous variables (untractable exact inference)

Gibbs sampling (1)

- Get samples from $p(x_1, x_2, x_3)$
- Basic idea to get a chain of samples
  - initialize x^0
  - then generate new samples by
    - selecting x_i in turn
    - sampling x_i from the local posterior
    - keeping the other x_j unchanged (i.e. just recopy them)
- After some time, the samples of the chain are true samples from the distribution

$$\mathbf{x}^{(0)} = (x_1^{(0)}, x_2^{(0)}, x_3^{(0)})$$

$$\mathbf{x}^{(1)} = (x_1^{(1)}, x_2^{(1)}, x_3^{(1)}) \text{ with } x_1^{(1)} \sim p(x_1|x_2^{(0)}, x_3^{(0)}), x_2^{(1)} = x_2^{(0)}, x_3^{(1)} = x_3^{(0)}$$

$$\mathbf{x}^{(2)} \text{ with } x_2^{(2)} \sim p(x_2|x_1^{(1)}, x_3^{(1)}), x_1^{(2)} = x_1^{(1)}, x_3^{(2)} = x_3^{(1)}$$

$$\mathbf{x}^{(3)} \text{ with } x_3^{(3)} \sim p(x_3|x_1^{(2)}, x_2^{(2)}), x_1^{(3)} = x_1^{(2)}, x_2^{(3)} = x_2^{(2)}$$

$$\mathbf{x}^{(4)} \text{ with } x_1^{(4)} \sim p(x_1|x_2^{(3)}, x_3^{(3)}), x_2^{(4)} = x_2^{(3)}, x_3^{(4)} = x_3^{(3)}$$

Gibbs sampling (2)

- Note: full conditional distribution

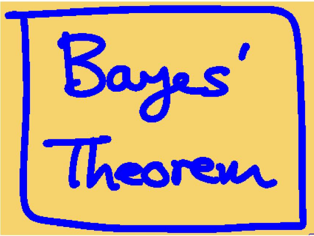
$$x_i \sim p(x_i|\mathbf{x}_{-i})$$

all components except i

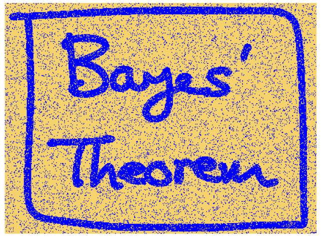
often, its computation depends only on some variables

e.g. graphical model: we only need to know the neighbors in the graph (Markov properties, i.e. conditional indepedance)

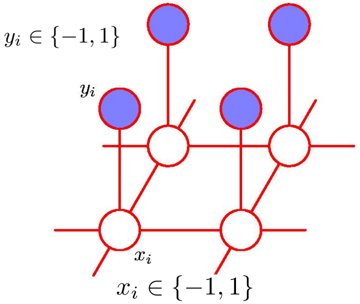
Gibbs sampling applied to the Ising model



Original Image x (hidden)
$\mathbf{x} = \{x_i, i = 1 \dots N\}$



Noisy Image y (observed)
$\mathbf{y} = \{y_i, i = 1 \dots N\}$



$\mathcal{N} = \{(k,l)$ such that x_k and x_l are neighbors $\}$

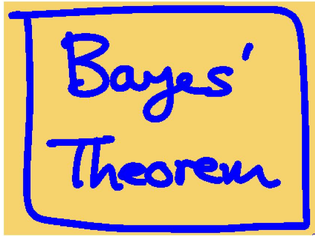
$$p(\mathbf{x}) \propto \prod_{(i,j)\in\mathcal{N}} \psi_{i,j}(x_i, x_j) \text{ with } \psi_{i,j}(x_i, x_j) = \exp(\beta x_i x_j)$$

$$p(\mathbf{y}|\mathbf{x}) \propto \prod_i \psi^d(y_i, x_i) \text{ with } \psi^d(y_i, x_i) = \exp(\alpha y_i x_i)$$

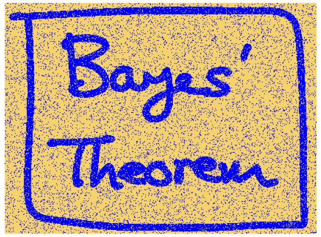
$$p(\mathbf{x}, \mathbf{y}) \propto \prod_{(i,j)\in\mathcal{N}} \psi_{i,j}(x_i, x_j) \prod_i \psi^d(y_i, x_i)$$

- Notes:
  - factors are higher when arguments are the same
  - computing full posterior $p(\mathbf{x}|\mathbf{y})$ (i.e. for all values of x) is intractable
  - get samples to approximate distribution, eg use gibbs samplings

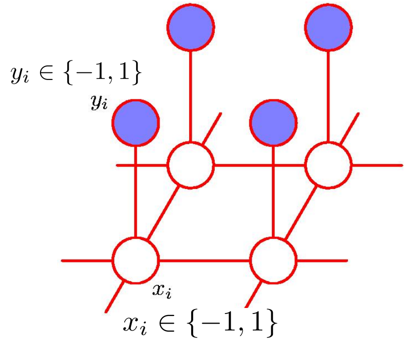
Gibbs sampling applied to the Ising model



Original Image x (hidden)
$\mathbf{x} = \{x_i, i = 1 \dots N\}$



Noisy Image y (observed)
$\mathbf{y} = \{y_i, i = 1 \dots N\}$



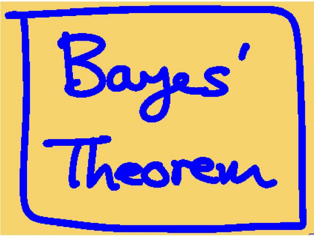
$$p(x_i|\mathbf{x}_{-i}, \mathbf{y}) = p(x_i|\mathbf{x}_{nei(i)}, y_i) \propto \exp(\eta x_i y_i) \prod_{j\in nei(i)} \psi_{i,j}(x_i, x_j)$$

$$p(x_i|\mathbf{x}_{nei(i)}, y_i) \propto h(x_i)$$

$$\text{with } h(x_i) = \exp(\eta x_i y_i + \sum_{j\in nei(i)} \beta x_i x_j) = \exp(x_i(\eta y_i + \beta\lambda_i)) \text{ with } \lambda_i = \sum_{j\in nei(i)} x_j$$

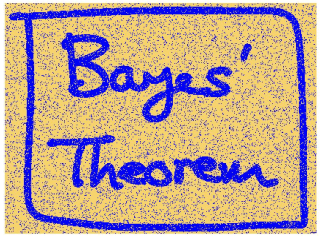
$$p(x_i = 1|\mathbf{x}_{nei(i)}, y_i) = \frac{h(x_i = 1)}{h(x_i = 1) + h(x_i = -1)} = \frac{\exp(\eta y_i + \beta\lambda_i)}{\exp(\eta y_i + \beta\lambda_i) + \exp(-\eta y_i - \beta\lambda_i)}$$

Gibbs sampling applied to the Ising model



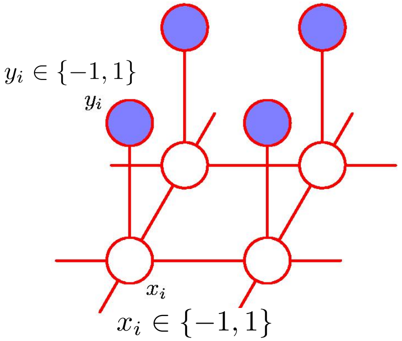
Original Image x (hidden)

$\mathbf{x} = \{x_i, i = 1 \dots N\}$



Noisy Image y (observed)

$\mathbf{y} = \{y_i, i = 1 \dots N\}$

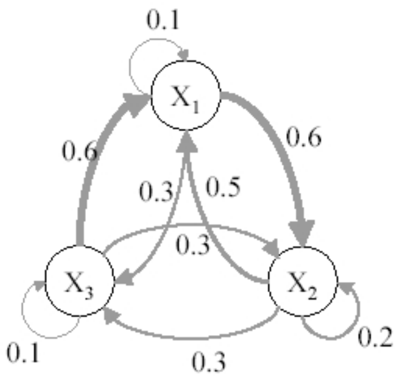


$$p(x_i = 1|\mathbf{x}_{nei(i)}, y_i) = \frac{h(x_i = 1)}{h(x_i = 1) + h(x_i = -1)} = \frac{\exp(\eta y_i + \beta\lambda_i)}{\exp(\eta y_i + \beta\lambda_i) + \exp(-\eta y_i - \beta\lambda_i)}$$

- so, sampling x_i=1 will be favored by having a y_i=+1 observation and $\lambda_i > 0$, i.e. having more neighbors with +1 than -1
- we can run the algorithms for enough iterations, and keep the samples (after removing initial samples, i.e. a burn-in period)
- we can use them to compute the mean of x according to the posterior

Markov Chains Monte Carlo

- Goal: sample from target distribution $p(\mathbf{X})$ noted here $\pi(\mathbf{X})$
- Approach: use Markov Chain
- Why ?



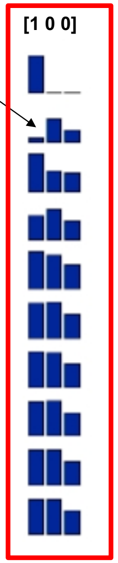
K= [

0.1	0.5	0.6
0.6	0.2	0.3
0.3	0.3	0.1

]

Markov Chains: stationary distribution

Probability of being in state one after one iteration of the transition kernel



3]

$$q_0$$
$$q_1 = K\, q_0$$
$$q_2 = K\, q_1 = K^2\, q_0$$
$$q_3 = K\, q_2 = K^2\, q_1 = K^3\, q_0$$

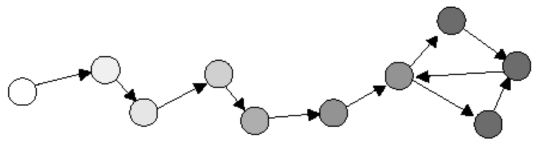
$$q_{10} = K\, q_9 = \dots K^{10}\, q_0$$

- the state distribution converges to a **stationary distribution**, whatever the starting distribution

$$K\pi = \pi$$

Metropolis-Hasting

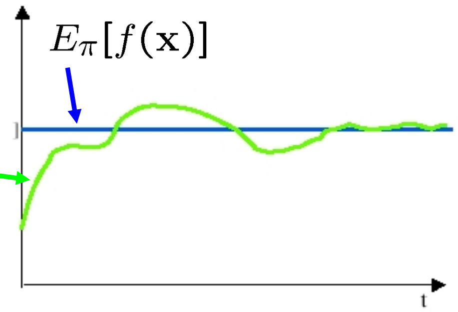
- Metropolis-Hasting algorithm : published in 1953
- Idea:
  - set-up a Markov Chain
  - run the chain until stationary
  - all subsequent samples are from stationary distribution
  - If the stationary distribution is our target, we get our samples !
- In high dimension space
  - start at $x_0 \sim q_0$
  - propose a move $K(x_t, x_{t+1})$
  - K never stored as a big matrix !
  - K can be seen as a function/search operator



MCMC inference

- Empirical average

$$\frac{1}{T}\sum_{t=1}^{T} f(\mathbf{x}^{(t)})$$



- converges to the expectation with respect to the stationary distribution $E_\pi[f(\mathbf{x})]$

- Reason: the chain is **ergodic**
  - we can forget the initial state value x_0

How do we get the right chain ? $K\pi = \pi$

- In practice, we are given the target distribution π
- How do we construct the kernel K such that the target distribution is the stationary distribution of K ?

- Idea: similar to importance sampling
  - select a proposal transition kernel q(x,x')
    - **irreductible**: you can reach any state from anywhere
    - **recurrent**: you will visit any state infinitely often
  - modify it to get the right stationary distribution
  - schematically

Detailed balance

- a sufficient condition to converge to $\pi(\mathbf{x})$

$$\pi(\mathbf{x})K(\mathbf{x},\mathbf{x}') = \pi(\mathbf{x}')K(\mathbf{x}',\mathbf{x})$$

"detailed balance"

- given q(x,x'), the detail balance is usually not satisfied
  - correction factor
    - reject a fraction of the moves to satisfy the detailed balance
    - insert a **factor a** in detailed balance

$$\pi(\mathbf{x})q(\mathbf{x},\mathbf{x}') \times a = \pi(\mathbf{x}')q(\mathbf{x}',\mathbf{x}))$$

Metropolis-Hasting algorithm

- Metropolis algorithm – produces a Markov Chain given our target distribution
  generated set of samples $\{\mathbf{x}^{(m)}\}$

1) Start with $x^{(0)}$ then iterate
2) propose x' from $q(x^{(m)},x')$
3) calculate acceptance ratio a

$$a = \frac{\pi(\mathbf{x}')q(\mathbf{x}',\mathbf{x}^{(m)})}{\pi(\mathbf{x}^{(m)})q(\mathbf{x}^{(m)},\mathbf{x}')}$$

1) accept new sample with probability min(a,1)
   if accepted set: $x^{(m+1)} = x'$
   else if rejected: set $x^{(m+1)} = x^{(m)}$

$$a = \frac{\pi(\mathbf{x}')}{\pi(\mathbf{x}^{(m)})}$$

- Note special case if q is symmetric, i.e. q(x,x')=q(x',x)
  - x' more probable than x: definitively accept (move to) the new sample
  - x' less probable than x: we may move there (not forbidden) depending on relative probabilities.

Special case: Gibbs sampling

- Notations:
  - **x** variable $\mathbf{x} = (x_1, \ldots, x_N)$
  - all components except the i^{th} one: $\mathbf{x}_{-i} = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_N)$
- Basic idea:
  - visit in turns each component x_i
  - draw sample from the local posterior of x_i given the all other variables
- More formally: define sequence of proposals

$$q(\mathbf{x}'|\mathbf{x}) = \underbrace{p(x_i'|\mathbf{x}_{-i})}_{\text{given the other component, draw component i from local}}\underbrace{\delta(\mathbf{x}'_{-i} - \mathbf{x}_{-i})}_{\text{all components different than } \mathbf{x}_i \text{ need to be unchanged in new sample}}$$

  - it can be shown that **under this assumption, a = 1** , so the new sample is always accepted

Some notes on sampling methods

- Advantages
  - often easier to implement than alternative methods
  - applicable to a broader range of models (e.g. models in which the number of components can vary)
  - sampling can be faster than other methods when applied to really huge models or datasets
- Notes
  - Issue: consecutive samples are correlated so the chain can be slow at exploring well the state space
  - Improved methods: block sampling, collapsed gibbs sampling, Rao-Blackwellization
  - can be used to find optimum of function (simulated annealing)

Conclusion

- Exact inference in Graphical Model
  - Possible for tree-structured graphs
  - Efficient algorithms existing that work with local message-passing algorithms (analog to dynamic programming)
  - Sum-product (and max-product) algorithms useful for computing marginal (and most likely) inference

  => useful for learning algorithms (cf HMM case)
  - Also possible when working with continuous variables (Gaussian)
- Approximate inference techniques
  - Sampling techniques : cf previous slide
  - Variational methods