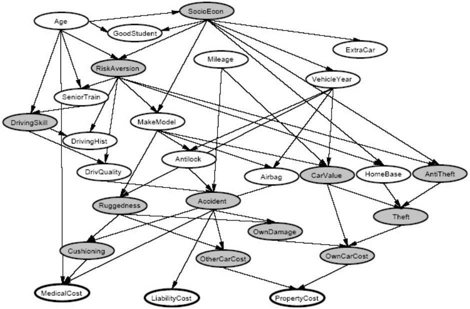
Topic 9: MESSAGE PASSING ALGORITHMS

STAT 37710/CAAM 37710/CMSC 35400 Machine Learning
Risi Kondor, The University of Chicago

Directed graphical models

Also called Bayes nets or Belief Networks. Each vertex $v \in V$ corresponds to a random variable. Graph must be acyclic but not necessarily a tree.



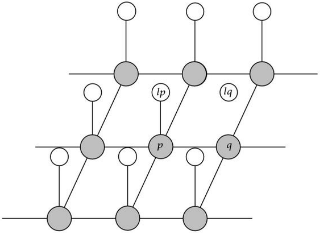
The general form of the joint distribution of all the variables is

$$p(\mathbf{x}) = \prod_{v \in V} p(x_v | \mathbf{x}_{\mathsf{pa}(v)}),$$

where $\mathsf{pa}(v)$ are all the parents of v in the graph.

Undirected graphical models

Also called Markov Random Fields. Graph can be any undirected graph.
Common example used for image segmentation:



The general form of the joint distribution over all the variables is

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{c \in \text{Cliques}(\mathcal{G})} \phi_c(\mathbf{x}_c)$$

where each ϕ_c is a potentially different **clique potential** (just a positive function) and Z is the **normalizing factor** $Z = \sum_{\mathbf{x}} \prod_{c \in \text{Cliques}(\mathcal{G})} \phi_c(\mathbf{x}_c)$.

Tasks for graphical models

- Model selection (i.e., learn the graph itself from data)
- Learn the parameters of the model from data (i.e., the individual conditionals or clique potentials)
- Deduce conditional independence relations
- Infer marginals and conditional distributions

Inference

Partition V , the set of nodes, into three sets:

1. the set O of observed nodes
2. the set Q of query nodes
3. the set L of latent nodes

Interested in $$p(\mathbf{x}_Q|\mathbf{x}_O) = \frac{\sum_{\mathbf{x}_L} p(\mathbf{x}_Q, \mathbf{x}_L, \mathbf{x}_O)}{\sum_{\mathbf{x}_L, \mathbf{x}_Q} p(\mathbf{x}_Q, \mathbf{x}_L, \mathbf{x}_O)}$$

Essential for both

- **Training**, when we are trying to learn the distribution of some of the nodes from data.
- **Prediction**, when we are trying to predict the values of some nodes (the output) given the values of some other nodes (the input)

Question: How can we do this in less than $m^{|Q|+|L|}$ time?

Inference

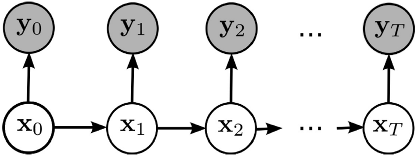
There are two approaches to inference:

- Exact inference
  - Variable elimination
  - Message passing algorithms
- Approximate inference
  - Sampling methods
  - Variational inference

In this course we focus on exact inference, specifically message passing.

Example: Hidden Markov Models

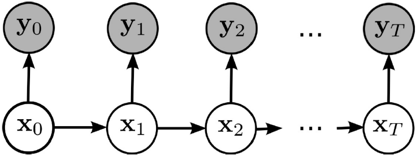
Hidden Markov Models (HMM)



Recall that in an HMM, $y_1, y_2, \ldots, y_t$ are observed and $x_1, x_2, \ldots, x_T$ are hidden (latent). Applications:

- speech recognition (which phoneme/word/etc.)
- part of speech tagging (is it a NP, VP, etc.)?
- biological sequence analysis (intron or extron)?
- time series analysis (finance, climate, etc.)
- robotics (what is the actual location of the robot)?
- tracking

Parametrizing HMMs: stationary case

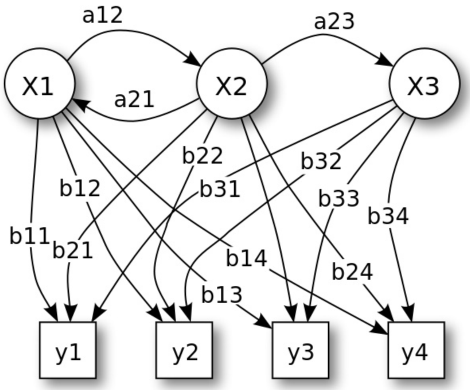


- Starting probabilities: $p(x_0 = i) = \pi_i$
- Emission probabilities: $p(\, y_t = j \,|\, x_t = i \,) = \omega_{i,j}$
- Transition probabilities: $p(\, x_{t+1} = j \,|\, x_t = i \,) = \theta_{i,j}$

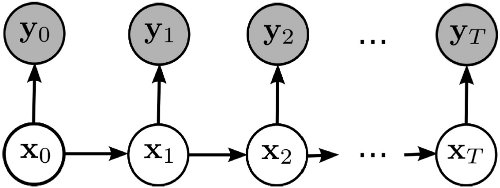
$$p(x_0, \ldots, x_T, y_0, \ldots, y_T) = p(x_0) \left(\prod_{t=1}^{T} p(x_t | x_{t-1}) \right) \left(\prod_{t=0}^{T} p(y_t | x_t) \right)$$

$$= \pi_{x_0} \left(\prod_{t=1}^{T} \theta_{x_{t-1}, x_t} \right) \left(\prod_{t=0}^{T} \omega_{x_t, y_t} \right)$$

Analogy with probabilistic finite state machines



HMM tasks



1. **Filtering:** compute $p(x_t | y_0, \ldots, y_t)$ $\rightarrow$ forward algorithm
2. **Smoothing:** compute $p(x_t | y_0, \ldots, y_t, \ldots, y_T)$ $\rightarrow$ forward-backward alg
3. **Most likely explanation:** compute $\arg\max_{x_0 \ldots x_T} p(x_0 \ldots x_T | y_0 \ldots y_T)$ $\rightarrow$ Viterbi algorithm
4. **Parameter learning:** estimate π, θ, ω $\rightarrow$ Baum–Welch algorithm (EM)

1. Filtering

The “forward” algorithm

Exploiting indepence

Goal: compute the conditional of a given hidden state x_t given all past observations:

$$p(x_t|y_0,\dots,y_t) = \frac{p(x_t, y_0, \dots, y_t)}{\sum_{x'_t} p(x'_t, y_0, \dots, y_t)} \, .$$

- Naive approach:

$$p(x_t|y_0,\dots,y_t) = \frac{\sum_{x_0,\dots,x_{t-1}} p(x_0,\dots,x_t, y_0, \dots, y_t)}{\sum_{x_0,\dots,x_{t-1},x'_t} p(x_0,\dots,x'_t, y_0, \dots, y_t)}$$

  $\rightarrow$ assuming $x_0,\dots,x_t \in \{1,\dots,k\}$, this takes $O(k^{t+1})$ time.

- Instead, exploit the independence $\{X_t, Y_t\} \perp\!\!\!\perp \{Y_0,\dots,Y_{t-1}\} \| X_{t-1}$.

Exploiting indepence

Using $\{X_t, Y_t\} \perp\!\!\!\perp \{Y_0, \dots, Y_{t-1}\} \| X_{t-1}$:

$$p(x_t, x_{t-1}, y_0, \dots, y_t) = p(x_t, y_t | x_{t-1}, y_0, \dots, y_{t-1})\, p(x_{t-1}, y_0, \dots, y_{t-1})$$

$$= p(x_t, y_t | x_{t-1})\, p(x_{t-1}, y_0, \dots, y_{t-1})$$

$$= p(y_t | x_t)\, p(x_t, x_{t-1})\, p(x_{t-1}, y_0, \dots, y_{t-1}).$$

Therefore,

$$p(x_t, y_0, \dots, y_t) = \sum_{x_{t-1}} p(y_t | x_t)\, p(x_t | x_{t-1})\, p(x_{t-1}, y_0, \dots, y_{t-1}).$$

Natural iterative algorithm: use this recursion to first compute $p(x_0, y_0)$, then $p(x_1, y_0, y_1)$, then $p(x_2, y_0, y_1, y_2)$, etc..

The "forward" algorithm

Define $\alpha_t(x_t) = p(x_t, y_0, \dots, y_t)$.

1. Seed: $\alpha_0(x_0) = p(y_0|x_0)\,\pi_{x_0}$
2. Iterate: $\alpha_t(x_t) = p(y_t|x_t)\sum_{x_{t-1}} p(x_t|x_{t-1})\,\alpha_{t-1}(x_{t-1})$

$$\boxed{p(x_t|y_0,\dots,y_t) = \frac{p(x_t, y_0, \dots, y_t)}{\sum_{x'_t} p(x'_t, y_0, \dots, y_t)} = \frac{\alpha_t(x_t)}{\sum_{x'_t} \alpha_t(x'_t)}}$$

$\rightarrow$ Reduces the complexity from $O(k^{t+1})$ to $O(tk^2)$!!!

2. Smoothing

The “forward–backward” algorithm

Exploiting indepence

Goal: compute the conditional of a hidden state x_t given all observations:

$$p(x_t|y_0,\ldots,y_T) = \frac{p(x_t, y_0, \ldots, y_T)}{\sum_{x'_t} p(x'_t, y_0, \ldots, y_T)} \,.$$

- Naive approach:

$$p(x_t|y_0,\ldots,y_{t-1}) = \frac{\sum_{x_0,\ldots,x_{t-1},x_{t+1},x_T} p(x_0,\ldots,x_T, y_0,\ldots,y_T)}{\sum_{x_0,\ldots,x_{t-1},x'_t,x_{t+1},x_T} p(x_0,\ldots,x'_T, y_0,\ldots,y_T)}$$

$\rightarrow$ assuming $x_0,\ldots,x_T \in \{1,\ldots,k\}$, this takes $O(k^{T+1})$ time.

- Instead, exploit the independences

$$\{X_t, Y_t\} \perp\!\!\!\perp \{Y_0,\ldots,Y_{t-1}\} \| X_{t-1}$$

$$\{Y_{t+1},\ldots,Y_T\} \perp\!\!\!\perp \{X_0,\ldots,X_{t-1}, Y_0,, Y_t\} \| X_t$$

.

Exploiting indepence

Using $\{X_t, Y_t\} \perp\!\!\!\perp \{Y_0, \ldots, Y_{t-1}, Y_{t+1}, \ldots, Y_T\} \| \{X_{t-1}, X_{t+1}\}$:

$$p(x_t, y_0, \ldots, y_T) = p(y_{t+1}, \ldots, y_T | x_t)\, \underbrace{p(x_t, y_0, \ldots, y_t)}_{\text{from forward alg}}.$$

Moreover,

$$p(y_{t+1}, \ldots, y_T | x_t) = \sum_{x_{t+1}} p(y_{t+2}, \ldots, y_T | x_{t+1})\, p(y_{t+1} | x_{t+1})\, p(x_{t+1} | x_t).$$

Natural iterative algorithm: use this recursion to first compute $p(y_T | x_{T-1})$, then $p(y_{T-1}, y_T | x_{T-2})$, then $p(y_{T-2}, y_{T-1}, y_T | x_{T-3})$, etc..

The "forward–backward" algorithm

Define $\beta_t(x_t) = p(y_{t+1}, \ldots, y_T | x_t)$.

1. Seed: $\beta_T(x_T) = 1$
2. Iterate: $\beta_t(x_t) = \sum_{x_{t+1}} p(x_{t+1}|x_t)\, p(y_{t+1}|x_{t+1})\, \beta_{t+1}(x_{t+1})$

$$\boxed{p(x_t|y_0, \ldots, y_T) = \frac{p(x_t, y_0, \ldots, y_T)}{\sum_{x'_t} p(x'_t, y_0, \ldots, y_T)} = \frac{\alpha_t(x_t)\beta_t(x_t)}{\sum_{x'_t} \alpha_t(x'_t)\beta_t(x'_t)}}$$

$\rightarrow$ Reduces the complexity from $O(k^{T+1})$ to $O(Tk^2)$!!!

3. Finding the most likely hidden sequenc

The Viterbi algorithm

Exploiting independence

Goal: find the most likely sequence of hidden variables

$$(\widehat{x}_0, \ldots, \widehat{x}_T) = \underset{(x_0,\ldots,x_T)}{\operatorname{argmax}}\, p(x_0, \ldots, x_T, y_0, \ldots, y_T).$$

Naively, this requires $O(Tk^T)$ time.

To exploit independence, define

$$V_t(x_t) = \max_{x_0 \ldots x_{t-1}} p(x_0, \ldots, x_t, y_0, \ldots, y_t),$$

i.e., the probability of the most likely way of ending up at x_t given the observations $y_0, \ldots, y_t$. Similar to the marginal $p(x_t, y_0, \ldots y_t)$.

Exploiting independence: forward

Since

$$p(x_0 \dots x_t, y_0 \dots y_t) = p(y_t|x_t)\, p(x_t|x_{t-1})\, p(x_0 \dots x_{t-1}, y_0 \dots y_{t-1}),$$

$$\begin{aligned}
\max_{x_0 \dots x_{t-1}} & \; p(x_0, \dots, x_t, y_0, \dots, y_t) \\
&= \max_{x_0 \dots x_{t-1}} \left[\, p(y_t|x_t)\, p(x_t|x_{t-1})\, p(x_0 \dots x_{t-1}, y_0 \dots y_{t-1})\right] \\
&= p(y_t|x_t) \max_{x_{t-1}} \left[\, p(x_t|x_{t-1}) \max_{x_0 \dots x_{t-2}} p(x_0 \dots x_{t-1}, y_0 \dots y_{t-1})\right].
\end{aligned}$$

So

$$V_t(x_t) = p(y_t|x_t)\, \max_{x_{t-1}} \left[p(x_t|x_{t-1})\, V_{t-1}(x_{t-1})\right]$$

This is the forward pass (same as in smoothing, except $\sum$ replaced by $\max$.

Exploiting independence: backward

The last element of the most likely sequence is $\widehat{x}_T = \mathrm{argmax}_{x_T} V_t(x_T)$. Furthermore,

$$\widehat{x}_{t-1} | (\widehat{x}_t, \ldots, \widehat{x}_T) = \underset{x_{t-1}}{\mathrm{argmax}} \big[\, p(\widehat{x}_t | x_{t-1}) V_{t-1}(x_{t-1}) \big].$$

So starting with $\widehat{x}_T$, we can recover $\widehat{x}_{T-1}, \widehat{x}_{T-2}, \ldots$ by the reursion

$$\widehat{x}_{t-1} = \arg\max_{x_{t-1}} \left(p(\widehat{x}_t | x_{t-1}) \, V_{t-1}(x_{t-1}) \right).$$

The Viterbi algorithm (1967)

Forward sweep:

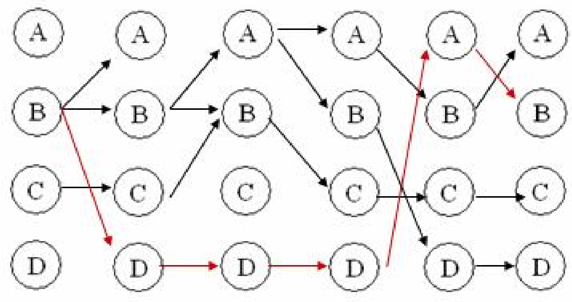
1. Seed: $V_0(x_0) = p(y_0|x_0)\,\pi_{x_0}$
2. Iterate: $V_t(x_t) = p(y_t|x_t)\ \max_{x_{t-1}} \left(p(x_t|x_{t-1})\,V_{t-1}(x_{t-1})\right)$

Backward sweep:

1. Seed: $\widehat{x}_T = \arg\max V_T(x_T)$
2. Iterate: $\widehat{x}_{t-1} = \arg\max_{x_{t-1}} \left(p(\widehat{x}_t|x_{t-1})\,V_{t-1}(x_{t-1})\right)$

Once again, the overal complexity is linear in $O(Tk^2)$.

The Viterbi algorithm



When computing each $V_t(x_t)$ if we store a pointer to $\max_{x_{t-1}} \left(p(x_t|x_{t-1})\, V_{t-1}(x_{t-1})\right)$, this is literally just retracing the pointers.

4. Learning the parameters

The Baum–Welch algorithm

The Baum–Welch algorithm

Need to jointly compute the distr. over the x_t's and the setting of the parameters $\Theta = (\pi, \omega, \theta)$ that maximizes the log likelihood $\ell(\pi, \omega, \theta | x_{0:T}, y_{0:T})$.

The expectation maximization (EM) approach is to iterate

1. **E–step:** compute the expectation of ℓ w.r.t. $x_{0:T}$:

$$\bar{\ell}_{\widehat{\Theta}_{\text{old}}}(\Theta) = \mathbb{E}(\ell_{\widehat{\Theta}_{\text{old}}}(\Theta)) = \sum_{x_{0:T}} p(x_{0:T},\, y_{0:T} | \widehat{\Theta}_{\text{old}})\, \ell(\Theta | x_{0:T}, y_{0:T})$$

2. **M-step:**

$$\widehat{\Theta} = \operatorname*{argmax}_{\Theta} \bar{\ell}_{\widehat{\Theta}_{\text{old}}}(\Theta)$$

until convergence (similar to k–means).

The Baum-Welch algorithm

Recall

$$p(x_t|y_0,\ldots,y_T) = \frac{\alpha_t(x_t)\,\beta_t(x_t)}{\sum_{x'_t}\alpha_t(x'_t)\,\beta_t(x'_t)} =: \gamma_t(x_t)$$

Also define

$$p(x_t,x_{t+1}|y_0,\ldots,y_T) = \frac{1}{Z}\,\alpha_t(x_t)\,p(x_{t+1}|x_t)\,p(y_{t+1}|x_{t+1})\,\beta_{t+1}(x_{t+1}) =$$

$$\frac{\gamma_t(x_t)}{\beta_t(x_t)}\,p(x_{t+1}|x_t)\,p(y_{t+1}|x_{t+1})\,\beta_{t+1}(x_{t+1}) =: \xi_t(x_t,x_{t+1})$$

The Baum–Welch algorithm

Doing the math, (in the stationary case) the EM approach boils down to

$$\pi_i^{(i+1)} = \gamma_0(i)$$

$$\omega_{i,j}^{(i+1)} = \frac{\sum_{t\,:\,y_t=j} \gamma_t(i)}{\sum_{t=0}^{T} \gamma_t(i)}$$

$$\theta_{i,j}^{(i+1)} = \frac{\sum_{t=0}^{T-1} \xi_t(i,j)}{\sum_{t=0}^{T-1} \gamma_t(i)},$$

which are effectively the “observed freuquencies” of transitions and emissions. Local minima a possibility, so initialization is important. Of course can use a Dirichlet prior or pseudocounts if necessary.

Numbers in logarithmic form

For any reasonable length chain, the $\alpha_t(x_t)$ and $\beta_t(x_t)$ numbers get too small for machine precision. The solution is to instead store their logarithms. We need two operations:

- **Multiplication:**

$$\log(xy) = \log x + \log y$$

- **Addition:** if $\log y \leq \log x$ use

$$\log(x+y) = \log\left(x\left(1+y/x\right)\right) = \log x + \log(1+e^{\log y - \log x}).$$

  If $\log x < \log y$ use

$$\log(x+y) = \log\left(y\left(1+x/y\right)\right) = \log y + \log(1+e^{\log x - \log y}).$$

You can write your own class "`logdouble`" for this.

Extensions of HMMs

- Hierarchical HMMs
- Auto-regressive HMMs
- Input-output HMMs
- Factorial HMMs
- Variable number of states
- State space models (continuous hidden states) $\rightarrow$ “Kalman filter”

MESSAGE PASSING IN UNDIRECTED TREES

Factorizing sums

Inference in graphical models involves computing huge sums of products like

$$p(x_1|x_6) = \sum_{x_2}\sum_{x_3}\sum_{x_4}\sum_{x_5} p(x_1)\, p(x_2|x_1)\, p(x_3|x_1)\, p(x_4|x_2)\, p(x_5|x_3)\, p(x_6|x$$

Factorizing such sums is a general problem in math. First we will find a solution for sums arising from undirected tree-shaped models.

Message passing in undirected trees

The marginal of x_r (regarded as the root) in a tree structured undirected model is

$$p(x_r) = \frac{1}{Z} \sum_{\mathbf{x}_{V \setminus \{r\}}} \prod_{(i,j) \in \text{Cliques}(\mathcal{G})} \phi_{i,j}(x_i, x_j)$$

Now factor it according to the children of the root:

$$p(x_r) = \frac{1}{Z} \prod_{u \in \text{chi}(r)} \underbrace{\left[\sum_{\mathbf{x}_{V_u}} \phi_{r,u}(x_r, x_u) \prod_{(i,j) \in E_u} \phi_{i,j}(x_i, x_j) \right]}_{m_{u \to r}(x_r)}$$

where $\text{chi}(r)$ is the set of children of the root, and V_u and E_u are the vertices and edges of the subtree rooted at u, respectively.

Message passing in undirected trees

Marginal:

$$p(x_r) = \frac{1}{Z} \prod_{u \in \text{chi}(r)} m_{u \to r}(x_r)$$

Messages to the root:

$$m_{u \to r}(x_r) = \sum_{x_u} \phi_{r,u}(x_r, x_u) \sum_{\mathbf{x}_{V_u \setminus \{u\}}} \prod_{(i,j) \in E_u} \phi_{i,j}(x_i, x_j)$$

Message passing in undirected trees

$$m_{u\to r}(x_r) = \sum_{x_u} \phi_{r,u}(x_r, x_u) \underbrace{\sum_{\mathbf{x}_{V_u\setminus\{u\}}} \prod_{(i,j)\in E_u} \phi_{i,j}(x_i, x_j)}_{\text{Factorizes further!}}$$

$$\prod_{v\in \text{chi}(u)} \Bigg[\overbrace{\sum_{x_v} \phi_{u,v}(x_u, x_v) \underbrace{\sum_{\mathbf{x}_{V_v\setminus\{v\}}} \prod_{(i,j)\in E_v} \phi_{i,j}(x_i, x_j)}_{\text{Factorizes further!}}}^{m_{v\to u}} \Bigg]$$

and so on...

Message passing in undirected trees

Overall we have the following recursive process:

- If v is a leaf (note: observed notes automatically become leaves), set

$$m_{v\to u}(x_u) = \sum_{x_v} \phi_{v,u}(x_v, x_u)$$

- else set

$$m_{v\to u}(x_u) = \sum_{x_v} \phi_{v,u}(x_v, x_u) \prod_{w\in\text{chi(v)}} m_{w\to v}(x_v).$$

- Read off result

$$p(x_r) = \frac{1}{Z} \prod_{u\in\text{chi(r)}} m_{u\to r}(x_r).$$

What if we want to compute marginals for several nodes? No problem! Just keep passing messages.

Message passing in undirected trees

Summary:

1. Collect phase (leaves to root)

$$m_{v\to u}(x_u) = \sum_{x_v} \phi_{v,u}(x_v, x_u) \qquad \text{or}$$

$$m_{v\to u}(x_w) = \sum_{x_v} \phi_{v,u}(x_v, x_u) \prod_{w\in \text{chi(v)}} m_{w\to v}(x_v)$$

2. Distribute phase

$$m_{u\to v}(x_v) = \sum_{x_u} \phi_{u,v}(x_u, x_v) \prod_{z\in \text{chi}(u)\cup \text{pa}(u)\setminus\{v\}} m_{z\to u}(x_u)$$

3. Read off marginals

$$p(x_v) = \frac{1}{Z} \prod_{u\in \text{chi}(v)\cup \text{pa}(v)} m_{u\to v}(x_v)$$

Semirings

A set R with two binary operations $+ : R \times R \to R$ and $\cdot : R \times R \to \mathbb{R}$ is a **semiring** if it satisfies the following axioms:

- $a + (b + c) = (a + b) + c$
- $\exists\, 0 \in R$ such that $0 + a = a + 0 = a$
- $a + b = b + a$
- $a \cdot (b \cdot c) = (a \cdot b) \cdot c$
- $\exists\, 1 \in R$ such that $1 \cdot a = a \cdot 1 = a$
- $a \cdot (b + c) = a \cdot b + a \cdot c$
- $(b + c) \cdot a = b \cdot a + c \cdot a$
- $0 \cdot a = a \cdot 0 = 0$

Since the only thing that message passing uses is distributivity, we can use it on any semiring $\to$ sum-product algorithm.

Semirings

Examples of semirings:

- Any ring, e.g., $(\mathbb{R}, +, \cdot)$, $(\mathbb{Q}, +, \cdot)$
- $(\mathbb{R}^+, +, \cdot)$, $(\mathbb{N}^+, +, \cdot)$
- $({\mathbb{R}^+}^{n\times n}, +, \cdot)$
- Booelan semiring: $(\{T, F\}, \vee, \wedge)$
- Log-semiring: $(\mathbb{R} \cup \{\pm\infty\}, \oplus, +)$ where $x \oplus y = -\log(e^{-x} + e^{-y})$
- Tropical semiring: $(\mathbb{R} \cup \{-\infty\}, \max, +)$
- $(\mathbb{R}^+ \cup \{0\}, \max, \cdot)$

Using the sum-product algorithm on $(\mathbb{R}^+ \cup \{0\}, \max, \cdot)$ $\rightarrow$ MAP (maximum a posteriori) inference.

Message passing in undirected trees

The above is a special case of a general procedure called the **sum-product algorithm.**

Hinges only on the fact that multiplication distributes over addition.

Therefore, has analogs for max-product and min-sum.

MESSAGE PASSING IN OTHER MODELS

Message passing in non-trees

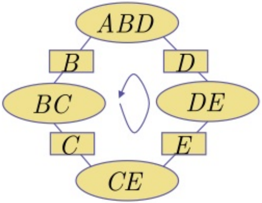
Can we generalize message passing to graphs with cycles?

- Why not just run the same algorithm? Iterate the same scheme and see if it converges to something reasonable $\rightarrow$ **Loopy belief propagation.** (approximate algorithm with no theoretical guarantees but often works suprisingly well)
- Convert graph into a tree by identifying its multiply connetcted parts and treating them as single nodes $\rightarrow$ **Junction tree algorithm.** (exact algorithm, but often unfeasible)

Message passing in non-trees

IDEA: Unify the cliques into single vertices and treat them as single variables. If some original variable is present in multiple cliques, use the conditionals to tie them together.

Question: Can a spanning tree of the resulting graph enforce all the ties? (running intersection property)

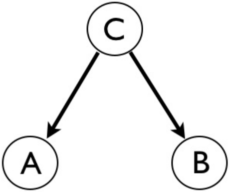


Yes, if the graph is a **junction tree**. Sufficient condition for this is that the original graph must be triangulated (no cycles of length 4 with no subcycles).

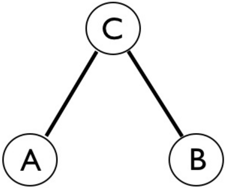
Junction tree algorithm: 1. Triangulate 2. Form junction tree from cliques 3. Run message passing.

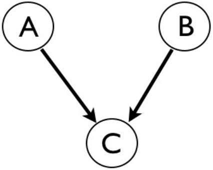
Message passing for directed models

IDEA: Convert directed model to undirected one!

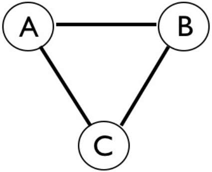


$\rightarrow$





$\rightarrow$



Marry the parents!!! (moralize)

Message passing for directed models

Summary:

1. Moralize (marry the parents)
2. Drop arrows
3. Triangulate
4. Form junction tree
5. Run message passing

FURTHER READING

- David Barber: **Bayesian Reasoning and Machine Learning** (online)
- Daphne Koller and Nir Friedman: **Probabilistic Graphical Models**
- Tutorial by Sam Roweis: `http://videolectures.net/mlss06tw_roweis_mlpgm/`
- Coursera course “Probabilistic Graphical Models” by Daphne Koller