10-708 L4 05/05/2020

- Review + readings

Jordan Ch 3

- Query node (unshaded)
- conditioning nodes - evidence nodes (dark shaded)
- marginalisation nodes - (lightly shaded)

(*) marginalisation of joint:-

$p(x_1, \dots, x_6)$

- naive → via C.P. table
- complexity $O(k^n)$

k - no. of states

n - no. of r.v.s.

(*) via factorisation law, use distrib.:-

- one-off summation has cost k^r - r - intermediate term (no. of variables)
- for n summations/elimin.
- overall cost $O(nk^r)$

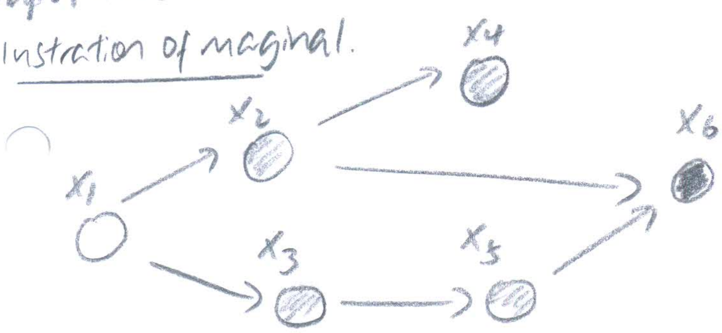
(*) $\sum_X \phi_1 \cdot \phi_2 = \phi_1 \sum_X \phi_2$ if $X \notin \text{scope}[\phi_1]$

- Graph theoretic methods for determ. of r.

(*) key idea:- 'pushing sums inside products'

- sum and product op. → individually commutative
- products → associative
- scope, summation, product

- Illustration of marginal.



- Fix $X_6 = \bar{x}_6$

$$p(x_1, \bar{x}_6) = \sum_{x_2}\sum_{x_3}\sum_{x_4}\sum_{x_5} p(x_1)p(x_2|x_1)p(x_3|x_1)p(x_4|x_2)p(x_5|x_3)p(\bar{x}_6|x_2,x_5) \quad (3.8)$$

$$= p(x_1)\sum_{x_2} p(x_2|x_1)\sum_{x_3} p(x_3|x_1)\sum_{x_4} p(x_4|x_2)\sum_{x_5} p(x_5|x_3)p(\bar{x}_6|x_2,x_5) \quad (3.9)$$

$$= p(x_1)\sum_{x_2} p(x_2|x_1)\sum_{x_3} p(x_3|x_1)\sum_{x_4} p(x_4|x_2)\, m_5(x_2,x_3)$$

i.e. $m_5(x_2,x_3) := \sum_{x_5} p(x_5|x_3)p(\bar{x}_6|x_2,x_5)$ - omit dependence on $\bar{x}_6$ for not. purp.

(*) Algebraic removal
of X_5 via elimination → node removal.

$$p(x_1, \bar{x}_6) = p(x_1) \sum_{x_2} p(x_2|x_1) \sum_{x_3} p(x_3|x_1) m_5(x_2|x_3) \sum_{x_4} p(x_4|x_2)$$

Define $m_4(x_2) := \sum_{x_4} p(x_4|x_2)$

$$= p(x_1) \sum_{x_2} p(x_2|x_1) m_4(x_2) \sum_{x_3} p(x_3|x_1) m_5(x_2, x_3)$$

Define $m_3(x_1, x_2) = \sum_{x_3} p(x_3|x_1) m_5(x_2, x_3)$

$$= p(x_1) \sum_{x_2} p(x_2|x_1) m_4(x_2) m_3(x_1, x_2)$$

Define $m_2(x_1) = \sum_{x_2} p(x_2|x_1) m_4(x_2) m_3(x_1, x_2)$

$$p(x_1, \bar{x}_6) = p(x_1) m_2(x_1) \qquad \text{(reduced joint)} \qquad (3.14)$$

$$p(\bar{x}_6) = \sum_{x_1} p(x_1) m_2(x_1) \qquad (3.15)$$

$$p(x_1|\bar{x}_6) = \frac{p(x_1, \bar{x}_6)}{p(\bar{x}_6)} = \frac{p(x_1) m_2(x_1)}{\sum_{x_1} p(x_1) m_2(x_1)} \qquad (3.16)$$

- formal trick for including conditioning on a fixed observation.

X_i - evidence node

$\bar{x}_i$ - observed value

<u>Evidence potential</u>: $\delta(x_i, \bar{x}_i) = \begin{cases} 1 & x_i = \bar{x}_i \\ 0 & \text{otherwise} \end{cases}$

Evaluate $g(x_i)$ at $\bar{x}_i$:-

$$g(\bar{x}_i) = \sum_{x_i} g(x_i) \delta(x_i, \bar{x}_i)$$

· A way of putting condition var into sum-product form

$m_6(x_2, x_5) = p(\bar{x}_6 | x_2, x_5) = \sum_{x_6} p(x_6 | x_2, x_5) \partial(x_6, \bar{x}_6)$

(*) In general:-

For subset cond. nodes E, config. $\bar{x}_E$, compute $p(x_F | \bar{x}_E)$

Total evidence pot: $\partial(x_E, \bar{x}_E) = \prod_{i \in E} \partial(x_i, \bar{x}_i)$

-(*) understanding the elimination algorithm in a formal, algorithmic sense

(it is a formalisation of the steps involved in the earlier example; by exploiting properties of summation, product operator; together with the 'scope' of a function.

(*)

-(*) Do you understand elimination:-

i) informal algebraic ✓

ii) formal algorithmic (~) ✓

iii) graph-theoretic ✓

(*) Algorithmic aspects of elim.

(*) Notes don't specify

- some supplements on elimination

(*) place all factors in active list F → sum-product variable elim algorithmically ✓

(*) check you understand elimination on non-chain structured)

(*) when eliminating a node z_i; we split active list into F' and F'' depending on whether the factor/potential/c.p.d. contains z_i.

- Take product of those factors ϕ that contain the node we want to eliminate as arguments i.e. $\phi(z_i, \dots)$ (i.e. those in F' on slides)

- Sum over z_i over the product of factors

$\sum_{z_i} \left(\prod_{\phi \in F'} \phi(z_i \dots) \right)$

- This resulting intermediate factor does NOT contain z_i and is placed in set $\{\tau\}$

(*) Then repeat the subroutine

- The ~~a~~ partition of factors with no z_i and $\{\tau\}$ are combined; z_i having become eliminated → this forms new F (active list)

· Non-chain structured elimination;
and graph-theoretic elim.

- understood in Jordan and in notes ✓

(*)

- ~~NB~~ Note that DGM → UGM in this process

(*)- Define elimination ordering

- remember conditioning/evidence potential.

- use scoping to take sum over a product of factors containing variable you wish to eliminate (delete on graph)

- This summation/marginalisation eliminates the node/variable.

- But creates an intermediate factor which ~~you need~~ is a function of new nodes. (these need connecting/moralising)

(*) Finally after all nodes other than query node eliminated; apply Bayes law over joint, conditional, normalised joint (marg.) to get query c.p.

Understand reconstituted graph as graph which is same as original; but with addition of edges that are newly created in every elimination step.

Elimination clique - the fully connected subset of nodes that include the neighbours of eliminated node (which get moralised and connected after elimination of relevant node); and eliminated node.

Marginalisation by removing r.v. from joint:

↳ sum of products of all factors that contain r.v. we wish to eliminate

↳ this couples all the other r.v.s. that appear in those factors

e.g. $\sum_{x_5} \psi(x_3,x_5) m_6(x_2,x_5)$ creates an intermediate factor involving (x_2,x_3)

i.e. inducing dependency between x_2 and x_3 (which get connected)

(*) In graph elimination algorithm, the elimination clique are those
(*) nodes/r.vs. which in the variable elimination algorithm, are the scope of the generated intermediate term/factor.
(excluding eliminated node)

- See slides → these are very intuitive and easy to understand

(*) Overall complexity of variable elimination algo is determined by the no. of variables in the largest elimin. clique

- recall $O(nR^{k})$ - r - no. of intermediate term r.vs.!

(*) That is computational complexity question → graph theoretic
(*) largest elimination clique → well studied ✓
(*) Tree width parameter $K = \left(\min_{I} w_{G,I} - 1\right)$ (minimum achievable value of cardinality of largest elim. clique over all elim. orderings)
(*) ~~largest elim. clique that is small as possible~~
(*) BUT finding best elim ordering of a graph (i.e. an elim. ordering) that achieves tree width → NP-hard.
(*) - NP-hardness → constraints on comput. efficiency of elimination algo.
→ comput. bottleneck on elimination algo
→ graph-theoretic (undirected graph eliminate) algo allows practically useful tool for assessing severity.

(*) For a given elimination ordering, cheap estimate of running time of var-eliminate algo by running graph-elimination algo (UGE).

- If graph-theoretic algo yields elim cliques of small cardinality; elim is viable.

(*) Limitations of variable elimination:
i) Single active list → inefficient traversal (S): Buckets.
ii) Single query node e.g. cond. prob of all non-evidence nodes in graph.

α a way of reducing redundant computation

- Next: i) Sum-product algo. (marginals for trees)
  ii) Junction-tree algo. (marginals for gen graphs)

(*) A clique tree : connects elimination cliques via undirected edges based on whether subsets of r.v.s. shared.

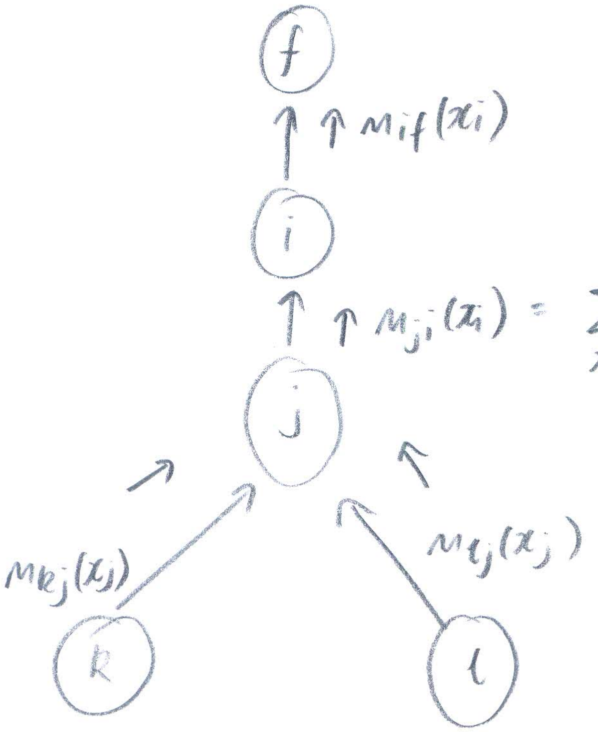
(*) Note correspondence between intermediate factors and elimination cliques

---

(*) Need to message passing on clique trees (Jordan)

(*) Reread/familiarise eliminate algo

L4 S.2020 10-708 (Message passing)



sum-product

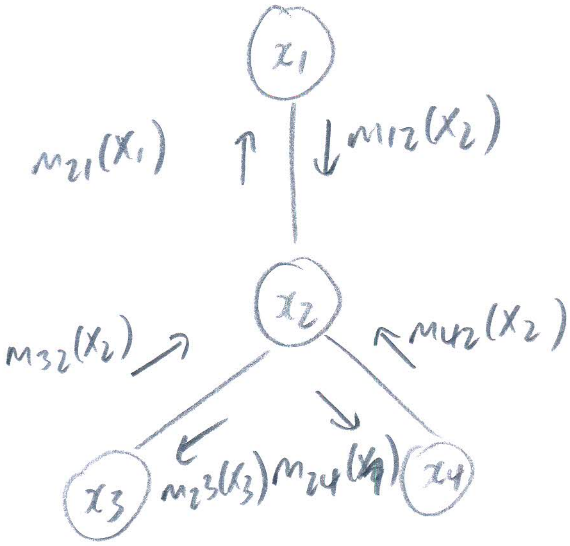
$$m_{ji}(x_i) = \sum_{x_j} \left(\psi(x_j)\psi(x_i, x_j) \prod_{k \in N(j) \setminus i} m_{kj}(x_j) \right)$$

$N(j)$ - neighbours of j

$N(j) \setminus i$ - neighbours of j excluding i

- Message from j to i i.e. $m_{ji}(x_i)$; collect messages flowing into it i.e. $m_{kj}(x_j)$ and $m_{lj}(x_j)$
- Don't use p, use ψ ; singleton, pairwise potentials

· message passing protocol



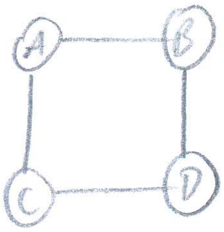
- 2 pass algorithm
  - Pick a node, hang it (make it root)
  - Generate messages from the leaves
  - collect them to the root node
    - Make contributions
      - 2 passes done
      - All necessary terms from message passing produced
        - Free to answer queries

· H M message passing

- MPP - collect messages from leaves, pass from top. (schedule)
- Belief propagation - parallel synchronous implementation
- localist view of doing message passing → inspired approximations
- Theorem: message passing guarantees all marginals on the tree:-

$$m_{ji}(x_i) = \sum_{x_j} \left(\psi(x_j) \psi(x_i, x_j) \prod_{k \in N(j) \setminus i} m_{kj}(x_j) \right)$$

Try message passing on a non-tree



· Message passing only on trees
· No guarantees on non-trees

Turn non tree → tree

↳ Do graph elim, identify elim cliques, turn into clique tree; messages on C.T.

- If you can turn an arbitrary gm → junction tree
  - can run message passing on tree cliques; but tree cliques may be huge