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:-

$p(x_1, \ldots, x_6)$

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

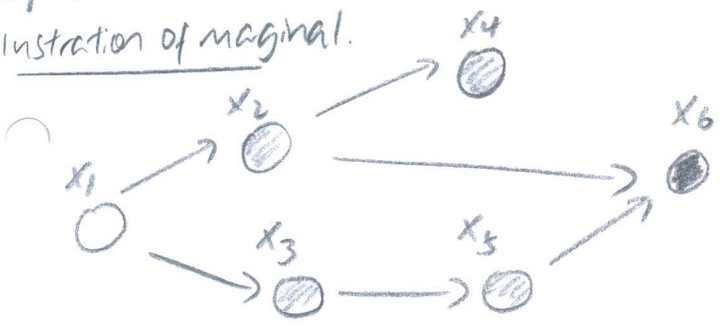
k - no. of states
n - no. of r.v.s.

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

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

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

- Graph theoretic methods for outcom. of r.
- Illustration of marginal.



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

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

- 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 prop.

(*) 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>: $\partial(x_i, \bar{x}_i) = \begin{cases} 1 & x_i = \bar{x}_i \\ 0 & \text{otherwise} \end{cases}$

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

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

- A way of putting conditional 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 earlie 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 ✓

(*) Notes don't specify
- some supplements on elimination
(*) check you understand elimination on non-chain structured F

(*) Algorithmic aspects of elim.
(*) flag all factors in active list → sum-product variable elim algorithmically ✓

(*) 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 agruments i.e. $\phi(z_i, \dots)$ (i.e. those in F' or slides)
- Sum over z_i our the product of factors $\left(\prod_{\phi \in F'} \phi(z_i \dots) \right)$

$$\sum_{z_i}$$

(*) Then repeat the subroutine

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

(*) - define elimination ordering
  - remember conditioning / evidence potential.
  - use scoping to take sum over a product of factors containg variable you wish to eliminate (delete on graph)
  - This summation/marginalisation eliminates the node/variable.
  - But creates an intermediate factor which ~~you~~ ~~want~~ 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. $\sum_{x_5} \psi(x_3, x_5) \mu_6(x_2, x_5)$ creates an intermediate factor involving (x_2, x_3)

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

- Note that DGM → UGM in this process

(*) 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 $O(nr^{\ell})$ - 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 of elimination algo

→ graph-theoretic (undirected graph eliminate) algo allows practically useful tool to assessing severity.

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

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

(*) Limitations of variable elimination:

i) Single active list → inefficient traversal ⑤: Buckets.

ii) single query node e.g. cond. prob of all non-evidence nodes in graph.

a means of reducing redundant computation

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

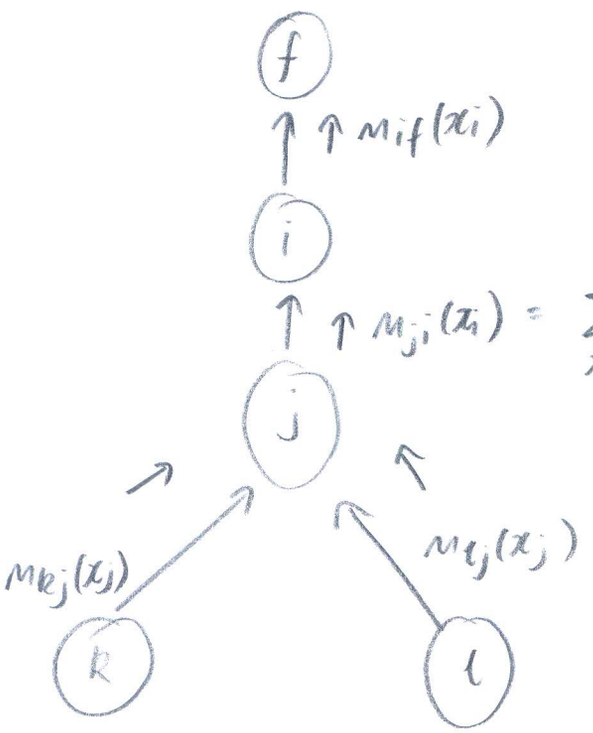
(*) A clique tree : connects elimination cliques via undirected edges based on whether subsets of r.v.s. shared.

(*) Note correspondence between intermediate factors and elimination cliques

---

(*) Need to message passing on clique trees (Jordan)
(*) Recap/familiarise eliminate algo

L4 S.2020 10.708 (message passing)



$$m_{if}(x_i)$$

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

sum-product

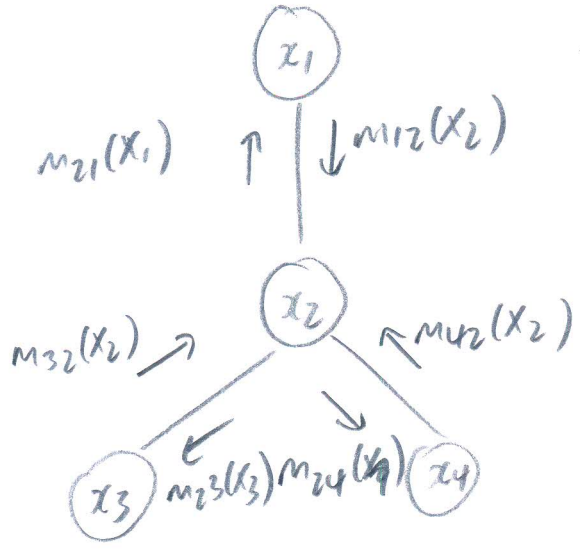
$m_{kj}(x_j)$ $\quad$ $m_{lj}(x_j)$

$N(j)$ - 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)
- create messages from the leaves
- collect them to the root node
  - make contributions
    - 2 passes done
    - All necessary items from message passing produced
    - Free to answer queries

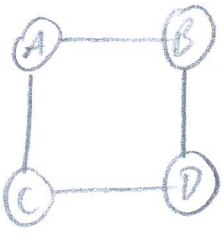
- ↑ ↓ message passing

- 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:-

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

✗ message passing on a non-tree



- Message passing only on trees
- NO guarantees on non-trees

↳ Turn non tree → tree

↳ Do graph chain, identify chain 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

10-708 06/05/2020

L5 - Parameter estimation (S.2020)

- Ex: Parameter estimation in PGMs (a clean reference point)

- start with named/indexed r.v.s.
- Avail training data (IID); fully observed - every r.v. has instantiation

1.) structure learning
  - In principle, possible to learn structure from data
  - Often experts.

2) Parameter estimation
  UGM - NOS on CPT or potential function values
  ~~UGM~~ BN

Ex: learning completely observed UGM - fairly trivial

PGMs (partially obs.)
- directed - focus on this

Estimation principles

MLE → classical setting statistics

Ex: ML hasn't gone beyond this'; present a universal, standardised view that
  ~~ifies~~ these.

Eg. reinforcement learning - intrinsic/extrinsic
  adversarial learning - adversarial score

- Traditionally, characterise learning/param est in terms of statistical
  consistency etc.
- In more modern engineering ML → may be compromised

- Simplest case
- COGMs where structure is known
- Parameter learning for BN

(*) Analytically write down loss fn → likelihood of data ~~g~~ as a function of the parameters

- Probability (likelihood)? as product of many local terms

Ex: Point you to further reading

- Building blocks of GM: W (11)

- Single node GM (e.g. root node in tree) → supp slides

- instances of exponential family distr.

(*) Ultimately; parrete (probability) is empirical frequency count

---

2 node graphical models → (10)(12)
↳ supp. slides

Ex: Patterns
modifying parrete learning for other use

Exponential family

Ex: Class interested in building blocks.

$$p(x|\eta) = h(x) \exp\{\eta^T T(x) - A(\eta)\}$$

$T(\cdot)$ and $\eta(\cdot)$ most important terms; note dot product

$A(\cdot)$ - log normaliser
(canonical)

Ex: Estimation of parrete η only require $T(\cdot)$ (i.e. sufficient statistic)

- Examples - MVG

- μ and Σ - moment parretes

- canonical, moments related

- (10)(11) → Review exp

Exponential family representation

vec(·) - ~~Inverse~~ / turn the entity into a vector

η and $T(\cdot)$ - of same dimensionality