CPSC 440: Advanced Machine Learning

DAG Models

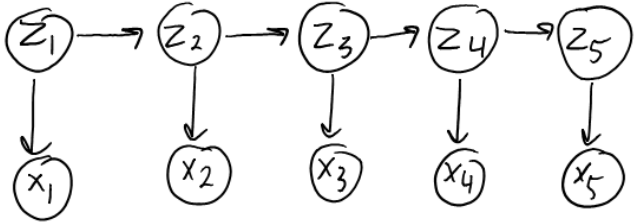
Mark Schmidt

University of British Columbia

Winter 2021

Last Time: Hidden Markov Models

- Hidden Markov models have each x_j depend on hidden Markov chain.



$$p(x_1, x_2, \ldots, x_d, z_1, z_2, \ldots z_d) = p(z_1) \prod_{j=2}^{d} p(z_j \mid z_{j-1}) \prod_{j=1}^{d} p(x_j \mid z_j).$$

- We're going to learn clusters z_j and the hidden dynamics.
  - Hidden cluster z_j could be "summer" or "winter" (we're learning the clusters).
  - Transition probability $p(z_j \mid z_{j-1})$ is probability of staying in "summer".
    - Initial probability $p(z_1)$ is probability of starting chain in "summer".
  - Emission probability $p(x_j \mid z_j)$ is probability of "rain" during "summer".

Who is Guarding Who?

- There is a lot of data on scoring/offense of NBA basketball players.
  - Every point and assist is recorded, more scoring gives more wins and $$$.
- But how do we measure defense ("stopping people from scoring")?
  - We need to know who each player is guarding.

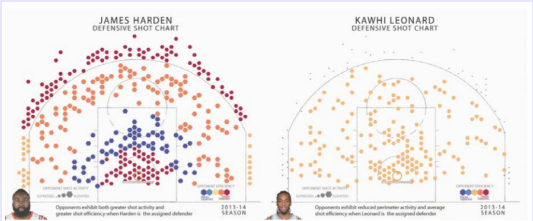


Figure 2a. Graphical depiction of a defender's volume (size) and disruption scores (color). Kawhi Leonard tends to suppress shots on the perimeter. More comparisons are provided in the Appendix.

http://www.lukebornn.com/papers/franks_ssac_2015.pdf

- HMMs can be used to model who is guarding who over time.
  - https://www.youtube.com/watch?v=JvNkZdZJBt4

Decoding in Hidden Markov Models

- The HMM model

$$p(x_1, x_2, \ldots, z_1, z_2, \ldots z_d) = p(z_1) \prod_{j=2}^{d} p(z_j \mid z_{j-1}) \prod_{j=1}^{d} p(x_j \mid z_j).$$

- Given a sequence $\{x_1, x_2, \ldots, x_d\}$, we can decode most likely z_j values.
  - "Which sequence of clusters was most likely".

- Variation on Vitberi decoding for HMMs:
  - Define $M_j(z_j) = \max_{z_1, z_2, \ldots, z_{j-1}} p(z_1) \prod_{j'=2}^{j} p(z_{j'} \mid z_{j'-1}) \prod_{j'=1}^{j} p(x_{j'} \mid z_{j'})$.
    - "Highest value sequence not depending on any future values, that ends in state z_j".
  - Base case: $M_1(z_1) = p(z_1)p(x_1 \mid z_1)$.
  - Recursion: $M_j(z_j) = \max_{z_{j-1}} p(z_j \mid z_{j-1})p(x_j \mid z_j)M_{j-1}(z_{j-1})$.
    - All terms are considered in the final $M_d(z_d)$ (intermediate M_j are not probabilities).

Learning and Inference in Hidden Markov Models

- HMMs are usually trained with EM, which requires $p(z_j \mid x_1, x_2, \ldots x_d)$.
  - Treating the z_j as nuissance variables, as in mixture models.
- But unlike Markov chains, CK equations don't work for HMMs.
  - Because $p(z_j = s \mid x_1, x_2, \ldots, x_d)$ may depend on all x_j values.
- You could compute $p(z_j = s \mid x_1, x_2, \ldots, x_d)$ using dynamic programming.
  - But this would cost $O(dk^2)$ for each value of the $O(dk)$ values j and s.
- Instead, we can use a generalization called the forward backward algorithm.
  - Allows you to compute $p(z_j = s \mid x_1, x_2, \ldots, x_d)$ for all j and s in $O(dk^2)$.
  - "Message passing" algoirthm for many variations of Markov chains, and beyond.

Notes

- This section seemed to go particularly badly. Alternative ways to warm-up to backward messages and forward-backward:
  - Do a variation on the Google problem?
  - Use DP to compute marginals of time 3 in 5-node HMM (or conditionals in a MC), show how computing marginals for time 4 has a bunch of redundant calculation (but no gain in time because "future" messages are still different?) then
    - Show how the same calculation can be done with backward messages, and how they are also redundant?
    - Show how the message factorizes out at time 3?

Message-Passing Algorithms

- We've discussed several algorithms with similar structure:
  - Viterbi decoding algorithm for decoding in discrete Markov chains.
  - CK equations for marginals in discrete Markov chains.
  - Gaussian updates for marginals in Gaussian Markov chains.

- These algorithms solve complicated problems using "forward messages" M_j:
  1. M_j summarizes all relevant past information, if you end at x_j at time j
  2. Use Markov property to write M_j recursively in terms of M_{j-1}.
  3. Solve task by computing M_1, M_2, ..., M_d.

- "Generalized distributive law" is a framework for describing when/why this works:
  - `https://authors.library.caltech.edu/1541/1/AJIieeetit00.pdf`

- In some cases we'll also use "backwards messages" V_j ("cost to go" or "value"):
  - V_j summarizes all relevant future information, if you start at x_j at time j.
  - Use Markov property to write V_j recursively in terms of V_{j+1}.

Forward Messages and Backwards Messages

- In the case of the CK equations for Markov chains we have $M_1(x_1) = p(x_1)$ and:

$$M_j(x_j) = \sum_{x_{j-1}} p(x_j \mid x_{j-1}) M_{j-1}(x_{j-1}),$$

  which are the forward messages.

- The backwards messages for Markov chains have $V_d(x_d) = 1$ and:

$$V_j(x_j) = \sum_{x_{j+1}} p(x_{j+1} \mid x_j) V_{j+1}(x_{j+1}),$$

  and you compute marginals at any time using $p(x_j) = M_j(x_j)V_j(x_j)$.
  - But here backwards messages do nothing, since $V_j(x_j) = 1$ for all j and x_j.

Forward-Backward Algorithm

- Forward-backward algorithm for computing marginals:
  - Compute all forward messages $M_j(x_j)$ and backward messages $V_j(x_j)$.
  - Compute all univariate marginals using the formula $p(x_j) \propto \frac{M_j(x_j)V_j(x_j)}{\kappa(x_j)}$.
    - Value $\kappa(x_j)$ is needed to avoid "double counting" (see HMM example below).
- Why do we care about backwards messages?
  - Compute Markov chain conditionals $p(x_j = s \mid x_{10} = 3)$ for all j and s in $O(dk^2)$.
    - Fix $M_{10}(3) = 1$ and $V_{10}(3) = 1$, and other $M_{10}(x_{10})$ and $V_{10}(x_{10})$ values to zero.
    - Then run forward-backward algorithm with these values ($V_j(x_j)$ won't be 1 for all j).
    - Backward messages modify CK equations with "what future information you need".
  - Can be used to compute probabilities in generalizations of Markov chains.
    - HMM forward message: $M_j(z_j) = \sum_{z_{j-1}} p(z_j \mid z_{j-1})p(x_j \mid z_j)M_{j-1}(z_{j-1})$.
    - HMM backward message: $V_j(z_j) = \sum_{z_{j+1}} p(z_{j+1} \mid z_j)p(x_j \mid z_j)V_{j+1}(x_{j+1})$.
    - HMM correction: $\kappa(z_j) = p(x_j \mid z_j)$ (divide by it to avoid counting twice).
  - In reinforcement learning, estimating the "cost to go" ("value") function is the goal.
    - We aren't covering RL, but understanding Markov chains will help you understand RL.

Outline

Higher-Order Markov Models

- Markov models use a density of the form

$$p(x) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_2)p(x_4 \mid x_3)\cdots p(x_d \mid x_{d-1}).$$

- They support efficient computation but Markov assumption is strong.

- A more flexible model would be a second-order Markov model,

$$p(x) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_2, x_1)p(x_4 \mid x_3, x_2)\cdots p(x_d \mid x_{d-1}, x_{d-2}),$$

  or even a higher-order models.

- General case is called directed acyclic graphical (DAG) models:
  - They allow dependence on any subset of previous features.

DAG Models

- As in Markov chains, DAG models use the chain rule to write

$$p(x_1, x_2, \ldots, x_d) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \cdots p(x_d \mid x_1, x_2, \ldots, x_{d-1}).$$

- We can alternately write this as:

$$p(x_1, x_2, \ldots, x_d) = \prod_{j=1}^{d} p(x_j \mid x_{1:j-1}).$$

- In Markov chains, we assumed x_j only depends on previous x_{j-1} given past.

- In DAGs, x_j can depend on any subset of the past $x_1, x_2, \ldots, x_{j-1}$.

DAG Models

- We often write joint probability in DAG models as

$$p(x_1, x_2, \ldots, x_d) = \prod_{j=1}^{d} p(x_j \mid x_{\text{pa}(j)}),$$

where $\text{pa}(j)$ are the "parents" of feature j.

  - For Markov chains the only "parent" of j is $(j-1)$.
  - If we have k parents we only need 2^{k+1} parameters (for binary states).

- This corresponds to a set of conditional independence assumptions,

$$p(x_j \mid x_{1:j-1}) = p(x_j \mid x_{\text{pa}(j)}),$$

that we're independent of previous non-parents given the parents.

From Probability Factorizations to Graphs

- DAG models are also known as "Bayesian networks" and "belief networks".
- "Graphical" name comes from visualizing parents/features as a graph:
  - We have a node for each feature j.
  - We place an edge into j from each of its parents.
- The DAG representation for a Markov chains is:

  - Different than "state transition diagrams": edges are between variables (not states).
- This graph is not just a visualization tool:
  - Can be used to test arbitrary conditional independences ("d-separation").
  - Graph structure tells us whether message passing is efficient ("treewidth").

Graph Structure Examples

With product of independent we have

$$p(x) = \prod_{j=1}^{d} p(x_j),$$

so $\text{pa}(j) = \varnothing$ and the graph is:

Graph Structure Examples

With Markov chain we have

$$p(x) = p(x_1) \prod_{j=2}^{d} p(x_j \mid x_{j-1}),$$

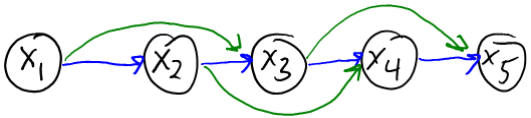
so $\text{pa}(j) = \{j-1\}$ and the graph is:

Graph Structure Examples

With second-order Markov chain we have

$$p(x) = p(x_1)p(x_2 \mid x_1) \prod_{j=3}^{d} p(x_j \mid x_{j-1}, x_{j-2}),$$

so $\text{pa}(j) = \{j-2, j-1\}$ and the graph is:

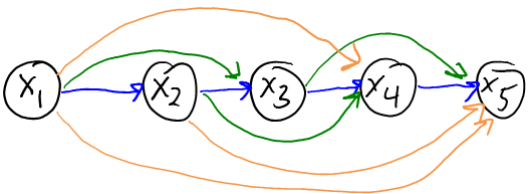


Graph Structure Examples

With general distribution we have

$$p(x) = \prod_{j=1}^{d} p(x_j \mid x_{1:j-1}).$$

so $\text{pa}(j) = \{1, 2, \ldots, j-1\}$ and the graph is:

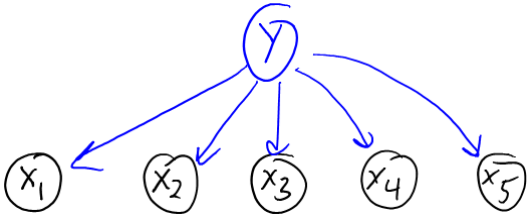


Graph Structure Examples

In naive Bayes (or GDA with diagonal Σ) we add an extra variable y and use

$$p(y, x) = p(y) \prod_{j=1}^{d} p(x_j \mid y),$$

which has $\text{pa}(y) = \emptyset$ and $\text{pa}(x_j) = y$ giving

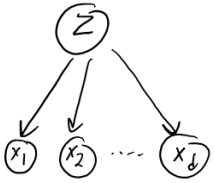


Graph Structure Examples

With mixture of independent models we have

$$p(z, x) = p(z) \prod_{j=1}^{d} p(x_j \mid z).$$

which has $\text{pa}(z) = \emptyset$ and $\text{pa}(x_j) = z$ giving same structure as naive Bayes:



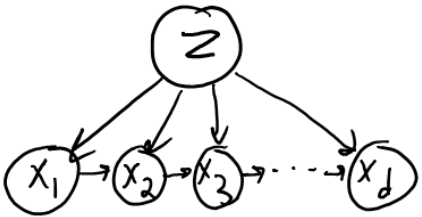
Since structure is the same, many computations will be similar.

Graph Structure Examples

With mixture of Markov chains models we have

$$p(x_1, x_2, \ldots, x_d, z) = p(z)p(x_1 \mid z) \prod_{j=2}^{d} p(x_j \mid x_{j-1}, z).$$

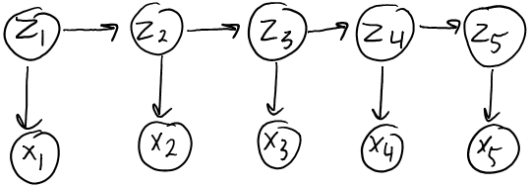
which has $\text{pa}(z) = \emptyset$ and $\text{pa}(x_j) = \{x_{j-1}, z\}$:



Graph Structure Examples

Sometimes it's easier to present a model using the graph.

In hidden Markov models we have this structure:

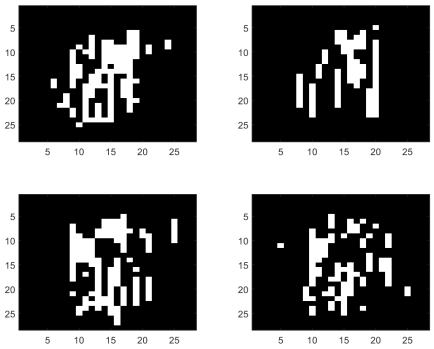


The graph and variable names already give you an idea of what this model does:

- We have hidden variables z_j that follow a Markov chain.
- Each feature x_j depends on corresponding hidden variable z_j.

MNIST DIgits with Markov Chains

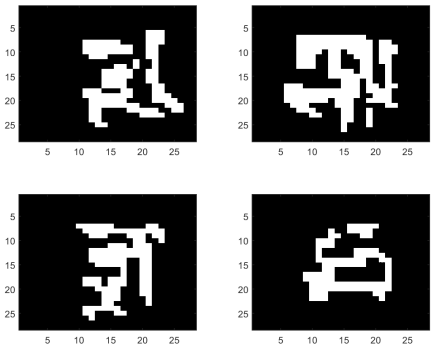
- Recall trying to model digits using an inhomogeneous Markov chain:



Only models dependence on pixel above, not on 2 pixels above nor across columns.

MNIST Digits with DAG Model (Sparse Parents)

- Samples from a DAG model with 8 parents per feature:



Parents of (i, j) are 8 other pixels in the neighbourhood ("up by 2, left by 2"):

$$\{(i-2, j-2), (i-1, j-2), (i, j-2), (i-2, j-1), (i-1, j-1), (i, j-1), (i-2, j), (i-1, j)\}.$$

Summary

- Forward-backward generalization of CK equations.
  - Allows you to solve many Markov-like problems.
  - Special case of a message passing algorithm.
- DAG models factorize joint distribution into product of conditionals.
  - Assume conditionals depend on small number of "parents".
- Next time: conditional independence in DAGs.
  (I am not going to pretend this is exciting, but its is really useful)

Chapman-Kolmogorov Equations as Message Passing

- We can view Chapman Kolmogorov equations as message passing:

$$
\begin{aligned}
p(x_4) = \sum_{x_3}\sum_{x_2}\sum_{x_1} p(x_1, x_2, x_3, x_4) &= \sum_{x_3}\sum_{x_2}\sum_{x_1} p(x_4 \mid x_3)p(x_3 \mid x_2)p(x_2 \mid x_1)p(x_1) \\
&= \sum_{x3} p(x_4 \mid x_3) \sum_{x_2} p(x_3 \mid x_2) \sum_{x_1} p(x_2 \mid x_1) M_1(x_1) \\
&= \sum_{x3} p(x_4 \mid x_3) \sum_{x_2} p(x_3 \mid x_2) M_2(x_2) \\
&= \sum_{x3} p(x_4 \mid x_3) M_3(x_3) \\
&= M_4(x_4),
\end{aligned}
$$

- Messages $M_j(x_j)$ are the marginals of the Markov chain.
  - So we can view CK equations as Viterbi decoding with "max" replace by "sum".
  - These two methods are also known as "max-product" and "sum-product" algorithms.

Backwards "Cost to Go" Messages

- Using backwards messages $V_j(x_j)$ to (innefficiently) compute $p(x_1)$:

$$
\begin{aligned}
p(x_1) = \sum_{x_2}\sum_{x_3}\sum_{x_4} p(x_1,x_2,x_3,x_4) &= \sum_{x_2}\sum_{x_3}\sum_{x_4} p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_2)p(x_4 \mid x_3) \\
&= p(x_1)\sum_{x_2} p(x_2 \mid x_1)\sum_{x_3} p(x_3 \mid x_2)\sum_{x_4} p(x_4 \mid x_3) \\
&= p(x_1)\sum_{x_2} p(x_2 \mid x_1)\sum_{x_3} p(x_3 \mid x_2)\sum_{x_4} p(x_4 \mid x_3)\underbrace{V_4(x_4)}_{=1} \\
&= p(x_1)\sum_{x_2} p(x_2 \mid x_1)\sum_{x_3} p(x_3 \mid x_2)\underbrace{V_3(x_3)}_{1} \\
&= p(x_1)\sum_{x_2} p(x_2 \mid x_1)\underbrace{V_2(x_2)}_{1} \\
&= p(x_1)\underbrace{V_1(x_1)}_{1}.
\end{aligned}
$$

- Observe that backwards messages $V_j(x_j)$ are not probabilities as in CK equations.
  - But they summarize everything you need to know about the future.
  - Can use this structure to condition on the future, and compute things like $p(x_1 \mid x_4)$.

Computing Conditional Probabilities

- Previously: Monte Carlo for approximating conditional probabilities
- For Gaussian/discrete Markov chains, we can do better than rejection sampling.
  1. We can generate exact samples from conditional distribution (bonus slide).
     - Rejection sampling is not needed, relies on "backwards sampling" in time.
  2. We can find conditional decoding $\max_{x \mid x_{j'}=c} p(x)$:
     - Run Viterbi decoding with $M_{j'}(c) = 1$ and $M_{j'}(c') = 0$ for $c \neq c'$.
  3. We can find univariate conditionals, $p(x_j \mid x_{j'})$.

- Example of computing $p(x_1 = c \mid x_3 = 1)$ in a length-4 discrete Markov chain:

$$\begin{aligned} p(x_1 = c \mid x_3 = 1) &\propto p(x_1 = c, x_3 = 1) \\ &= \sum_{x_4} \sum_{x_2} p(x_1 = c, x_2, x_3 = 1, x_4), \end{aligned}$$

  where the normalizing constant is the marginal $p(x_3 = 1)$.
- This is a sum over k^{d-2} possible assignments to other variables.

Distributing Sum across Product

- Fortunately, the Markov property makes the sums simplify as before:

$$\begin{aligned}
\sum_{x_4}\sum_{x_2} p(x_1 = c, x_2, x_3 = 1, x_4) &= \sum_{x4}\sum_{x3=1}\sum_{x_2}\sum_{x_1=c} p(x_4 \mid x_3)p(x_3 \mid x_2)p(x_2 \mid x_1)p(x_1) \\
&= \sum_{x4}\sum_{x3=1}\sum_{x_2} p(x_4 \mid x_3)p(x_3 \mid x_2)\sum_{x_1=c} p(x_2 \mid x_1)p(x_1) \\
&= \sum_{x4}\sum_{x3=1} p(x_4 \mid x_3)\sum_{x_2} p(x_3 \mid x_2)\sum_{x_1=c} p(x_2 \mid x_1)M_1(x_1) \\
&= \sum_{x4}\sum_{x3=1} p(x_4 \mid x_3)\sum_{x_2} p(x_3 \mid x_2)M_2(x_2) \\
&= \sum_{x4}\sum_{x3=1} p(x_4 \mid x_3)M_3(x_3) \\
&= \sum_{x4} M_4(x_4),
\end{aligned}$$

where $M_j(x_j)$ now sums over paths ending in x_j instead of maximizing.

- And we set $M_1(c') = 0$ if $c' \neq c$ and $M_3(c') = 0$ for $c' \neq 1$.

Conditionals via Backwards Messages

- Performing our conditional calculation using backwards messages.

$$\begin{aligned}
\sum_{x_4}\sum_{x_2} p(x_1 = c, x_2, x_3 = 1, x_4) &= \sum_{x_1=c}\sum_{x_2}\sum_{x3=1}\sum_{x4} p(x_4 \mid x_3)p(x_3 \mid x_2)p(x_2 \mid x_1)p(x_1) \\
&= \sum_{x_1=c} p(x_1)\sum_{x_2} p(x_2 \mid x_1)\sum_{x3=1} p(x_3 \mid x_2)\sum_{x4} p(x_4 \mid x_3) \\
&= \sum_{x_1=c} p(x_1)\sum_{x_2} p(x_2 \mid x_1)\sum_{x3=1} p(x_3 \mid x_2)\sum_{x4} p(x_4 \mid x_3)\underbrace{V_4(x_4)}_{=1} \\
&= \sum_{x_1=c} p(x_1)\sum_{x_2} p(x_2 \mid x_1)\sum_{x3=1} p(x_3 \mid x_2)V_3(x_3) \\
&= \sum_{x_1=c} p(x_1)\sum_{x_2} p(x_2 \mid x_1)V_2(x_2) \\
&= \sum_{x_1=c} p(x_1)V_1(x_1).
\end{aligned}$$

Forward-Backward Algorithm

- Generic forward and backward messages for discrete marginals have the form

$$M_j(x_j) = \sum_{x_{j-1}} p(x_j \mid x_{j-1}) M_{j-1}(x_{j-1}), \quad V_j(x_j) = \sum_{x_{j+1}} p(x_{j+1} \mid x_j) V_{j+1}(x_{j+1}).$$

- We can compute $p(x_j = c \mid x_{j'} = c')$ using only forward messages:
  - Set $M_j(c) = 1$ and $M_{j'}(c') = 1$.

- Why we would need backward messages?

Forward-Backward Algorithm

- We can compute $p(x_j = c \mid x_{j'} = c')$ for all j in $O(dk^2)$ with both messages.
- First compute all message normally with $M_{j'}(c') = 1$ and $V_{j'}(c') = 1$.

  (Set $M_{j'}(c)$ and $V_{j'}(c)$ to 0 for other values of c.)
- We then have that
  - $M_j(x_j)$ sums up all the paths that end in state x_j (with $x_{j'} = c'$).
  - $V_j(x_j)$ sums up all the paths that start in state x_j (with $x_{j'} = c'$).
  - We can combine these values to get

$$p(x_j \mid x_{j'}) \propto M_j(x_j)V_j(x_j),$$

  - Computing all M_j and V_j is called the forward-backward algorithm.

Conditional Samples from Gaussian/Discrete Markov Chain

Generating exact conditional samples from Gaussian/discrete Markov chains:

1. If we're only conditioning on first j states, $x_{1:j}$, just fix these values and start ancestral sampling from time $(j+1)$.
2. If we have the marginals $p(x_j)$, we can get the "backwards" transition probabilities using Bayes rule,

$$p(x_j \mid x_{j+1}) = \frac{p(x_{j+1} \mid x_j)p(x_j)}{p(x_{j+1})},$$

   which lets us run ancestral sampling in reverse: sample x_d from $p(x_d)$, then x_{d-1} from $p(x_{d-1} \mid x_d)$, and so on.
3. If we're only conditioning on last j states $x_{d-j:d}$, run CK equations to get marginals and then start ancestral sampling "backwards" starting from $(d-j-1)$ to sample the earlier states.

Conditional Samples from Gaussian/Discrete Markov Chain

4. If we're conditioning on contiguous states in the middle, $x_{j:j'}$, run ancestral sampling forward starting from position $(j'+1)$ and backwards starting from position $(j-1)$.
5. If you condition on non-contiguous positions j and j' with $j < j'$, need to do (i) forward sampling starting from $(j'+1)$, (ii) backward sampling starting from $(j-1)$, and (iii) CK equations on the sequence $(j:j')$ to get marginals conditioned on value of j then backwards sampling back to j starting from $(j'-1)$.

The above are all special cases of conditioning in an undirected graphical model (UGM), followed by applying the "forward-filter backward-sampling" algorithm on each of the resulting chain-structured UGMs.