CPSC 540: Machine Learning
Directed Acyclic Graphical Models

Mark Schmidt
University of British Columbia
Winter 2018
Directed Acyclic Graphical Models

Last Time: Viterbi Decoding and Message Passing

- **Decoding** in density models: finding $x$ with highest joint probability:

  $$
  \arg\max_{x_1, x_2, \ldots, x_d} p(x_1, x_2, \ldots, x_d).
  $$

- For Markov chains, we find decoding by writing maximization as

  $$
  \max_{x_1, x_2, x_3, x_4} p(x_1, x_2, x_3, x_4) = \max_{x_4} \max_{x_3} p(x_4 \mid x_3) \max_{x_2} p(x_3 \mid x_2) \max_{x_1} p(x_2 \mid x_1) p(x_1),
  $$

  where $M_1(x_1)$, $M_2(x_2)$, $M_3(x_3)$, and $M_4(x_4)$ are messages (summarize everything about past).

- **Viterbi decoding** computes $M_1(x_1)$ for all $x_1$, $M_2(x_2)$ for all $x_2$, and so on.

The $M_j(x_j)$ functions are called messages (summarize everything about past).
Chapman-Kolmogorov Equations as Message Passing

- We can also view Chapman Kolmogorov equations as message passing:

\[
\sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} p(x_1, x_2, x_3, x_4) = \sum_{x_4} \sum_{x_3} \sum_{x_2} \sum_{x_1} p(x_4 \mid x_3)p(x_3 \mid x_2)p(x_2 \mid x_1)p(x_1)
\]

\[
= \sum_{x_4} \sum_{x_3} p(x_4 \mid x_3) \sum_{x_2} p(x_3 \mid x_2) \sum_{x_1} p(x_2 \mid x_1) M_1(x_1)
\]

\[
= \sum_{x_4} \sum_{x_3} p(x_4 \mid x_3) \sum_{x_2} p(x_3 \mid x_2) M_2(x_2)
\]

\[
= \sum_{x_4} \sum_{x_3} p(x_4 \mid x_3) M_3(x_3)
\]

\[
= \sum_{x_4} M_4(x_4)
\]

- 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”.
  - Also known as “max-product” and “sum-product” algorithms.
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 are all special cases of message-passing algorithms:
  1. Define $M_j$ summarizing all relevant information about the past at time $j$.
  2. Use Markov property to write $M_j$ recursively in terms of $M_{j-1}$.
  3. Solve task by computing $M_1, M_2, \ldots, M_d$.

- “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 need backwards message $V_j$ (“cost to go”):
  - $V_j$ summarizes all relevant information about the future at time $j$. 
Conditionals via Backwards Messages

- Markov chain decoding using backwards messages $V_j(x_j)$:

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

$$= \max_{x_1} p(x_1) \max_{x_2} p(x_2 | x_1) \max_{x_3} p(x_3 | x_2) \max_{x_4} p(x_4 | x_3)$$

$$= \max_{x_1} p(x_1) \max_{x_2} p(x_2 | x_1) \max_{x_3} p(x_3 | x_2) \max_{x_4} p(x_4 | x_3) V_4(x_4)$$

$$= \max_{x_1} p(x_1) \max_{x_2} p(x_2 | x_1) \max_{x_3} p(x_3 | x_2) V_3(x_3)$$

$$= \max_{x_1 = c} p(x_1) \max_{x_2} p(x_2 | x_1) V_2(x_2)$$

$$= \max_{x_1} p(x_1) V_1(x_1).$$

- Computing all $M_j(x_j)$ and $V_j(x_j)$ is called forward backward algorithm.
  - Important later to compute marginals in generalizations of Markov chains.
  - Can be used to efficiently compute conditionals (bonus).
Outline

1 Directed Acyclic Graphical Models

2 D-Separation
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

- **DAG** models use product rule, $p(a, b) = p(a)p(b \mid a)$, to write

\[
p(x_1, x_2, \ldots, x_d) = p(x_1)p(x_2, x_3, \ldots, x_d \mid x_1) \\
= p(x_1)p(x_2 \mid x_1)p(x_3, x_4, \ldots, x_d \mid x_1, x_2) \\
= p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_2, x_1)p(x_4, x_5, \ldots, x_d \mid x_1, x_2, x_3),
\]

and so on until we get

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

- This **factorization** holds for any distribution.
- But it leads to **complicated conditionals**:
  - For binary $x_j$, we need $2^d$ parameters for $p(x_d \mid x_1, x_2, \ldots, x_{d-1})$ alone.
DAG Models

- To reduce number of parameters, in DAG models we use

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

where \( \text{pa}(j) \) are the “parents” of node \( j \).

- If we have \( k \) parents we only need \( 2^{k+1} \) parameters.
- For Markov chains the only “parent” of \( j \) is \( (j - 1) \).

- 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.
MNIST Digits with Markov Chains

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

  ![Images of MNIST digits](image)

  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 \((i - 2 : i, j - 2 : j)\):
\[
\{(i-2, j-2), (i-1, j-2), (i, j-2), (i-2, j-1), (i-1, j-1), (i, j-1), (i-2, j), (i-1, j)\}.
\]
Directed Acyclic Graphical Models

From Probability Factorizations to Graphs

- DAG models are also known as “Bayesian networks” and “belief networks”.

- “Graphical” name comes from visualizing features/parents as a graph:
  - We have a node for each variable $j$.
  - We place an edge into $j$ from each of its parents.

- The graph for Markov chains is:

```
 X_1 -> X_2 -> X_3 -> X_4 -> X_5
```

- 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”).
With product of independent we have

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

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

\[ \begin{array}{cccccc}
\text{x}_1 & \text{x}_2 & \text{x}_3 & \text{x}_4 & \text{x}_5 \\
\end{array} \]
Directed Acyclic Graphical Models

Graph Structure Examples

With Markov chain we have

\[ p(x) = p(x_1) \prod_{j=2}^{d} p(x_j | x_{j-1}), \]

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

```
  X_1 -----> X_2 -----> X_3 -----> X_4 -----> X_5
```
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 \(pa(j) = \{j - 2, j - 1\}\) and the graph is:
Directed Acyclic Graphical Models

Graph Structure Examples

With **general distribution** we have

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

so \( pa(j) = \{1, 2, \ldots, j - 1\} \) and the graph is:
In naive Bayes we add an extra variable $y$ and use

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

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

With mixture of independent models we have

\[ p(z, x) = p(z) \prod_{j=1}^{d} p(x_j | z). \]

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

![Directed Acyclic Graphical Models](image)
Graph Structure Examples

- Instead of factorizing by variables $j$, could factor into blocks $b$:
  $$p(x) = \prod_b p(x_b | x_{pa(b)}),$$
  and have the nodes be blocks (we assume full connectivity within the block).

- With mixture of Gaussian and full covariances we have
  $$p(z, x) = p(z)p(x | z).$$
  The corresponding graph structure is:
Graph Structure Examples

With Gaussian generative classifier (GDA) we have the same structure:

\[ p(y, x) = p(y)p(x \mid y). \]
With probabilistic PCA we have

\[
p(z, x) = p(x \mid z) \prod_{c=1}^{k} p(z_c).
\]

The corresponding graph structure is:
Sometimes it’s easier to present a model using the graph.

Later in the course we’ll see hidden Markov models which have this structure:

You should already be able to get 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 $z_j$. 
Graph Structure Examples

We can consider less-structured examples,

\[ p(S, V, R, W, G, D) = p(S)p(V)p(R \mid V)p(W \mid S, R)p(G \mid V)p(D \mid G). \]

The corresponding graph structure is:
We can consider **phylogeny** (family trees):
DAGs and Conditional Independence

- In DAGs we make the **conditional independence assumption** that
  \[
p(x_j \mid x_{j-1}, x_{j-2}, \ldots, x_1) = p(x_j \mid x_{pa(j)}).
\]

- But these conditional independence assumptions **can imply other assumptions**.
  - For example, in Markov chains we directly assume for all \(j\) that
    \[
p(x_j \mid x_{j-1}, x_{j-2}, \ldots, x_1) = p(x_j \mid x_{j-1}),
    \]
    but this also implies that
    \[
p(x_j \mid x_{j-2}, x_{j-3}, \ldots, x_1) = p(x_j \mid x_{j-2}),
    \]
    and it implies that
    \[
p(x_j \mid x_{j+1}, x_{j+2}, \ldots, x_d) = p(x_j \mid x_{j+1}).
    \]

- Knowing which assumptions hold can help **identify which operations are efficient**.
  - For example, decoding in general DAGs is NP-hard but it's easy in Markov chains.
Review of Independence

Let $A$ and $B$ are random variables taking values $a \in A$ and $b \in B$.

We say that $A$ and $B$ are independent if we have

$$p(a, b) = p(a)p(b),$$

for all $a$ and $b$.

To denote independence of $x_i$ and $x_j$ we use the notation

$$x_i \perp x_j.$$
Review of Independence

- For independent $a$ and $b$ we have

$$
p(a \mid b) = \frac{p(a, b)}{p(b)} = \frac{p(a)p(b)}{p(b)} = p(a).
$$

- This gives us a more intuitive definition: $A$ and $B$ are independent if

$$
p(a \mid b) = p(a)
$$

for all $a$ and $b \neq 0$.

- In words: knowing $b$ tells us nothing about $a$ (and vice versa).

- Useful fact: $a \perp b$ iff $p(a, b) = f(a)g(b)$ for some functions $f$ and $g$. 
Example: Independence in Product Models

- Let’s show independence of pairs $x_i$ and $x_j$ in product of independent models:

\[ p(x_1, x_2, \ldots, x_d) = p(x_1)p(x_2)\cdots p(x_d). \]

- From marginalization rule we have

\[ p(x_i, x_j) = \sum_{x_{-ij}} p(x_1, x_2, \ldots, x_d), \]

where $x_{-ij}$ is “over all variables except $i$ and $j$”.

- Using the definition of $p(x)$ above we get

\[ p(x_i, x_j) = \sum_{x_{-ij}} p(x_1)p(x_2)\cdots p(x_d) = p(x_i)p(x_j) \sum_{x_{-ij}} \prod_{j' \neq i, j' \neq j} p(x_{j'}) = p(x_i)p(x_j). \]

because the sum is over a joint probability distribution.
Example: Independence in Product of Bernoullis Model

- In a product of Bernoullis probabilities model we have

\[ p(x_1, x_2, \ldots, x_d) = p(x_1)p(x_2) \cdots p(x_d), \]

which we showed implies

\[ p(x_i, x_j) = p(x_i)p(x_j), \]

so we have \( x_i \perp x_j \) for all \( i \) and \( j \).

- In mixture of Bernoullis \( x_i \) is not independent of \( x_j \) (\( x_i \nparallel x_j \)):
  - Knowing \( x_j \) tells you something about \( x_i \).
  - But similar notation-heavy steps give the conditional independence that

\[ p(x_i, x_j \mid z) = p(x_i \mid z)p(x_j \mid z), \]

“variables \( x_i \) and \( x_j \) are conditionally independent given the cluster \( z \)”. 
Conditional Independence

- We say that $A$ is **conditionally independent** of $B$ **given** $C$ if
  
  $$p(a, b \mid c) = p(a \mid c)p(b \mid c),$$

  for all $a$, $b$, and $c \neq 0$.

- Equivalently, we have
  
  $$p(a \mid b, c) = p(a \mid c).$$

- “If you know $C$, then also knowing $B$ would tell you nothing about $A$”.
  - In mixture of Bernoullis, given cluster there is no dependence between variables.

- We often write this as
  
  $$A \perp B \mid C.$$  

- Most models have some sort of conditional independence.
  - They were used to simplify calculations in the EM notes.
  - They determine whether message passing is efficient.
D-Separation: From Graphs to Conditional Independence

- All conditional independences implied by a DAG can be read from the graph.

- In particular: variables $A$ and $B$ are conditionally independent given $C$ if:
  - “All paths from any variable in $A$ to any $B$ are blocked by d-separation by $C$”.

- In the special of product of independent models our graph is:

- Here there are no paths to block, which implies the variables are independent.

- Checking paths in a graph tends to be faster than tedious calculations.
  - We can start connecting properties of graphs to computational complexity.
D-Separation as Genetic Inheritance

- The rules of d-separation are intuitive in a simple model of gene inheritance:
  - Each person has a single number, which we'll call a "gene".
  - If you have no parents, your gene is a random number.
  - If you have parents, your gene is a sum of your parents plus noise.

- For example, think of something like this:

![Graph diagram](image)

- Graph corresponds to the factorization $p(x_1, x_2, x_3) = p(x_1)p(x_2)p(x_3 \mid x_1, x_2)$.
  - Are $x_1$ and $x_2$ independent here?
Genes of people are independent if knowing one says nothing about the other:
- Knowing your parent’s “gene” gives you information about your gene.
- Knowing your friend’s gene tells doesn’t say anything about your gene.

Genes of people can be conditionally independent given a third person:
- Knowing your grandparent’s gene tells you something about your gene.
- But grandparent’s gene isn’t useful if you know parent’s gene.
Are genes in person $x$ independent of the genes in person $y$?

- **No path:** $x$ and $y$ are not related (independent),
  
  We have $x \perp y$: there are no paths to be blocked.

- **Direct link:** $x$ is the parent of $y$,
  
  We have $x \not\perp y$: knowing $x$ tells you about $y$ (direct paths aren’t blockable).
D-Separation Case 0 (No Paths and Direct Links)

Neither case changes if we have a third independent person $z$:

- **No path:** If $x$ and $y$ are independent,
  
  ![Diagram showing independence](image)

  We have $x \perp y$: adding $z$ doesn’t make a path.

- **Direct link:** $x$ is the parent of $y$,

  ![Diagram showing direct link](image)

  We have $x \not\perp y \mid z$: adding $z$ doesn’t block path.

  - We use **black or shaded** nodes to denote values we condition on (in this case $z$).
D-Separation Case 1: Chain

Case 1: \( x \) is the grandparent of \( y \).
- If \( z \) is the mother we have:
  - We have \( x \not\perp y \): knowing \( x \) would give information about \( y \) because of \( z \)
- But if \( z \) is observed:
  - In this case \( x \perp y \mid z \): knowing \( z \) “breaks” dependence between \( x \) and \( y \).
D-Separation Case 1: Chain

- Consider weird case where parents $z_1$ and $z_2$ share parent $x$:
  - If $z_1$ and $z_2$ are observed we have:

$$x \perp y \mid z_1, z_2$$

- But if only $z_1$ is observed:

$$x \not\perp y \mid z_1$$

We have $x \perp y \mid z_1, z_2$: knowing both parents breaks dependency.
We have $x \not\perp y \mid z_1$: dependence still “flows” through $z_2$. 
Case 2: \( x \) and \( y \) are siblings.

- If \( z \) is a common unobserved parent:

\[ x \perp y \text{ if } z \text{ is observed.} \]

We have \( x \not\perp y \): knowing \( x \) would give information about \( y \).

- But if \( z \) is observed:

\[ x \perp y \mid z \text{ if } z \text{ is observed.} \]

In this case \( x \perp y \mid z \): knowing \( z \) “breaks” dependence between \( x \) and \( y \).
Case 2: \( x \) and \( y \) are siblings.

- If \( z_1 \) and \( z_2 \) are common observed parents:

  We have \( x \perp y \mid z_1, z_2 \): knowing \( z_1 \) and \( z_2 \) breaks dependence between \( x \) and \( y \).

- But if we only observe \( z_2 \):

  Then we have \( x \not\perp y \mid z_2 \): dependence still “flows” through \( z_1 \).
D-Separation Case 3: Common Child

- **Case 3**: \( x \) and \( y \) share a child \( z \):
  - If we observe \( z \) then we have:
    
    We have \( x \perp y \mid z \): if we know \( z \), then knowing \( x \) gives us information about \( y \).
  - But if \( z \) is not observed:

    We have \( x \perp y \): if you don’t observe \( z \) then \( x \) and \( y \) are independent.

- **Different from Case 1 and Case 2**: not observing the child blocks path.
D-Separation Case 3: Common Child

- Case 3: $x$ and $y$ share a child $z_1$:
  - If there exists an unobserved grandchild $z_2$:
    - We have $x \perp y$: the path is still blocked by not knowing $z_1$ or $z_2$.
  - But if $z_2$ is observed:
    - We have $x \not\perp y \mid z_2$: grandchild creates dependence even with unobserved parent.

- Case 3 needs to consider descendants of child.
We say that \( A \) and \( B \) are d-separated if for all paths \( P \) from \( A \) to \( B \), at least one of the following holds:

1. \( P \) includes a “chain” with an observed middle node:
   ![Chain Diagram]

2. \( P \) includes a “fork” with an observed parent node:
   ![Fork Diagram]

3. \( P \) includes a “v-structure” or “collider”:
   ![V-Structure Diagram]

where child and all its descendants are unobserved.
Summary

- **Message-passing** allow efficient calculations with Markov chains.

- **DAG models** factorize joint distribution into product of conditionals.
  - Assume conditionals depend on small number “parents”.
  - Joint distribution of models we’ve discussed can be written as DAG models.

- **Conditional independence** of $A$ and $B$ given $C$:
  - Knowing $B$ tells us nothing about $A$ if we already know $C$.

- **D-separation** allows us to test conditional independences based on graph.

- Next time: the IID assumption as a graphical model?
Computing Conditional 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 \( \max_x | x_j' = c \ p(x) \):
     - Run Viterbi decoding with \( M_{j'}(c) = 1 \) and \( M_{j'}(c') = 0 \) for \( c \neq c' \).
  3. We can find univariate conditionals, \( p(x_j | x_{j'}) \).

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

\[
p(x_1 = c | x_3 = 1) \propto p(x_1 = c, x_3 = 1) = \sum_{x_4} \sum_{x_2} p(x_1 = c, x_2, x_3 = 1, x_4),
\]

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:

\[
\sum_{x_4} \sum_{x_2} p(x_1 = c, x_2, x_3 = 1, x_4) = \sum_{x_4} \sum_{x_3=1} \sum_{x_2} \sum_{x_1=c} p(x_4 | x_3)p(x_3 | x_2)p(x_2 | x_1)p(x_1)
\]

\[
= \sum_{x_4} \sum_{x_3=1} \sum_{x_2} p(x_4 | x_3)p(x_3 | x_2) \sum_{x_1=c} p(x_2 | x_1)p(x_1)
\]

\[
= \sum_{x_4} \sum_{x_3=1} p(x_4 | x_3) \sum_{x_2} p(x_3 | x_2) \sum_{x_1=c} p(x_2 | x_1)M_1(x_1)
\]

\[
= \sum_{x_4} \sum_{x_3=1} p(x_4 | x_3) \sum_{x_2} p(x_3 | x_2)M_2(x_2)
\]

\[
= \sum_{x_4} \sum_{x_3=1} p(x_4 | x_3)M_3(x_3)
\]

\[
= \sum_{x_4} M_4(x_4),
\]

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.

\[
\sum_{x_4} \sum_{x_2} p(x_1 = c, x_2, x_3 = 1, x_4) = \sum_{x_1 = c} \sum_{x_2} \sum_{x_3 = 1} \sum_{x_4} p(x_4 | x_3)p(x_3 | x_2)p(x_2 | x_1)p(x_1)
\]

\[
= \sum_{x_1 = c} p(x_1) \sum_{x_2} p(x_2 | x_1) \sum_{x_3 = 1} p(x_3 | x_2) \sum_{x_4} p(x_4 | x_3)
\]

\[
= \sum_{x_1 = c} p(x_1) \sum_{x_2} p(x_2 | x_1) \sum_{x_3 = 1} p(x_3 | x_2) V_4(x_4)
\]

\[
= \sum_{x_1 = c} p(x_1) \sum_{x_2} p(x_2 | x_1) V_3(x_3)
\]

\[
= \sum_{x_1 = c} p(x_1) V_2(x_2)
\]

\[
= \sum_{x_1 = c} p(x_1) V_1(x_1).
\]
Directed Acyclic Graphical Models

**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?
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 \( V_{j'}(c') = 1 \).

(Other \( M_{j'} \) and \( V_{j'} \) are set to 0)

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,

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

which lets us run ancestral sampling in reverse: sample $x_d$ from $p(x_d)$, then $x_{d-1}$ from $p(x_{d-1} | x_d)$, 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.
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.