# CPSC 540: Machine Learning

## Message Passing, Directed Acyclic Graphical Models

Mark Schmidt

University of British Columbia

Winter 2017

# Admin

- Assignment 3:
  - 1 late day to hand in today, 2 for Monday.
- Assignment 4:
  - Due March 20.
- For graduate students planning to graduate in May:
  - Send me a private message on Piazza ASAP.

# Last Time: Monte Carlo Methods

- If we want to approximate expectations of random functions,

$$\mathbb{E}[g(X)] = \underbrace{\sum_{x \in \mathcal{X}} g(x)p(x)}_{\text{discrete } x} \quad \text{or} \quad \underbrace{\mathbb{E}[g(X)] = \int_{x \in \mathcal{X}} g(x)p(x)dx}_{\text{continuous } x},$$

the Monte Carlo estimate is

$$\mathbb{E}[g(X)] \approx \frac{1}{n} \sum_{i=1}^{n} g(x^i),$$

# Last Time: Monte Carlo Methods

- If we want to approximate expectations of random functions,

$$\underbrace{\mathbb{E}[g(X)] = \sum_{x \in \mathcal{X}} g(x)p(x)}_{\text{discrete } x} \quad \text{or} \quad \underbrace{\mathbb{E}[g(X)] = \int_{x \in \mathcal{X}} g(x)p(x)dx}_{\text{continuous } x},$$

the Monte Carlo estimate is

$$\mathbb{E}[g(X)] \approx \frac{1}{n}\sum_{i=1}^{n} g(x^i),$$

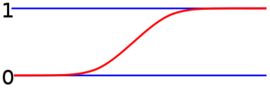
where the $x^i$ are independent samples from $p(x)$.

- We can use this to approximate marginals,

$$p(x_j = c) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{I}_{x_j^i = c}.$$

# Last Time: Inverse Transform Sampling Method

- The cumulative distribution function (CDF) $F$ is $p(X \leq x)$.
  - $F(x)$ is between $0$ and $1$ a gives proportion of times $X$ is below $x$.

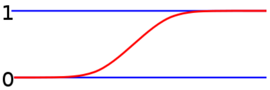



https://en.wikipedia.org/wiki/Cumulative_distribution_function

- The inverse CDF (or quantile function) $F^{-1}$ is its inverse:
  - Given a number $u$ between $0$ and $1$, gives $x$ such that $p(X \leq x) = u$.

# Last Time: Inverse Transform Sampling Method

- The cumulative distribution function (CDF) $F$ is $p(X \leq x)$.
  - $F(x)$ is between $0$ and $1$ a gives proportion of times $X$ is below $x$.




https://en.wikipedia.org/wiki/Cumulative_distribution_function

- The inverse CDF (or quantile function) $F^{-1}$ is its inverse:
  - Given a number $u$ between $0$ and $1$, gives $x$ such that $p(X \leq x) = u$.

- Inverse transfrom method for exact sampling in 1D:
  1. Sample $u \sim \mathcal{U}(0, 1)$.
  2. Compute $x = F^{-1}(u)$.

# Last Time: Markov Chains

- We can use Markov chains for density estimation,

$$p(x) = \underbrace{p(x_1)}_{\text{initial prob.}} \prod_{j=2}^{d} \underbrace{p(x_j | x_{j-1})}_{\text{transition prob.}},$$

which model dependency between adjacent features.

# Last Time: Markov Chains

- We can use Markov chains for density estimation,

$$p(x) = \underbrace{p(x_1)}_{\text{initial prob.}} \prod_{j=2}^{d} \underbrace{p(x_j | x_{j-1})}_{\text{transition prob.}},$$

  which model dependency between adjacent features.

- Homogeneous chains use same transition probability for all $j$ (parameter tieing).
  - Gives more data to estimate transitions, allows examples of different sizes.
- Inhomogeneous chains allow different transitions at different times.

# Last Time: Markov Chains

- We can use Markov chains for density estimation,

$$p(x) = \underbrace{p(x_1)}_{\text{initial prob.}} \prod_{j=2}^{d} \underbrace{p(x_j | x_{j-1})}_{\text{transition prob.}},$$

  which model dependency between adjacent features.

- Homogeneous chains use same transition probability for all $j$ (parameter tieing).
  - Gives more data to estimate transitions, allows examples of different sizes.
- Inhomogeneous chains allow different transitions at different times.

- Ancestral sampling from a Markov chain:
  - Sample $x_1$, then $x_2$ given $x_1$, then $x_3$ given $x_2$, and so on.

# Last Time: Chapman-Kolmogorov Equations

- We can compute marginals like $p(x_j = c)$ recursively in a Markov chain,

$$p(x_j) = \sum_{x_{j-1}} p(x_j|x_{j-1})p(x_{j-1}),$$

# Last Time: Chapman-Kolmogorov Equations

- We can compute marginals like $p(x_j = c)$ recursively in a Markov chain,

$$p(x_j) = \sum_{x_{j-1}} p(x_j|x_{j-1})p(x_{j-1}),$$
$$p(x_j) = \int_{x_{j-1}} p(x_j|x_{j-1})p(x_{j-1}) = \int_{x_{j-1}} p(x_j, x_{j-1}),$$

  which are called the Chapman-Kolmogorov equations.

- Yields closed-form solutions for marginals in discrete or Gaussian Markov chains.

# Outline

# Decoding: Maximizing Joint Probability

- The decoding problem in density models is finding most probable $x$:

$$\underset{x}{\operatorname{argmax}}\, p(x).$$

- For example, for binary $x$ we might have $\begin{bmatrix}1 & 0 & 0 & 1\end{bmatrix}$.

# Decoding: Maximizing Joint Probability

- The decoding problem in density models is finding most probable $x$:

$$\operatorname*{argmax}_x p(x).$$

- For example, for binary $x$ we might have $\begin{bmatrix}1 & 0 & 0 & 1\end{bmatrix}$.

- For independent models this is easy:
  - The log-likelihood is separable so we can optimize each $x_j$ independently.

# Decoding: Maximizing Joint Probability

- The decoding problem in density models is finding most probable $x$:

$$\underset{x}{\operatorname{argmax}}\, p(x).$$

- For example, for binary $x$ we might have $\begin{bmatrix}1 & 0 & 0 & 1\end{bmatrix}$.

- For independent models this is easy:
  - The log-likelihood is separable so we can optimize each $x_j$ independently.
  - For example, with three variables we have

$$\max_{x_1,x_2,x_3,x_4}\{p(x_1)p(x_2)p(x_3)p(x_4)\} = \left(\max_{x_1} p(x_1)\right)\left(\max_{x_2} p(x_2)\right)\left(\max_{x_3} p(x_3)\right)\left(\max_{x_4} p(x_4)\right).$$

- Can we also maximize the marginals to decode a Markov chain?

# Decoding vs. Maximizing Marginals

- Consider the "plane of doom" 2-variable Markov chain:
  - Initial probabilities are given by

$$p(x_1 = \text{"land"}) = 0.4, \quad p(x_1 = \text{"crash"}) = 0.3, \quad p(x_1 = \text{"explode"}) = 0.3,$$

# Decoding vs. Maximizing Marginals

- Consider the "plane of doom" 2-variable Markov chain:
  - Initial probabilities are given by

$$p(x_1 = \text{“land”}) = 0.4, \quad p(x_1 = \text{“crash”}) = 0.3, \quad p(x_1 = \text{“explode”}) = 0.3,$$

and the transition probabilities are such that

$$x_2 = \begin{cases} \text{“alive”} & \text{If } x_1 = \text{“land”} \\ \text{“dead”} & \text{otherwise} \end{cases}$$

# Decoding vs. Maximizing Marginals

- Consider the "plane of doom" 2-variable Markov chain:
  - Initial probabilities are given by

$$p(x_1 = \text{"land"}) = 0.4, \quad p(x_1 = \text{"crash"}) = 0.3, \quad p(x_1 = \text{"explode"}) = 0.3,$$

    and the transition probabilities are such that

$$x_2 = \begin{cases} \text{"alive"} & \text{If } x_1 = \text{"land"} \\ \text{"dead"} & \text{otherwise} \end{cases}$$

  - The decoding is given by ("land", "alive"), which has probability $0.4$.

# Decoding vs. Maximizing Marginals

- Consider the "plane of doom" 2-variable Markov chain:
  - Initial probabilities are given by

$$p(x_1 = \text{“land”}) = 0.4, \quad p(x_1 = \text{“crash”}) = 0.3, \quad p(x_1 = \text{“explode”}) = 0.3,$$

  and the transition probabilities are such that

$$x_2 = \begin{cases} \text{“alive”} & \text{If } x_1 = \text{“land”} \\ \text{“dead”} & \text{otherwise} \end{cases}$$

  - The decoding is given by (“land”, “alive”), which has probability $0.4$.

  - The marginals for the different values of $x_2$ are given by

$$p(x_2 = \text{“alive”}) = 0.4, \quad p(x_2 = \text{“dead”}) = 0.6,$$

  so maximizing the marginals gives (“land”, “dead”) which has probability $0$.

# Distributing Max across Product

- Note that decoding can't be done forward in time as in CK equations.
  - Even if $p(x_0 = 2) = 0.99$, the most likely sequence could have $x_0 = 1$.
  - We need to optimize over all $k^d$ length-$d$ paths.

# Distributing Max across Product

- Note that decoding can't be done forward in time as in CK equations.
  - Even if $p(x_0 = 2) = 0.99$, the most likely sequence could have $x_0 = 1$.
  - We need to optimize over all $k^d$ length-$d$ paths.

- Fortunately, the Markov property makes the max simplify:

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

# Distributing Max across Product

- Note that decoding can't be done forward in time as in CK equations.
  - Even if $p(x_0 = 2) = 0.99$, the most likely sequence could have $x_0 = 1$.
  - We need to optimize over all $k^d$ length-$d$ paths.

- Fortunately, the Markov property makes the max simplify:

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

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

# Distributing Max across Product

- Note that decoding can't be done forward in time as in CK equations.
  - Even if $p(x_0 = 2) = 0.99$, the most likely sequence could have $x_0 = 1$.
  - We need to optimize over all $k^d$ length-$d$ paths.

- Fortunately, the Markov property makes the max simplify:

$$\begin{aligned}
\max_{x_1,x_2,x_3,x_4} p(x) &= \max_{x_1,x_2,x_3,x_4} p(x_4|x_3)p(x_3|x_2)p(x_2|x_1)p(x_1) \\
&= \max_{x_4}\max_{x_3}\max_{x_2}\max_{x_1} p(x_4|x_3)p(x_3|x_2)p(x_2|x_1)p(x_1) \\
&= \max_{x_4}\max_{x_3}\max_{x_2} p(x_4|x_3)p(x_3|x_2) \max_{x_1} p(x_2|x_1)p(x_1)
\end{aligned}$$

where we're using that $\max_i \alpha a_i = \alpha \max_i a_i$ for non-negative $\alpha$.

# Distributing Max across Product

- Note that decoding can't be done forward in time as in CK equations.
  - Even if $p(x_0 = 2) = 0.99$, the most likely sequence could have $x_0 = 1$.
  - We need to optimize over all $k^d$ length-$d$ paths.

- Fortunately, the Markov property makes the max simplify:

$$\begin{aligned}
\max_{x_1,x_2,x_3,x_4} p(x) &= \max_{x_1,x_2,x_3,x_4} p(x_4|x_3)p(x_3|x_2)p(x_2|x_1)p(x_1) \\
&= \max_{x_4}\max_{x_3}\max_{x_2}\max_{x_1} p(x_4|x_3)p(x_3|x_2)p(x_2|x_1)p(x_1) \\
&= \max_{x_4}\max_{x_3}\max_{x_2} p(x_4|x_3)p(x_3|x_2)\max_{x_1} p(x_2|x_1)p(x_1) \\
&= \max_{x_4}\max_{x_3} p(x_4|x_3)\max_{x_2} p(x_3|x_2)\max_{x_1} p(x_2|x_1)p(x_1),
\end{aligned}$$

where we're using that $\max_i \alpha a_i = \alpha \max_i a_i$ for non-negative $\alpha$.

# Decoding with Memoization

- The Markov property writes decoding as a sequence of max problems:

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

but note that we can't just "solve" $\max_{x_1}$ once because it depends on $x_2$.

# Decoding with Memoization

- The Markov property writes decoding as a sequence of max problems:

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

but note that we can't just "solve" $\max_{x_1}$ once because it depends on $x_2$.

  - Instead, we'll memoize solution $M_2(x_2) = \max_{x_1} p(x_2|x_1)p(x_1)$ for all $x_2$,

$$\max_{x_1,x_2,x_3,x_4} p(x) = \max_{x_4} \max_{x_3} p(x_4|x_3) \max_{x_2} p(x_3|x_2) M_2(x_2).$$

# Decoding with Memoization

- The Markov property writes decoding as a sequence of max problems:

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

  but note that we can't just "solve" $\max_{x_1}$ once because it depends on $x_2$.
  - Instead, we'll memoize solution $M_2(x_2) = \max_{x_1} p(x_2|x_1)p(x_1)$ for all $x_2$,

$$\max_{x_1,x_2,x_3,x_4} p(x) = \max_{x_4} \max_{x_3} p(x_4|x_3) \max_{x_2} p(x_3|x_2) M_2(x_2).$$

- Now we memoize solution $M_3(x_3) = \max_{x_2} p(x_3|x_2)M_2(x_2)$ for all $x_2$,

$$\max_{x_1,x_2,x_3,x_4} p(x) = \max_{x_4} \max_{x_3} p(x_4|x_3) M_3(x_3).$$

# Decoding with Memoization

- The Markov property writes decoding as a sequence of max problems:

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

but note that we can't just "solve" $\max_{x_1}$ once because it depends on $x_2$.

  - Instead, we'll memoize solution $M_2(x_2) = \max_{x_1} p(x_2|x_1)p(x_1)$ for all $x_2$,

$$\max_{x_1,x_2,x_3,x_4} p(x) = \max_{x_4}\max_{x_3} p(x_4|x_3)\max_{x_2} p(x_3|x_2)M_2(x_2).$$

- Now we memoize solution $M_3(x_3) = \max_{x_2} p(x_3|x_2)M_2(x_2)$ for all $x_2$,

$$\max_{x_1,x_2,x_3,x_4} p(x) = \max_{x_4}\max_{x_3} p(x_4|x_3)M_3(x_3).$$

- And defining $M_4(x_4) = \max_{x_3} p(x_4|x_3)M_2(x_3)$ the maximum value is given by

$$\max_{x_1,x_2,x_3,x_4} p(x) = \max_{x_4} M_4(x_4).$$

# Decoding Example

- Consider the "plane of doom" 2-variable Markov chain:
  - Initial probabilities are given by

$$p(x_1 = \text{"land"}) = 0.4, \quad p(x_1 = \text{"crash"}) = 0.3, \quad p(x_1 = \text{"explode"}) = 0.3,$$

and $x_2 =$"alive" if we land and is "dead" otherwise.

# Decoding Example

- Consider the "plane of doom" 2-variable Markov chain:
  - Initial probabilities are given by

    $$p(x_1 = \text{"land"}) = 0.4, \quad p(x_1 = \text{"crash"}) = 0.3, \quad p(x_1 = \text{"explode"}) = 0.3,$$

    and $x_2 =$"alive" if we land and is "dead" otherwise.
  - We have $M_1(x_1) = p(x_1)$ so we get

    $$M_1(\text{"land"}) = 0.4, \quad M_1(\text{"crash"}) = 0.3, \quad M_1(\text{"explode"}) = 0.3.$$

# Decoding Example

- Consider the "plane of doom" 2-variable Markov chain:
  - Initial probabilities are given by

    $$p(x_1 = \text{"land"}) = 0.4, \quad p(x_1 = \text{"crash"}) = 0.3, \quad p(x_1 = \text{"explode"}) = 0.3,$$

    and $x_2 =$"alive" if we land and is "dead" otherwise.
  - We have $M_1(x_1) = p(x_1)$ so we get

    $$M_1(\text{"land"}) = 0.4, \quad M_1(\text{"crash"}) = 0.3, \quad M_1(\text{"explode"}) = 0.3.$$
  - We have $M_2(x_2) = \max_{x_2} p(x_2|x_1)M_1(x_1)$ so we get

    $$M_2(\text{"alive"}) = 0.4, \quad M_2(\text{"dead"}) = 0.3.$$

# Decoding Example

- Consider the "plane of doom" 2-variable Markov chain:
  - Initial probabilities are given by

$$p(x_1 = \text{“land”}) = 0.4, \quad p(x_1 = \text{“crash”}) = 0.3, \quad p(x_1 = \text{“explode”}) = 0.3,$$

    and $x_2 =$"alive" if we land and is "dead" otherwise.

  - We have $M_1(x_1) = p(x_1)$ so we get

$$M_1(\text{“land”}) = 0.4, \quad M_1(\text{“crash”}) = 0.3, \quad M_1(\text{“explode”}) = 0.3.$$

  - We have $M_2(x_2) = \max_{x_2} p(x_2|x_1)M_1(x_1)$ so we get

$$M_2(\text{“alive”}) = 0.4, \quad M_2(\text{“dead”}) = 0.3.$$

  - This means the optimal decoding has probability $0.4$ and ends with "alive".
    - We now need need to backtrack to find the state that lead to "alive", giving "land".

# Vitebi Decoding

- What is $M_j(x_j)$ in words?
  - "Probability of most likely length-$j$ sequence ending in $x_j$ (ignoring future)".

# Vitebi Decoding

- What is $M_j(x_j)$ in words?
  - "Probability of most likely length-$j$ sequence ending in $x_j$ (ignoring future)".
- The Viterbi decoding algorithm (special case of dynamic programming):
  1. Set $M_1(x_1) = p(x_1)$ for all $x_1$.

# Vitebi Decoding

- What is $M_j(x_j)$ in words?
  - "Probability of most likely length-$j$ sequence ending in $x_j$ (ignoring future)".

- The Viterbi decoding algorithm (special case of dynamic programming):
  1. Set $M_1(x_1) = p(x_1)$ for all $x_1$.
  2. Compute $M_2(x_2)$ for all $x_2$, and store the $x_1$ leading to each $x_2$ in $D_2(x_2)$.

# Vitebi Decoding

- What is $M_j(x_j)$ in words?
  - "Probability of most likely length-$j$ sequence ending in $x_j$ (ignoring future)".

- The Viterbi decoding algorithm (special case of dynamic programming):
  1. Set $M_1(x_1) = p(x_1)$ for all $x_1$.
  2. Compute $M_2(x_2)$ for all $x_2$, and store the $x_1$ leading to each $x_2$ in $D_2(x_2)$.
  3. Compute $M_3(x_3)$ for all $x_3$, and store the $x_2$ leading to each $x_3$ in $D_3(x_3)$.
  4. ...

# Vitebi Decoding

- What is $M_j(x_j)$ in words?
  - "Probability of most likely length-$j$ sequence ending in $x_j$ (ignoring future)".

- The Viterbi decoding algorithm (special case of dynamic programming):
  1. Set $M_1(x_1) = p(x_1)$ for all $x_1$.
  2. Compute $M_2(x_2)$ for all $x_2$, and store the $x_1$ leading to each $x_2$ in $D_2(x_2)$.
  3. Compute $M_3(x_3)$ for all $x_3$, and store the $x_2$ leading to each $x_3$ in $D_3(x_3)$.
  4. ...
  5. Maximize $M_d(x_d)$ to find $x_d$ in decoding, bactrack with $D_j$'s to find decoding.

# Vitebi Decoding

- What is $M_j(x_j)$ in words?
  - "Probability of most likely length-$j$ sequence ending in $x_j$ (ignoring future)".

- The Viterbi decoding algorithm (special case of dynamic programming):
  1. Set $M_1(x_1) = p(x_1)$ for all $x_1$.
  2. Compute $M_2(x_2)$ for all $x_2$, and store the $x_1$ leading to each $x_2$ in $D_2(x_2)$.
  3. Compute $M_3(x_3)$ for all $x_3$, and store the $x_2$ leading to each $x_3$ in $D_3(x_3)$.
  4. ...
  5. Maximize $M_d(x_d)$ to find $x_d$ in decoding, bactrack with $D_j$'s to find decoding.

- Computing all $M_j(x_j)$ given all $M_{j-1}(x_{j-1})$ costs $O(k^2)$.
  - Total cost is only $O(dk^2)$ to search over all $k^d$ paths.

# Conditional Probabilities

- We often want to compute conditional probabilities in Markov chains.
  - We can ask "what lead to $x_{10} = 4$?" with queries like $p(x_1|x_{10})$.
  - We can ask "where does $x_{10} = 4$ lead?" with queries like $p(x_d|x_{10})$.

# Conditional Probabilities

- We often want to compute conditional probabilities in Markov chains.
  - We can ask "what lead to $x_{10} = 4$?" with queries like $p(x_1|x_{10})$.
  - We can ask "where does $x_{10} = 4$ lead?" with queries like $p(x_d|x_{10})$.

- Monte Carlo approach to estimating $p(x_j|x_{j'})$:
  1. Generate a large number of samples from the Markov chain, $x^i \sim p(x_1, x_2, \dots, x_d)$.
  2. Use Monte Carlo estimates of $p(x_j, x_{j'} = c)$ and $p(x_{j'} = c)$ to give

$$p(x_j|x_{j'} = c) = \frac{p(x_j, x_{j'} = c)}{p(x_{j'} = c)}.$$

# Conditional Probabilities

- We often want to compute conditional probabilities in Markov chains.
  - We can ask "what lead to $x_{10} = 4$?" with queries like $p(x_1|x_{10})$.
  - We can ask "where does $x_{10} = 4$ lead?" with queries like $p(x_d|x_{10})$.

- Monte Carlo approach to estimating $p(x_j|x_{j'})$:
  1. Generate a large number of samples from the Markov chain, $x^i \sim p(x_1, x_2, \dots, x_d)$.
  2. Use Monte Carlo estimates of $p(x_j, x_{j'} = c)$ and $p(x_{j'} = c)$ to give

$$p(x_j|x_{j'} = c) = \frac{p(x_j, x_{j'} = c)}{p(x_{j'} = c)}.$$

- This is a special case of rejection sampling (we'll see general case later).
  - Unfortunately, if $x_{j'} = c$ is rare then most samples are "rejected" (ignored).

# 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.

# 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 $c \neq c'$.

# 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, $p(x_j|x_{j'})$.

# 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:

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

# 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:

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

  where the normalizing constant is $p(x_3 = 1)$.

- 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:

$$\sum_{x_4}\sum_{x_2} p(x_1 = c, x_2, x_3 = 1, x_4) = \sum_{x4}\sum_{x3=1}\sum_{x_2}\sum_{x_1=c} p(x_4|x_3)p(x_3|x_2)p(x_2|x_1)p(x_1)$$

# Distributing Sum across Product

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

$$\begin{aligned}\sum_{x_4}\sum_{x_2} p(x_1 = c, x_2, x_3 = 1, x_4) &= \sum_{x4}\sum_{x3=1}\sum_{x_2}\sum_{x_1=c} p(x_4|x_3)p(x_3|x_2)p(x_2|x_1)p(x_1) \\ &= \sum_{x4}\sum_{x3=1}\sum_{x_2} p(x_4|x_3)p(x_3|x_2)\sum_{x_1=c} p(x_2|x_1){\color{green}p(x_1)}\end{aligned}$$

# Distributing Sum across Product

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

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

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

# Distributing Sum across Product

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

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

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

# Distributing Sum across Product

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

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

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

# Distributing Sum across Product

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

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

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

# Message-Passing Algorithms

- We've just discussed several algorithms with similar structure:
  - CK equations for computing univariate marginals in discrete Markov chains.
  - Recursive marginal updates for Gaussian Markov chains (Assignment 4).
  - Viterbi decoding algorithm for discrete Markov chains.
  - Conditional inference in discrete Markov chains.

# Message-Passing Algorithms

- We've just discussed several algorithms with similar structure:
  - CK equations for computing univariate marginals in discrete Markov chains.
  - Recursive marginal updates for Gaussian Markov chains (Assignment 4).
  - Viterbi decoding algorithm for discrete Markov chains.
  - Conditional inference in discrete Markov chains.

- These are all special cases of message-passing algorithms:
  1. Define $M_j$ summarizing all relevant information about the past at time $j$.

# Message-Passing Algorithms

- We've just discussed several algorithms with similar structure:
  - CK equations for computing univariate marginals in discrete Markov chains.
  - Recursive marginal updates for Gaussian Markov chains (Assignment 4).
  - Viterbi decoding algorithm for discrete Markov chains.
  - Conditional inference in discrete 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 $M_1$, $M_2$, $\ldots$, $M_d$.

# Message-Passing Algorithms

- We've just discussed several algorithms with similar structure:
  - CK equations for computing univariate marginals in discrete Markov chains.
  - Recursive marginal updates for Gaussian Markov chains (Assignment 4).
  - Viterbi decoding algorithm for discrete Markov chains.
  - Conditional inference in discrete 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 $M_1$, $M_2$, . . . , $M_d$.

- 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

- Performing our conditional calculation using backwards messages.

$$\sum_{x_4}\sum_{x_2} p(x_1 = c, x_2, x_3 = 1, x_4) = \sum_{x_1=c}\sum_{x_2}\sum_{x3=1}\sum_{x4} p(x_4|x_3)p(x_3|x_2)p(x_2|x_1)p(x_1)$$

# Conditionals via Backwards Messages

- Performing our conditional calculation using backwards messages.

$$\sum_{x_4} \sum_{x_2} p(x_1 = c, x_2, x_3 = 1, x_4) = \sum_{x_1=c} \sum_{x_2} \sum_{x3=1} \sum_{x4} p(x_4|x_3)p(x_3|x_2)p(x_2|x_1)p(x_1)$$

$$= \sum_{x_1=c} p(x_1) \sum_{x_2} p(x_2|x_1) \sum_{x3=1} p(x_3|x_2) \sum_{x4} p(x_4|x_3)$$

# Conditionals via Backwards Messages

- Performing our conditional calculation using backwards messages.

$$\sum_{x_4} \sum_{x_2} p(x_1 = c, x_2, x_3 = 1, x_4) = \sum_{x_1=c} \sum_{x_2} \sum_{x3=1} \sum_{x4} p(x_4|x_3)p(x_3|x_2)p(x_2|x_1)p(x_1)$$

$$= \sum_{x_1=c} p(x_1) \sum_{x_2} p(x_2|x_1) \sum_{x3=1} p(x_3|x_2) \sum_{x4} p(x_4|x_3)$$

$$= \sum_{x_1=c} p(x_1) \sum_{x_2} p(x_2|x_1) \sum_{x3=1} p(x_3|x_2) \sum_{x4} p(x_4|x_3) \underbrace{V_4(x_4)}_{=1}$$

# Conditionals via Backwards Messages

- Performing our conditional calculation using backwards messages.

$$\sum_{x_4}\sum_{x_2} p(x_1=c, x_2, x_3=1, x_4) = \sum_{x_1=c}\sum_{x_2}\sum_{x3=1}\sum_{x4} p(x_4|x_3)p(x_3|x_2)p(x_2|x_1)p(x_1)$$

$$= \sum_{x_1=c} p(x_1)\sum_{x_2} p(x_2|x_1)\sum_{x3=1} p(x_3|x_2)\sum_{x4} p(x_4|x_3)$$

$$= \sum_{x_1=c} p(x_1)\sum_{x_2} p(x_2|x_1)\sum_{x3=1} p(x_3|x_2)\sum_{x4} p(x_4|x_3)\underbrace{V_4(x_4)}_{=1}$$

$$= \sum_{x_1=c} p(x_1)\sum_{x_2} p(x_2|x_1)\sum_{x3=1} p(x_3|x_2)V_3(x_3)$$

# Conditionals via Backwards Messages

- Performing our conditional calculation using backwards messages.

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

# Conditionals via Backwards Messages

- Performing our conditional calculation using backwards messages.

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

# Forward-Backward Algorithm

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

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

# Forward-Backward Algorithm

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

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

- We can compute $p(x_j = c|x_{j'} = c')$ 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 $p(x_j = c | x_{j'} = c')$ for all $j$ in $O(nk^2)$ with both messages.

# Forward-Backward Algorithm

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

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

# Forward-Backward Algorithm

- We can compute $p(x_j = c | x_{j'} = c')$ for all $j$ in $O(nk^2)$ with both messages.

- 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 $x_j$ (wth $x_{j'} = c'$).
  - We can combine these values to get

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

  - Special case of the forward-backward algorithm.

# Outline

# Higher-Order Markov Models

- Markov models use a density of the form

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

- They support efficient computation but Markov assumption is strong.

# Higher-Order Markov Models

- Markov models use a density of the form

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

- They support efficient computation but Markov assumption is strong.

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

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

  or even a higher-order models.

# Higher-Order Markov Models

- Markov models use a density of the form

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

- They support efficient computation but Markov assumption is strong.

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

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

or even a higher-order models.

- Directed acyclic graphical (DAG) models generalize Markov models:
  - They allow dependence on any subset of previous features.

# DAG Models

- DAG models use product rule, $p(a, b, c) = p(b, c|a)p(a)$, to write

$$p(x_1, x_2, \dots, x_d) = p(x_1)p(x_2, x_3, \dots, x_d|x_1)$$

# DAG Models

- DAG models use product rule, $p(a,b,c) = p(b,c|a)p(a)$, to write

$$\begin{aligned} p(x_1, x_2, \dots, x_d) &= p(x_1)p(x_2, x_3, \dots, x_d | x_1) \\ &= p(x_1)p(x_2|x_1)p(x_3, x_4, \dots, x_d | x_1, x_2) \end{aligned}$$

# DAG Models

- DAG models use product rule, $p(a, b, c) = p(b, c|a)p(a)$, to write

$$\begin{aligned}
p(x_1, x_2, \dots, x_d) &= p(x_1)p(x_2, x_3, \dots, x_d|x_1) \\
&= p(x_1)p(x_2|x_1)p(x_3, x_4, \dots, x_d|x_1, x_2) \\
&= p(x_1)p(x_2|x_1)p(x_3|x_2, x_1)p(x_4, x_5, \dots, x_d|x_1, x_2, x_3),
\end{aligned}$$

# DAG Models

- DAG models use product rule, $p(a, b, c) = p(b, c|a)p(a)$, to write

$$\begin{aligned} p(x_1, x_2, \dots, x_d) &= p(x_1)p(x_2, x_3, \dots, x_d|x_1) \\ &= p(x_1)p(x_2|x_1)p(x_3, x_4, \dots, x_d|x_1, x_2) \\ &= p(x_1)p(x_2|x_1)p(x_3|x_2, x_1)p(x_4, x_5, \dots, x_d|x_1, x_2, x_3), \end{aligned}$$

  and so on until we get

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

- This factorization holds for any distribution.

# DAG Models

- DAG models use product rule, $p(a,b,c) = p(b,c|a)p(a)$, to write

$$\begin{aligned} p(x_1,x_2,\dots,x_d) &= p(x_1)p(x_2,x_3,\dots,x_d|x_1) \\ &= p(x_1)p(x_2|x_1)p(x_3,x_4,\dots,x_d|x_1,x_2) \\ &= p(x_1)p(x_2|x_1)p(x_3|x_2,x_1)p(x_4,x_5,\dots,x_d|x_1,x_2,x_3), \end{aligned}$$

  and so on until we get

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

- This factorization holds for any distribution.
- But it has too many parameters:
  - For binary $x_i$, we need $2^d$ parameters for $p(x_j|x_1,x_2,\dots,x_{j-1})$ alone.

# DAG Models

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

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

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 $(j-1)$.

# DAG Models

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

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

  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 $(j-1)$.

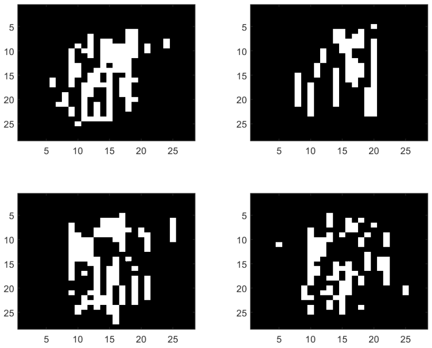
- This corresponds to a set of conditional independence assumptions,

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

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

# MNIST DIgits with Markov Chains

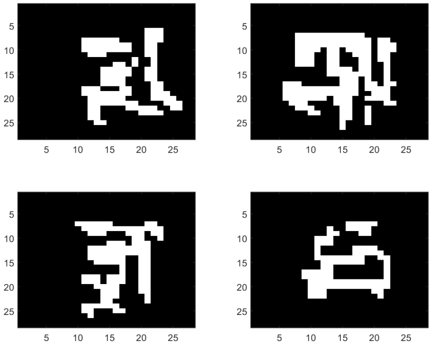
- Recall trying to model digits using an inhomogeneous Markov chain:




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 $(i-2:i, j-2:j)$:

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

# From Probability Factorizations to Graphs

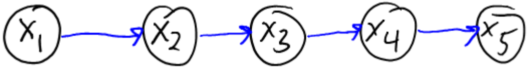
- DAG models are also known as "Bayesian networks" and "belief networks".
- "Graphical" name comes from visualizing features/parents as a graph:
  - We have a node for each variable $j$.

# 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.

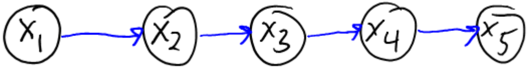
# From Probability Factorizations to Graphs

- 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:



# 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:




- 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

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

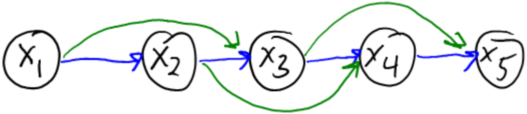
so $\text{pa}(j) = \{j - 1\}$ and the graph is:

# Graph Structure Examples

With second-order Markov chain we have

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

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

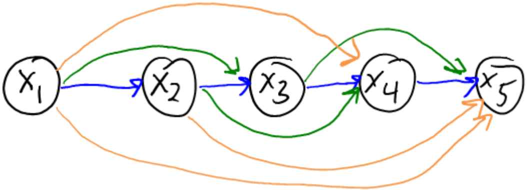


# Graph Structure Examples

With general distribution we have

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

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

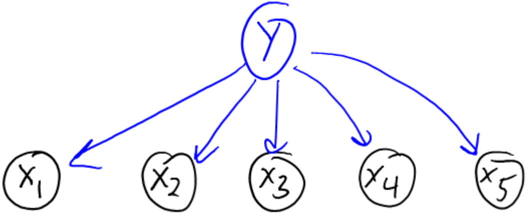


# Graph Structure Examples

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

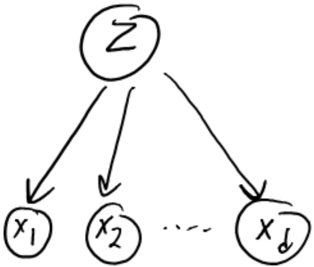


# Graph Structure Examples

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



# Graph Structure Examples

Instead of factorizing by variables $j$, could factor into blocks $b$:

$$p(x) = \prod_b p(x_b | x_{\text{pa}(b)}),$$

and have the nodes be blocks.

# Graph Structure Examples

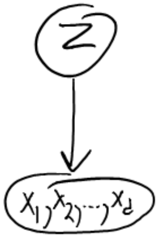
Instead of factorizing by variables $j$, could factor into blocks $b$:

$$p(x) = \prod_b p(x_b | x_{\mathsf{pa}(b)}),$$

and have the nodes be blocks. With mixture of Gaussian we have

$$p(z, x) = p(z)p(x|z).$$

The corresponding graph structure is:



# Graph Structure Examples

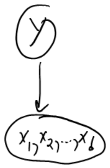
Instead of factorizing by variables $j$, could factor into blocks $b$:

$$p(x) = \prod_b p(x_b | x_{\text{pa}(b)}),$$

and have the nodes be blocks. With Gaussian generative classifier we have

$$p(y, x) = p(y)p(x|y).$$

The corresponding graph structure is:

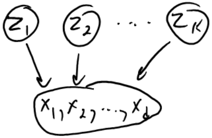


# Graph Structure Examples

With probabilistic PCA we have

$$p(z,x) = p(x|z)\prod_{c=1}^{k} p(z_c).$$

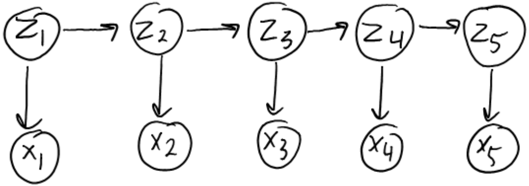
The corresponding graph structure is:



# Graph Structure Examples

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

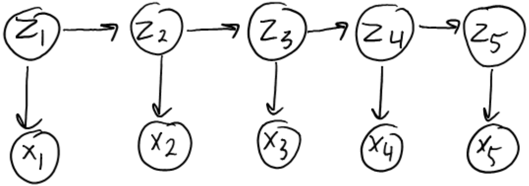
Later in the course we'll see hidden Markov models which have the structure



# Graph Structure Examples

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

Later in the course we'll see hidden Markov models which have the structure




You should already be able to get an idea of what this model does:

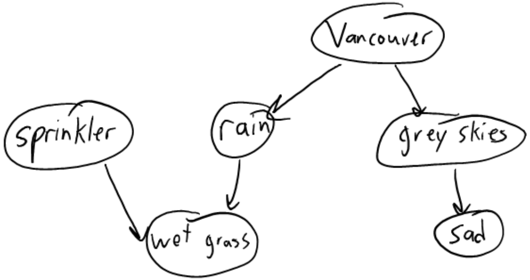
- We have hidden variables $z_j$ that follow a Markov chain.
- Each feature $x_j$ depends on corresponding feature $z_j$.

# Graph Structure Examples

We can consider less-structured examples,

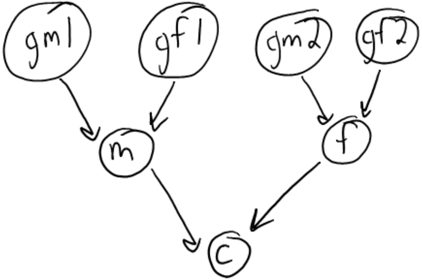
$$p(S, V, R, W, G, D) = p(S)p(V)p(R|V)p(W|S, R)p(G|V)p(D|G).$$

The corresponding graph structure is:



# Graph Structure Examples

We can consider phylogeny (family trees):



# Outline

# Review of Independence

- Let $A$ and $B$ are random variables taking values $a \in \mathcal{A}$ and $b \in \mathcal{B}$.
- We say that $A$ and $B$ are independent if we have

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

  for all $a$ and $b$.
- This is true iff $p(a, b) = f(a)g(b)$ for some functions $f$ and $g$.

# Review of Independence

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

$$x_i \perp x_j.$$

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

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

# Review of Independence

- To denote independence of $x_i$ and $x_j$ we use the notation
$$x_i \perp x_j.$$
- For independent $a$ and $b$ we have
$$p(a|b) = \frac{p(a,b)}{p(b)} = \frac{p(a)p(b)}{p(b)} = p(a).$$
- This gives us a more intuitive definition: $A$ and $B$ are independent if
$$p(a|b) = p(a)$$
for all $a$ and $b$.
  - In words: knowing $b$ tells us nothing about $a$ (and vice versa).

# Independence in "Independent Bernoulli" Model

- In a product of Bernoullis model we have

$$p(x) = \prod_{j=1}^{n} p(x_j).$$

From marginalization rule we have

$$p(x_i, x_j) = \sum_{x_{-ij}} p(x),$$

where $x_{-ij}$ is "all variables except $i$ and $j$".

# Independence in "Independent Bernoulli" Model

- In a product of Bernoullis model we have

$$p(x) = \prod_{j=1}^{n} p(x_j).$$

  From marginalization rule we have

$$p(x_i, x_j) = \sum_{x_{-ij}} p(x),$$

  where $x_{-ij}$ is "all variables except $i$ and $j$".
- Using the definition of $p(x)$ above we get

$$p(x_i, x_j) = \sum_{x_{-ij}} \prod_{j'=1}^{d} p(x_{j'})$$

# Independence in "Independent Bernoulli" Model

- In a product of Bernoullis model we have

$$p(x) = \prod_{j=1}^{n} p(x_j).$$

From marginalization rule we have

$$p(x_i, x_j) = \sum_{x_{-ij}} p(x),$$

where $x_{-ij}$ is "all variables except $i$ and $j$".

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

$$p(x_i, x_j) = \sum_{x_{-ij}} \prod_{j'=1}^{d} p(x_{j'}) = p(x_i)p(x_j) \underbrace{\sum_{x_{-ij}} \prod_{j' \neq i, j' \neq j} p(x_{j'})}_{=1}$$

# Independence in "Independent Bernoulli" Model

- In a product of Bernoullis model we have
$$p(x) = \prod_{j=1}^{n} p(x_j).$$
From marginalization rule we have
$$p(x_i, x_j) = \sum_{x_{-ij}} p(x),$$
where $x_{-ij}$ is "all variables except $i$ and $j$".
- Using the definition of $p(x)$ above we get
$$p(x_i, x_j) = \sum_{x_{-ij}} \prod_{j'=1}^{d} p(x_{j'}) = p(x_i)p(x_j) \underbrace{\sum_{x_{-ij}} \prod_{j' \neq i, j' \neq j} p(x_{j'})}_{=1} = p(x_i)p(x_j).$$
because the sum is over a joint probability distribution.

# Independence in Product of Bernoullis Model

- In a product of Bernoullis model we have

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

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 we have $x_i \not\perp x_j$:
  - Knowing $x_j$ tells you something about $x_i$.
  - But there are conditional independences in mixture of Bernoulli...

# Conditional Independence

- We say that $A$ is conditionally independent of $B$ given $C$ if

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

  for all $a$, $b$, and $c$.

- Equivalently, we have

$$p(a|b, c) = p(a|c).$$

# Conditional Independence

- We say that $A$ is conditionally independent of $B$ given $C$ if

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

  for all $a$, $b$, and $c$.

- Equivalently, we have

$$p(a|b, c) = p(a|c).$$

- "If you know $C$, then *also* knowing $B$ would tell you nothing about $A$"'.

- We often write this as

$$A \perp B \mid C.$$

# Conditional Indpendence in Mixture of Bernoulli

- In a mixture of Bernoulli model

$$p(x) = \sum_{c=1}^{k} p(z = c) \prod_{j=1}^{d} p(x_j | z = c),$$

we have that $x_i \perp x_j \mid z$ ("conditional independence given the cluster")

# Conditional Indpendence in Mixture of Bernoulli

- In a mixture of Bernoulli model

$$p(x) = \sum_{c=1}^{k} p(z = c) \prod_{j=1}^{d} p(x_j | z = c),$$

we have that $x_i \perp x_j \mid z$ ("conditional independence given the cluster")

- In particular, the same tedious notation-heavy cancellation gives that

$$p(x_i, x_j | z) = p(x_i | z) p(x_j | z).$$

- But we can also show [conditional] independencies using the graph...

# D-Separation: From Graphs to Conditional Independence

- In DAGs we make the conditional independence assumption that

$$p(x_j|x_{j-1}, x_{j-2}, \ldots, x_1) = p(x_j|x_{\text{pa}}(j)).$$

- But these conditional independence assumptions imply other assumptions.

# D-Separation: From Graphs to Conditional Independence

- In DAGs we make the conditional independence assumption that

$$p(x_j|x_{j-1}, x_{j-2}, \ldots, x_1) = p(x_j|x_{\mathsf{pa}}(j)).$$

- But these conditional independence assumptions imply other assumptions.
  - For example, in Markov chains we assume

$$p(x_j|x_{j-1}, x_{j-2}, \ldots, x_1) = p(x_j|x_{j-1}),$$

but this implies that

$$p(x_j|x_{j-2}, x_{j-3}, \ldots, x_1) = p(x_j|x_{j-2}),$$

# D-Separation: From Graphs to Conditional Independence

- In DAGs we make the conditional independence assumption that

$$p(x_j|x_{j-1}, x_{j-2}, \ldots, x_1) = p(x_j|x_{\mathsf{pa}}(j)).$$

- But these conditional independence assumptions imply other assumptions.
  - For example, in Markov chains we assume

$$p(x_j|x_{j-1}, x_{j-2}, \ldots, x_1) = p(x_j|x_{j-1}),$$

but this implies that

$$p(x_j|x_{j-2}, x_{j-3}, \ldots, x_1) = p(x_j|x_{j-2}),$$

and it implies that

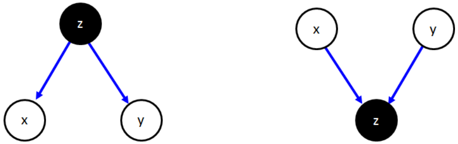
$$p(x_j|x_{j+1}, x_{j+2}, \ldots, x_d) = p(x_j|x_{j+1}).$$

# D-Separation: From Graphs to Conditional Independence

- All implied conditional independences can be read from the graph.
  - 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 $C$".

# D-Separation: From Graphs to Conditional Independence

- All implied conditional independences can be read from the graph.
  - 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 $C$".
- E.g., consider three $\{X, Y, Z\}$ variables and the following graph structure:




- We use **black or shaded** nodes to denote observed values (we condition on $Z$).
- D-separation will tell us that $X \perp Y | Z$ on the left but $X \not\perp Y | Z$ on the right.

# D-Separation: From Graphs to Conditional Indpendence

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

# D-Separation: From Graphs to Conditional Indpendence

- The rules of d-separation are intuitive in a simple model of gene inheritance:
  - Each person has single number, which we'll call a "gene".
  - If you have no parents, your gene is random.
  - If you have parents, your gene is a linear combination of your parents plus noise.
- Genes of people are independent if knowing one says nothing about the other:
  - Knowing your mom's gene tells you something about your gene (same up to noise).
  - Knowing your friend's gene tells doesn't say anything about your gene.

# D-Separation: From Graphs to Conditional Indpendence

- The rules of d-separation are intuitive in a simple model of gene inheritance:
  - Each person has single number, which we'll call a "gene".
  - If you have no parents, your gene is random.
  - If you have parents, your gene is a linear combination of your parents plus noise.
- Genes of people are independent if knowing one says nothing about the other:
  - Knowing your mom's gene tells you something about your gene (same up to noise).
  - 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 grandma's gene tells you something about your gene.
  - If you know you mom's gene, then grandma's gene isn't useful.
    - You are conditionally independent of grandma given mom.

# D-Separation Case 0 (No Paths and Direct Links)

Are genes in person $x$ independent of the genes in person $y$?

# D-Separation Case 0 (No Paths and Direct Links)

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.

# D-Separation Case 0 (No Paths and Direct Links)

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).

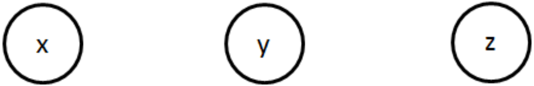
# D-Separation Case 0 (No Paths and Direct Links)

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

# D-Separation Case 0 (No Paths and Direct Links)

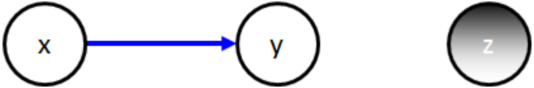
Neither case changes if we have a third independent person $z$:

- No path: If $x$ and $y$ are independent,




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

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



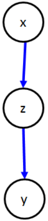

  We have $x \not\perp y$: adding $z$ doesn't block path.

## D-Separation Case 1: Chain

- Case 1: $x$ is the grandmother of $y$.

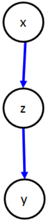
# D-Separation Case 1: Chain

- Case 1: $x$ is the grandmother of $y$.
  - If $z$ is the mother we have:



# D-Separation Case 1: Chain

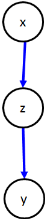
- Case 1: $x$ is the grandmother of $y$.
  - If $z$ is the mother we have:



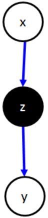

We have $x \not\perp y$: knowing $x$ would give information about $y$ because of $z$

# D-Separation Case 1: Chain

- Case 1: $x$ is the grandmother of $y$.
  - If $z$ is the mother we have:

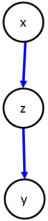



    We have $x \not\perp y$: knowing $x$ would give information about $y$ because of $z$
  - But if $z$ is *observed*:



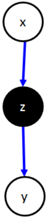
# D-Separation Case 1: Chain

- Case 1: $x$ is the grandmother 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*:



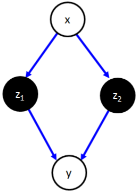

In this case $x \perp y \mid z$: knowing $z$ "breaks" dependence between $x$ and $y$.

# D-Separation Case 1: Chain

- Consider weird case where parents $z_1$ and $z_2$ share mother $x$:

# D-Separation Case 1: Chain

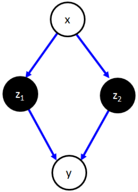
- Consider weird case where parents $z_1$ and $z_2$ share mother $x$:
    - If $z_1$ and $z_2$ are observed we have:



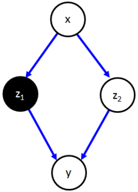

We have $x \perp y \mid z_1, z_2$: knowing both parents breaks dependency.

# D-Separation Case 1: Chain

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



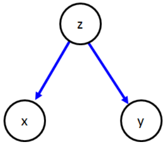

    We have $x \perp y \mid z_1, z_2$: knowing both parents breaks dependency.
  - But if only $z_1$ is *observed*:




    We have $x \not\perp y \mid z_1$: dependence still "flows" through $z_2$.

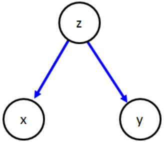
# D-Separation Case 2: Common Parent

- Case 2: $x$ and $y$ are sibilings.
  - If $z$ is a common unobserved parent:



# D-Separation Case 2: Common Parent

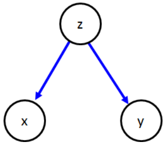
- Case 2: $x$ and $y$ are sibilings.
  - If $z$ is a common unobserved parent:



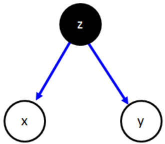

We have $x \not\perp y$: knowing $x$ would give information about $y$.

# D-Separation Case 2: Common Parent

- Case 2: $x$ and $y$ are sibilings.
  - If $z$ is a common unobserved parent:

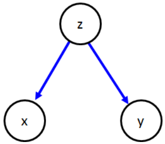



    We have $x \not\perp y$: knowing $x$ would give information about $y$.
  - But if $z$ is *observed*:

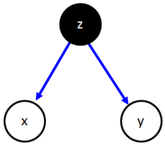


# D-Separation Case 2: Common Parent

- Case 2: $x$ and $y$ are sibilings.
  - If $z$ is a common unobserved parent:



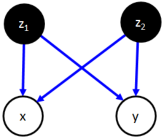

    We have $x \not\perp y$: knowing $x$ would give information about $y$.
  - But if $z$ is *observed*:




    In this case $x \perp y \mid z$: knowing $z$ "breaks" dependence between $x$ and $y$.

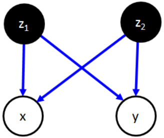
# D-Separation Case 2: Common Parent

- Case 2: $x$ and $y$ are sibilings.
  - If $z_1$ and $z_2$ are common observed parents:



# D-Separation Case 2: Common Parent

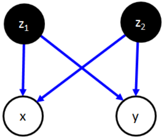
- Case 2: $x$ and $y$ are sibilings.
  - If $z_1$ and $z_2$ are common observed parents:



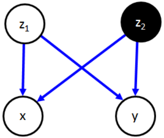

We have $x \perp y \mid z_1, z_2$: knowing $z_1$ and $z_2$ breaks dependence between $x$ and $y$.

# D-Separation Case 2: Common Parent

- Case 2: $x$ and $y$ are sibilings.
  - 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 $z_2$:



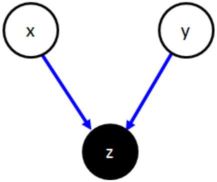

    Then we have $x \not\perp y \mid z_2$: dependence still "flows" through $z_1$.

# D-Separation Case 3: Common Child

- Case 3: $x$ and $y$ share a child $z$:

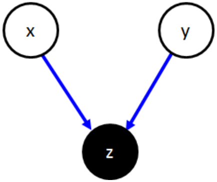
# D-Separation Case 3: Common Child

- Case 3: $x$ and $y$ share a child $z$:
  - If we observe $z$ then we have:



# D-Separation Case 3: Common Child

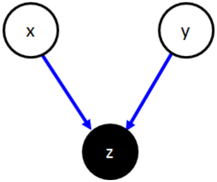
- Case 3: $x$ and $y$ share a child $z$:
  - If we observe $z$ then we have:




We have $x \not\perp y \mid z$: if we know $z$, then knowing $x$ gives us information about $y$.

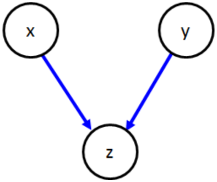
# D-Separation Case 3: Common Child

- Case 3: $x$ and $y$ share a child $z$:
  - If we observe $z$ then we have:



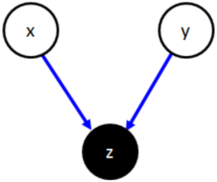

We have $x \not\perp y \mid z$: if we know $z$, then knowing $x$ gives us information about $y$.

  - But if $z$ is not observed:

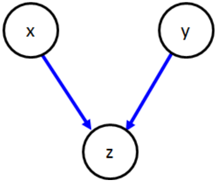


# D-Separation Case 3: Common Child

- Case 3: $x$ and $y$ share a child $z$:
  - If we observe $z$ then we have:



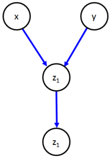

    We have $x \not\perp y \mid z$: if we know $z$, then knowing $x$ gives us information about $y$.
  - But if $z$ is not observed:




    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.

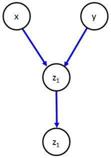
# D-Separation Case 3: Common Child

- Case 3: $x$ and $y$ share a child $z_1$:
  - If there exists an unobserved grandchild $z_2$:



# D-Separation Case 3: Common Child

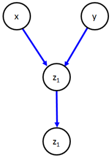
- Case 3: $x$ and $y$ share a child $z_1$:
  - If there exists an unobserved grandchild $z_2$:




We have $x \perp y$: the path is still blocked by not knowing $z_1$ or $z_2$.

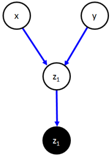
# D-Separation Case 3: Common Child

- Case 3: $x$ and $y$ share a child $z_1$:
  - If there exists an unobserved grandchild $z_2$:



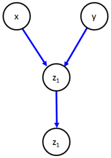

We have $x \perp y$: the path is still blocked by not knowing $z_1$ or $z_2$.

  - But if $z_2$ is observed:

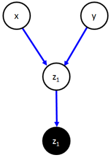


# D-Separation Case 3: Common Child

- 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:



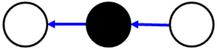

    We have $x \not\perp y \mid z_2$: grandchild creates dependence even with unobserved parent.
- Case 3 needs to consider descendants of child.

# D-Separation

- 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:

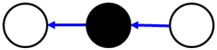
# D-Separation

- 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:



# D-Separation

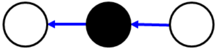
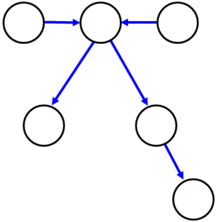
- We say that $A$ and $B$ are d-separated if
  for *all paths* $P$ from $A$ to $B$, *at least one* of the following holds:
  1. $P$ includes a "chain" with an observed middle node:

  2. $P$ includes a "fork" with an observed parent node:

# D-Separation

- 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:
     
  2. $P$ includes a "fork" with an observed parent node:
     
  3. $P$ includes a "v-structure" or "collider":
     

     where child $C$ and all its descendants are unobserved.

# Summary

- Decoding is task of finding most probable $x$.
- Message-passing allow efficient calculations with Markov chains.

# Summary

- Decoding is task of finding most probable $x$.
- 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.

# Summary

- Decoding is task of finding most probable $x$.
- 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 $C$.

# Summary

- Decoding is task of finding most probable $x$.
- 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: undirected graphical models and how we use graphical models.

# Bonus Slide: 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,

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

   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.

# Bonus Slide: 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.