Directed Acyclic Graphical Models

Suggested Reading

- MLPP: Chapters 10-12 (excluding * sections)
- Kevin Murphy's page on graphical models

Overview

- Graphical models notation
- Conditional independence
- Bayes Ball Algorithm
- Latent variables
- Common motifs

Joint distributions.

The joint distribution of N random variables is a very general way to encode knowledge about a system.

In general, a discrete distribution of N variables each of which can take K states requires $K^N - 1$ parameters to specify.

They can always be decomposed into a set of simpler conditional distributions by the chain rule of probability

$$p(x_1, x_2, \ldots, x_N) = p(x_1)p(x_2|x_1)p(x_3|x_2, x_1)\ldots p(x_N|x_{N-1}, \cdots, x_1)$$

this is true for any joint distribution over any random variables. For example, an application of the chain rule for two random variables gives

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

and for N random variables

$$p(x_1, x_2, \ldots, x_N) = \prod_{j=1}^{N} p(x_j|x_1, x_2, \ldots, x_{j-1})$$

for all possible orderings.

This decomposition doesn't reduce the number of parameters.

Number of parameters for joint distributions

The size of array representing a joint distribution can be huge if there are many variables and if the variables take on many states. This is true for both discrete and continuous variables, but is simpler in the discrete setting, so we'll focus on that in this lecture.

To see this, it is helpful to think of the joint distribution over a set of n random variables as a table with k^n parameters, where k the number of states each variable can take.
For simplicity, this assumes each variable takes on the same number of states, k.

Example

For example, the joint distribution over weather, $W = \{sun, rain\}$, and temperature $T = \{cold, hot\}$
where each variable can take 2 states can be represented by a 2×2 grid of parameters.

If we extend the possible weather states to include snow and fog,
$W = \{sun, rain, snow, fog\}$
then the parameterization of the joint would require a 2×4 grid of parameters.

If we also want our model to capture the mode of transportation for how we will get to class, $M = \{walk, bike, ttc\}$, then the parameterization would now require a $2 \times 4 \times 3$ cube of parameters.

> Actually, this is not quite accurate. In all cases we would require 1 fewer parameters to fully specify these distributions. This is due to the requirement that $\sum_x P(X = x) = 1$. If we know all but 1 parameter, then we can always solve for the remaining parameter.

How to reduce the number of parameters?

Most of the art of building models is maintaining the necessary flexiblity while using as few parameters as possible. One of the main tools we have is enforcing independence between variables. This is the same as enforcing factorizations, such as
$p(a, b) = p(a)p(b) \quad \forall a \forall b$.

The above discussion made no factorizations, giving the worst case of k^n.
This is maximally expressive, but requires us to parameterize every possible state.
Independence restricts the expressiveness of our model, but reduces the number of parameters required to specify a joint distribution.

For example, we can assume that T and W are independent (this is maybe a bad assumption!), but that our mode of transportation depends on temperature and weather.

$$P(T, W, M) = P(T)P(W)P(M|T, W)$$

This *joint factorization* encodes the independence assumptions, and requires fewer parameters to represent.

Conditional Independence

Reminder: Two random variables X_A, X_B are conditionally independent given a third variable X_C, denoted

$$X_A \perp X_B | X_C$$

if

$$\begin{aligned} &\Leftrightarrow p(X_A, X_B|X_C) = p(X_A|X_C)p(X_B|X_C) \\ &\Leftrightarrow p(X_A|X_B, X_C) = p(X_A|X_C) \\ &\Leftrightarrow p(X_B|X_A, X_C) = p(X_A|X_B) \end{aligned}$$

for all X_c.

Only a subset of all joint distributions respect any given conditional independence statement.

Directed acyclic graphical models

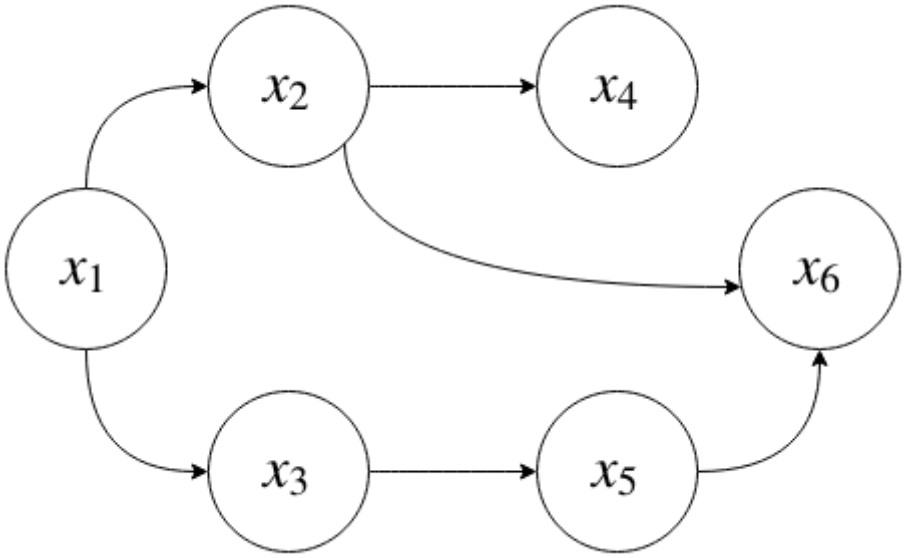
Reminder: The meaning of any particular directed acyclic graphical model D is that

$$p(x_1, x_2, \ldots, x_N) = \prod_{i=1}^{N} p(x_i|\text{parents}_M(x_i))$$

where $\text{parents}_M(x_i)$ is the set of nodes with edges pointing to x_i.

In other words, the joint distribution of a DAGM factors into a product of local conditional distributions, where each node (a random variable) is conditionally dependent on its parent node(s).

For example, the graphical model



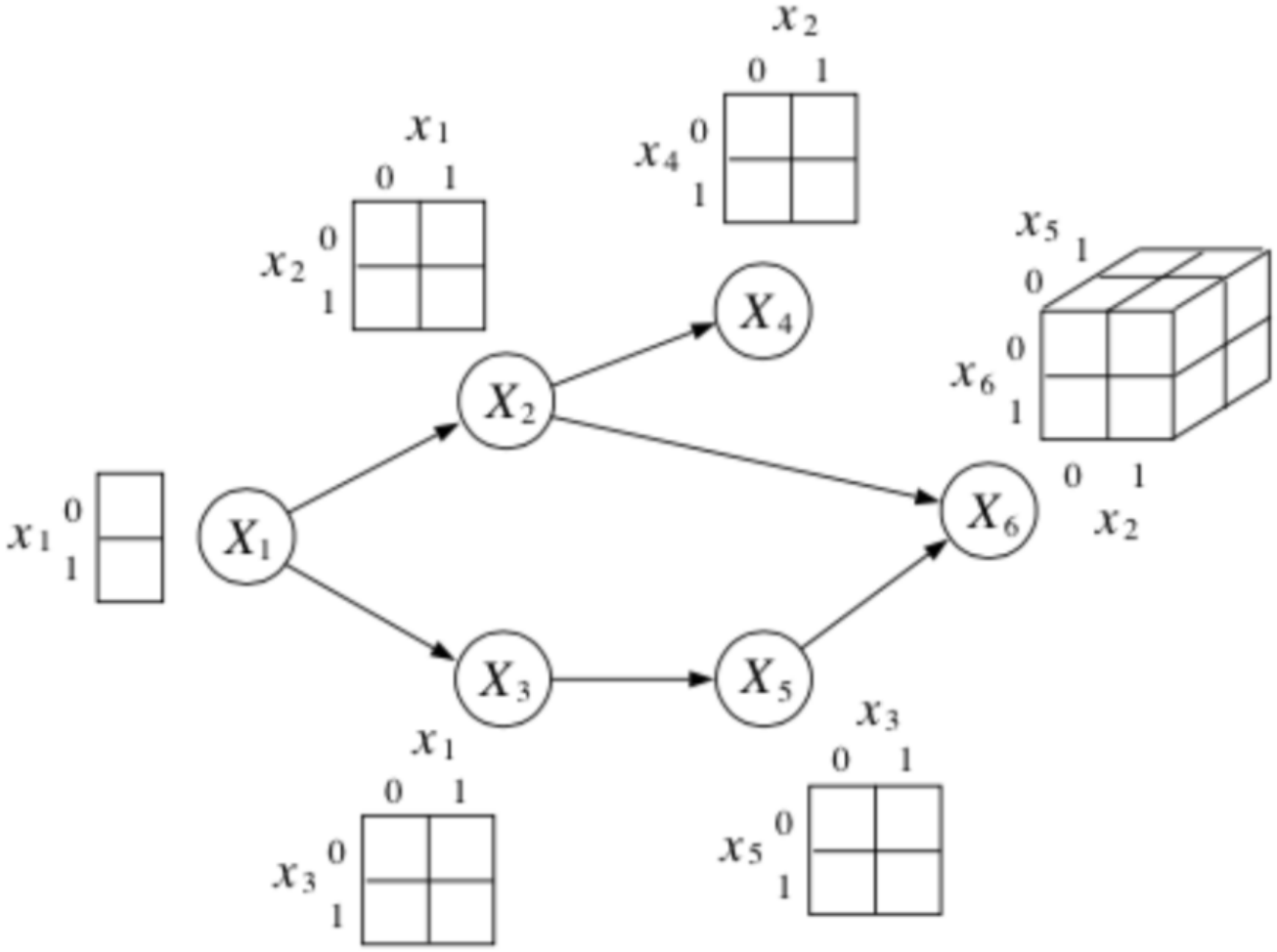
corresponds to the following factorization of the joint distribution:

$$p(x_1, x_2, \ldots, x_6) = p(x_1)p(x_2|x_1)p(x_3|x_1)p(x_4|x_2)p(x_5|x_3)p(x_6|x_2, x_5)$$

The general formula for the number of parameters necessary to specify the conditional probability table (CPT) of a variable with N parents each of which can take K states is:

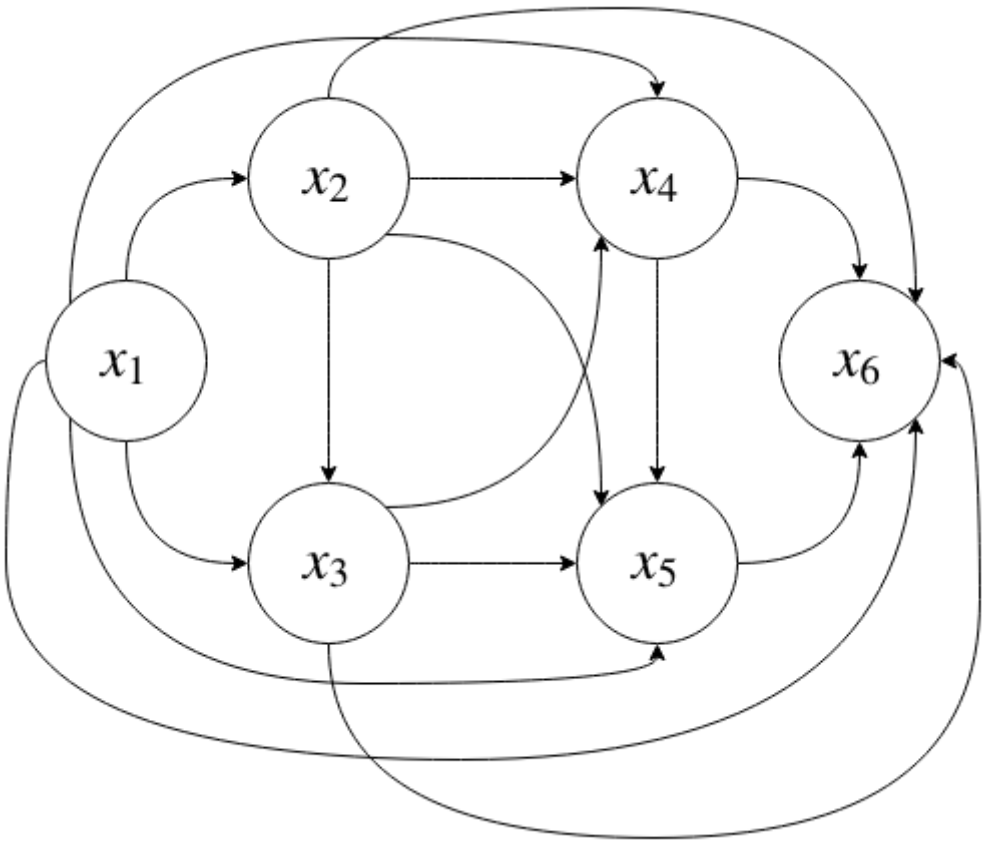
$$\underbrace{(K-1)}_{\text{pmf of node}} \times \underbrace{N^K}_{\text{possible states of parents}}$$

Suppose each is x_i is a binary random variable. How many parameters does it take to represent this joint distribution?



where each conditional probabilty table with K parents requires 2^K parameters.

If we allow all possible conditional dependencies, that corresponds to a fully-connected DAG:



which will require $2^N - 1$ parameters to specify.

Since we only condition on *parent nodes* as opposed to *every node*, this distribution is exponential in the fan-in of the nodes (the number of nodes in the parent set), instead of in N.

In general, it's computationally infeasible to work with fully-flexible distributions like this one, both because of the computational burden, and because it's hard to fit so many parameters accurately.

We can reduce the number of parameters in a model, and also reduce the computational burden of making inferences by introducing conditional independencies. This might not be a bad approximation in some settings.

Conditional Independence in DAGMs

From Kevin Murphy: The simplest conditional independence relationship encoded in a Bayesian network can be stated as follows: a node is independent of its ancestors given its parents:

$$x_i \perp x_{\tilde{\pi}_i} \mid x_{\pi_i}$$

In general, missing edges *imply conditional independence*. The next part of this lecture will be about how to determine which conditional independencies hold given a DAG.

D-Separation

D-separation, or **directed-separation** is a notion of connectedness in DAGs in which two (sets of) variables *may or may not be connected* conditioned on a third (set of) variable(s).
D-connection implies conditional *dependence* and d-separation implies conditional *independence*.

For set $A \subset \{1, 2, \ldots, N\}$, we denote by $x_A = \{x_i \ : \ i \in A\}$. In particular, we say that

$$x_A \perp x_B \mid x_C$$

if every variable in A is d-separated from every variable in B conditioned on all the variables in C. We will look at two methods for checking if an independence is true: A depth-first search algorithm and Bayes Balls.

DFS Algorithm for checking independence

To check if an independence is true, we can cycle through each node in A, do a depth-first search to reach every node in B, and examine the path between them. If all of the paths are d-separated (i.e., conditionally independent), then

$$x_i \perp x_j \mid x_k$$

It will be sufficient to consider triples of nodes.
Let's go through some of the most common triples.

1. Chain

Question: When we condition on y, are x and z independent?

Answer:

From the graph, we get

$$P(x, y, z) = P(x)P(y|x)P(z|y)$$

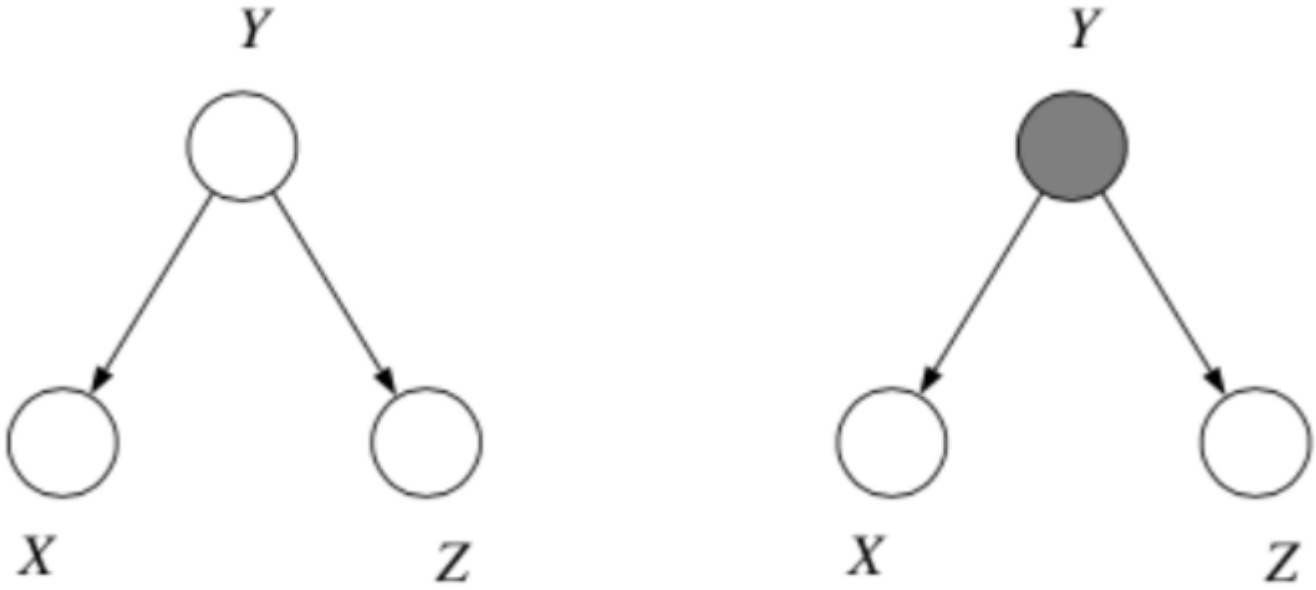
which implies

$$\begin{aligned} P(z|x,y) &= \frac{P(x,y,z)}{P(x,y)} \\ &= \frac{P(x)P(y|x)P(z|y)}{P(x)P(y|x)} \\ &= P(z|y) \end{aligned}$$

$\therefore P(z|x,y) = P(z|y)$ and so by $\star\star$, $x \perp z|y$.

> It is helpful to think about x as the past, y as the present and z as the future when working with chains such as this one.

2. Common Cause



Where we think of y as the "common cause" of the two independent effects x and z.

Question: When we condition on y, are x and z independent?

Answer:

From the graph, we get

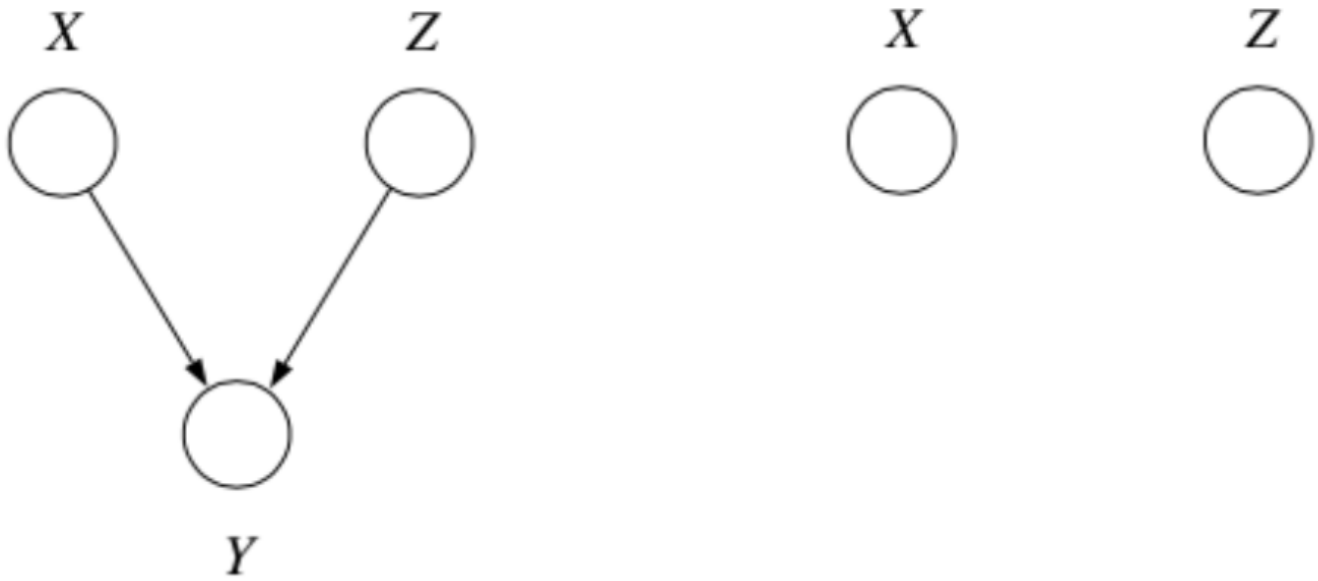
$$P(x,y,z) = P(y)P(x|y)P(z|y)$$

which implies

$$
\begin{aligned}
P(x, z|y) &= \frac{P(x, y, z)}{P(y)} \\
&= \frac{P(y)P(x|y)P(z|y)}{P(y)} \\
&= P(x|y)P(z|y)
\end{aligned}
$$

$\therefore P(x, z|y) = P(x|y)P(z|y)$ and so by $\star$, $x \perp z|y$.

3. Explaining Away



Question: When we condition on y, are x and z independent?

Answer:

From the graph, we get

$$
P(x, y, z) = P(x)P(z)P(y|x, z)
$$

which implies

$$
\begin{aligned}
P(z|x, y) &= \frac{P(x)P(z)P(y|x, z)}{P(x)P(y|x)} \\
&= \frac{P(z)P(y|x, z)}{P(y|x)} \\
&\neq P(z|y)
\end{aligned}
$$

$\therefore P(z|x, y) \neq P(z|y)$ and so by $\star\star$, $x \not\perp z|y$.

In fact, x and z are *marginally independent*, but given y they are *conditionally dependent*. This important effect is called explaining away (Berkson's paradox).

> Imaging flipping two coins independently, represented by events x and z. Furthermore, let $y = 1$ if the coins come up the same and $y = 0$ if they come up differently. Clearly, x and z are independent, but if I tell you y, they become coupled!

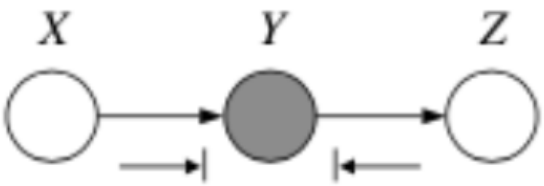
Bayes-Balls Algorithm

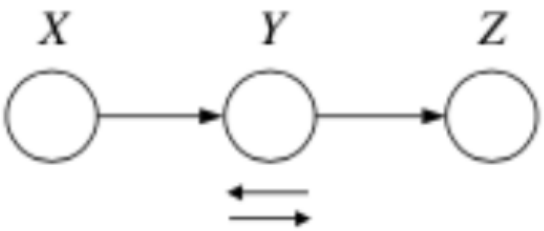
A particular algorithm for determining conditional independence in a DAGM is the Bayes Ball algorithm. To check if $x_A \perp x_B | x_C$ we need to check if every variable in A is d-seperated from every variable in B conditioned on all variables in C. In other words, given that all the nodes in x_C are "clamped", when we "wiggle" nodes x_A can we change any of the nodes in x_B?

In general, the algorithm works as follows:

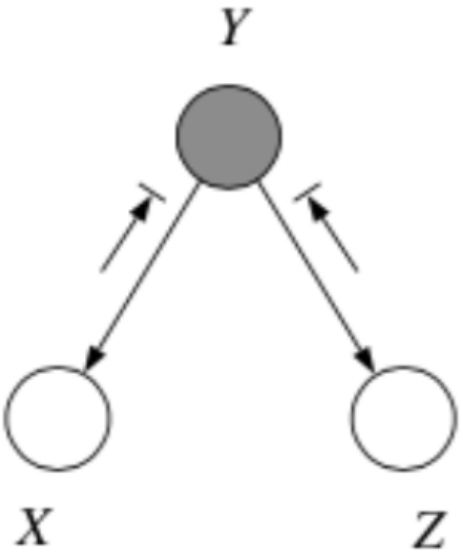
1. Shade all nodes x_C
2. Place "balls" at each node in x_A (or x_B)
3. Let the "balls" "bounce" around according to some rules
    - If any of the balls reach any of the nodes in x_B from x_A (or x_A from x_B) then $x_A \not\perp x_B | x_C$
    - Otherwise $x_A \perp x_B | x_C$

The rules are as follows:

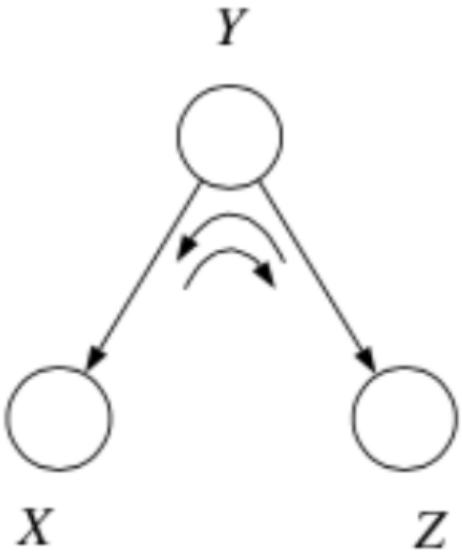




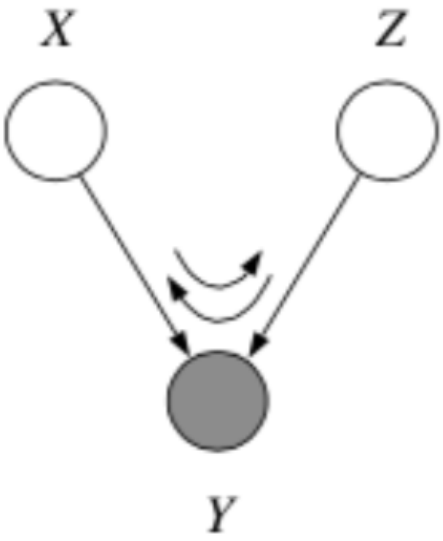
(a)



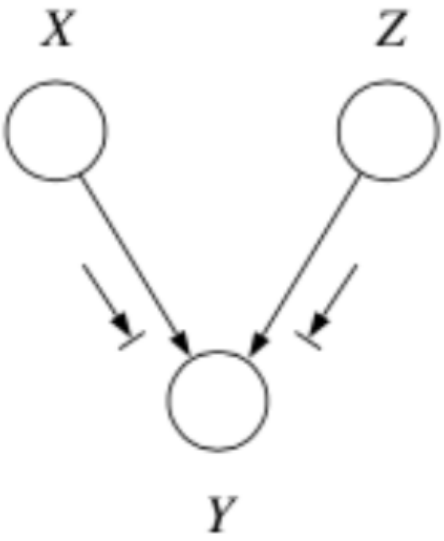
(b)



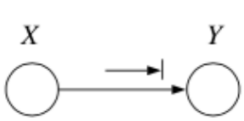
(a)

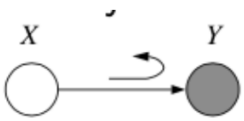


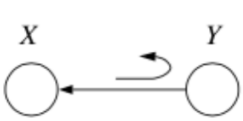
(b)

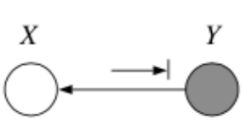


including the *boundary rules*:





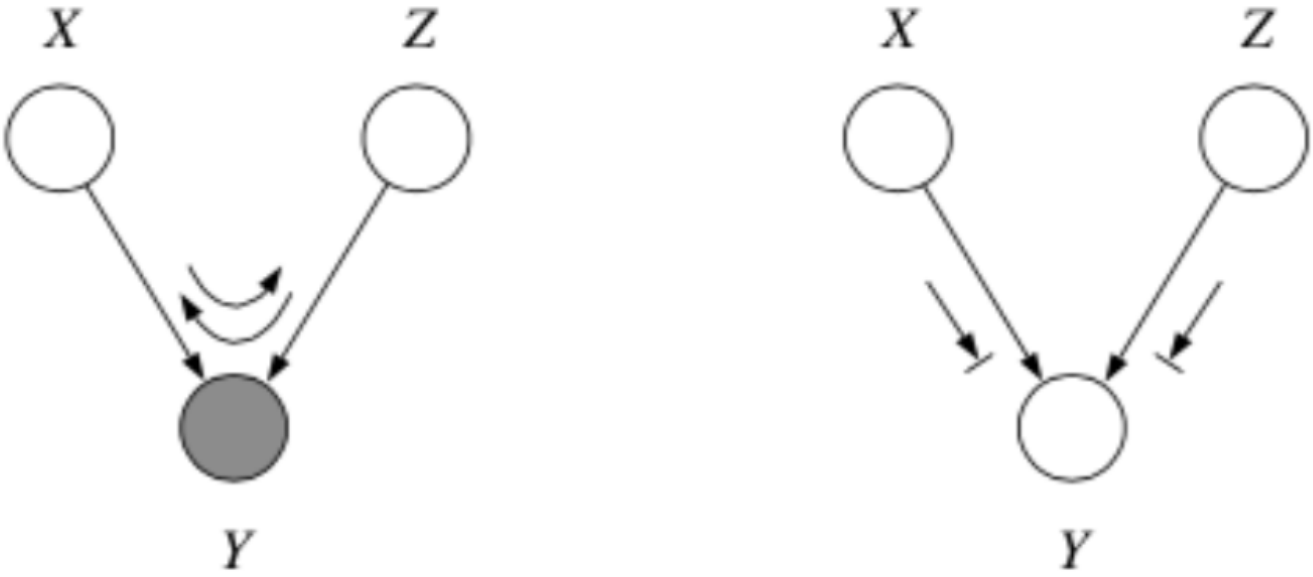




where **arrows** indicate paths the balls *can* travel, and **arrows with bars** indicate paths the balls *cannot* travel.

> Notice balls can travel opposite to edge directions!

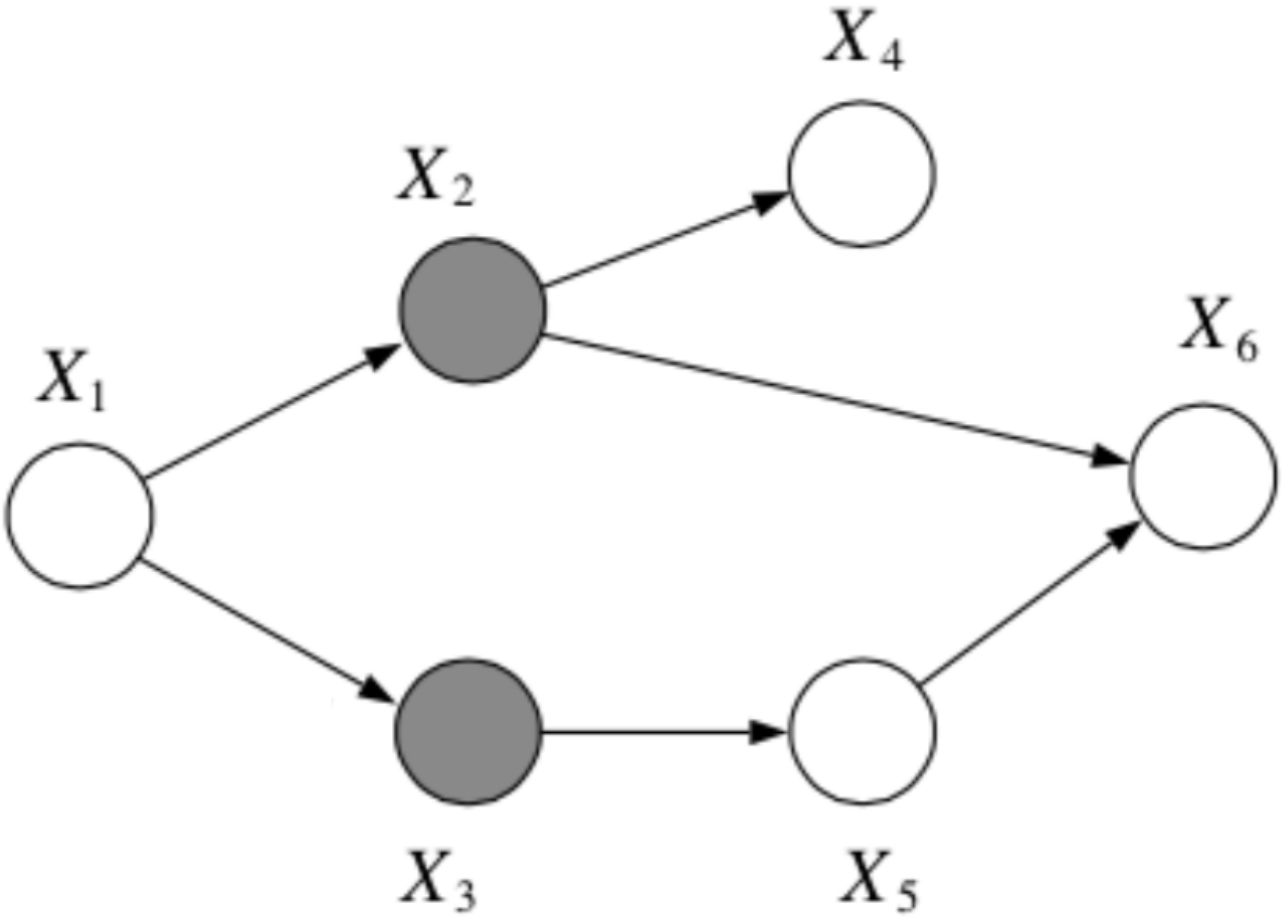
Here's a trick for the explaining away case: If y *or any of its descendants* is **shaded**, the ball passes through.



> See this video for an easy way to remember all 10 rules.

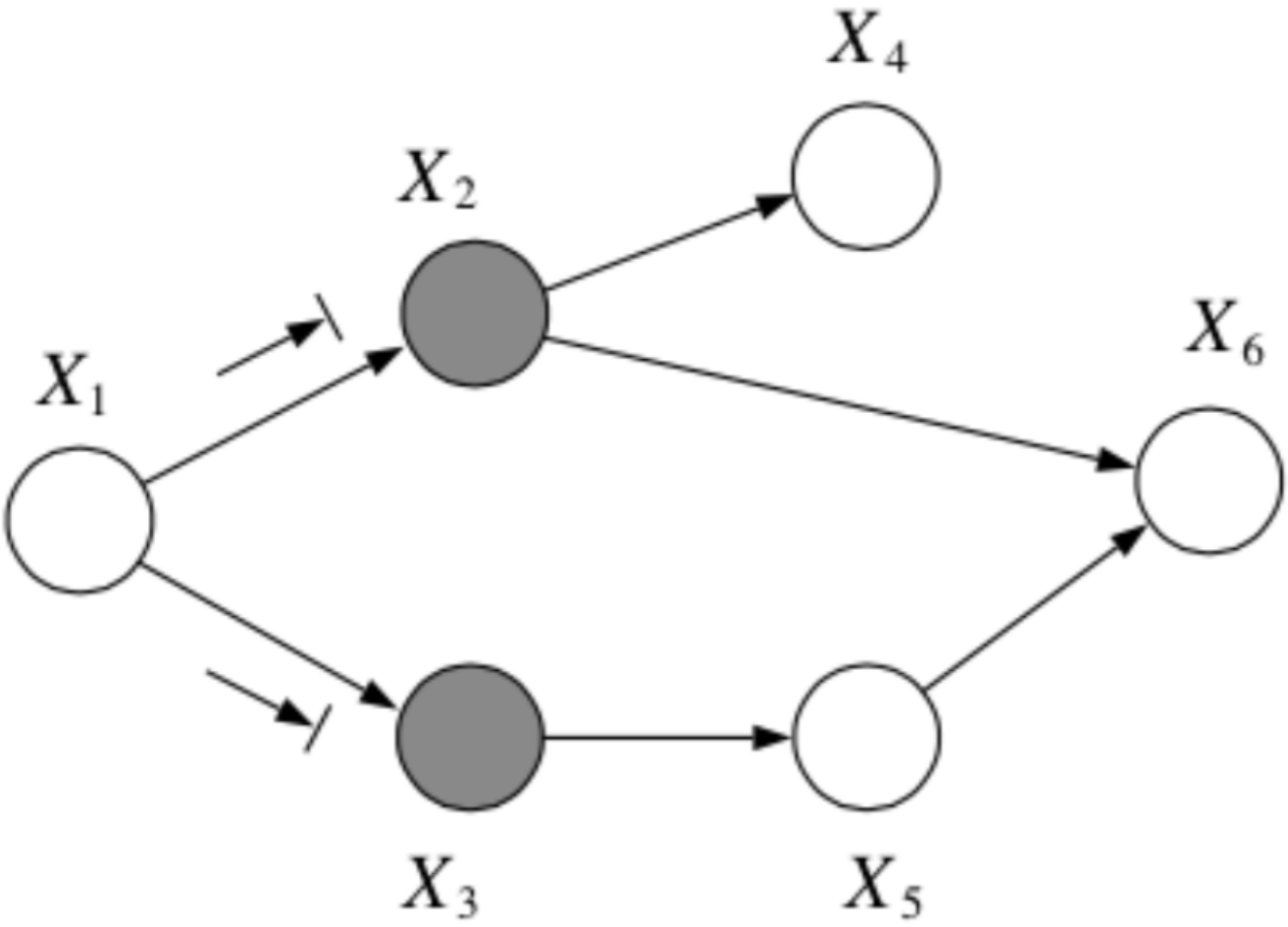
Examples

Question: In the following graph, is $x_1 \perp x_6 | \{x_2, x_3\}$?

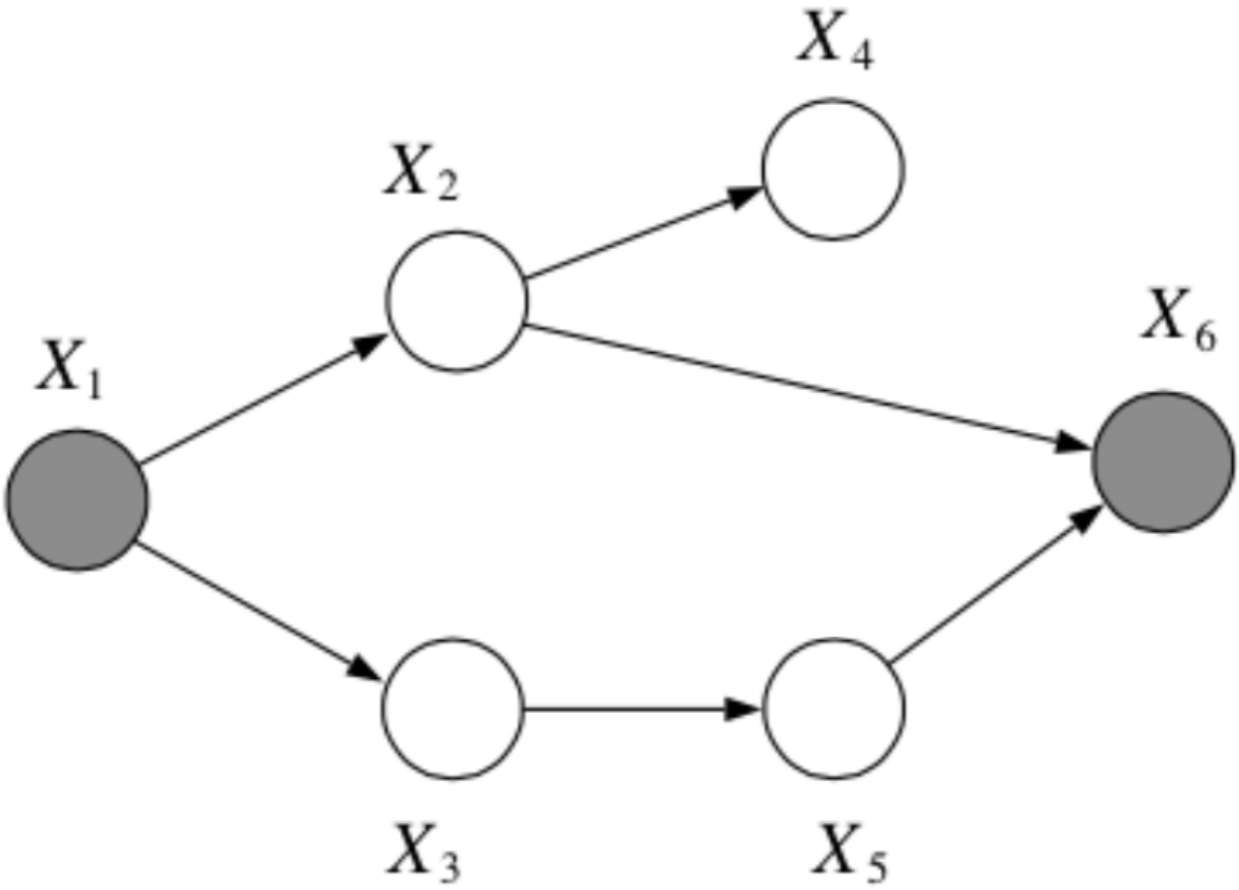


Answer:

Yes, by the Bayes Balls algorithm.

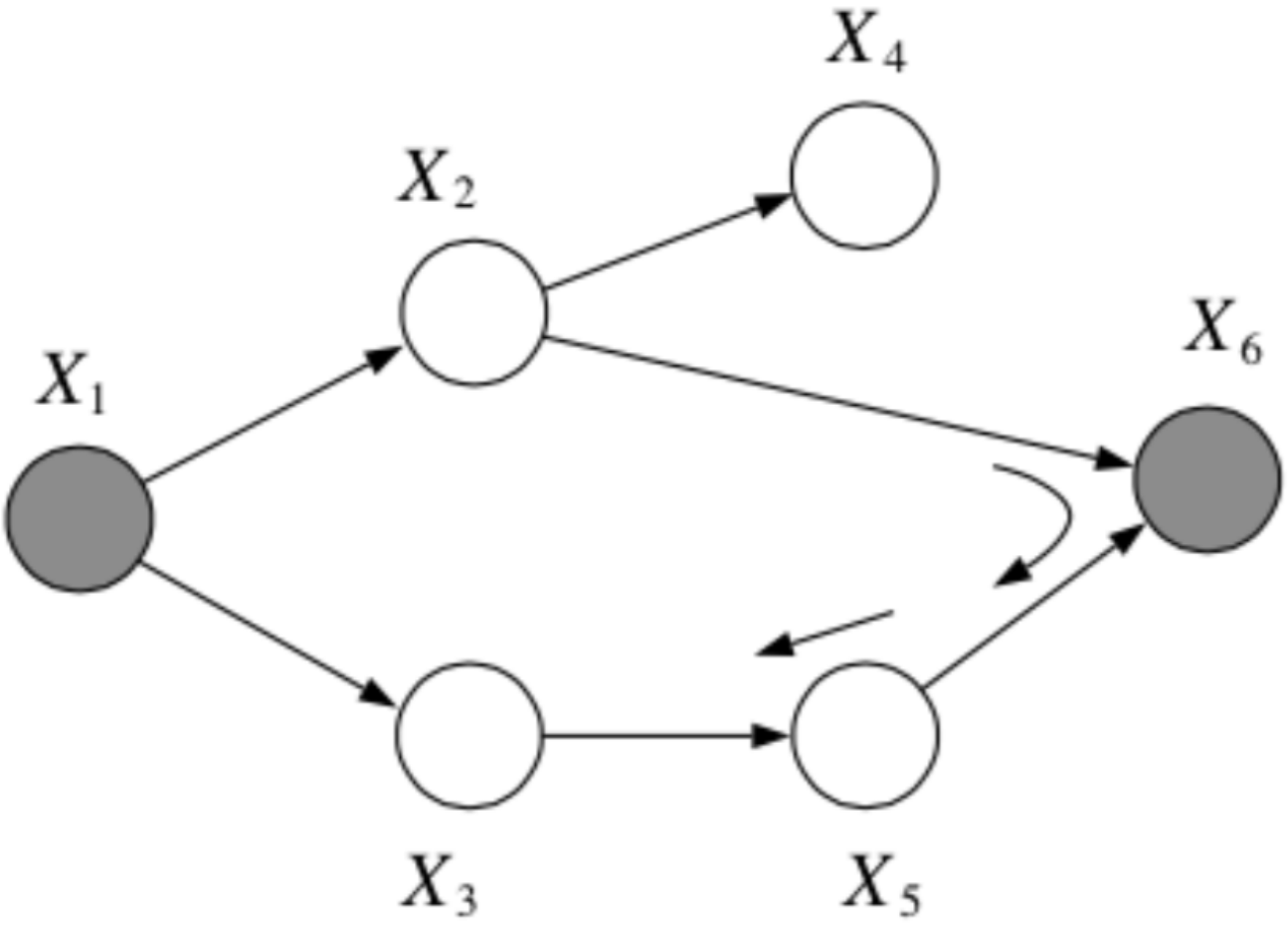


Question: In the following graph, is $x_2 \perp x_3 | \{x_1, x_6\}$?



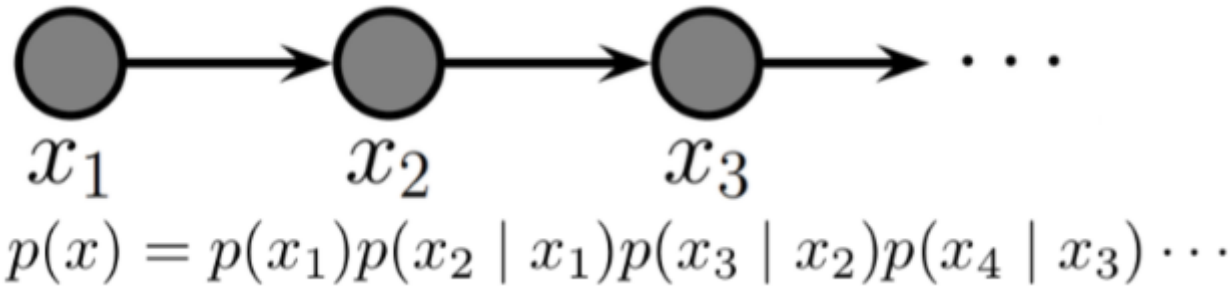
Answer:

No, by the Bayes Balls algorithm.



Example of a DAGM: Markov Chain

Markov chains are a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event.



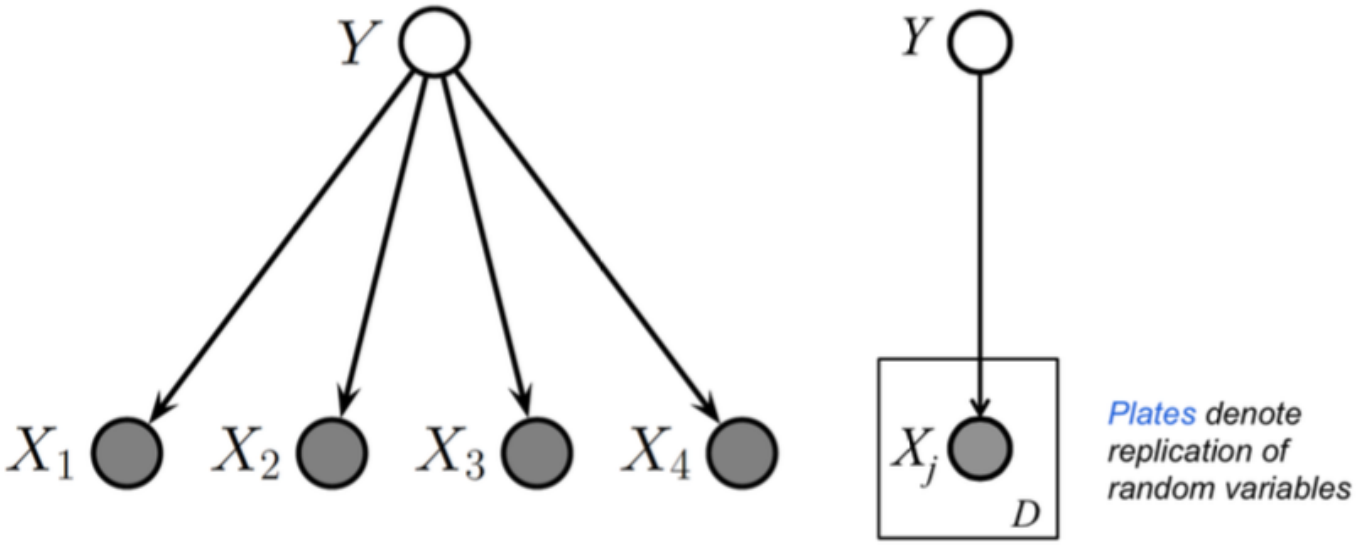
$$p(x) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_2)p(x_4 \mid x_3)\cdots$$

In other words, it is a model that satisfies the Markov property, i.e., conditional on the present state of the system, its future and past states are independent.

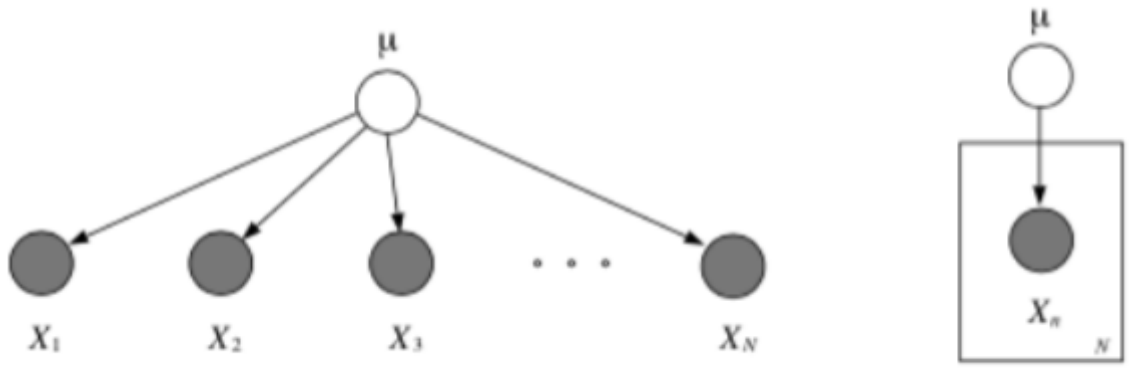
Plates

Because Bayesian methods treat parameters as random variables, we would like to include them in the graphical model. One way to do this is to repeat all the iid

observations explicitly and show the parameter only once. A better way is to use **plates**, in which repeated quantities that are iid are put in a box

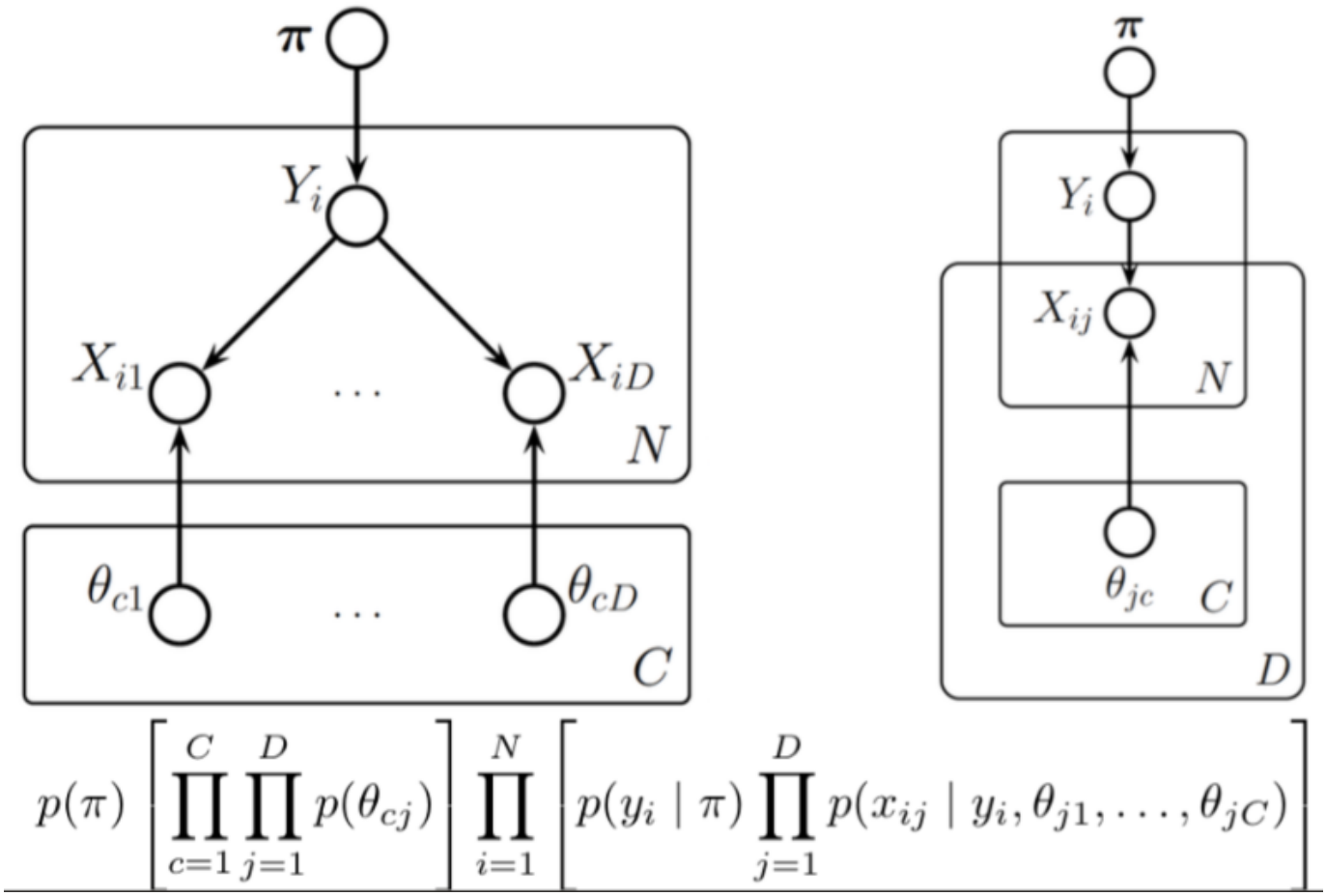


Plates are like “macros” that allow you to draw a very complicated graphical model with a simpler notation.
The rules of plates are simple: repeat every structure in a box a number of times given by the integer in the corner of the box (e.g. N), updating the plate index variable (e.g. n) as you go. Duplicate every arrow going into the plate and every arrow leaving the plate by connecting the arrows to each copy of the structure.



Nested Plates

Plates can be nested, in which case their arrows get duplicated also, according to the rule: draw an arrow from every copy of the source node to every copy of the destination node.



Plates can also cross (intersect), in which case the nodes at the intersection have multiple indices and get duplicated a number of times equal to the product of the duplication numbers on all the plates containing them.

Unobserved Variables

Certain variables in our models may be unobserved (Q in the example given below), either some of the time or always, at training time or at test time.

Graphically, we use shading to indicate observation.

Partially Unobserved (Missing) Variables

If variables are *occasionally unobserved* then they are *missing data*, e.g., undefined inputs, missing class labels, erroneous target values. In this case, we can still model the joint distribution, but we marginalize the missing values:

$$\ell(\theta; \mathcal{D}) = \sum_{\text{complete}} \log p(x^c, y^c|\theta) + \sum_{\text{missing}} \log p(x^m|\theta)$$

$$= \sum_{\text{complete}} \log p(x^c, y^c|\theta) + \sum_{\text{missing}} \log \sum_{y} p(x^m, y|\theta)$$

> Recall that $p(x) = \sum_q p(x, q)$.

Latent variables

What to do when a variable z is *always* unobserved? Depends on where it appears in our model. If we never condition on it when computing the probability of the variables we do observe, then we can just forget about it and integrate it out.

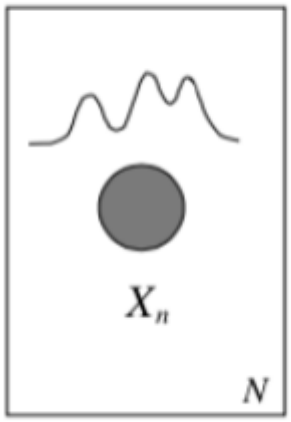
E.g., given y, x fit the model $p(z, y|x) = p(z|y)p(y|x, w)p(w)$. In other words if it is a leaf node.

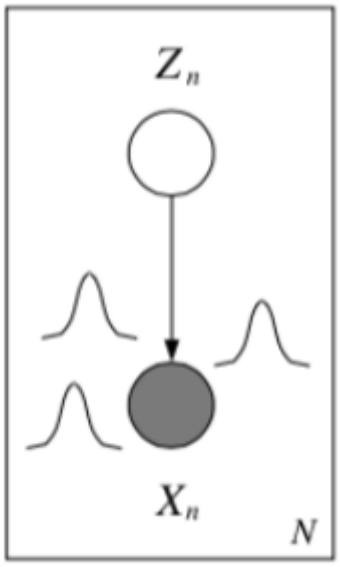
However, if z is not a leaf node, marginalizing over it will induce dependencies between its children.

E.g. given y, x fit the model $p(y|x) = \sum_z p(y|x, z)p(z)$.

Where do latent variables come from?

Latent variables may appear naturally, from the structure of the problem (because something wasn't measured, because of faulty sensors, occlusion, privacy, etc.). But we also may want to *intentionally* introduce latent variables to model complex dependencies between variables without specifying the dependencies between them directly.





Mixture models

What if the class is *unobserved*? Then we sum it out

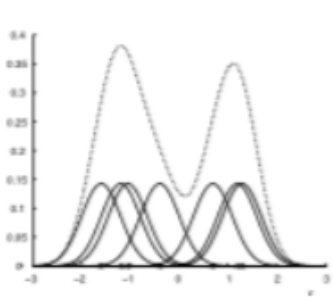
$$p(x|\theta) = \sum_{k=1}^{K} p(z=k|\theta_z)p(x|z=k,\theta_k)$$

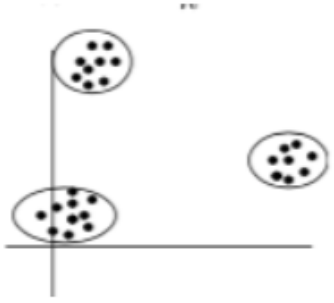
We can use Bayes' rule to compute the posterior probability of the mixture component given some data:

$$p(z=k|x,\theta_z) = \frac{p(z=k|\theta_z)p_k(x|\theta_k)}{\sum_j p(z=j|\theta_z)p_j(x|\theta_j)}$$

these quantities are called **responsibilities**.

Example: Gaussian Mixture Models





Consider a mixture of K Gaussian componentns

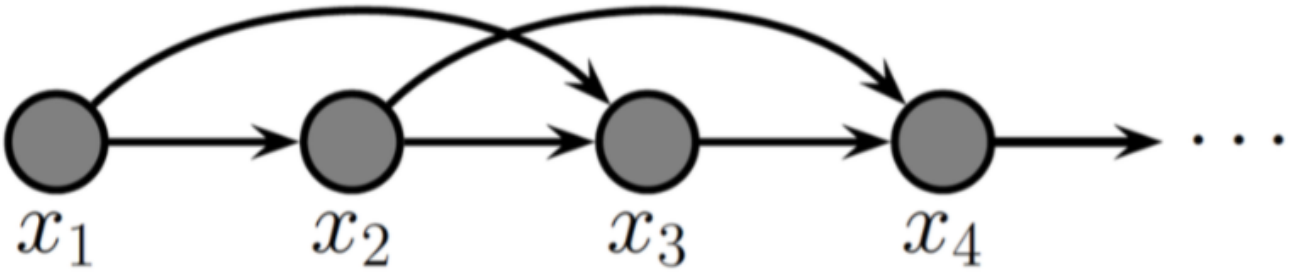
$$p(x|\theta) = \sum_k \alpha_k \mathcal{N}(x|\mu_k, \Sigma_k)$$
$$\log(x_1, x_2, \ldots, x_N)|\theta) = \sum_n \log \sum_k \alpha_k \mathcal{N}(x^{(n)}|\mu_k, \Sigma_k)$$
$$p(z = k|x, \theta) = \frac{\alpha_k \mathcal{N}(x|\mu_k, \Sigma_k)}{\sum_j \alpha_j \mathcal{N}(x|\mu_j, \Sigma_j)}$$

- Density model: $p(x|\theta)$ is the marginal density.
- Clustering: $p(z|x, \theta)$ is cluster assignment probability.
- Fitting: $p(z = k|x, \theta)$ is the log marginal likelihood.

Tutorial

Below are some detailed examples of the above concepts.

Second-order Markov chain

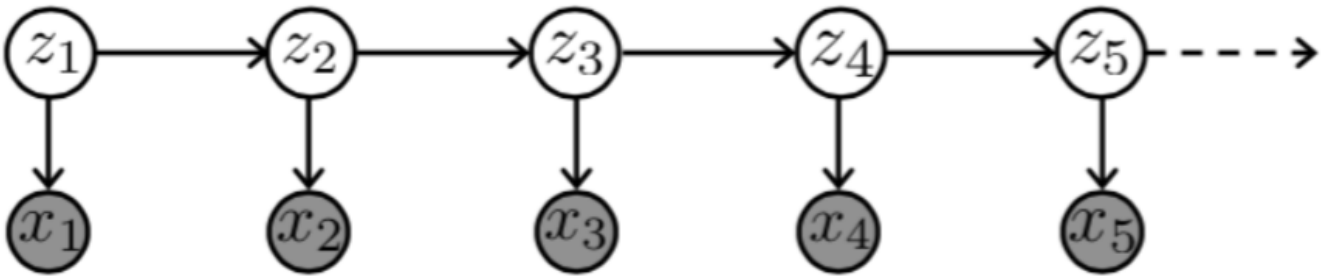


$$p(\mathbf{x}_{1:T}) = p(x_1, x_2)p(x_3|x_1, x_2)p(x_4|x_2, x_3)\ldots = p(x_1, x_2)\prod_{t=3}^{T} p(x_t|x_{t-1}, x_{t-2})$$

The earlier images depicts a *first*-order Markov chain, this is a *second*-order Markov chain.

Hidden Markov Models (HMMs)

Hidden Markov Model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (i.e. hidden) states. It is a very popular type of latent variable model



where

- Z_t are *hidden states* taking on one of K discrete values
- X_t are *observed variables* taking on values in any space

the joint probability represented by the graph factorizes according to

$$p(X_{1:T}, Z_{1:T}) = p(Z_{1:T})p(X_{1:T}|Z_{1:T}) = p(Z_1)\prod_{t=2}^{T} p(Z_t|Z_{t-1})\prod_{t=1}^{T} p(X_t|Z_t)$$

Conditional independence examples and Bayes Ball

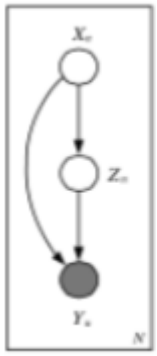
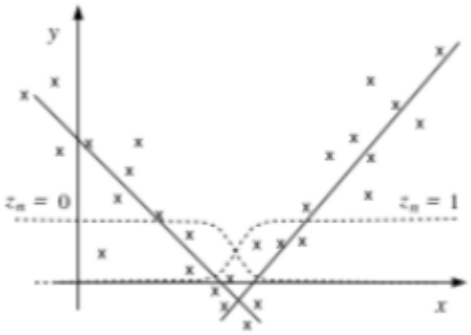
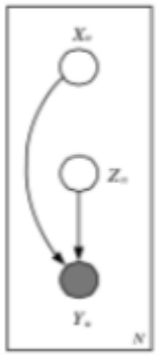
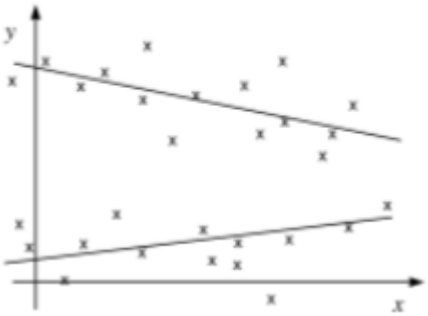
Example: Explaining away from Kevin Murphy's tutorial

From Kevin Murphy's page on graphical models:
Consider a college which admits students who are either brainy or sporty (or both!). Let C denote the event that someone is admitted to college, which is made true if they are either brainy (B) or sporty (S). Suppose in the general population, B and S are independent. We can model our conditional independence assumptions using a graph which is a V structure, with arrows pointing down.

Now look at a population of college students (those for which C is observed to be true). It will be found that being brainy makes you less likely to be sporty and vice versa, because either property alone is sufficient to explain the evidence on C.

Example: Mixtures of Experts

Think about the following two sets of data, and notice how there is some underlying structure *not dependent on x*.

The most basic latent variable model might introduce a single discrete node, z, in order to better model the data. This allows different submodels (experts) to contribute to the (conditional) density model in different parts of the space (known as a mixture of experts).

Mixtures of experts, also known as conditional mixtures are exactly like a class-conditional model, but the class is unobserved and so we sum it out:

$$p(y|x,\theta) = \sum_{k=1}^{K} p(z=k|x,\theta_z)p(y|z=k,x,\theta_K) = \sum_k \alpha_k(x|\theta_z)p_k(y|x,\theta_k)$$

where $\sum_k \alpha_k(x) = 1 \; \forall x$. This is a harder problem than the previous example, as we must learn $\alpha(x)$, often called the **gating function** (unless we chose z to be independent of x). However, we can still use Bayes' rule to compute the posterior probability of the mixture components given some data:

$$p(z=k|x,y,\theta) = \frac{\alpha_k(x)p_k(y|x,\theta_k)}{\sum_j \alpha_j(x)p_j(y|x_j,\theta_j)}$$

Example: Mixtures of Linear Regression Experts

In this model, each expert generates data according to a linear function of the input plus additive Gaussian noise

$$p(y|x,\theta) = \sum_k \alpha_k \mathcal{N}(y|\beta_k^T x, \sigma_k^2)$$

where the gating function can be a softmax classifier

$$\alpha_k(x) = p(z=k|x) = \frac{e^{\eta_k^T x}}{\sum_j e^{\eta_k^T x}}$$

Remember: we are *not* modeling the marginal density of the inputs x.

Gradient learning with mixtures

We can learn mixture densities using gradient descent on the likelihood as usual.

$$\begin{aligned}
\ell(\theta) = \log p(x|\theta) &= \log \sum_k \alpha_k p_k(x_k|\theta_k) \\
\Rightarrow \frac{\partial \ell}{\partial \theta} &= \frac{1}{p(x|\theta)} \sum_k \alpha_k \frac{\partial p_k(x|\theta)}{\partial \theta} \\
&= \sum_k \alpha_k \frac{1}{p(x|\theta)} p_k(x|\theta_k) \frac{\partial \log p_k(x|\theta_k)}{\partial \theta} \\
&= \sum_k \alpha_k \frac{p_k(x|\theta_k)}{p(x|\theta)} \frac{\partial \ell_k}{\partial \theta_k} \\
&= \sum_k \alpha_k r_k \frac{\partial \ell_k}{\partial \theta_k}
\end{aligned}$$

In other words, the gradient is the responsibility weighted sum of the individual log likelihood gradients

!!! tip
We used two tricks here to derive the gradient, $\frac{\partial \log f(\theta)}{\partial \theta} = \frac{1}{f(\theta)} \cdot \frac{\partial f(\theta)}{\partial \theta}$ and

$$\frac{\partial f(\theta)}{\partial \theta} = f(\theta) \cdot \frac{\partial \log f(\theta)}{\partial \theta}$$

Useful Resources

- Metacademy lesson on Bayes Balls. In fact, that link will bring you to a short course on a couple important concepts for this course, including conditional probability, conditional independence, Bayesian networks and d-separation.
- A video on how to memorize the Bayes Balls rules (this is linked in the above course).