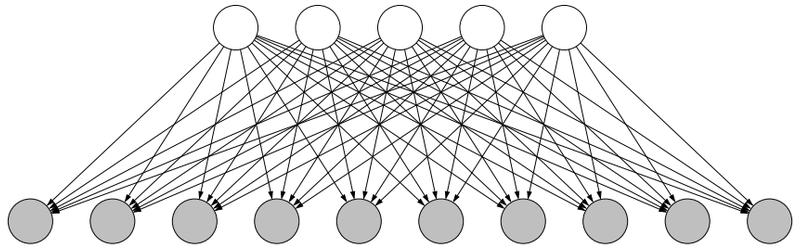
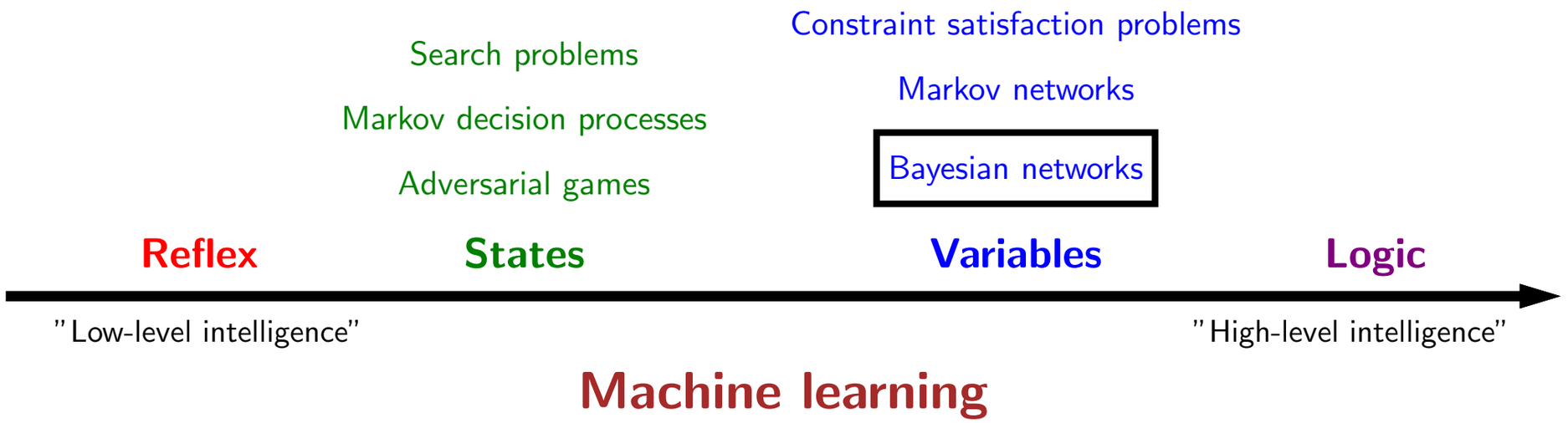
Bayesian networks: overview



- In this module, I'll introduce Bayesian networks, a new framework for modeling.

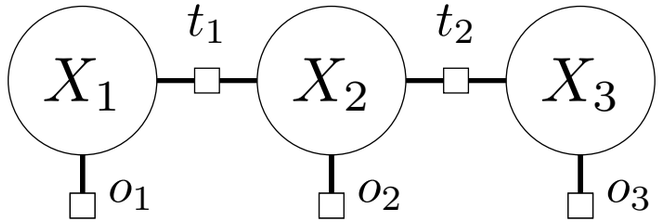
Course plan

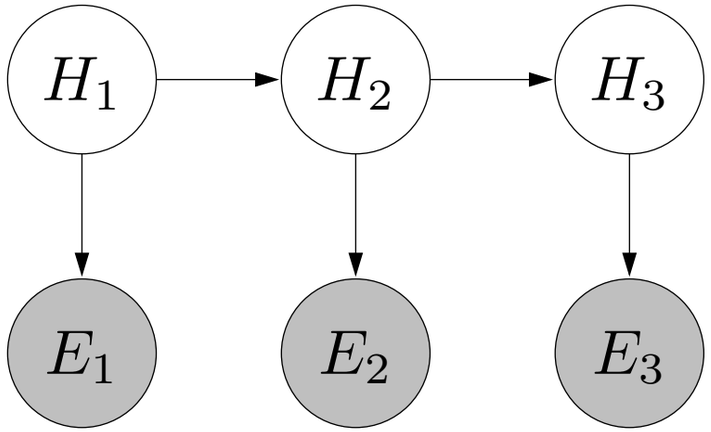

Constraint satisfaction problems
Search problems
Markov networks
Markov decision processes
Bayesian networks
Adversarial games
Reflex
States
Variables
Logic
"Low-level intelligence"
"High-level intelligence"
Machine learning

- We have talked about two types of variable-based models.
- In constraint satisfaction problems, the objective is to find the maximum weight assignment given a factor graph.
- In Markov networks, we use the factor graph to define a joint probability distribution over assignments and compute marginal probabilities.
- Now we will present Bayesian networks, where we still define a probability distribution using a factor graph, but the factors have special meaning.
- Bayesian networks were developed by Judea Pearl in the 1980s, and have evolved into the more general notion of generative modeling that we see today.

Markov networks versus Bayesian networks

Both define a joint probability distribution over assignments





Markov networks	Bayesian networks
arbitrary factors	local conditional probabilities
set of preferences	generative process

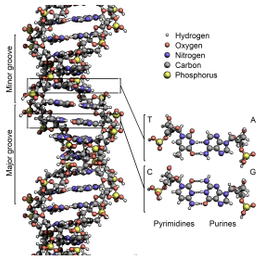
- Before defining Bayesian networks, it is helpful to compare and contrast Markov networks and Bayesian networks at a high-level.
- Both define a joint probability distribution over assignments, and in the end, both are backed by factor graphs.
- But the way each approaches modeling is different. In Markov networks, the factors can be arbitrary, so you should think about being able to write down an arbitrary set of preferences and constraints and just throw them in. In the object tracking example, we slap on observation and transition factors.
- Bayesian networks require the factors to be a bit more coordinated with each other. In particular, they should be local conditional probabilities, which we'll define in the next module.
- We should think about a Bayesian network as defining a generative process represented by a directed graph. In the object tracking example, we think of an object as moving from position H_{i-1} to position H_i and then yielding a noisy sensor reading E_i.

Applications

Topic modeling: unsupervised discovery of topics in text

Vision as inverse graphics: recover semantic description given image

Error correcting codes: recover data over a noisy channel



DNA matching: identify people based on relatives

- There are a huge number of applications of Bayesian networks, or more generally, generative models. One application is topic modeling, where the goal is to discover the hidden structure in a large collection of documents. For example, Latent Dirichlet Allocation (LDA) posits that each document can be described by a mixture of topics.
- Another application is a very different take on computer vision. Rather than modeling the bottom-up recognition using neural networks, which is the dominant paradigm today, we can encode the laws of physics into a graphics engine which can generate an image given a semantic description of an object. Computer vision is "just" the inverse problem: given an image, recover the hidden semantic information (e.g., objects, poses, etc.). While the "vision as inverse graphics" perspective hasn't been scaled up beyond restricted environemnts, the idea seems tantalizing.
- Switching gears, in a wireless or Ethernet network, nodes must send messages (a sequence of bits) to each other, but these bits can get corrupted along the way. The idea behind error correcting codes (Low-Density Parity Codes in particular) is that the sender also sends a set of random parity checks on the data bits. The receiver obtains a noisy version of the data and parity bits. A Bayesian network can then be defined to relate the original bits to the noisy bits, and the receiver can use inference (usually loopy belief propagation) to recover the original bits.
- The final application that we'll discuss is DNA matching. For example, Bonaparte is a software tool developed in the Netherlands that uses Bayesian networks to match DNA based on a candidate's family members. There are two use cases, the first one is controversial and the second one is grim. The first use case is in forensics: given DNA found at a crime site, even if the suspect's DNA is not in the database, one can match it against the family members of a suspect, where the Bayesian network is structured according to the family tree of the suspect and models the relationship between the family members's DNA using Mendelian inheritance. While this technology has been used to solve crime cases, there are some tricky ethical concerns about this expanded DNA matching, especially since an individual's decision to release their own DNA can impact the privacy of family members. The second use case is in disaster victim identification. After a big airplane crash (e.g., Malaysia Airlines flight MH17 in the Ukraine in 2014), a victim's DNA found at the crash site can be matched against their family members using the same mechanism above to identify the victim.

Why Bayesian networks?

- Handle **heterogenously** missing information, both at training and test time
- Incorporate **prior** knowledge (e.g., Mendelian inheritance, laws of physics)
- Can **interpret** all the intermediate variables
- Precursor to **causal** models (can do interventions and counterfactuals)

- These days, it's hard not to think about problems exclusively through the lens of standard supervised learning such as training a deep neural network on a pile of data.. Bayesian networks operate in a different paradigm which offers several advantages that are important to understand so that you can pick the right tool for the task.
- First, in traditional machine learning (e.g., linear models or neural networks), the input is usually of a fixed size (homogenous). With Bayesian networks, the types of inputs one can handle can be **hetereogenous** (e.g., missing features), both during training and test times.
- Second, Bayesian networks offer most leverage when you have rich **prior knowledge** (e.g., Mendelian inheritance, laws of physics). This allows one to often learn from very few samples and extrapolate beyond distribution of the training data. In contrast, deep neural networks generallly requires much more data to be effective.
- Third, because Bayesian networks are often carefully constructed based on prior knowledge, the variables in the Bayesian network are **interpretable** (more so that hidden units in a neural network), and you can ask questions about any of them via the laws of probability.
- Finally, Bayesian networks are an important precursor to developing **causal** models, which allow us to answer questions about interventions ("what would happen if we gave this drug to this patient?") and counterfactuals ("what would have happened if we had given this drug?"). These are extremely tricky and deep questions that standard machine learning or any methods that only view the world through prediction are unable to answer. For an easy introduction to some of these ideas, check out Judea Pearl's *The Book of Why*.
- Finally, Bayesian networks aren't suitable in every situation. In many vision, speech, and language problems, we have large datasets, mostly care about prediction, and it is extremely hard to incorporate prior knowledge about these very complex domains. In such cases, Bayesian networks have largely been supplanted with deep learning.

Roadmap

Modeling

Definitions

Probabilistic programming

Inference

Probabilistic inference

Forward-backward

Particle filtering

Learning

Supervised learning

Smoothing

EM algorithm

- In the remaining modules on Bayesian networks, I will first introduce a formal definition of Bayesian networks and explore some of its formal properties. Then I'll talk about probabilistic programming, a way to define Bayesian networks as (probabilistic) programs, which will provide a new perspective that allows to develop more powerful models.
- Then we turn to inference, which is what we do once we have a Bayesian network. We first define probabilistic inference, the problem of computing conditional and marginal probabilities and reduce this to the problem of inference in Markov networks. We then specialize to Hidden Markov Models (HMMs), an important special case of Bayesian networks, and show that the **forward-backward** algorithm can leverage the graph structure and do exact inference efficiently. Then we introduce particle filtering, which allows us to do approximate inference but scale up to HMMs where variables have larger domains.
- Finally, we talk about learning Bayesian networks from data. First we show how to do **supervised learning**, where all the variables are observed, which turns out to be very easy (just count and normalize). Then we show how to guard against overfitting in Bayesian networks by **smoothing**. Finally, we show how to do learning where some of the variables are unobserved using the **EM algorithm**.

Review: probability

Random variables: sunshine $S \in \{0, 1\}$, rain $R \in \{0, 1\}$

Joint distribution (probabilistic database):

$\mathbb{P}(S, R) =$

s	r	$\mathbb{P}(S = s, R = r)$
0	0	0.20
0	1	0.08
1	0	0.70
1	1	0.02

Marginal distribution:

(aggregate rows)

$\mathbb{P}(S) =$

s	$\mathbb{P}(S = s)$
0	0.28
1	0.72

Conditional distribution:

(select rows, normalize)

$\mathbb{P}(S \mid R = 1) =$

s	$\mathbb{P}(S = s \mid R = 1)$
0	0.8
1	0.2

- Before introducing Bayesian networks, let's review some basic probability. We start with an example about the weather. Suppose we have two boolean random variables, S and R representing whether there is sunshine and whether there is rain, respectively. Think of an assignment to (S, R) as representing a possible state of the world.
- The **joint distribution** specifies a probability for each assignment to (S, R) (state of the the world). We use lowercase letters (e.g., s and r) to denote values and uppercase letters (e.g., S and R) to denote random variables. Note that $\mathbb{P}(S = s, R = r)$ is a probability (a number) while $\mathbb{P}(S, R)$ is a distribution (represented by a table of probabilities). We don't know what state of the world we're in, but we know what the probabilities are (there are no unknown unknowns). Think of the joint distribution as one giant (probabilitsic) database that contains full information about how the world works.
- Sometimes, we might only be interested in a subset of the variables, e.g., sunshine S. From the joint distribution, we can derive a **marginal distribution** over that. In the case of S, we get this by summing the probabilities of the rows in the joint distribution table that share the same value of S. The interpretation is that we are interested in (the marginal probability of) S. We don't explicitly care about R, but we still need to take into account R's effect on S. We say in this case that R is **marginalized out**.
- Sometimes, we might observe evidence; for example, suppose we know that there's rain ($R = 1$). Again from the joint distribution, we can derive a **conditional distribution** of the remaining variables (S) given this evidence $R = 1$. We do this by selecting rows of the table matching the condition and then normalizing the remaining probabilities so that they sum to 1. Note that this normalization constant is exactly $\mathbb{P}(R = 1)$.

Review: probability

Variables: S (sunshine), R (rain), T (traffic), A (autumn)

Joint distribution (probabilistic database):

$$\mathbb{P}(S, R, T, A)$$

Marginal conditional distribution (probabilistic inference):

- **Condition** on evidence (traffic, autumn): $T = 1, A = 1$
- Interested in **query** (rain?): R

$$\mathbb{P}(\underbrace{R}_{\text{query}} \mid \underbrace{T = 1, A = 1}_{\text{condition}})$$

(S is **marginalized out**)

- Let us augment our running example with two other random variables, T (whether there is traffic) and A (whether it's autumn).
- We have a joint distribution, which again can be thought of as a probabilistic database that tells us how the world works.
- Probabilistic inference is the process of answering questions against this database. In general, we can both condition on evidence and be interested in a subset of the remaining variables at the same time.
- For example, we might **condition** on there being traffic and the fact that it's autumn.
- And we might be interested in whether there is rain (called the **query** variable), marginalizing out **sunshine**.
- The set of conditioning variables, query variables, and variables that are marginalized out should form a partitioning of all the variables.

A puzzle

Problem: earthquakes, burglaries, and alarms

Earthquakes and **burglaries** are independent events (probability ϵ).

Either will cause an **alarm** to go off.

Suppose you get an **alarm**.

Does hearing that there's an **earthquake** increase, decrease, or keep constant the probability of a **burglary**?

Joint distribution:

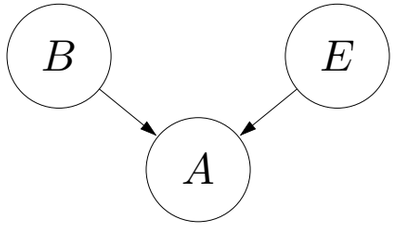
$$\mathbb{P}(E, B, A)$$

Questions:

$$\mathbb{P}(B = 1 \mid A = 1) \quad \textbf{?} \quad \mathbb{P}(B = 1 \mid A = 1, E = 1)$$

- Let's consider a classic puzzle, which we will tackle with Bayesian networks. Suppose that in the world, earthquakes and burglaries are independent (and hopefully rare) events, and for the sake of simplicity, assume that each one has a probability ϵ (say 0.05) of happening. You have installed an alarm that will notify you if either one happens.
- Now suppose you are away on vacation and you get an alarm notification on your phone. You would expect at this point that the probability of your home being burglarized has gone up. But suppose then you see breaking news saying that there was an earthquake near your home. How does that change your beliefs about the burglary?
- One could try to intuit the answer, but this is risky because sometimes the right answer is counterintuitive. In this case, you might think since earthquakes and burglaries are independent, that the probability shouldn't change. But that would be wrong. So let's use Bayesian networks instead to perform this type of **reasoning under uncertainty** in a principled way.
- Let us try to write down this question using the language of probability. The first step is to always figure out the variables of interest, which in this case are earthquake E, burglary B, and alarm A.
- We then have a joint distribution over these variables, which we will define later. But first the questions. We are interested in comparing the probability of a burglary given an alarm only versus given alarm and earthquake.

Bayesian network (alarm)



b	$p(b)$
1	ϵ
0	$1-\epsilon$

e	$p(e)$
1	ϵ
0	$1-\epsilon$

b	e	a	$p(a \mid b,e)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$$p(b) = \epsilon \cdot [b = 1] + (1 - \epsilon) \cdot [b = 0]$$

$$p(e) = \epsilon \cdot [e = 1] + (1 - \epsilon) \cdot [e = 0]$$

$$p(a \mid b, e) = [a = (b \vee e)]$$

$$\mathbb{P}(B = b, E = e, A = a) \stackrel{\text{def}}{=} p(b)p(e)p(a \mid b, e)$$

- Now let us define the joint distribution. Recall the first step was just to define the three variables, B (burglary), E (earthquake), and A (alarm).
- Second, we connect up the variables to model the dependencies. Unlike in factor graphs, these dependencies are represented as **directed** edges. You can intuitively think about the directionality as representing causality, though what this actually means is a more complex issue and beyond the scope of this module.
- Third, for each variable, we specify a **local conditional distribution** of that variable given its parent variables. In this example, B and E have no parents while A has two parents, B and E. This local conditional distribution is what governs how a variable is generated.
- Fourth, we define the joint distribution over all the random variables as the product of all the local conditional distributions.
- Note that we write the local conditional distributions using p, while $\mathbb{P}$ is reserved for the joint distribution over all random variables, which is defined as the product.

Probabilistic inference (alarm)

Joint distribution

b	e	a	$\mathbb{P}(B=b, E=e, A=a)$
0	0	0	$(1-\epsilon)^2$
0	0	1	0
0	1	0	0
0	1	1	$(1-\epsilon)\epsilon$
1	0	0	0
1	0	1	$\epsilon(1-\epsilon)$
1	1	0	0
1	1	1	ϵ^2

Questions:

$$\mathbb{P}(B=1) = \epsilon(1-\epsilon) + \epsilon^2 = \epsilon$$

$$\mathbb{P}(B=1 \mid A=1) = \frac{\epsilon(1-\epsilon)+\epsilon^2}{\epsilon(1-\epsilon)+\epsilon^2+(1-\epsilon)\epsilon} = \frac{1}{2-\epsilon}$$

$$\mathbb{P}(B=1 \mid A=1, E=1) = \frac{\epsilon^2}{\epsilon^2+(1-\epsilon)\epsilon} = \epsilon$$

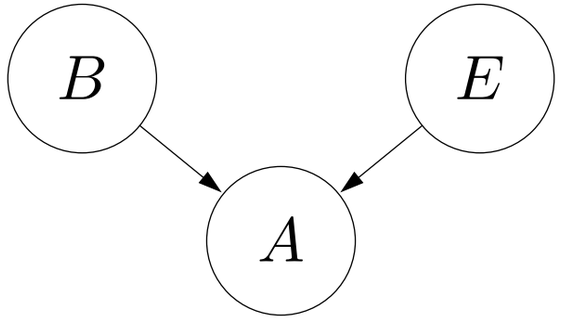
[demo]

News flash: earthquakes decrease burglarlies!*

*This is not a causal statement!

- We multiply all the local conditional distributions together to produce the joint distribution. Recall this is the probabilistic that is the source of all truth, and from it we can answer all sorts of questions.
- Let us start with the simplest query, $\mathbb{P}(B = 1)$: what is the probability of burglary without any evidence? We can sum up all the rows with $B = 1$ to get ϵ.
- Now suppose we hear the alarm $A = 1$. Let us first filter out all the rows where $A = 1$ does not hold. Then we look at the sum of the probabilities of rows where $B = 1$ over the sum of all the probabilities. The resulting probability of burglary is now $\mathbb{P}(B = 1 \mid A = 1) = \frac{1}{2-\epsilon}$.
- Now let us condition on alarm $(A = 1)$ and earthquake $(E = 1)$. Filter out rows that don't satisfy the condition, and look at the fraction of probabilities of remaining rows on $B = 1$. The resulting probability of burglary goes **down** to $\mathbb{P}(B = 1 \mid A = 1, E = 1) = \epsilon$ again.
- So in the end, observing that there's an earthquake does actually decrease the probability of the burglary. This might be counterintuitive because we said that burglaries and earthquakes are independent. But it's important to not interpret this causally. Creating more earthquakes clearly will not make the burglars disappear. When dealing with slippery questions such as these, we need a sound mathematical framework like Bayesian networks to ensure that we get the right answers.

Explaining away



Key idea: explaining away

Suppose two causes positively influence an effect. Conditioned on the effect, further conditioning on one cause reduces the probability of the other cause.

$$\mathbb{P}(B = 1 \mid A = 1, E = 1) < \mathbb{P}(B = 1 \mid A = 1)$$

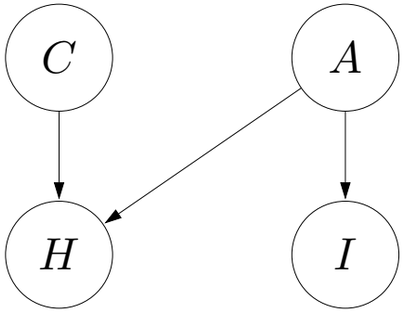
Note: happens even if causes are independent!

- This last phenomenon is so important for reasoning under uncertainty that it has a special name: **explaining away**. Suppose we have two **cause** variables B and E, which are parents of an **effect** variable A. Futher, assume the causes influence the effect positively (e.g., through the OR function).
- Let us condition on the evidence $A = 1$. We are trying to seek an explanation for $A = 1$ (what caused the alarm to go off?).
- Further conditioning on one of the causes ($E = 1$) decreases the probability of the other cause, because $E = 1$ alone **explains away** $A = 1$, and there's no more pressure on B.
- Note that in our setting, the probability of $B = 1$ returns to the original $\mathbb{P}(B = 1)$, but this need not be the case in general.
- Conditioning on $A = 1$ is important for explaining away. If you didn't, then the probability of $B = 1$ would not change. You can verify for yourself that $\mathbb{P}(B = 1 \mid E = 1) = \mathbb{P}(B = 1)$, which just follows from the definition of B and E being independent.

Medical diagnosis

Problem: cold or allergies?

You are coughing and have itchy eyes. Do you have a cold?



Random variables:

cold C, allergies A, cough H, itchy eyes I

Joint distribution:

$$\mathbb{P}(C = c, A = a, H = h, I = i) = p(c)p(a)p(h \mid c, a)p(i \mid a)$$

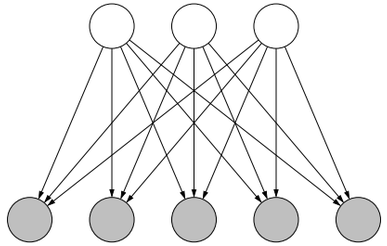
Questions:

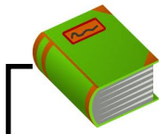
$$\mathbb{P}(C = 1 \mid H = 1) = 0.28 \qquad \mathbb{P}(C = 1 \mid H = 1, I = 1) = 0.13$$

[demo]

- Here is another example (a cartoon version of Bayesian networks for medical diagnosis).
- Step 1: identify all the relevant variables.
- Step 2: draw arrows between them, using prior knowledge. Using our simplistic medical knowledge, suppose that a cough can be either because of a cold or because of allergies, but itchy eyes are generally only caused by allergies.
- Step 3: define a local conditional distribution for each variable.
- Step 4: multiply all the local conditional distributions to form the joint distribution.
- Now we have our probabilistic database and we can ask questions about it. Our motivating question is $\mathbb{P}(C, A \mid H = 1, I = 1)$.
- You can try the demo to get a quantitative answer. Note that $\mathbb{P}(C = 1 \mid H = 1) = 0.28$, which is another example of explaining away. Observing itchy eyes provides evidence for A, which explains away the cough ($H = 1$), resulting in a reduced probability of cold ($C = 1$).
- Note that even qualitatively reasoning about even a four-node Bayesian network can be quite subtle, let alone getting quantitative answers on large Bayesian networks. But we can rest at ease since the laws of probability make sure that all these calculations are internally consistent provided we defined the Bayesian network correctly (which in practice is an admittedly hard modeling task).

Bayesian network (definition)





Definition: Bayesian network

Let $X = (X_1, \ldots, X_n)$ be random variables.

A **Bayesian network** is a directed acyclic graph (DAG) that specifies a joint distribution over X as a product of local conditional distributions, one for each node:

$$\mathbb{P}(X_1 = x_1, \ldots, X_n = x_n) \stackrel{\text{def}}{=} \prod_{i=1}^{n} p(x_i \mid x_{\text{Parents}(i)})$$

- Without further ado, let's define a Bayesian network formally. A Bayesian network defines a joint distribution over a set of random variables.
- Second, we have a directed **acyclic** graph over the variables that captures the qualitative dependencies.
- Third, we specify a local conditional distribution for each variable X_i, which is a function that specifies a distribution over X_i given an assignment $x_{\mathsf{Parents}(i)}$ to its parents in the graph (possibly no parents).
- Finally, the joint distribution is simply **defined** to be the product of all of the local conditional distributions.
- Notationally, we use lowercase p (in $p(x_i \mid x_{\mathsf{Parents}(i)})$) to denote a local conditional distribution, and uppercase $\mathbb{P}$ to denote the induced joint distribution over all variables. While we will see that the two coincide, it is important to keep these things separate in your head!

Probabilistic inference (definition)

Input

Bayesian network: $\mathbb{P}(X_1, \ldots, X_n)$

Evidence: $E = e$ where $E \subseteq X$ is subset of variables

Query: $Q \subseteq X$ is subset of variables

Output

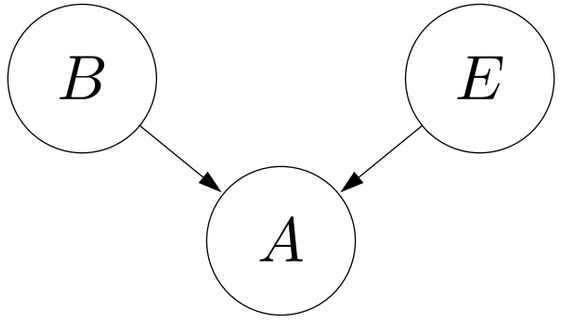
$\mathbb{P}(Q \mid E = e)$  $\mathbb{P}(Q = q \mid E = e)$ for all values q

Example: if coughing and itchy eyes, have a cold?

$$\mathbb{P}(C \mid H = 1, I = 1)$$

- Now given a Bayesian network representing a probabilistic database, we can answer questions on it.
- In particular, we are given a set of evidence variables E and values e. We are also given a set of query variables Q. What a probabilistic inference algorithm should output given this is the marginal conditional distribution $\mathbb{P}(Q \mid E = e)$.
- Note that this output is a table that specifies a probability for each assignment of values to Q.
- So far, we have shown examples of probabilistic inference on small Bayesian networks. The bad news is that in general, answering arbitrary probabilistic inference questions on arbitrary Bayesian networks is computationally intractable. The good news it that the core probabilistic inference in Bayesian networks is identical to Markov networks (which we will see later).

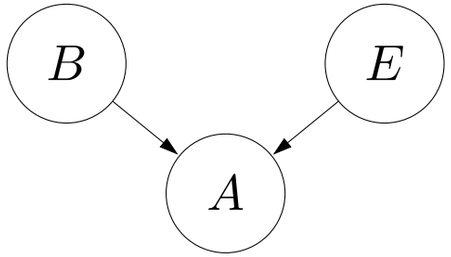
Summary



- Random variables capture state of world
- Directed edges between variables represent dependencies
- Local conditional distributions $\Rightarrow$ joint distribution
- Probabilistic inference: ask questions about world
- Captures reasoning patterns (e.g., explaining away)

- In summary, we have introduced Bayesian networks.
- It's important to think about an assignment to random variables as capturing the state of the world.
- Directed edges represent qualitative (sometimes causal) dependencies.
- Quantitatively, we specify a local conditional distribution for each variable conditioned on its parents, and multiply them together to get a joint distribution.
- Now we have our probabilistic database on which we can ask all sorts of questions, marginal conditional probabilities.
- Hopefully through the alarm and medical diagnosis examples, you are able to appreciate that the framework can capture intuitive or counter-intuitive reasoning patterns such as explaining away in a mathematically sound way so you can sleep well at night.

Probabilistic programs



Joint distribution:

$$\mathbb{P}(B = b, E = e, A = a) = p(b)p(e)p(a \mid b, e)$$

Probabilistic program: alarm

$B \sim \text{Bernoulli}(\epsilon)$
$E \sim \text{Bernoulli}(\epsilon)$
$A = B \vee E$

```
def Bernoulli(epsilon):
    return random.random() < epsilon
```

Key idea: probabilistic program

A randomized program that sets the random variables.

- Recall that a Bayesian network is given by (i) a set of random variables, (ii) directed edges between those variables capturing qualitative dependencies, (iii) local conditional distributions of each variable given its parents which captures these dependencies quantitatively, and (iv) a joint distribution which is produced by multiplying all the local conditional distributions together. Now the joint distribution is your probabilistic database, which you can answer all sorts of questions on it using probabilistic inference.
- There is another way of writing down Bayesian networks other than graphically or mathematically, and that is as a probabilistic program.
- Let's go through the alarm example. We can sample B and E independently from a Bernoulli distribution with parameter ϵ, which produces 1 (true) with probability ϵ. Then we just set $A = B \vee E$.
- In general, a **probabilistic program** is a randomized program that invokes a random number generator. Executing this program will assign values to a collection of random variables $X_1, \ldots, X_n$; that is, generating an assignment.
- We then define probability under the joint distribution of an assignment to be exactly the probability that the program generates an assignment.
- While you can run the probabilistic program to generate samples, it's important to think about it as a mathematical construct that is used to define a joint distribution.

Probabilistic program: example

Probabilistic program: object tracking

$X_0 = (0, 0)$

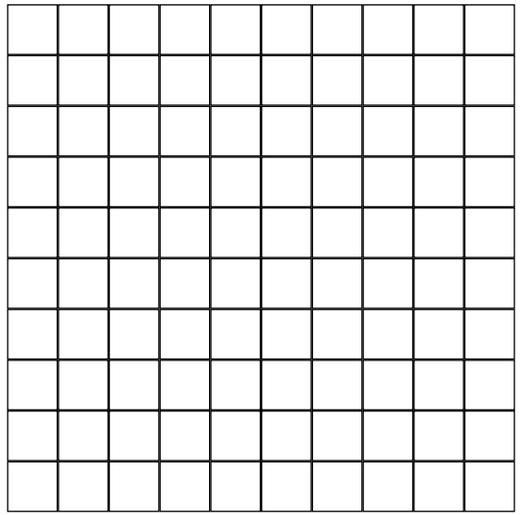
For each time step $i = 1, \ldots, n$:

    if Bernoulli(α):

        $X_i = X_{i-1} + (1, 0)$ [go right]

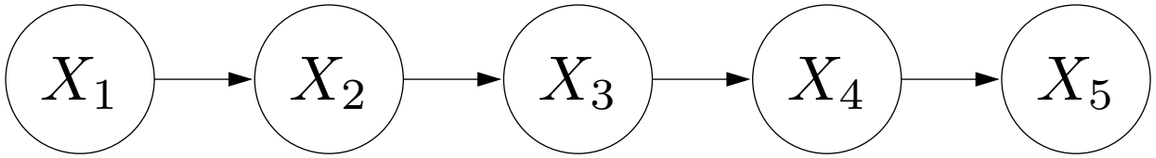
    else:

        $X_i = X_{i-1} + (0, 1)$ [go down]



(press ctrl-enter to save)

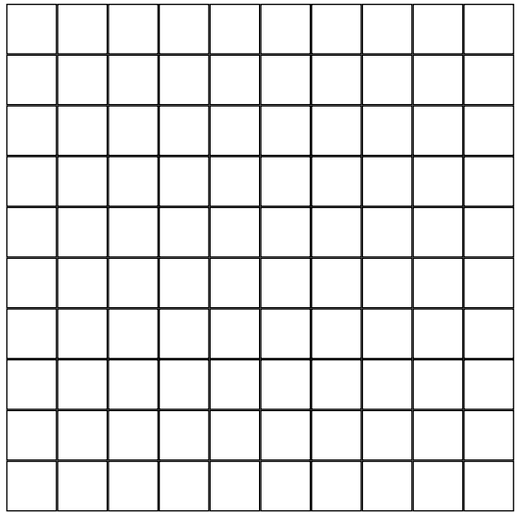
Run



- This is a more interesting example showcasing the convenience of probabilistic **programming**.
- In this program, we'll use a for loop, which allows us to compactly specify the distribution over an unboundedly large (n) set of variables.
- In the object tracking example, we define a program that generates the trajectory of an object. At each time step i, we take the previous X_{i-1} location and move it right with probability α and down with probability $1-\alpha$, yielding X_i.
- This program is a full specification of the local conditional distribution and thus the joint distribution!
- Try clicking [Run] to run the program. Each time a new assignment of $(X_1, \ldots, X_n)$ is chosen, and recall that the probability of the program generating an assignment is the probability under the joint distribution by definition.
- We can also draw the Bayesian network, which allows us to visualize the dependencies. Here, each X_i only depends on X_{i-1}. This chain-structured Bayesian network is called a **Markov model**. However, note that the graphical representation doesn't specify the local conditional distributions.

Probabilistic inference: example

Question: what are possible trajectories given **evidence** $X_{10} = (8, 2)$?



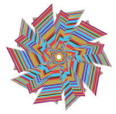
(press ctrl-enter to save)

Run

- Having used the program to define a joint distribution, we can now answer questions about that distribution.
- For example, suppose that we observe evidence $X_{10} = (8, 2)$. What is the distribution over the other variables?
- In the demo, we condition on the evidence and observe the distribution over all trajectories, which are constrained to go through $(8, 2)$ at time step 10.

Application: language modeling

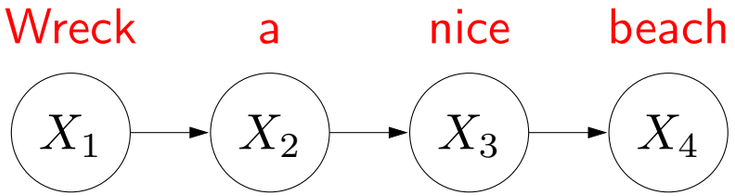
Can be used to score sentences for speech recognition or machine translation



Probabilistic program: Markov model

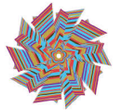
For each position $i = 1, 2, \ldots, n$:

Generate word $X_i \sim p(X_i \mid X_{i-1})$



- Now I'm going to quickly go through a set of examples of Bayesian networks or probabilistic programs and talk about the applications they are used for.
- A natural language sentence can be viewed as a sequence of words, and a language model assigns a probability to each sentence, which measures the "goodness" of that sentence.
- Markov models and higher-order Markov models (called n-gram models in NLP), were the dominant paradigm for language modeling before deep learning, and for a while, they outperformed neural language models since they were computationally much easier to scale up.
- While they could be used to generate text unconditionally, they were often used in the context of a speech recognition or machine translation system to score the fluency of the output.
- A Markov model generates each word given the previous word according to some local conditional distribution $p(X_i \mid X_{i-1})$ which we're not specifying right now.

Application: object tracking

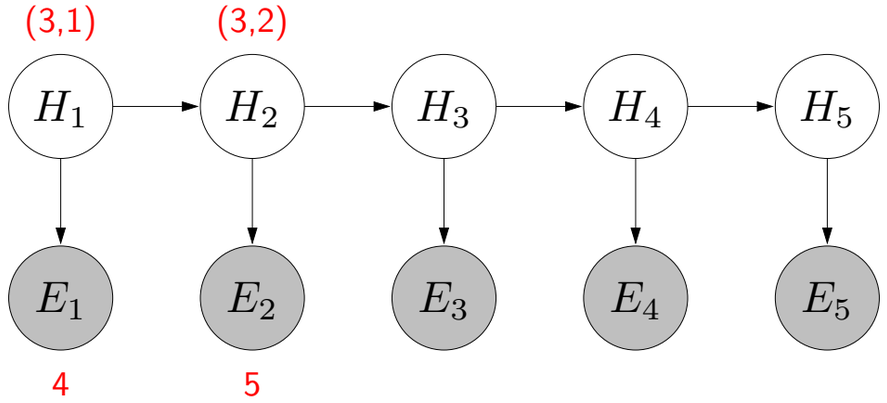


Probabilistic program: hidden Markov model (HMM)

For each time step $t = 1, \ldots, T$:

Generate object location $H_t \sim p(H_t \mid H_{t-1})$

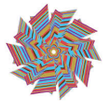
Generate sensor reading $E_t \sim p(E_t \mid H_t)$



Inference: given sensor readings, where is the object?

- Markov models are limiting because they do not have a way of talking about noisy evidence (sensor readings). They can be extended to hidden Markov models, which introduce a parallel sequence of observation variables.
- For example, in object tracking, H_t denotes the true object location, and E_t denotes the noisy sensor reading, which might be (i) the location H_t plus noise, or (ii) the distance from H_t plus noise, depending on the type of sensor.
- In speech recognition, H_t would be the phonemes or words and E_t would be the raw acoustic signal.

Application: multiple object tracking



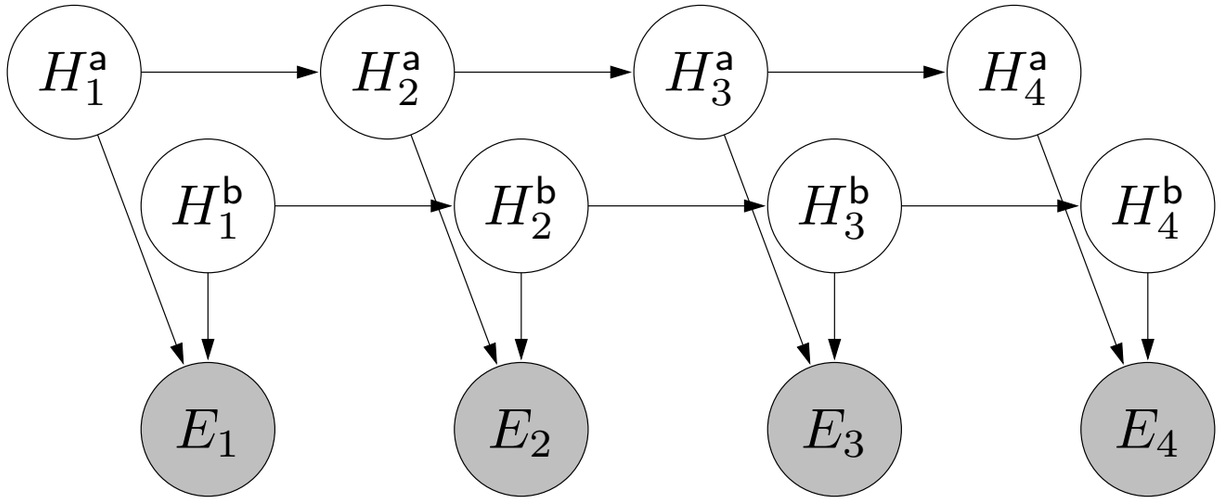
Probabilistic program: factorial HMM

For each time step $t = 1, \ldots, T$:

  For each object $o \in \{\mathsf{a}, \mathsf{b}\}$:

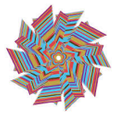
    Generate location $H_t^o \sim p(H_t^o \mid H_{t-1}^o)$

  Generate sensor reading $E_t \sim p(E_t \mid H_t^{\mathsf{a}}, H_t^{\mathsf{b}})$



- An extension of an HMM, called a **factorial HMM**, can be used to track multiple objects.
- We assume that each object moves independently according to a Markov model, but that we get one sensor reading which is some noisy aggregated function of the true positions.
- For example, E_t could be the set $\{H_t^{\mathsf{a}}, H_t^{\mathsf{b}}\}$, which reveals where the objects are, but doesn't say which object is responsible for which element in the set.

Application: document classification

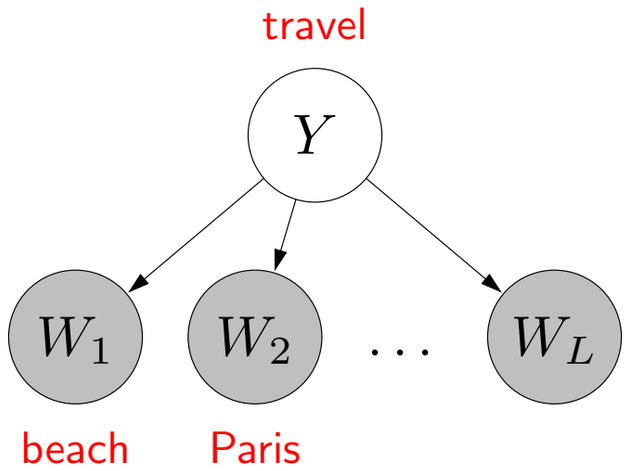


Probabilistic program: naive Bayes

Generate label $Y \sim p(Y)$

For each position $i = 1, \ldots, L$:

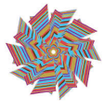
Generate word $W_i \sim p(W_i \mid Y)$



Inference: given a text document, what is it about?

- Naive Bayes is a very simple model which is often used for classification. For document classification, we generate a label and all the words in the document given that label.
- Note that the words are all generated independently, which is not a very realistic model of language, but naive Bayes models are surprisingly effective for tasks such as document classification.
- These types of models are traditionally called generative models as opposed to discriminative models for classification. Rather than thinking about how you take the input and produce the output label (e.g., using a neural network), you go the other way around: think about how the input is generated from the output (which is usually the purer, more structured form of the input).
- One advantage of using Naive Bayes for classification is that "training" is extremely easy and fast and just requires counting (as opposed to performing gradient descent).

Application: topic modeling



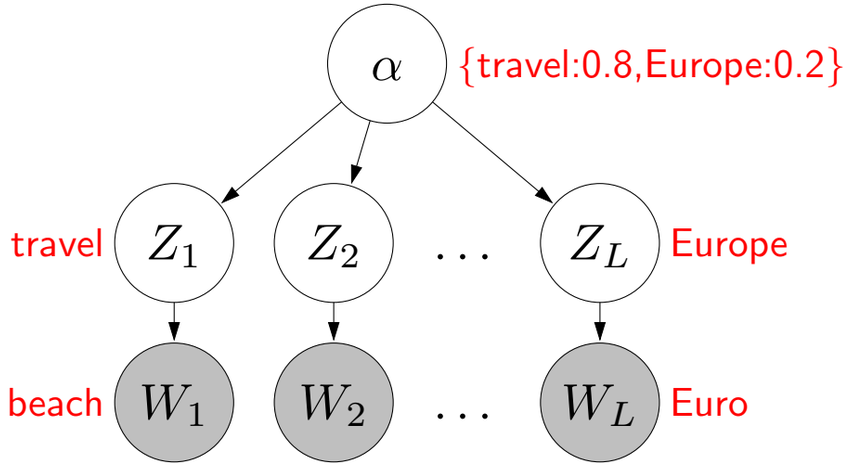
Probabilistic program: latent Dirichlet allocation

Generate a distribution over topics $\alpha \in \mathbb{R}^K$

For each position $i = 1, \ldots, L$:

Generate a topic $Z_i \sim p(Z_i \mid \alpha)$

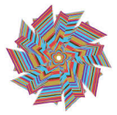
Generate a word $W_i \sim p(W_i \mid Z_i)$



Inference: given a text document, what topics is it about?

- A more sophisticated model of text is Latent Dirichlet Allocation (LDA), which allows a document to not just be about one topic or class (which was true in naive Bayes), but about multiple topics.
- Here, the distribution over topics α is chosen per document from a Dirichlet distribution. Note that α is a continuous-valued random variable. For each position, we choose a topic according to that per-document distribution and generate a word given that topic.
- Latent Dirichlet Alloction (LDA) has been very infuential for modeling not only text but images, videos, music, etc.; any sort of data with hidden structure. It is very related to matrix factorization.

Application: medical diagnosis



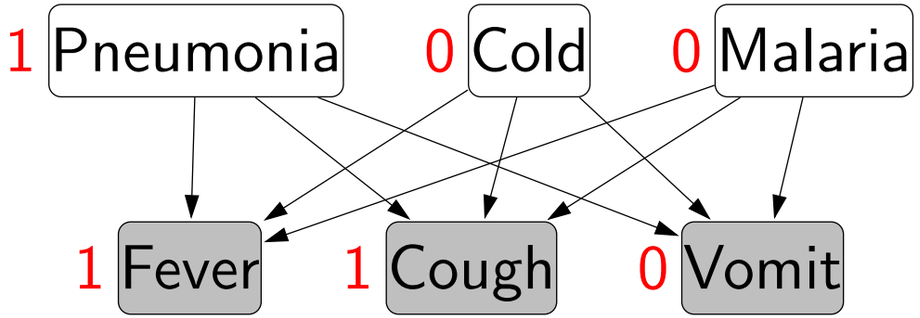
Probabilistic program: diseases and symptoms

For each disease $i = 1, \ldots, m$:

Generate activity of disease $D_i \sim p(D_i)$

For each symptom $j = 1, \ldots, n$:

Generate activity of symptom $S_j \sim p(S_j \mid D_{1:m})$



Inference: If a patient has some symptoms, what diseases do they have?

- We already saw a special case of this model. In general, we would like to diagnose many diseases and might have measured many symptoms and vitals.

Application: social network analysis

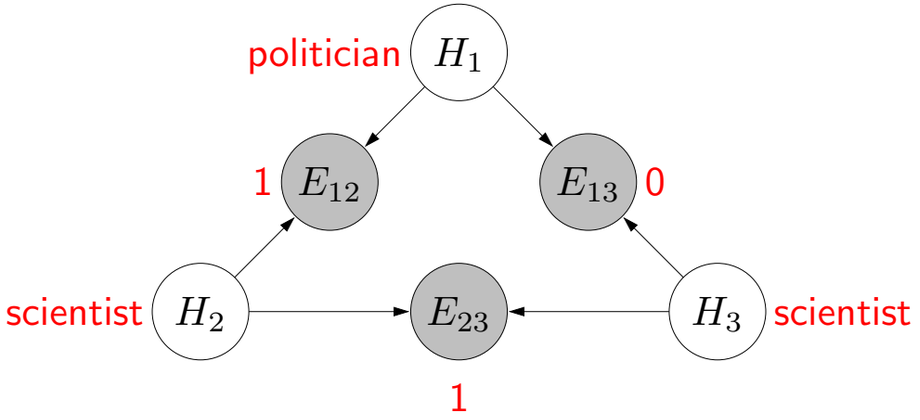
Probabilistic program: stochastic block model

For each person $i = 1, \ldots, n$:

  Generate person type $H_i \sim p(H_i)$

For each pair of people $i \neq j$:

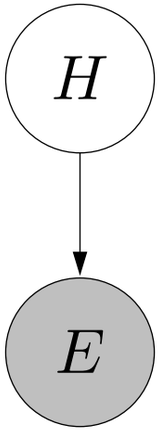
  Generate connectedness $E_{ij} \sim p(E_{ij} \mid H_i, H_j)$



Inference: Given a social network graph, what types of people are there?

- One can also model graphs such as social networks. A very naive-Bayes-like model is that each node (person) has a "type". Whether two people interact with each other (there is an edge beween the two people) is determined solely by their types and random chance.
- Note: there are extensions called mixed membership models which, like LDA, allow each person to have multiple types.

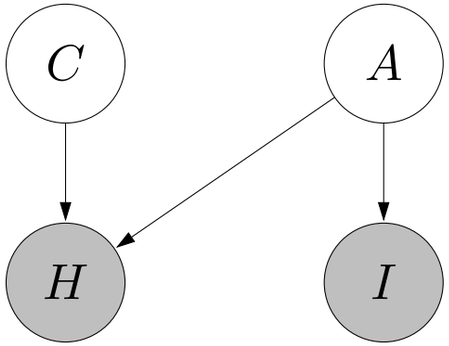
Summary



- Probabilistic program specifies a Bayesian network
- Many different types of models
- Common paradigm: come up with stories of how the quantities of interest (output) generate the data (input)
- Opposite of how we normally do classification!

- In summary, we've seen how we can define Bayesian networks (and therefore joint distributions) by writing down probabilistic programs.
- Using this powerful tool, we then did a whirlwind tour of lots of probabilistic programs that exist in the literature (though not often introduced under this general framework).
- The common theme of these probabilistic programs is that each attempts to produce **stories** of how certain quantities of interest H (e.g., actual location of an object) generate (or give rise to) observations E (e.g., usually noisy verions).
- After defining such a model, one can do probabilistic inference to compute $\mathbb{P}(H \mid E = e)$. Note that we can see how Bayesian networks allow us to handle heterogenous inputs (e.g., missing information). We can simply condition on partial evidence.
- Bayesian networks therefore provide quite a different paradigm compared to normal classification (e.g., neural networks). You have to think about going from the output to the input rather than input to output, which takes some getting used to.

Review: probabilistic inference



Question: $\mathbb{P}(C \mid H = 1, I = 1)$

Input

Bayesian network: $\mathbb{P}(X_1, \ldots, X_n)$

Evidence: $E = e$ where $E \subseteq X$ is subset of variables

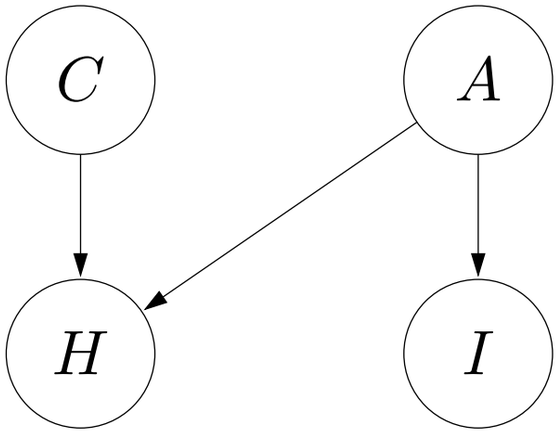
Query: $Q \subseteq X$ is subset of variables

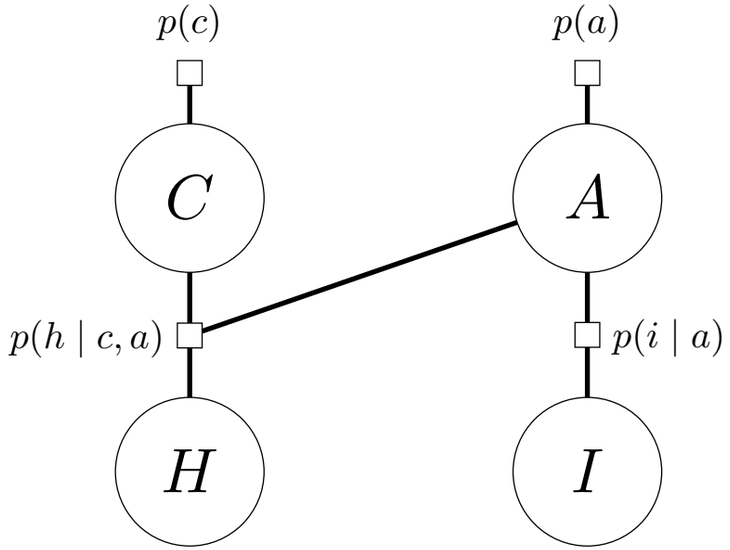
Output

$\mathbb{P}(Q \mid E = e)$ $\longleftrightarrow$ $\mathbb{P}(Q = q \mid E = e)$ for all values q

- Given the joint distribution representing your probabilistic database, you can answer all sorts of questions on it using probabilistic inference.
- Given a set of evidence variables and values, a set of query variables, we want to compute the probability of the query variables given the evidence, marginalizing out all other variables.

Reduction to Markov networks





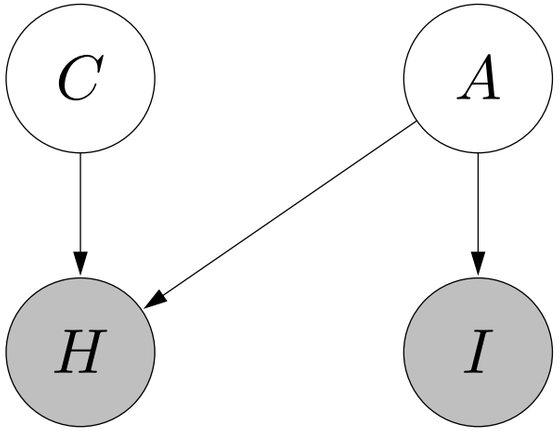
$$\mathbb{P}(C = c, A = a, H = h, I = i) = \frac{1}{Z} p(c)p(a)p(h \mid c, a)p(i \mid a)$$

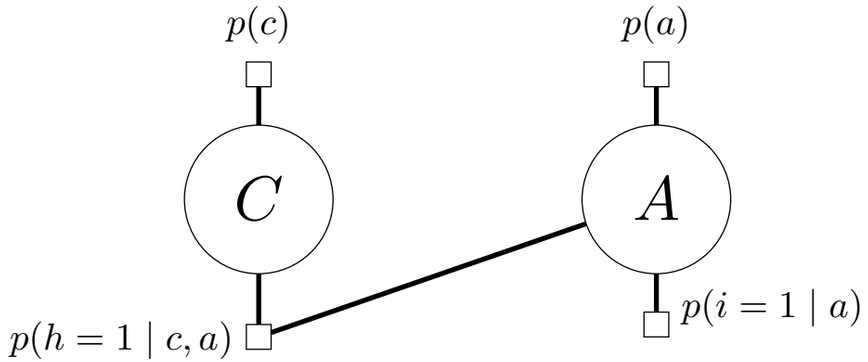
Bayesian network = Markov network with normalization constant $Z = 1$

Reminder: single factor that connects **all** parents!

- Our overarching strategy for performing inference in Bayesian networks is to convert them into Markov networks.
- Recall that the joint distribution is just the product of all the local conditional distributions. The local conditional distributions (e.g., $p(a \mid b, e)$) are all non-negative so they can be interpreted as simply factors in a factor graph.
- Recall that a Markov network defines the joint distribution as the product of all the factors divided by some normalization constant Z. But in this case, $Z = 1$ because the factors are local conditional distributions of a Bayesian network! Put it another way, Bayesian networks are just instances of Markov networks where the normalization constant $Z = 1$.
- It's important to remember that there is a single factor that connects all the parents. Don't let the directed graph in the Bayesian network deceive you into thinking that there are two factors, one per arrow, which is a common mistake.
- Now we can run any inference algorithm for Markov networks (e.g., Gibbs sampling) on this so-called Markov network and obtain quantities such as $\mathbb{P}(H = 1)$. But there is one important thing that's missing, which is the ability to condition on evidence...

Conditioning on evidence





Markov network:

$$\mathbb{P}(C = c, A = a \mid H = 1, I = 1) = \frac{1}{Z} p(c) p(a) p(h = 1 \mid c, a) p(i = 1 \mid a)$$

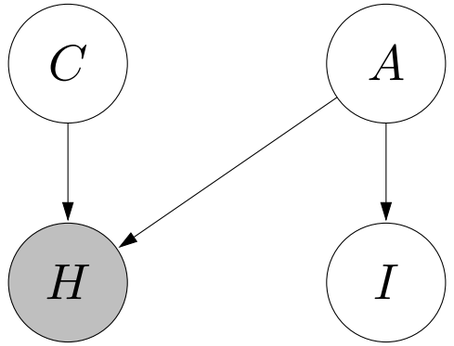
Bayesian network with evidence = Markov network with $Z = \mathbb{P}(H = 1, I = 1)$

Solution: run any inference algorithm for Markov networks (e.g., Gibbs sampling)!

[demo]

- Suppose we condition on evidence $H = 1$ and $I = 1$.
- We can define a new Markov network over the remaining variables (C and A) by simply plugging in the values to H and I. The normalization constant Z is the sum over all values of C and A, which is no longer 1, but rather the probability of the evidence $\mathbb{P}(H = 1, I = 1)$.
- To understand why this relationship holds, recall that the desired conditional probability is the joint probability over the marginal probability. The factors simply represent the joint probability, and thus the normalization constant must be the marginal probability.
- Now we can again run any inference algorithm for Markov networks (e.g., Gibbs sampling), and this allows us to do probabilistic inference in any Bayesian network.
- In the demo, we will run Gibbs sampling to compute $\mathbb{P}(C = 1 \mid H = 1, I = 1)$, and we see that it converges to the right answer (0.13).

Leveraging additional structure: unobserved leaves



Markov network:

$$\mathbb{P}(C = c, A = a, I = i \mid H = 1) = \tfrac{1}{Z} p(c)p(a)p(h = 1 \mid c, a)p(i \mid a),$$

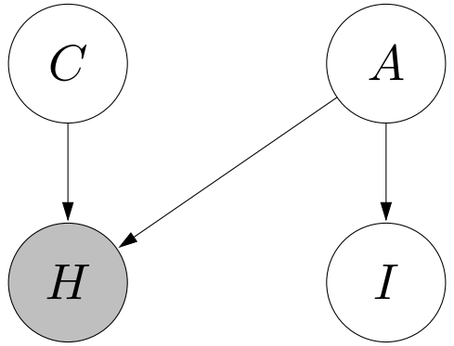
where $Z = \mathbb{P}(H = 1)$

Question: $\mathbb{P}(C = 1 \mid H = 1)$

Can we reduce the Markov network before running inference?

- We could stop there, but there are two more ways we can leverage the structure of Bayesian networks to optimize things a bit.
- Suppose we are now just conditioning on $H = 1$. As before we can form a Markov network over the remaining variables.
- But what if we knew we were only interested in $\mathbb{P}(C = 1 \mid H = 1)$?
- Is there a way to reduce the size of the Markov network before running inference?

Leveraging additional structure: unobserved leaves



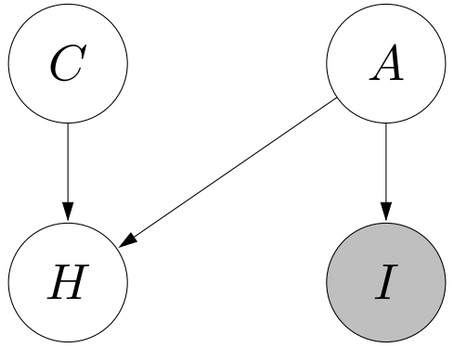
Markov network:

$$\begin{aligned}\mathbb{P}(C = c, A = a \mid H = 1) &= \textstyle\sum_i \mathbb{P}(C = c, A = a, I = i \mid H = 1) \\ &= \textstyle\sum_i \frac{1}{Z} p(c)p(a)p(h = 1 \mid c, a)p(i \mid a) \\ &= \frac{1}{Z} p(c)p(a)p(h = 1 \mid c, a) \textstyle\sum_i p(i \mid a) \\ &= \frac{1}{Z} p(c)p(a)p(h = 1 \mid c, a)\end{aligned}$$

Throw away any unobserved leaves before running inference!

- The answer is yes.
- Let us try marginalizing out I. We expand using the definition of marginal probability, definition of the Bayesian network, pushing the $\sum_i$ inwards past factors that don't depend on i, and noting that $\sum_i p(i \mid a) = 1$ by definition of local conditional distributions.
- But if we stare at the last equation, it is what we would have gotten if we had just ignored I in the first place!
- The general principle here is that marginalization of any unobserved leaf node produces 1, and thus all such nodes can be simply ignored. And we can keep on iterating this until all leaves are observed.
- This is practically very useful because it means that whenever we have a large Bayesian network, we might be able to remove large swaths of the network.
- This property establishes a bridge between marginalization (algebraic operations, usually involves hard work) with removal (graph operations, usually more intuitive).

Leveraging additional structure: independence



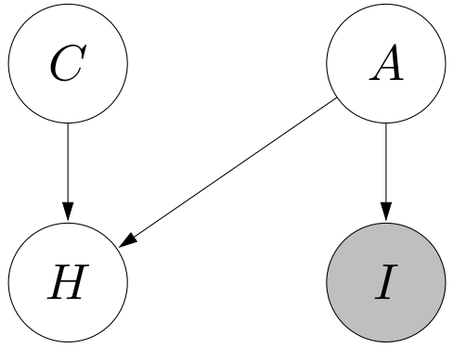
Markov network:

$$\begin{aligned}
\mathbb{P}(C = c \mid I = 1) &= \textstyle\sum_{a,h} \mathbb{P}(C = c, A = a, H = h \mid I = 1) \\
&= \textstyle\sum_{a,h} \frac{1}{Z} p(c) p(a) p(h \mid c, a) p(i = 1 \mid a) \\
&= \textstyle\sum_{a} \frac{1}{Z} p(c) p(a) p(i = 1 \mid a) \\
&= p(c) \textstyle\sum_{a} \frac{1}{Z} p(a) p(i = 1 \mid a) \\
&= p(c)
\end{aligned}$$

Throw away any disconnected components before running inference!

- There is another type of structure we can exploit, which is not specific to Bayesian networks, but shows up generally in Markov networks.
- Suppose we now condition on $I = 1$. Let us expand the marginal probability into the joint probability, expand into the local conditional probabilities, marginalize out the unobserved leaf H using the same idea we just discussed,
- Now at this point, C is completely disconnected from A and I. Algebraically, we can pull $p(c)$ out of the expression.
- We have this mess involving a and i, but this quantity does not depend on c so it is a constant. In this case, we know this constant must be 1 because both $p(c)$ and the LHS are probability distributions.
- So we can throw away any disconnected components. Note that it is advantageous to do this after removing all unobserved leaves, because removing those leaves can help disconnect the graph, as it did in this example.
- Now we have a Markov network, and we would run a standard inference algorithm on it. But in this case, it only has one factor which is already a local probability distribution, so we're done.

Summary



- Condition on evidence (e.g., $I = 1$)
- Throw away unobserved leaves (e.g., H)
- Throw away disconnected components (e.g., A and I)
- Define Markov network out of remaining factors
- Run your favorite inference algorithm (e.g., manual, Gibbs sampling)

- In summary, we tackled the problem of how to perform probabilistic inference in Bayesian networks, by reducing the problem to that of inference in Markov networks.
- To prepare the Markov network, we condition on the evidence (substitute the values into the factors), throw away any unobserved leaves, and throw away any disconnected components.
- Then we just define the Markov network over the remaining factors. If the resulting Markov network is small enough, we can do inference manually. Otherwise, we can run an algorithm like Gibbs sampling.