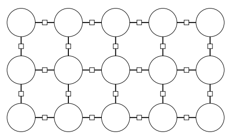
Markov Networks and Bayesian Networks I



Lecture

Markov Networks: Overview

Markov Networks: Gibbs Sampling

Bayesian Networks: Overview

Bayesian Networks: Definitions

- In this module, I will introduce Markov networks.

Course plan

Search problems
Markov decision processes
Adversarial games

Constraint satisfaction problems
Markov networks
Bayesian networks

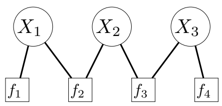
Reflex **States** **Variables** **Logic**

Low-level → High-level

Machine learning

- So far, we have introduced CSPs, the first of our variable-based models.
- Markov networks are the second type of variable-based model, which will connect factor graphs with **probability** and serve as a stepping stone on the way to Bayesian networks.

Review: factor graphs



Definition: factor graph

Variables:

$X = (X_1, \dots, X_n)$, where $X_i \in \text{Domain}_i$

Factors:

$f_1, \dots, f_m$, with each $f_j(X) \geq 0$

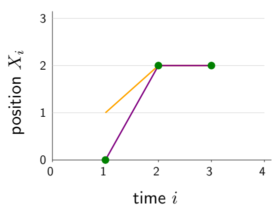
Definition: assignment weight

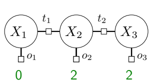
Each **assignment** $x = (x_1, \dots, x_n)$ has a **weight**:

$$\text{Weight}(x) = \prod_{j=1}^{m} f_j(x)$$

- Markov networks, like all variable-based models, are based on factor graphs.
- Recall that a factor graph contains a set of variables whose relationships are determined by a set of factors. For each assignment to all the variables, we have a non-negative weight, which captures how "good" a particular assignment is.
- Aside: Markov networks are also known as Markov random fields. They are typically defined as an undirected graph over variables, where we have a factor for each clique in the graph. But we use factor graphs to make the factors more explicit.

Example: object tracking





x_1	$o_1(x_1)$
0	2
1	1
2	0

x_2	$o_2(x_2)$
0	0
1	1
2	2

x_3	$o_3(x_3)$
0	0
1	1
2	2

$\lvert x_i - x_{i+1}\rvert$	$t_i(x_i, x_{i+1})$
0	2
1	1
2	0

[demo]

- Recall the object tracking example in which we observe noisy sensor readings 0, 2, 2.
- We have observation factors o_i that encourage the position X_i and the corresponding sensor reading to be nearby.
- We also have transition factors t_i that encourage the positions X_i and X_{i+1} to be nearby.

Maximum weight assignment

CSP objective: find the maximum weight assignment

$$\max_x \text{Weight}(x)$$

x_1	x_2	x_3	Weight(x)
0	1	1	4
0	1	2	4
1	1	1	4
1	1	2	4
1	2	1	2
1	2	2	8

Maximum weight assignment: $\{x_1 : 1, x_2 : 2, x_3 : 2\}$ (weight 8)

But this doesn't represent all the other possible assignments...

- In constraint satisfaction problems, we are interested in finding the maximum weight assignment.
- For the object tracking example, we show all the assignments with non-zero weight. The maximum weight assignment here is $\{x_1 : 1, x_2 : 2, x_3 : 2\}$ with weight 8.
- However, just returning this one assignment doesn't give us a sense of the alternatives, and how likely they are. In other words, we are not representing our **uncertainty**.

Definition

Definition: Markov network

A Markov network is a factor graph which defines a joint distribution over random variables $X = (X_1, \dots, X_n)$:

$$\mathbb{P}(X = x) = \frac{\text{Weight}(x)}{Z}$$

where $Z = \sum_{x'} \text{Weight}(x')$ is the normalization constant.

x_1	x_2	x_3	Weight(x)	$\mathbb{P}(X = x)$
0	1	1	4	0.15
0	1	2	4	0.15
1	1	1	4	0.15
1	1	2	4	0.15
1	2	1	2	0.08
1	2	2	8	0.31

$$Z = 4 + 4 + 4 + 4 + 2 + 8 = 26$$

Represents uncertainty!

- We now introduce **Markov networks** to capture the uncertainty over assignments.
- We've done most of the hard work by defining factor graphs, which endows each assignment $x = (x_1, \dots, x_n)$ with a weight Weight(x).
- We define the probability of an assignment x to be the fraction of weight relative to all assignments.
- Operationally, we first compute the **normalization constant** (also known as the partition function) Z, which is the sum of the weights over all assignments.
- Then we simply divide each weight by this normalization constant to get the probability.
- So the maximum weight assignment here only has 31% of the total probability.

Marginal probabilities

Example question: where was the object at time step 2 (X_2)?

Definition: Marginal probability

The marginal probability of $X_i = v$ is given by:

$$\mathbb{P}(X_i = v) = \sum_{x : x_i = v} \mathbb{P}(X = x)$$

Object tracking example:

x_1	x_2	x_3	Weight(x)	$\mathbb{P}(X = x)$
0	1	1	4	0.15
0	1	2	4	0.15
1	1	1	4	0.15
1	1	2	4	0.15
1	2	1	2	0.08
1	2	2	8	0.31

$\mathbb{P}(X_2 = 1) = 0.15 + 0.15 + 0.15 + 0.15 = 0.62$

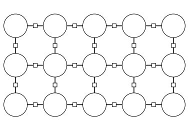
$\mathbb{P}(X_2 = 2) = 0.08 + 0.31 = 0.38$

Note: different than max weight assignment!

- The language of probability allows us to do more than just ask for the probability of complete assignments.
- It allows you to also ask for the **marginal probability** of partial assignments. In particular, we will focus on probability of single variables. This means asking for the probability of one variable X_i while marginalizing out others. Intuitively, while we don't ask for particular values on the marginalized variables, they still have a influence since factors still get multiplied into the weight.
- In the object tracking example, suppose we are interested in where the object was at time step 2 only, not caring about its position at other times.
- Then we would ask for the marginal probabilities $\mathbb{P}(X_2 = 1)$ and $\mathbb{P}(X_2 = 2)$. We compute these quantities by summing the probabilities of the complete assignment that match the condition on X_2.
- Interestingly, the result is that the object is 62% likely to be at position 1, even though the most likely complete assignment says the object is at position 2! Intuitively, this is because there are multiple assignments with $x_2 = 1$ with moderate weight (4), even though they don't have the maximum weight (4). There is kind of a "strength in numbers" phenomenon.
- The lesson is that you might get different answers depending on what you're asking.

Application: Ising model

Ising model: classic model from statistical physics to model ferromagnetism



$X_i \in \{-1, +1\}$: atomic spin of site i

$f_{ij}(x_i, x_j) = \exp(\beta x_i x_j)$ wants same spin

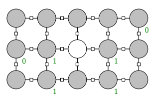
Samples as β increases:

Figure 2 from Perez (1998)

- A canonical example of a Markov network is the Ising model from statistical physics, which was developed by physicists in the 1920s to model ferromagnetism.
- The idea is that you have a large set of sites, each of which can either have an up or down spin.
- Assignments in which adjacent sites tend to have the same spin (resulting in a lower energy configuration) are favored, where the strength is given by β.
- Ising models are used to study phase transitions in physical systems. If $\beta = 0$, then the factors all evaluate to 1 independent of the assignment. Therefore, all assignments are equally likely, and there is simply no structure; every variable is completely random (probability $\frac{1}{2}$ up and probability $\frac{1}{2}$ down). As β increases, there starts to be more cohesion between sites, leading to larger blobs. As $\beta \to \infty$, equality becomes more like a hard constraint.
- Here we are showing samples from the Ising model (how we do this we will talk about in a future module).

Application: image denoising

Example: image denoising



- $X_i \in \{0, 1\}$ is pixel value in location i
- Subset of pixels are observed

$o_i(x_i) = [x_i = \text{observed value at } i]$

- Neighboring pixels more likely to be same than different

$t_{ij}(x_i, x_j) = [x_i = x_j] + 1$

- As another example, consider the problem of image denoising. This is one of the classic applications of Markov networks in computer vision before deep learning.
- In our stylized example, suppose we have a noisy image where only some of the pixels are observed and our goal is to recover our best guess of the clean image.
- We define a variable X_i for each pixel $i \in \{(1,1),(1,2),(1,3),\dots\}$.
- We then define an observation factor o_i on each pixel that is observed that constrains that pixel to be the observed value. For example, $o_{(1,1)}(x_i) = [x_i = 1]$.
- Then for every pair of neighboring pixels i and j (e.g., $i = (1,1)$ and $j = (2,1)$), we define a transition factor $t_{ij}(x_i, x_j)$ that encourages the pixel values to agree (both be 0 or both be 1). Weight 2 is given to those pairs which are the same and 1 if the pair is different.
- Note that the observation and transition factors should be reminiscent of the object tracking example, just in two dimensions. In general, having factors that incorporate external evidence (observations) and factors that incorporate internal consistency (transitions) is a common template for building Markov networks, and variable-based models more generally.

Summary

Markov networks = factor graphs + probability

- Normalize weights to get probability distribution
- Can compute marginal probabilities to focus on variables

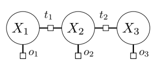
CSPs	Markov networks
variables	random variables
weights	probabilities
maximum weight assignment	marginal probabilities

- In summary, we have introduced Markov networks, which connect factor graphs with probability.
- The connection is very natural: factor graphs already provide a way of specifying non-negative weights over assignments, which gets us most of the way there. We then normalize the weights to make them sum to 1 to get a probability distribution.
- Once we have a joint probability distribution, we can compute marginal probabilities of individual (or subsets of) variables.
- We can compare CSPs with Markov networks. Variables become random variables, which means that they have probabilities associated with them. Instead of weights, we have their normalized versions, a.k.a., probabilities. The big difference is that instead of focusing on just finding the maximum weight assignment, which might be not representative of the full set of possibilities, the goal is to look at marginal probabilities.

Lecture

- In this module, I will present Gibbs sampling, a simple algorithm for approximately computing marginal probabilities.

Review: Markov networks



Definition: Markov network

A Markov network is a factor graph which defines a joint distribution over random variables $X = (X_1, \dots, X_n)$:

$$\mathbb{P}(X = x) = \frac{\text{Weight}(x)}{Z}$$

where $Z = \sum_{x'} \text{Weight}(x')$ is the normalization constant.

Objective: compute marginal probabilities $\mathbb{P}(X_i = v) = \sum_{x : x_i = v} \mathbb{P}(X = x)$

x_1	x_2	x_3	Weight(x)	$\mathbb{P}(X = x)$
0	1	1	4	0.15
0	1	2	4	0.15
1	1	1	4	0.15
1	1	2	4	0.15
1	2	1	2	0.08
1	2	2	8	0.31

$Z = 4 + 4 + 4 + 4 + 2 + 8 = 26$

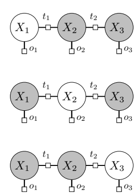
$\mathbb{P}(X_2 = 1) = 0.15 + 0.15 + 0.15 + 0.15 = 0.62$
$\mathbb{P}(X_2 = 2) = 0.08 + 0.31 = 0.38$

- Recall that a Markov network is defined by a factor graph, which provides a non-negative weight to each assignment x.
- If we compute the normalization constant Z and divide, then we get a probability distribution over joint assignments.
- For the object tracking example, we can compute the normalization factor to get joint probabilities. Note that this gives us a notion of uncertainty over the possible assignments.
- The main objective after defining a joint distribution is to compute marginal probabilities, which allow us to ask pointed questions about variables (e.g., X_2). Marginal probabilities are computed by summing the probabilities over all assignments satisfying the given condition.

Gibbs sampling

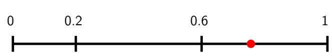
Algorithm: Gibbs sampling

Initialize x to a random complete assignment
Loop through $i = 1, \dots, n$ until convergence:
  Set $x_i = v$ with prob. $\mathbb{P}(X_i = v \mid X_{-i} = x_{-i})$
  (X_{-i} denotes all variables except X_i)
  Increment count$_i(x_i)$
Estimate $\hat{\mathbb{P}}(X_i = x_i) = \frac{\text{count}_i(x_i)}{\sum_v \text{count}_i(v)}$



Example: sampling one variable

Weight$(x \cup \{X_2 : 0\}) = 1$ prob. 0.2
Weight$(x \cup \{X_2 : 1\}) = 2$ prob. 0.4
Weight$(x \cup \{X_2 : 2\}) = 2$ prob. 0.4

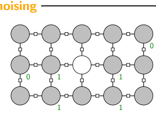


[demo]

- Now we present Gibbs sampling, a simple algorithm for approximately computing marginal probabilities. The algorithm follows the template of local search, where we change one variable at a time, but unlike Iterated Conditional Modes (ICM), Gibbs sampling is a randomized algorithm.
- Gibbs sampling proceeds by going through each variable X_i, considering all the possible assignments of X_i with some $v \in$ Domain$_i$, and setting $X_i = v$ with probability equal to the conditional probability of $X_i = v$ given everything else.
- To perform this step, we can rewrite this expression using laws of probability: $\mathbb{P}(X_i = v \mid X_{-i} = x_{-i}) = \frac{\text{Weight}(x \cup \{X_i : v\})}{Z \mathbb{P}(X_{-i} = x_{-i})}$, where the denominator is a new normalization constant. We don't need to compute it directly. Instead, we first compute the weight of $x \cup \{X_i : v\}$ for each v, and then normalize to get a distribution. Finally we sample a v according to that distribution.
- Along the way, for each variable X_i that we're interested in tracking, we keep a counter count$_i(v)$ of how many times we've seen $X_i = v$. These counts can be normalized at any time to produce an estimate $\hat{\mathbb{P}}(X_i = x_i)$ of the marginal probability.

Application: image denoising

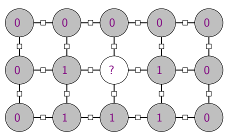
Example: image denoising



- $X_i \in \{0, 1\}$ is pixel value in location i
- Subset of pixels are observed
$o_i(x_i) = [x_i = \text{observed value at } i]$
- Neighboring pixels more likely to be same than different
$t_{ij}(x_i, x_j) = [x_i = x_j] + 1$

- Let's apply Gibbs sampling to the image denoising application.
- Recall that we have a grid of pixels, a subset of which are observed, and we wish to fill in the remaining pixels.
- The unknown pixels are represented by a variable X_i for each pixel i. We have observation factors can constrain the observed pixels, and transition factors that encourage neighboring pixels to agree.

Gibbs sampling for image denoising



$$t_{ij}(x_i, x_j) = [x_i = x_j] + 1$$

Scan through image and update each pixel given rest:

v	weight	$\mathbb{P}(X_i = v \mid X_{-i} = x_{-i})$
0	$2 \cdot 1 \cdot 1 \cdot 1$	0.2
1	$1 \cdot 2 \cdot 2 \cdot 2$	0.8

- Let us compute the Gibbs sampling update. We go through each pixel X_i and try to update its value.
- For the given example, we consider both values 0 and 1, and multiply exactly the transition factors that depend on that value. Assume there are no observation factors here.
- The factor returns 2 if the pixel values agree and 1 if they disagree.
- We then normalize the weights to form a distribution and then sample v.
- Intuitively, the neighbors are all trying to pull $X_{(3,2)}$ towards their values, and 0.8 reflects the fact that the pull towards 1 is stronger.

Image denoising demo

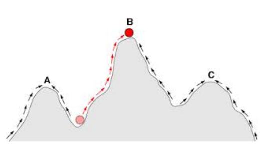
[see web version]

- Let's actually play around with Gibbs sampling for image denoising in the browser.
- Try playing with the demo by modifying the settings to get a feeling for what Gibbs sampling is doing. Each iteration corresponds to resampling each pixel (variable).
- When you hit ctrl-enter for the first time, red and black correspond to 1 and 0, and white corresponds to unobserved.
- `showMarginals` allows you to either view the assignments produced or the marginals estimated from the particles (this gives you a smoother probability estimate of what the pixel values are).
- If you decrease `missingFrac` to 0.3, the problem becomes easier, and the reconstruction looks pretty good.
- If you set `coherenceFactor` to 10, then there will be coupling between neighboring variables, and you'll see sharper lines, although the reconstruction is not perfect.
- If you set `icm` to true, we will use local search rather than Gibbs sampling, which produces very bad solutions.

Search versus sampling

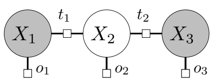
Iterated Conditional Modes	Gibbs sampling
maximum weight assignment	marginal probabilities
choose best value	sample a value
converges to local optimum	marginals converge to correct answer*

*under technical conditions (sufficient condition: all weights positive), but could take exponential time



- It is instructive to compare Gibbs sampling with its cousin, Iterated Conditional Modes (ICM). Both iteratively go through the variables and tries to update each one of them holding the others fixed.
- Recall that the goals are different: ICM tries to find the maximum weight assignment while Gibbs sampling is trying to compute marginal probabilities.
- Accordingly, ICM will choose the value for a variable X_i with the highest weight, whereas Gibbs sampling will use the weights to form a distribution to sample from.
- ICM converges to local optimum, an assignment that can't be improved on. Note that Gibbs sampling is stochastic so in some sense never converges. However, the estimates of the marginal probabilities do in fact converge under some technical assumptions. The simplest sufficient condition if all weights are positive, but it also suffices that the probability of Gibbs sampling going between any two assignments is positive. A major caveat is that the time it takes to converge can be exponential in the number of variables.
- Advanced: Gibbs sampling is an instance of a Markov Chain Monte Carlo (MCMC) algorithm which generates a sequence of particles $X^{(1)}, X^{(2)}, X^{(3)}, \dots$. A Markov chain is irreducible if there is positive probability of getting from any assignment to any other assignment (now the probabilities are over the random choices of the sampler). When the Gibbs sampler is irreducible, then in the limit as $t \to \infty$, the distribution of $X^{(t)}$ converges to the true distribution $\mathbb{P}(X)$. MCMC is a very rich topic which we will not talk about very much here.

Summary



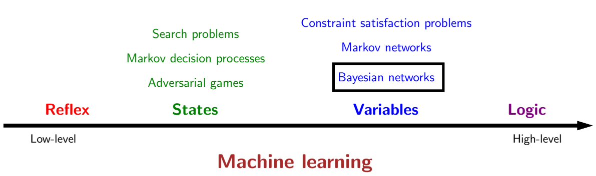
- Objective: compute marginal probabilities $\mathbb{P}(X_i = x_i)$
- Gibbs sampling: sample one variable at a time, count visitations
- More generally: Markov chain Monte Carlo (MCMC) powerful toolkit of randomized procedures

- In summary, we are trying to compute the marginal probabilities of a Markov network.
- Gibbs sampling allows us to do this by exploring all the assignments randomly, but very carefully controlled probabilities, so that the visitation frequencies of various values converge to the right answer.
- Gibbs sampling is part of a beautiful and rich set of tools for using randomness to do inference on Markov networks, which I encourage you to check out.

Lecture

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

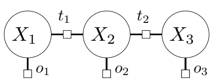
Course plan

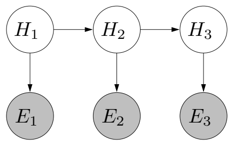


- We have talked about two types of variable-based models.
- In constraint satisfaction problems, the objective is to find the maximum weight assignment given a factor graph.
- In Markov networks, we use the factor graph to define a joint probability distribution over assignments and compute marginal 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

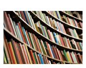




Markov networks	Bayesian networks
arbitrary factors	local conditional probabilities
set of preferences	generative process

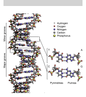
- Before defining Bayesian networks, it is helpful to compare and contrast Markov networks and Bayesian networks at a high-level.
- Both define a joint probability distribution over assignments, and in the end, both are backed by factor graphs.
- But the way each approaches modeling is different. In 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 environments, the idea is intriguing.
- 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 **heterogeneously** 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 (homogeneous). With Bayesian networks, the types of inputs one can handle can be **heterogeneous** (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 generally 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: Bayesian Networks

Modeling

Definitions

Probabilistic programming

Inference	**Learning**
Probabilistic inference	Supervised learning
Forward-backward	Smoothing
Particle filtering	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**.

Lecture

Markov Networks: Overview

Markov Networks: Gibbs Sampling

Bayesian Networks: Overview

Bayesian Networks: Definitions

- In this module, I'll present the formal definition of Bayesian networks, give a few examples, and talk about an important property called explaining away.

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 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 (probabilistic) 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

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

(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:

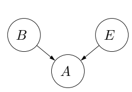
$$\mathbb{P}(E, B, A)$$

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:

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

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

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

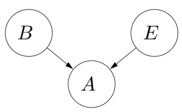
[demo]

News flash: earthquakes decrease burglaries!*

*This is not a causal statement!

- We multiply all the local conditional distributions together to produce the joint distribution. Recall that this probability 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.

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

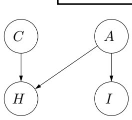
Note: happens even if causes are independent!

- 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. Further, 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:

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

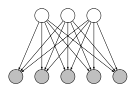
Questions:

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

[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 $X = (X_1, \dots, X_n)$ 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:

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

- 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 $x_{\text{Parents}(i)}$ 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 $p(x_i \mid x_{\text{Parents}(i)})$) 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: $\mathbb{P}(X_1, \dots, X_n)$

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

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

Output

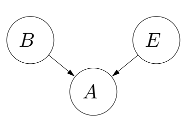
$\mathbb{P}(Q \mid E = e) \longleftrightarrow \mathbb{P}(Q = q \mid E = e)$ for all values q

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, i.e., 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.

Summary: Markov and Bayesian Networks I

- Markov Networks: Factor graphs + Probability
- Gibbs sampling is an algorithm for estimating marginal probabilities
- Bayesian Networks, represent generative processes, related to Factor graphs and Markov Networks
- Bayesian Networks Definitions: explaining away
- Next: Inference in Bayesian networks

- In summary, we started be defining Markov Networks, which connect factor graphs with probability, useful for capturing uncertainty.
- Next, we covered Gibbs sampling for computing marginal probabilities of a Markov network
- Then, we discussed Bayesian Networks which define a generative process represented by a directed graph
- Finally, we discussed definitions in Bayesian Networks, including probabilistic inference and explaining away.
- Next Lecture, we will continue with some inference techniques for Bayesian networks, including the forward backward algorithm and particle filtering