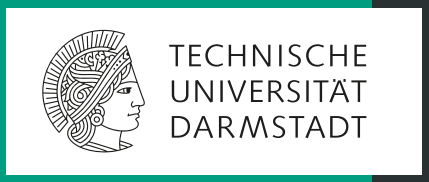
Probabilistic Graphical Models

Summary

Fabian Damken

November 8, 2023



Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

Probabilistic graphical models (PGMs) are one of the most exciting development in machine learning, knowledge representation, artificial intelligence, statistics, electrical engineer, and much more research areas that has evolved in the last two decades. This summary covers a wide range of topics of this field, including Bayesian networks, Markov networks, factor graphs, junction trees, parameter learning, structure learning, exact inference, variable elimination, approximate inference, sampling, MCMC, Gibbs, loopy belief propagation, Bayesian learning, missing data, EM, temporal models, hidden Markov models, forward-backward propagation, Viterbi, Baum-Welch, DBNs, and many more.

Some of the fundamental questions of graphical models are:

- What are model types? What does a model mean/imply/assume? How to incorporate conditional independencies to reduce representation size?
- How to answer questions/queries with the model? What is the probability of X given some observations? What is the most likely observation of a phenomenon? What decision to make?
- How to learn the model from data? What model fits to the data? What are right/good parameters? How to obtain the structure of a model?

This summary starts off, after some foundations, by introducing Bayesian networks and how to perform inference (in discrete domains). Later, undirected graphical models (Markov random fields), continuous random variables, and approximate inference is covered.

2 Foundations

This chapter covers fundamental concepts of probability theory and machine learning that are required for the later chapters. Note that not all relevant concepts of probability theory are covered.

2.1 Probability Theory

This section covers some very important concepts of probability theory, however, one should already be familiar with some basics like probability measures, density functions, joint distributions, marginalization, etc.[1]

One note on notation: whenever a sum represents a marginalization over some random variable X, it is written as

$$P(Y) = \sum_{X}^{\text{marg.}} P(X, Y) \coloneqq \sum_{x \in \text{val}(X)} P(X = x, Y)$$

for brevity.

2.1.1 (Conditional) Independence

The most important concept leveraged in probabilistic graphical models is (conditional) independence of random variables. Two random variables X and Y are *statistically independent* if knowing either does not change the belief/probability of the other, i.e.,

$$P(X \mid Y) = P(X) \quad \text{and} \quad P(Y \mid X) = P(Y).$$

This is equivalent to the definition of independence, $P(X, Y) = P(X)\,P(Y)$. Independence is denoted $X \perp Y$ and is a symmetric properties. A milder property is *conditional* independence, i.e., two random variables X and Y are independent if Z is given:

$$P(X \mid Y, Z) = P(X \mid Z) \quad \text{and} \quad P(Y \mid X, Z) = P(Y \mid Z).$$

Again, this property can be written as $P(X, Y \mid Z) = P(X \mid Z)\,P(Y \mid Z)$ by the chain rule. Conditional independency is denoted $X \perp Y \mid Z$.

The following properties hold and can be useful for some proofs later on:

$$\begin{aligned}
X \perp Y \mid Z &\iff Y \perp X \mid Z && \text{(Symmetry)} \\
X \perp (Y, W) \mid Z &\implies (X \perp Y \mid Z) \wedge (X \perp W \mid Z) && \text{(Decomposition)} \\
X \perp (Y, W) \mid Z &\implies X \perp Y \mid (Z, W) && \text{(Weak Union)} \\
(X \perp W \mid (Y, Z)) \wedge (X \perp Y \mid Z) &\implies X \perp (Y, W) \mid Z && \text{(Contraction)} \\
(X \perp Y \mid (W, Z)) \wedge (X \perp W \mid (Y, Z)) &\implies X \perp (Y, W) \mid Z && \text{(Intersection)}
\end{aligned}$$

Note that the intersection property only holds for positive distributions (i.e., $P(X = x) > 0$ for all x).

[1] Take a look the chapter of statistics fundamentals of `https://fabian.damken.net/summaries/cs/elective/vc/statml/statml-summary.pdf`.

Monty Hall Problem

2.1.2 Inference

Statistical *inference* is concerned with computing all kinds of statistical properties given a model, e.g., the probability of an event. This section covers same fundamental concepts needed to understand the later chapters.

Information Theory

Information theory is trying to quantify how much information is encoded in some distribution $P(X)$. The central measure is *entropy*:

$$H_P(X) = \mathbb{E}\big[\log(1/P(X))\big] = \sum_{x \in \mathrm{val}(X)} P(X) \log \frac{1}{\log P(X)} = - \sum_{x \in \mathrm{val}(X)} P(X) \log P(X)$$

If the logarithm is of base two, the entropy encodes how much bits are required *on average* to encode X when X follows the distribution $P(X)$. Similarly, *conditional entropy* can be defined as

$$H_P(X \mid Y) = \mathbb{E}\big[\log(1/P(X \mid Y))\big] = H_P(X, Y) - H_P(X)$$

where $H_P(X, Y)$ is the joint entropy over X and Y. Like for probabilities, a *chain rule of entropies* is derivable:

$$H_P(X, Y, Z) = H_P(X) + H_P(Y \mid X) + H_P(Z \mid X, Y).$$

To quantify (in)dependency between two variables X and Y, the *mutual information*

$$I_P(X; Y) = H_P(X) - H_P(X \mid Y)$$

can be used. This quantity is symmetric and is zero if and only if X and Y are independent.

2.1.3 Potentials

A *potential* is an alternative way of representing (conditional) probabilities aside from conditional probability tables (CPTs). A potential $\phi_{X,Y,Z}$ is is a function that maps each configuration $(x, y, z) \in \mathrm{val}(X) \times \mathrm{val}(Y) \times \mathrm{val}(Z)$ to a non-negative real number. The set of random variables targeted by a potential is its *domain*, i.e., $\mathrm{dom}\, f_{X,Y,Z} = \{X, Y, Z\}$. Note that a (conditional) probability distribution is a special case of potentials where the potential is normalized. Vice versa, a potential can always be normalized into a CPT. This is illustrated in Figure 2.1.

Similar to CPTs, potentials can be multiplied by pairing up the entries and marginalized by summing up the corresponding entries. Compared to CPTs, it is not necessary to normalize a potential into a probability distribution afterwards, easing some calculations.

Potentials will come in handy later on when covering inference in junction trees (section 5.4).

$P(X = x \mid Y = y, Z = z)$	$x = \mathfrak{f}$	$x = \mathfrak{t}$
$(Y, Z) = (\mathfrak{t}, \mathfrak{t})$	0.8	0.2
$(Y, Z) = (\mathfrak{t}, \mathfrak{f})$	0.5	0.5
$(Y, Z) = (\mathfrak{t}, \mathfrak{t})$	0.2	0.8
$(Y, Z) = (\mathfrak{t}, \mathfrak{f})$	0.7	0.3

$\longleftrightarrow$

X	Y	Z	$\phi_{X,Y,Z}$	$\tilde{\phi}_{X,Y,Z}$
$\mathfrak{t}$	$\mathfrak{t}$	$\mathfrak{t}$	0.8	8
$\mathfrak{t}$	$\mathfrak{t}$	$\mathfrak{f}$	0.2	2
$\mathfrak{t}$	$\mathfrak{f}$	$\mathfrak{t}$	0.5	5
$\mathfrak{t}$	$\mathfrak{f}$	$\mathfrak{f}$	0.5	5
$\mathfrak{f}$	$\mathfrak{t}$	$\mathfrak{t}$	0.2	2
$\mathfrak{f}$	$\mathfrak{t}$	$\mathfrak{f}$	0.8	8
$\mathfrak{f}$	$\mathfrak{f}$	$\mathfrak{t}$	0.7	7
$\mathfrak{f}$	$\mathfrak{f}$	$\mathfrak{f}$	0.3	3

Figure 2.1: Comparison of a CPT (left) and the corresponding potential (right). The rightmost column in the potential $\tilde{\phi}$ is equivalent to ϕ as it can be normalized accordingly.

3 Bayesian Networks

Bayesian networks provide a compact representation for exponentially-large distribution by exploiting conditional independencies. When representing a joint distribution $P(X_1, X_2, \ldots, X_n)$ with $|\mathrm{val}(X_i)| = k$ for simplicity, the CPT has $k^n - 1$ entries[1]! But lots of information is redundant in the joint if some of the variables exhibit independencies. Assume, for instance, that all subsets $\mathcal{X}, \mathcal{Y} \subseteq \{X_1, X_2, \ldots, X_n\}$ are independent ($\mathcal{X} \perp \mathcal{Y}$). Then the joint can be written as

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i),$$

requiring only $n(k-1) \in \mathcal{O}(n)$ parameters! This is linear in the number of random variables! While this assumption seems rather simplistic, it is used with success in practice: this is the assumption of the naive Bayed classifier covered in section 3.1.

3.1 The Naive Bayes Model

In the *naive Bayes model*, it is assumed that some feature random variables $\{X_1, X_2, \ldots, X_n\}$ are all conditionally independent given the class C; $\forall \mathcal{X}, \mathcal{Y} \subseteq \{X_1, X_2, \ldots, X_n\} : \mathcal{X} \perp \mathcal{Y} \,|\, C$. The joint distribution is therefore simply

$$P(X_1, X_2, \ldots, X_n, C) = P(X) \prod_{i=1}^{n} P(X_i). \tag{3.1}$$

Using Bayesian networks, the independencies are represented using a graph as shown in Figure 3.1.

To classify a new instance $\boldsymbol{x}$ (a vector composed of the individual instances of $X_1, X_2, \ldots, X_n$), the class $c \in \mathrm{val}(C)$ with the highest posterior probability is selected:

$$c_{\mathrm{MAP}} = \operatorname*{arg\,max}_{c \in \mathrm{val}(C)} P(c \,|\, \boldsymbol{x}) = \operatorname*{arg\,max}_{c \in \mathrm{val}(C)} \frac{P(\boldsymbol{x} \,|\, c)\, P(c)}{P(\boldsymbol{x})} = \operatorname*{arg\,max}_{c \in \mathrm{val}(C)} P(\boldsymbol{x} \,|\, c)\, P(c) = \operatorname*{arg\,max}_{c \in \mathrm{val}(C)} P(X) \prod_{i=1}^{n} P(X_i).$$

[1] The -1 comes from the requirement of the probability being normalized, hence the probability of the "last" configuration is implicit.

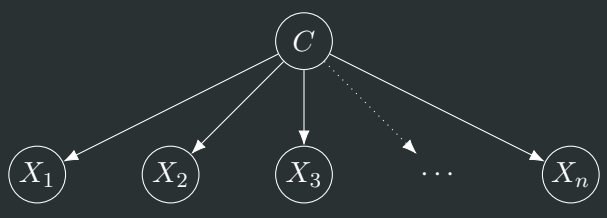


Figure 3.1: Bayesian network for the naive Bayes classifier.

3.1.1 Maximum Likelihood Parameter Estimation

To learn the probabilities required to perform classification using the naive Bayes model, the straightforward approach is to use maximum likelihood estimates, i.e., the frequencies in the data:

$$\hat{P}(C = c_j) = \frac{N(C = c_j)}{N} \qquad \hat{P}(X_i = x_{ik} \mid c_j) = \frac{N(X_i = x_{ik}, C = C_j)}{N(C = c_j)}.$$

here, $N(C = c_j)$ is the number of data points where the class is c_j and $N(X_i = x_{ik}, C = C_j)$ is the number of data points with class c_j and value x_{ik} in the i-th feature. Using the identity function $\mathbb{1}[\psi]$ that is one iff ψ is true and zero otherwise, these quantities can be expressed as

$$N(C = c_j) = \sum_{((x_1,x_2,\ldots,x_n),c)\in\mathcal{D}} \mathbb{1}[c = c_j] \qquad N(X_i = x_{ik}, C = C_j) = \sum_{(x_1,x_2,\ldots,x_n,c)\in\mathcal{D}} \mathbb{1}[x_i = x_{ik}, c = c_j]$$

where $\mathcal{D} \subseteq \text{val}(X_1) \times \text{val}(X_2) \times \cdots \times \text{val}(X_n) \times \text{val}(C)$ is the data set. Note that the formality of the data set and values is quite rigorous in this section. From now on, the notation will be abused from time to time for brevity.

This estimation method poses a major challenge: if no instances have been observed of a given feature/class-configuration, the evidence is zero and no matter how high other evidence is in the joint (3.1), the joint will be zero. This problem can be reducing by *smoothing* the distributions to avoid overfitting,

$$\hat{P}(X_i = x_{ik} \mid c_j) = \frac{N(X_i = x_{ik}, C = C_j) + m\big(N(X_i = x_{ik})/N\big)}{N(C = c_j) + m},$$

where m controls the extend of smoothing and is a hyper-parameter.

3.1.2 Application

Even though the assumption of all features being conditionally independent is often false, the naive Bayes classifier turns out to be quite effective in practice. While not being the best classification method, is usually serves as a good and dependable baseline. *If* the assumption holds, the Bayes classifier is optimal (and can be tuned for different misclassification costs, for example).

Also, it is very fast: learning is done in a single pass over the data (by counting) and testing is linear in the number of attributes and data size. For the same reason (small CPTs), it requires very little storage, making it suitable for on-device classification.

3.2 Definition and Independence Assumptions

After some introductory treatment of the naive Bayes classifier and its associated Bayesian network, this section now focuses on a more rigorous treatment of the independencies that are actually encoded in a Bayesian network and a formal definition of what a Bayesian network is.

The key idea for Bayesian networks is the incorporation of independence assumptions into their structure. A formal Bayesian network consists of the following components:

- Set of random variables $\{X_1, X_2, \ldots, X_n\}$.
- Directed acyclic graph (DAG) encoding the (in-) dependencies.

- Conditional probability table for each random variable, $P\big(X_i \mid \mathrm{Pa}(X_i)\big)$. $\mathrm{Pa}(X_i)$ contains all parents of the random variable in the given DAG.

Then, the joint distribution is given by

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P\big(X_i \mid \mathrm{Pa}(X_i)\big) \tag{3.2}$$

and the local Markov assumption holds. The *local Markov assumption* states that a random variable X_i is independent of all its non-descendants $\mathrm{ND}(X_i)$ given its parents $\mathrm{Pa}(X_i)$:

$$X_i \perp \mathrm{ND}(X_i) \mid \mathrm{Pa}(X_i).$$

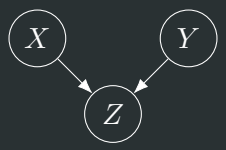
Let $\mathrm{Ch}(X_i)$ be the children of X_i, then X_i's non-descendants are all random variables that are not in $\mathrm{Des}(X_i)$, X_i's descendants:

$$\mathrm{ND}(X_i) = \{X_1, X_2, \ldots, X_n\} \setminus \mathrm{Des}(X_i), \qquad \mathrm{Des}(X_i) = \mathrm{Ch}(X_i) \cup \bigcup_{X_j \in \mathrm{Ch}(X_i)} \mathrm{Des}(X_j).$$

This is a recursive definition for which $\mathrm{Des}(X_i) = \emptyset$ holds iff X_i does not have children.

3.2.1 "Explaining Away" / Berkson's Paradox

Berkson's paradox describe the phenomenon that, by the local Markov assumption, for a network like



the independence $X \perp Y$ holds but $X \perp Y \mid Z$ does not. This implies that information can flow from X to Y iff Z is given, i.e., having evidence on either X or Y is already sufficient to explain the evidence on Z:

$$P(X = x \mid Z = z, Y = y) \leq P(X = x \mid Z = z).$$

3.2.2 Representation Theorem

All this work on independencies raise the question of whether it is actually worth it: what distributions can be represented using a Bayesian network? And what networks are required to represent a distribution? Also, what other independencies apart from the local Markov assumption are encoded in a network? Some independencies can certainly be derived using the relations presented in subsection 2.1.1. Let P be the real distribution and let $I(P)$ be the independencies encoded in the true distribution. Similarly, let $I_\ell(G)$ be the local independencies (induces by the local Markov assumption) encoded in a graph G. Then the key assumption of Bayesian networks is that

$$I_\ell(G) \subseteq I(P).$$

That is, the local independencies only capture such that are actual present in the true distribution. If this relation holds, G is said to be an *I-map* (independency map) of P.

Theorem The *representation theorem* states that G if an I-map of P if and only if P factorizes according to G via (3.2).

Figure 3.2: Illustration of a Bayesian network capturing more independencies than induced by the local Markov assumption. In this BN, the local Markov assumption encodes $I_\ell(G) = \{ B \perp A \,|\, A, C \perp (A, B) \,|\, B, D \perp (A, B, C) \,|\, C \}$. However, also $A \perp D \,|\, C$ is a captured independency, for example.

Proof The proof is split into proofing "if" and "only if".

"if":

"only if": Assume, w.l.o.g., a topological ordering $X_1, X_2, \ldots, X_n$ where $j < i$ for all children X_j of some random variable X_i. Applying the chain rule to the joint yields $P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_1 \,|\, X_1, X_2, \ldots, X_{i-1})$. By the topological ordering, $\mathrm{Pa}(X_1) \subseteq \{X_1, X_2, \ldots, X_{i-1}\}$ and hence, using the local Markov assumption, $P(X_i \,|\, X_1, X_2, \ldots, X_{i-1}) = P\big(X_1 \,|\, \mathrm{Pa}(X_i)\big)$. Therefore, P factorized according to (3.2).

3.2.3 "Adding Edges Does Not Hurt"

Theorem Let G be an I-map for P. Then any DAG G' having the same directed edges as G, then G' is also an I-map of P.

From this theorem it follows that

- G' is strictly more expressive than G by capturing more independencies, and
- adding edges to G still results in an I-map.

Proof Idea Let $I_\ell(G) \subseteq I(P)$. To show that $I_\ell(G') \subseteq I(P)$ holds, it is enough to show $I_\ell(G') \subsetneq I_\ell(G)$. Note that is enough to show that this property holds if a single edge is added as adding two edges can be understood as two modifications, resulting in a third graph G''. It therefore follows from $I_\ell(G'') \subsetneq I_\ell(G') \subseteq I_\ell(G)$ that also $I_\ell(G'') \subsetneq I_\ell(G)$.

Let X and Y be some random variables of G such that $Y \notin \mathrm{Pa}_G(X)$. Then adding an edge $Y \to X$ only removes local independencies, but does not induce new ones.

3.3 Encoded Independencies

So far, only local independencies have been considered. However, a Bayesian network encodes even more dependencies by applying the algebra of conditional independencies (subsection 2.1.1; see Figure 3.2 for an example).

The findings of this section are summarized in Figure 3.3 which gives an overview of the different variants to getting the set of independencies (read this section first before inspecting the figure).

3.3.1 Dependency Structures

In a three-node BN, four types of structures are possible. The first, second, and third are indirect causal and evidential effects and a common cause, respectively:

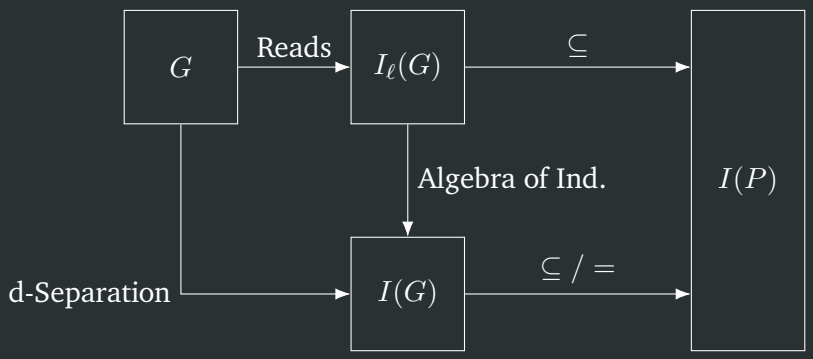


Figure 3.3: Illustration of different methods to capture the independencies of a distribution. Note that $I(G)$ and $I(P)$ are equal (i.e., P is faithful to G) for almost all P that factor over G.

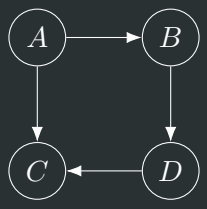
In both cases, the conditional independence $X \perp Y \mid Z$ holds, but *not* the simple independence $X \perp Y$.

The last and most special structure is a v-structure, where two variables have a common effect:

In this case, the independencies are inverted: the simple independence $X \perp Y$ holds, but $X \perp Y \mid Z$ *does not*. This is the same effect as the "explain away" phenomenon / Berkson's paradox (subsection 3.2.1).

3.3.2 d-Separation

d-Separation (dependence separation) is a principled way of finding all independencies modeled by a Bayesian network. The main component are trails. A *trail* is a non-cyclic path $X'_1 \leftrightarrow X'_2 \leftrightarrow \cdots \leftrightarrow X'_k$ from one random variable X'_1 to another X'_k. Note that a trail can also be along an edge in opposite order (i.e., the trail itself is undirected) and that every edge and note can only be visited once. For example, the network



has the following trails:

- $A \to B,\ A \to C \leftarrow D \leftarrow B;\ A \to C,\ A \to B \to D \to C;\ A \to B \to D,\ A \to C \leftarrow D$
- $B \leftarrow A,\ B \to D \to C \leftarrow A;\ B \leftarrow A \to C,\ B \to D \to C;\ B \to D,\ B \leftarrow A \to C \leftarrow D$
- $C \leftarrow A,\ C \leftarrow D \leftarrow B \leftarrow A;\ C \leftarrow A \to B,\ C \leftarrow D \leftarrow B;\ C \leftarrow D,\ C \leftarrow A \to B \to D$
- $D \leftarrow B \leftarrow A,\ D \to C \leftarrow A;\ D \leftarrow B,\ D \to C \leftarrow A \to B;\ D \to C,\ D \leftarrow B \leftarrow A \to C$

Note that as trails are themselves undirected, the graph itself really has half as many trails, but they are kept for consistency.

d-Separation now uses these trails to extract independencies. Consider any trail $X'_1 \leftrightarrow X'_2 \leftrightarrow \cdots \leftrightarrow X'_k$ between two variables X'_1 and X'_k this trail is *active* if for each triplet $X'_{i-1} \leftrightarrow X'_i \leftrightarrow X'_{i+1}$, the following holds:

- for $X_{i-1} \to X_i \to X_{i+1}$, X_i is *not observed*.
- for $X_{i-1} \leftarrow X_i \leftarrow X_{i+1}$, X_i is *not observed*.
- for $X_{i-1} \leftarrow X_i \to X_{i+1}$, X_i is *not observed*.
- for $X_{i-1} \to X_i \leftarrow X_{i+1}$, X_i or one of its descendants *is observed*.

Theorem Two variables X and Y are independent given a set of variables $\mathcal{Z}$ if there is no active trail between X and Y if $\mathcal{Z}$ are observed. They are said to be *d-separated*. The set of independencies obtained via d-separation is called $I(G)$.

Soundness

Theorem d-Seperation is *sound*. That is, if P factorizes over G, then $I(G) \subseteq I(P)$.
This means that d-separation only captured independencies that the true distributions exhibits.

Proof

Completeness

A distribution P is said to be *faithful* if it does not have independencies that cannot be read from G.

Theorem For almost all distributions P that factorize over G, $I(G) = I(P)$. "Almost all" means that the possible distributions are a set of measure zero parametrizations of the CPTs.

Hence, the Bayesian network is usually sufficient for capturing all independence properties of the distribution! But this only holds for complete independence, but there might be context-specific independencies that are not captured.

Proof

3.3.3 Context-Specific Independence (CSI)

As already said, d-separation only captures complete independencies, i.e., once for which

$$\forall x, \in \text{val}(X), y \in \text{val}(Y), z \in (Z) : (X = x \perp Y = y \,|\, Z = z)$$

holds but context-specific independencies,

$$\exists x, \in \text{val}(X), y \in \text{val}(Y), z \in (Z) : (X = x \perp Y = y \,|\, Z = z),$$

are not captured.

One option for representing CSIs are Tree CPDs. *Tree CPDs* encode a distribution $P\big(X \mid \text{Pa}(X)\big)$ using a decision-tree like structure, where the paths are an assignment of (a subset of) $\text{Pa}(X)$ and leaves represent the distributions given the assignments on the path. This way, representation size can be drastically reduced when CSIs are present.

Another variant where CSIs occur is determinism. While determinism already makes the CPT sparse, it has even greater influences on inference as propagating the zeros often leads to a bunch of new zeros.

3.3.4 The Bayes' Ball Algorithm

The *Bayes' ball algorithm* is a algorithmic approach to applying d-separation.

3.4 SOTA Modal

Current state-of-the-art (SOTA) models are often characterized by richness in their local structures (determinism, CSI), massive sizes (thousand of variables), and high connectivity (treewidth, see **??**). These developments are enabled by high-level modeling tools (relational and first-order logic), the overall advancement in machine learning, and new application areas (e.g., bioinformatics and sensor networks).

In these big models, it is a must to exploit local and relational structure!

4 Inference

Inference is concerned with answering statistical queries in a probabilistic model. As this summary is on probabilistic model, it is concerned with inference in graphical models. However, inference in Bayesian networks is—in general—pretty hopeless aka. NP-hard (even approximate inference). In practice, however, special structures can be exploited producing many effective (approximate) inference algorithms. This chapter covers exact inference whereas approximate inference is covered later in chapter 8.

A general probabilistic query is $P(X \mid e)$ computing some probability given some evidence e. The straightforward way to compute the conditional is to directly use its definition,

$$P(X \mid e) = \frac{P(X, e)}{P(e)}.$$

This requires computing the joint distribution and marginalizing out X to get the normalization factor. In general, such a computation has exponential complexity in the number of random variables!

For all complexity arguments in the following sections, let k be the number of configurations of a single random variable, $k = |\mathrm{val}(X)|$, and assume that all random variables have the same number of values.

4.1 Chain Models

The simplest BN to perform efficient exact inference in is a chain model

that factors as

$$P(X_1, X_2, \ldots, X_n) = P(X_1) \prod_{i=2}^{n} P(X_i \mid X_{i-1}).$$

However, to computing $P(X_n)$ still requires an exponential number of operations, $\mathcal{O}(k^n)$:

$$\begin{aligned} P(X_n) &= \sum_{X_1}^{\text{marg.}} \sum_{X_2}^{\text{marg.}} \cdots \sum_{X_{n-1}}^{\text{marg.}} P(X_1, X_2, \ldots, X_{n-1}, X_n) \\ &= \sum_{X_1}^{\text{marg.}} \sum_{X_2}^{\text{marg.}} \cdots \sum_{X_{n-1}}^{\text{marg.}} P(X_1)\, P(X_2 \mid X_1) \cdots P(X_n \mid X_{n-1}). \end{aligned}$$

Instead, it is advisable to reverse the sums and to pull the independent factors out of the marginalization. This corresponds to iteratively computing $P(X_1)$, $P(X_2)$, …, $P(X_n)$:

$$P(X_{i+1}) = \sum_{X_i}^{\text{marg.}} P(X_{i+1} \mid X_i)\, P(X_i).$$

This process is linear in the number of random variables and quadratic in the number of values of a random variable, $\mathcal{O}(nk^2)$! Hence, at least for simple chain models, inference is tractable.

4.2 Variable Elimination

Variable elimination extends the idea introduced in the chain models of pulling factors out of the marginalization sums. To incorporate evidence e, the corresponding RVs are simply set to the respective values and marginalization over these variables is removed. See section 8.2.1 in *Data Mining and Machine Learning*[1] for a more thorough treatment of this topic. By convention, the factor that appears by eliminating a variable X that still depends on other RVs $Y_1, Y_2, \ldots, Y_k$ is named $f_X(Y_1, Y_2, \ldots, Y_k)$. Pseudocode for variable elimination is shown in algorithm 1.

One of the most important steps in variable elimination is pruning the non-active variables (line 2). This means removing all random variables that are independent of all variables in $\boldsymbol{X} \cup \boldsymbol{E}$ as the distributions over these random variables simply reduce to a multiplication with 1, not changing the results. However, this can reduce the number of required operations to a tractable amount.

The most important aspect determining the efficiency of variable elimination is, intuitively, the choice of the elimination ordering (this is covered in more detail in section 4.5). For relatively simple networks like polytrees[2], inference is linear in *representation* size (i.e., in the number of entries across all CPTs)! In general, however, variable elimination exponential in the number of variables in the intermediate factors—hence its complexity is dominated by the largest intermediate factor. This again is discussed in more detail in section 4.5.

While variable elimination on a general Bayesian network is still exponential in time w.r.t. the number of variables in the intermediate factors, it is linear (in representation size!) in polytree networks.

Also note that variable elimination can also directly be applied to potentials. It is not required that the original factors actually represent CPTs.

Algorithm 1: Variable Elimination

Input: Bayesian network with RVs $X_1, X_2, \ldots, X_n$ and query $P(\boldsymbol{X} \mid \boldsymbol{E} = \boldsymbol{e})$
Output: Probability $P(\boldsymbol{X} \mid \boldsymbol{E} = \boldsymbol{e})$

1 Instantiate evidence $\boldsymbol{E} = \boldsymbol{e}$.
2 Prune non-active variables (w.r.t. $\boldsymbol{X} \cup \boldsymbol{E}$), giving $X_1, X_2, \ldots, X_{\tilde{n}}$, $\tilde{n} \leq n$.
3 Choose an ordering $X_1, X_2, \ldots, X_{\tilde{n}}$.
// Initialize factors:
4 $\mathcal{F}^{(0)} \leftarrow \left\{ f_i = P\big(X_i \mid \mathrm{Pa}(X_i)\big) : i = 1, 2, \ldots, \tilde{n} \right\}$
5 **for** $i = 1, 2, \ldots, \tilde{n}$ **do**
6 **if** $X_i \in \boldsymbol{X} \cup \boldsymbol{E}$ **then**
7 **skip**
8 Collect factors $f_1, f_2, \ldots, f_k \in \mathcal{F}^{(i-1)}$ containing X_i.
  // Generate a new factor g by eliminating X_i:
9 $g = \sum_{X_i}^{\text{marg.}} \prod_{i=1}^{k} f_j$
  // Replace factors in factor set:
10 $\mathcal{F}^{(i)} \leftarrow \left(\mathcal{F}^{(i-1)} \setminus \{f_1, f_2, \ldots, f_k\}\right) \cup \{g\}$
11 Read only remaining factor $P(\boldsymbol{X}, \boldsymbol{E} = \boldsymbol{e})$ from $\mathcal{F}^{(\tilde{n})}$.
12 Normalize $P(\boldsymbol{X}, \boldsymbol{E} = \boldsymbol{e})$ to get $P(\boldsymbol{X} \mid \boldsymbol{E} = \boldsymbol{e})$.

[1] `https://fabian.damken.net/summaries/cs/elective/iws/dmml/dmml-summary.pdf`
[2] A polytree is the directed variant of a tree. That is, a DAG is a polytree if its underlying undirected graph is a tree.

4.3 Abductive Inference

So far, only the calculation of a-posterior beliefs/probabilities has been covered. That is, computing queries of the form $P(\boldsymbol{X} \mid \boldsymbol{E} = \boldsymbol{e})$ (where $\boldsymbol{E}$ is a, possibly empty, set of evidence). While this type of inference is useful in many cases (e.g., to perform predictions by computing the probability of an outcome, diagnosing a disease by computing the probability given some symptoms, ...), it is also desirable to perform *abductive* inference. *Abductive inference* is concerned with finding a configuration of a set of random variables that "best" explains the evidence at hand. This can be done in two fashions:

- Most Probable Explanation (MPE): finding the most probable configuration of *all* variables in a Bayesian network given some evidence.
- Maximum A Posteriori (MAP): finding the most probable configuration of a *subset* of variables in a Bayesian network given some evidence.

Note that being able to perform either does not necessarily solve the other (i.e., the MPE cannot be found by individually maximizing all probabilities for all variables and the MAP cannot be found by taking the corresponding subset of the MPE). It is said that MPE and MAP are *not consistent*.

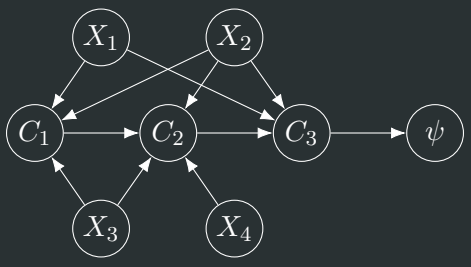
Finding (approximate) MPEs is quite straightforward by replacing the sums (marginalizations) in variable elimination by $\max$'s. In a second backward pass, the sums are replace by $\arg\max$-operators to find the configuration. A pseudocode version of this approach is given in algorithm 2.

4.4 Complexity of Conditional Queries

In general, inference on conditional query $P(\boldsymbol{X} \mid \boldsymbol{E} = \boldsymbol{e})$ is NP-hard. This can be shown by reducing 3-SAT to inference in a Bayesian network. 3-SAT is concerned with the question whether a propositional formula (in conjunctive normal form) with at most three literals per clause is satisfiable. For instance, is the following formula satisfiable (with boolean $X_1, X_2, \ldots, X_4$)?

$$\psi = (\neg X_1 \vee X_2 \vee X_3) \wedge (X_2 \vee \neg X_3 \vee X_4) \wedge (X_1 \vee \neg X_2) \tag{4.1}$$

This problem is known to be NP-hard. This problem can be expressed using a Bayesian network:



The CPTs of C_1, C_2, and C_3 then encode the clauses of (4.1) and ψ summarizes them into a single random variable. While it seems possible to just direct all random variables into ψ, this would make the reduction non-polynomial, invalidating the proof of NP-hardness. By putting a prior $P(X_i = \mathrm{t}) = 0.5$ over all (boolean) RVs, satisfiability can be queried as $P(\psi = \mathrm{t}) > 0$. Hence, if conditional inference would be tractable, 3-SAT would be too. By contradiction, this shows that conditional inference is NP-hard. □

However, while NP-hard is already pretty hard, it can also be shown that inference is #P-complete involving counting all satisfying configurations.

Even though this shows that inference is hard in the general case, it can be solved for problems with special structures (which are very common). For these, efficient (exact) inference algorithms exist.

Algorithm 2: Most Probable Explanations via Variable Elimination

Input: Bayesian network with RVs $X_1, X_2, \ldots, X_n$ and evidence $\boldsymbol{E} = \boldsymbol{e}$
Output: Most probable explanation $\boldsymbol{x}^*$

Instantiate evidence $\boldsymbol{E} = \boldsymbol{e}$.
Prune non-active variables (w.r.t. $\boldsymbol{X} \cup \boldsymbol{E}$), giving $X_1, X_2, \ldots, X_{\tilde{n}}$, $\tilde{n} \leq n$.
Choose an ordering $X_1, X_2, \ldots, X_{\tilde{n}}$.
// Initialize factors:
$\mathcal{F}^{(0)} \leftarrow \{f_i = P(X_i \mid \mathrm{Pa}(X_i)) : i = 1, 2, \ldots, \tilde{n}\}$
for $i = 1, 2, \ldots, \tilde{n}$ **do**
    if $X_i \in \boldsymbol{E}$ **then**
        skip
    Collect factors $f_1, f_2, \ldots, f_k \in \mathcal{F}^{(i-1)}$ containing X_i.
    // Generate a new factor g by maximizing over X_i:
    $g = \max_{X_i} \prod_{i=1}^{k} f_j$
    // Replace factors in factor set:
    $\mathcal{F}^{(i)} \leftarrow \left(\mathcal{F}^{(i-1)} \setminus \{f_1, f_2, \ldots, f_k\}\right) \cup \{g\}$
// Initilize vector of MPE-values:
$\boldsymbol{x}^* \in \mathrm{val}(X_1) \times \mathrm{val}(X_2) \times \cdots \times \mathrm{val}(X_n)$
for $i = \tilde{n}, \tilde{n} - 1, \ldots, 1$ **do**
    if $X_i \in \boldsymbol{E}$ **then**
        // Copy evident value to result vector:
        $x_i^* \leftarrow e_i$
    else
        // Compute $\arg\max$ over X_i, incorporating the evidence and already determined values:
        $x_i^* \leftarrow \arg\max_{X_i} \prod_{f \in \mathcal{F}^{i-1}} f(\boldsymbol{x}_{1:i-1}^*, \boldsymbol{e})$
Set pruned variables $X_{\tilde{n}+1}, X_{\tilde{n}+2}, \ldots, X_n$ so arbitrary values.

(a) Bayesian Network

(b) Moral Graph

Figure 4.1: Example for creating a moral graph (right) from a Bayesian network (left) by moralizing. All edges are turned into undirected edges and a moral link (depicted in red) has to be introduced between the "unmarried" parents X_2 and X_3 or X_5.

(a) About to eliminate X_3. The affected node and edges are depicted in red.

(b) After eliminating X_3. The introduces fill-ins are depicted in red.

Figure 4.2: Example for eliminating a variable in a moral graph.

4.5 Variable Elimination in Moral Graphs

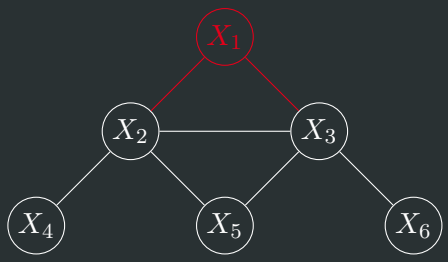
The induces moral graph of a Bayesian network is an undirected graph where the members of each clique correspond to the domain of a potential in the Bayesian network. To extract the moral graph from a Bayesian network, it might be necessary to introduce moral links to fulfill this (defining) criterion. A *moral link* is be introduced between unconnected parents of a node[3]. The rest of the moral graph is then given as the skeleton (i.e., the underlying undirected graph of a DAG) of the network. An example is given in Figure 4.1. More details on why this moral link is necessary are covered in section 5.1.

As variable elimination introduces new factors/potentials into the joint probability, these changes have to be reflected in the moral graph after the elimination. If an elimination step introduces a potential with random variables that have not yet been together in a potential, a *fill-in* is introduced to the moral graph representing this new connection. An example of this process is shown in Figure 4.2. An elimination ordering is said to be *perfect* if it does not introduce any fill-ins (see Figure 4.3 for an example with the same graph used in Figure 4.2).

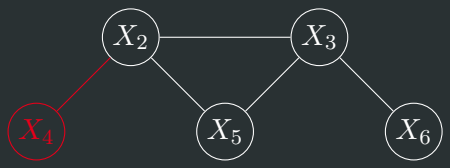
4.5.1 Complexity

As already pointed out in section 4.2, the driving factor of the complexity of VE is the number of variables in the intermediate factors. Hence, for VE on moral graphs, the complexity is determined by the number of variables in the domains of the potentials producing during the elimination. This leads to a set of domains

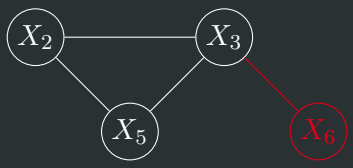
[3]The term "moralizing" has quite outdated origins and refers to the unmorality of "unmarried" parents.



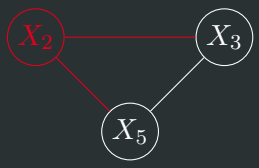
(a) About to eliminate X_1.



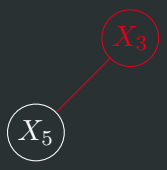
(b) After eliminating X_1, about to eliminate X_4.



(c) After eliminating X_4, about to eliminate X_6.



(d) After eliminating X_6, about to eliminate X_2.



(e) After eliminating X_2, about to eliminate X_3.

(f) After eliminating X_3, elimination finished.

Figure 4.3: Example for a perfect elimination sequence executed in a moral graph. The variable that should be eliminated is always depicted in red and is eliminated in the upcoming graph.

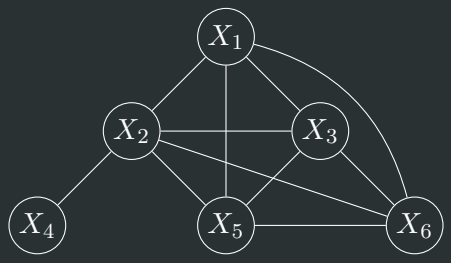


Figure 4.4: Induced graph for the imperfect elimination ordering $(X_3, X_4, X_6, X_1, X_2, X_5)$.

where all sets that are subsets of other sets can be removed (as they do not affect the complexity). For the elimination ordering shown in Figure 4.3, these are, in order of appearance, the following sets:

$$\begin{aligned} & \big\{\{X_1, X_2, X_3\},\ \{X_2, X_4\}, \{X_3, X_6\}, \{X_2, X_3, X_5\}, \{X_3, X_5\}, \{X_5\}\big\} \\ \longrightarrow \quad & \big\{\{X_1, X_2, X_3\},\ \{X_2, X_4\}, \{X_3, X_6\}, \{X_2, X_3, X_5\}\big\} \end{aligned}$$

It turns out that all perfect elimination orderings (i.e., those that do not produce fill-ins), have the same set of potential domains. Also, all perfect elimination orderings ending some random variable X are *optimal* in terms of computing $P(X)$.

For the imperfect elimination ordering $(X_3, X_4, X_6, X_1, X_2, X_5)$ (for which the first elimination is shown in Figure 4.2), the following sets of potentials are created:

$$\begin{aligned} & \big\{\{X_1, X_2, X_3, X_5, X_6\},\ \{X_2, X_4\}, \{X_1, X_2, X_5, X_6\}, \{X_1, X_2, X_5\}, \{X_2, X_5\}, \{X_5\}\big\} \\ \longrightarrow \quad & \big\{\{X_1, X_2, X_3, X_5, X_6\},\ \{X_2, X_4\}\big\} \end{aligned}$$

Intuitively, the number of items of this clique are closely related to the complexity of VE (a small clique set corresponds to large connectivity in the graph, hence large factors).

Induced Graph and Treewidth

To further formalize the aforementioned idea, the notion of an induced graph is introduced. The *induced graph* of some moral graph is the graph including all connections of the clique set, i.e., the moral graph with all fill-ins inserted. Hence, the moral graph is also a sub-graph of the induced graph. For the perfect elimination ordering of Figure 4.3, the induced graph is also the moral graph (as no fill-ins were added). However, for the imperfect elimination sequence $(X_3, X_4, X_6, X_1, X_2, X_5)$, the induced graph has more connections as shown in Figure 4.4.

It is clear that the size of the maximum clique differs for the perfect and imperfect elimination sequence. This quantity (minus one) is called the *treewidth* of a graph and is called *induced* treewidth for a Bayesian network combined with an elimination ordering (which highly influences the treewidth of the moral graph). Hence, finding a good elimination ordering is equivalent to minimizing the treewidth of the induced graph.

(Poly-) Trees

A direct consequence of the tight coupling between the complexity of VE and the treewidth is that inference on a "tree" Bayesian network (one in which every node has at most one parent) is linear in the number of variables (as the treewidth is 1 for a tree)! Finding an elimination ordering is also straightforward: by starting from the leaves, each factor only involves two RVs. It is also guaranteed that a tree has a leave, so the ordering cannot stall.

Query	Graph Type	
	General	Low Treewidth
Probabilistic Inference	#P-Complete	Easy
Most Probable Explanation (MPE)	NP-Complete	Easy
Maximum A Posteriori (MAP)	NP^{PP}-Complete	NP-Hard

For approximate probabilistic inference, hardness depends on the error:
Absolute Error, $|\hat{P} - P| \leq \epsilon$: NP-Hard for $\epsilon < 0.5$
Relative Error, $|\hat{P} - P|/P \leq \epsilon$: NP-Hard for $\epsilon > 0$

Table 4.1: Summary of complexities for various inference queries.

For polytree Bayesian networks (ones of which the skeleton is a tree but a node may have more than one parent), the treewidth is 2 and inference is linear in CPT size (i.e., representation size of the network). Opposed to a tree BN, the treewidth *is not* 1 as moral links have to be added between "unmarried" parents.

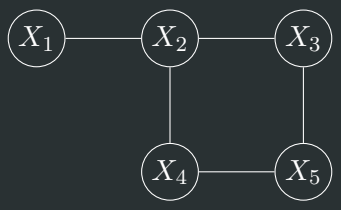
General Networks

For general networks, the treewidth is usually greater than 1. In fact, for any graph that has a cycle, the treewidth is at least 2. Form graph theory, it is known that finding an optimal ordering minimizing the treewidth is NP-hard, so again, in general, inference is intractable. However, there are some neat heuristics that can be applied for finding "good" (but not necessarily optimal) orderings. There also exist algorithm that—if the graph has low treewidth—find the optimal ordering in polynomial time. Also, incorporating CSIs can reduce the complexity to be tractable even for graphs with large treewidth.

A summary of the complexity results is given in Table 4.1.

5 Markov Random Fields

Bayesian networks are not allowed to have cycles as they would not make any sense. Another variant of PGMs, Markov Random Fields (MRFs), allow cycles and are undirected. Instead of encoding dependencies, the edges in a Markov network encode which random variables occur together in a potential. For example, the Markov network



corresponds to the distribution

$$P(X_1, X_2, X_3, X_4, X_5) \propto \phi(X_1, X_2)\, \phi(X_2, X_3)\, \phi(X_2, X_4)\, \phi(X_3, X_5)\, \phi(X_4, X_5)$$

up to normalization. In general, a distribution $P(X_1, X_2, \ldots, X_n)$ specified by a MRF G factors as

$$P(X_1, X_2, \ldots, X_n) = \frac{1}{Z} \prod_{C \in \text{Cliques}(G)} \phi_C(X_1, X_2, \ldots, X_n)$$

where $\text{Cliques}(G)$ is the set of maximal cliques in G and $1/Z$ is the normalization factor. In an MRF, the *Markov property* holds: two sets of random variables, $\mathcal{X}$ and $\mathcal{Y}$ are independent given a *separating subset* $\mathcal{Z}$ ($\mathcal{X} \perp \mathcal{Y} \,|\, \mathcal{Z}$). This subset has the property that every path from any variable in $\mathcal{X}$ to any variable in $\mathcal{Y}$ contains a variable from $\mathcal{Z}$.

Note that a random variable X is independent from all other variables Y in the network givens its parents, its children, and, to care about v-structures, its children's parents:

$$X \perp Y \,|\, \left(\text{Pa}(X) \cup \text{Ch}(X) \cup \left(\bigcup_{Z \in \text{Ch}(X)} \text{Pa}(Z) \right) \right)$$

This set of variables is known as the *Markov blanket* $M(X)$ of X.

Proof

5.1 Bayesian Networks as MRFs

It is possible to convert Bayesian networks into MRFs for leveraging the theory that evolved around them. This process was already introduced as *moralizing* before (section 4.5). This section builds up a more formal view on how the addition of a moral link is necessary. The following paragraphs show to convert the fundamental building blocks (pair, chain, common cause, and common effect) of a Bayesian network into a Markov network.

Pair

This conversion makes sense as from the Bayesian model,

$$P(X, Y) = P(X)\, P(Y \mid X),$$

and from the Markov network,

$$P(X, Y) \propto \phi(X, Y)$$

with $\phi(X, Y) \propto P(X)\, P(Y \mid X)$.

Chain

Like for the pair of variables, the joint probability is

$$P(X, Y, Z) = P(X)\, P(Y \mid X)\, P(Z \mid Y)$$

by the Bayesian model and

$$P(X, Y, Z) \propto \phi(X, Y)\, \phi(Y, Z)$$

by the Markov network with $\phi(X, Y) \propto P(X)\, P(Y \mid X)$ and $\phi(Y, Z) \propto P(Z \mid Y)$. Note that the parametrization of the potentials is not unique.

Common Cause

For a common cause, the conversion is again similar to the chain model. By the Bayesian network,

$$P(X, Y, Z) = P(X \mid Y)\, P(Y)\, P(Z \mid Y)$$

and by the Markov network,

$$P(X, Y, Z) \propto \phi(X, Y)\, \phi(Y, Z)$$

with $\phi(X, Y) \propto P(X \mid Y)\, P(Y)$ and $\phi(Y, Z) \propto P(Z \mid Y)$. Note again that the parametrization is not unique.

Common Effect

For the v-structure, the conversion is a tad more complicated. By the Bayesian network,

$$P(X, Y, Z) = P(X)\, P(Y \mid X, Z)\, P(Z).$$

But if the moral link (depicted in red) would not be present, the Markov network would model the distribution as

$$P(X, Y, Z) = \phi(X, Y)\, P(Y, Z).$$

However, this would entail $X \perp Y \mid Z$ which is—prominently as Berkson's paradox—not true. Hence, a moral link has to be included to ensure that the said conditional independency does not hold. With the moral link, the Markov network entails

$$P(X, Y, Z) \propto \phi(X, Y)\, \phi(Y, Z)\, \phi(X, Z).$$

Note that this formulation *does not* encode the same independencies: in the Markov network formulation, $X \perp Y$ does not hold anymore. However, it is more important to not capture invalid independencies than missing out on some.

This observation depends on the fact that two variables X and Y are conditionally dependent given Z if and only if their distribution $P(X, Y, Z)$ can be written as $P(X, Y, Z) = \phi(X, Y)\, \phi(Y, Z)$ with some $\phi(\cdot)$.

5.2 Triangulated Graphs and Simplicial Nodes

For the upcoming sections, some nomenclature is needed:

- A graph is *triangulated* if it has a perfect elimination ordering (i.e., one that does not introduce any fill-inds).
- A node is *simplicial* if its family (itself in addition to its neighbors) is equivalent to its maximum clique.

It follows that in a triangulated graph, the elimination of a simplicial node again yields a triangulated graph. Also, every triangulated graph (with at least two nodes) has at least two simplicial nodes. Hence, it directly follows that a perfect elimination ordering exists for every random variable in the graph.

Note that it is possible to triangulate every graph by applying any elimination ordering and consider the induced graph instead. However, while this creates a triangulated graph, it does not induce high-performance inference as the produces graph might have very large treewidth!

5.3 Join Trees

A *join tree* is a special kind of tree where each node (also called *cluster* in join trees) holds a set of variables. Additionally, the *running intersection property* (RIP) holds: for each pair of nodes, $\mathcal{X}$ and $\mathcal{Y}$, each node along each path between the two nodes contains the intersection $\mathcal{X} \cap \mathcal{Y}$. An example is shown in Figure 5.1. It can be shown that if the (maximum) cliques of a undirected graph can be organized into a join tree, then the graph is triangulated and vice versa.

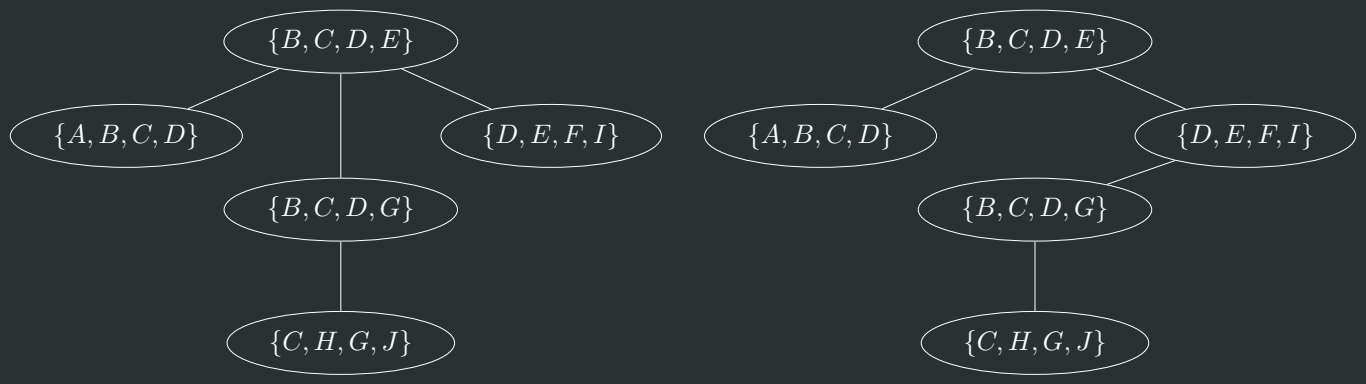
5.3.1 Markov Network as a Join Tree: Junction Tree

Organizing a triangulated graph into a join tree is straightforward by exploiting the special properties of such a graph. Pseudocode for this process is given in algorithm 3 and. A join tree G_J built from a Markov network G' also models the joint distribution as

$$P(X_1, X_2, \ldots, X_n) = \frac{\prod_{C \in \text{Cliques}(G_J)} \phi_C(X_1, X_2, \ldots, X_n)}{\prod_{S \in \text{Sep}(G_j)} \phi_S(X_1, X_2, \ldots, X_n)}$$

That is, the joint distribution is the product of the clique potentials divided by the product of the separator potentials.

Proof



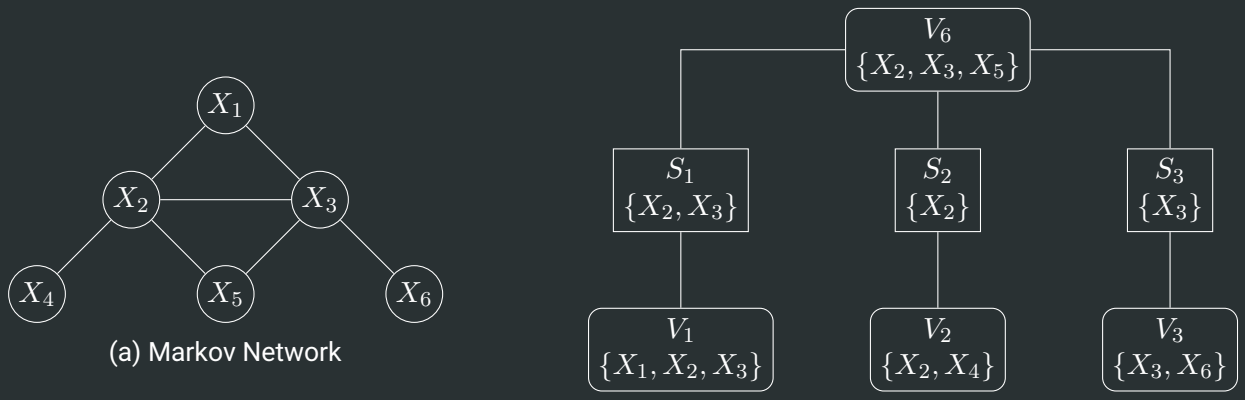
(a) Join Tree (b) No Join Tree

Figure 5.1: Illustration of a join tree. The left is a join tree, but the right is not as on the path between $\{A, B, C, D\}$ and $\{C, H, G, J\}$ has to contain $\{C\}$ in every node, which is not a subset of $\{D, E, F, I\}$.

Algorithm 3: Triangulated, Undirected Graph $\rightarrow$ Join Tree

Input: Triangulated undirected graph G with random variables $X_1, X_2, \ldots, X_n$
Output: Join tree with clique and separator nodes

Initialize set of nodes $\mathcal{X}$.
Initialize count of eliminated nodes $n \leftarrow 0$
while $\mathcal{X}$ *is not empty* **do**
    Pop simplicial node X from $\mathcal{X}$.
    // Construct set of variables to eliminate:
    $\mathcal{E} \leftarrow \{Y \in \text{Family}(X) : \text{Family}(Y) \subseteq \text{Family}(X)\}$.
    Eliminate all variables in $\mathcal{E}$.
    // Increment counter for number of eliminated variables:
    $n \leftarrow n + |\mathcal{E}|$
    // Create clique and seperator nodes for X:
    $V_n \leftarrow \text{Family}(X)$
    $S_n \leftarrow \text{Family}(X) \setminus \mathcal{E}$
    Connect V_n and S_n.
Connect each separator S_i to *exactly one* clique V_j with $j > i$ and $S_i \subseteq V_j$.



(a) Markov Network

(b) Junction Tree

Figure 5.2: Example of the junction tree (right) produced using algorithm 3 with the elimination ordering $(X_1, X_4, X_6, X_2, X_3, X_5)$ showed in Figure 4.3.

5.4 Junction Trees

A *junction tree* of a Markov network is a join tree for which each factor ϕ in the Markov network is associated to a cluster V in the join tree such that $\operatorname{dom} \phi \subseteq V$. This is called the *family preservation property* of junction trees. Such a join tree can, for example, be generated using algorithm 3 if an elimination ordering is given; see Figure 5.2 for an example. However, it is not necessary to build up a triangulated graph beforehand in general (it would be, for example, possible to create a junction tree with a node containing all nodes of the original tree; however, this would be useless as it does not leverage any theory of junction trees at all).

Ideally, the clusters are as small and modular as possible. Unfortunately, it is NP-hard to find the optimal tree... However, some useful heuristics and special cases exists. For example, tree-BNs can directly be converted to an efficient junction tree by converting each edge into a cluster.

5.4.1 Inference

To perform inference in a junction tree, each node sends exactly one *message* to exactly one of its neighbors once it has received messages from all its other neighbors. Hence, message passing must start at the leaves of the tree. To define a leave, however, first an arbitrary root node must be picked. Messages are then collected at this node and afterwards distributed back across the tree to the leaves. After this process, all clusters have the relevant information to compute the potential for its variables.

To pass messages between the clusters, each separator S_i has two "mailboxes" $\phi_{\overrightarrow{S_i}}$ and $\phi_{\overleftarrow{S_i}}$, one for the forward- and one for the backward-pass, respectively. That is, $\phi_{\overrightarrow{S_i}}$ is filled with messages flowing from the leaves to the root while $\phi_{\overleftarrow{S_i}}$ is filled with messages flowing from the root to the leaves. Note that from the construction of the junction tree, each separator has exactly two neighboring clusters, making the message boxes unambiguous. Each clique C_i is associated with a potential ϕ_{C_i} that initially holds the potential of the clique in the Markov network[1] and after running message passing holds the marginal potential for $\operatorname{dom} \phi_{C_i}$. All of this is illustrated in Figure 5.3.

For the complete pseudocode, see algorithm 4.

[1] For a junction tree stemming from a Bayesian network, this is the product of the potentials forming a clique in the Markov network.

Algorithm 4: Junction Tree Message Passing

Input: Junction tree with clusters C_i and separators S_i
Output: Probabilities $P(C_i)$ for each cluster C_i

Initialize all separator potentials to one.
Initialize all cluster potentials to the Markov networks' clique potentials.
Choose a root node.
// Forward Pass
foreach *cluster or separator* C_i, S_i *whose children were processed* **do**
    if *a cluster was chosen* **then**
        // Update cluster potential:
        $\phi_{C_i} \leftarrow \phi_{C_i} \prod_{S_j \in \mathrm{Ch}(C_i)} \phi_{\overrightarrow{S_j}}$
    if *a separator was chosen* **then**
        Let C_j be the child cluster of S_i.
        // Update separator potential:
        $\phi_{\overrightarrow{S_i}} \leftarrow \sum_{C_j \setminus S_i}^{\text{marg.}} \phi_{C_j}$
// Backward Pass
foreach *cluster or separator* C_i, S_i *whose parents were processed* **do**
    if *a cluster was chosen* **then**
        // Update cluster potential:
        $\phi_{C_i} \leftarrow \phi_{C_i} \prod_{S_j \in \mathrm{Pa}(C_i)} \phi_{\overleftarrow{S_j}}$
    if *a separator was chosen* **then**
        Let C_j be the parent cluster of S_i.
        // Update separator potential:
        $\phi_{\overleftarrow{S_i}} \leftarrow \sum_{C_j \setminus S_i}^{\text{marg.}} \phi_{C_j}$
Normalize potentials at all clusters C_i to get $P(C_i)$.

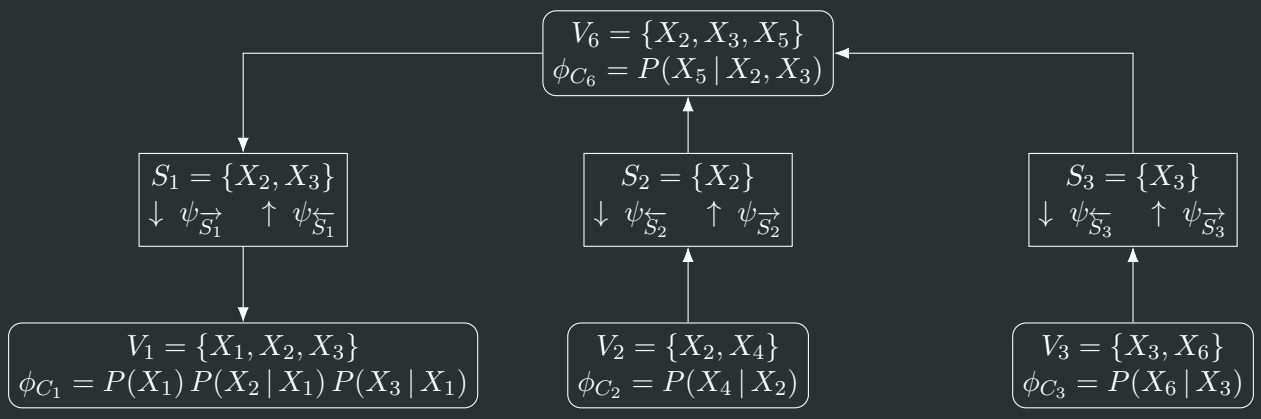


Figure 5.3: Illustration of the message boxes and potentials used for message passing in junction trees, based on the junction tree shown in Figure 5.2. For this tree, V_1 is chosen as the root node. The arrow directions depict the forward pass, the backward pass goes in reverse.

6 Learning

Until now, it was assumed that both the network structure and the CPTs are given somehow. This chapter is concerned with learning these from data. It is important to be able to learn structures from data as usually, no expert is available that can craft a network by hand or she is really expensive. And in the age of big data, data is cheap and huge amounts of data are available.

Learning Bayesian network is especially appealing as detecting and modeling conditional independencies captures the structure of many real-world distributions and can lead to knowledge discovery by capturing causal relationships. Also, the learned model can be used for handling missing data and hidden variables.

In a *complete* dataset, every data point has the values for all attributes that are getting investigated. On the other hand, *incomplete* datasets lack some values of random variables (this is usually the case in real-world data…). Also, a special kind of incompleteness, are hidden variables. *Hidden variables* never have assigned values in the whole dataset. One example for this is the cluster ID in an unsupervised clustering problem. They can also present a low-dimensional embedding that summarizes some other variables and separates them from a second set of RVs (dimensionality reduction).

The relation of the different problem types are summarized in Table 6.1.

6.1 Parameter Estimation

In *parameter estimation*, it is assumed that the structure of the Bayesian network (i.e., the random variables and dependencies between them) is fixed and only the entries of the CPTs have to be estimated. This field further subdivides into the cases of complete and incomplete data.

6.1.1 Known Structure, Complete Data

With known structure and complete data, the problem simply reduces to basically counting. The most basic method to do so are maximum likelihood estimators (where the likelihood can be computed using the Bayesian network). Some advantages of ML estimated are that they are (often) intuitive, consistent (the estimate converges to the best possible value with increasing number of data points), and asymptotic efficiency (given a dataset, the estimate is as close as possible to the true value).

Observed	Fixed Structure	Fixed Variables	Hidden Variables
Fully	Easy (Counting, MLE)	Selection of Edges	—
Partially	Numerical Optimization (difficult for large networks)	Encompasses for Difficult Subproblem (only structural EM known)	Scientific Discovery

Table 6.1: Overview over different learning problems and solution approaches.

Decomposability of the Likelihood

An important property of the likelihood for a Bayesian network is that it is *decomposable*. Let θ_i be the CPT of $P\big(X_i \mid \mathrm{Pa}(X_i)\big)$ and let $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)}$ be the data points. Then the log-likelihood is

$$\log P(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)} \mid \theta_1, \theta_2, \ldots, \theta_n) = \sum_{j=1}^{N} \log P(\boldsymbol{x}^{(j)} \mid \theta_j) = \sum_{j=1}^{N} \sum_{i=1}^{n} \log P\big(X_i = x_i^{(j)} \mid \mathrm{Pa}(X_i), \theta_j\big).$$

By swapping the sums, the log-likelihood can be rewritten as

$$\log P(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)} \mid \theta_1, \theta_2, \ldots, \theta_n) = \sum_{i=1}^{n} \log P(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)} \mid \theta_i).$$

Hence, it is possible to maximize each likelihood function independently and subsequently combine the solutions to get an ML estimate. This allows efficient solutions by distributing the calculations.

Likelihood for (Conditional) and Multinomials

An important special case for the CPTs are (conditional) multinomials, i.e., distributions for which

$$P(X = k) = \theta_k$$

where $k = 1, 2, \ldots, |\mathrm{val}(X)|$ are indices for the values of X. Maximizing the likelihood of this distribution yields the solution

$$\theta_k^* = \frac{N_k}{N}$$

where N_k is the number of data points for which $X = k$, i.e., $N_k = \sum_x \mathbb{1}[x = k]$ and N is the total number of data points.

For *conditional* multinomials, the distribution has the form $P(X = k \mid Y = \ell) = \theta_{k|\ell}$. Similarly, the ML solution is

$$\theta_{k|\ell} = \frac{N_{k|\ell}}{N_\ell} \tag{6.1}$$

where $N_{k|\ell}$ is the number of data points for which $X = k$ and $Y = \ell$ and N_ℓ is the number of data points for which N_ℓ. Hence, conditional multinomials are completely analogous to non-conditional multinomials but the dataset is constrained to the respective conditional values.

Proof

6.1.2 Known Structure, Incomplete Data (Expectation-Maximization)

When the dataset is incomplete, the algorithm somehow has to deal with the incompleteness. One idea is to use an EM algorithm to complete the data in the E-step and to estimate the parameters in the M-step. For multinomial distributions and discrete data, this involves computing *expected counts* in the E-step and using the ML estimated (6.1) in the M-step to tune the model. Let $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)}$ be the data points, then the expected count $\mathbb{E}[N_{k \mid \ell}]$, where $N_{k \mid \ell}$ would be the number of data points for which $X = k$ and $Y = \ell$ The

expected count $\mathbb{E}[N_{k|\ell}]$, where $N_{k|\ell}$ would be the number of data points for which $X = k$ and $Y = \ell$ holds (assuming a distribution $P(X = k \mid Y = \ell) = \theta_{k|\ell}$), is

$$\mathbb{E}[N_{k|\ell}] = \sum_{i=1}^{N} P(k, \ell \mid \boldsymbol{x}^{(i)}). \tag{6.2}$$

If $\boldsymbol{x}^{(i)}$ is complete, the probability would reduce to $\mathbb{1}[x_x^{(i)} = k \wedge x_y^{(i)} = \ell]$, i.e., counting. However, if the data is not complete, inference has to be run to actually compute the probability. Using the expected counts, the parameters can easily be estimated in the M-step using

$$\theta_{k|\ell} = \frac{\mathbb{E}[N_{k|\ell}]}{\mathbb{E}[N_\ell]},$$

where $\mathbb{E}[N_\ell]$ is analogous to (6.2).

In practice, some more things are needed to apply EM. Firstly, the parameters have to be initialized some how (e.g., randomly, or as a best guess from other sources, by estimating from just the complete data points, ...). Secondly, a stopping criterion has to be defined (only small changes in the likelihood or parameter values, for example). Thirdly, bad local maxima (which the EM algorithm is prone to converge to) have to be avoided (e.g., by running multiple times, pruning unpromising runs soon, ...). Also, various methods are available to speed up convergence.

6.1.3 Gradient Ascent

An alternative, rather bruteforcy method for dealing with incomplete data is to just ignore it and use that the gradient

$$\frac{\partial}{\partial \theta_{k \mid \ell}} \log P(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)} \mid \Theta) = \frac{1}{\theta_{k \mid \ell}} \sum_{j=1}^{N} \log P(k, \ell \mid \boldsymbol{x}^{(i)}, \Theta)$$

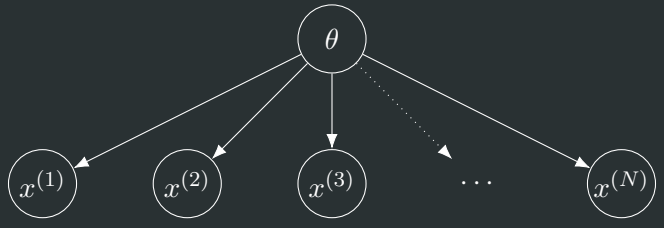
can be computed in closed form. Then, gradient ascent can be applied. This has the advantage that training is well-known from training neural networks, coming with the disadvantages that the parameters somehow have to be constrained to be legal and the requirement for smart optimization techniques to ensure convergence.

6.1.4 Bayesian Parameter Estimation

In Bayesian estimation, the parameters are assumed to be random variables as well. Hence, by putting a prior on the parameters, learning can be understood as a form of inference using Bayes rule:

$$P(\Theta \mid \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)}) = \frac{P(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)} \mid \Theta)\, P(\Theta)}{P(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)})}.$$

Assume (for now) that the model only has a single binomial random variable X with the parameter θ. Then, by the i.i.d. assumption, all data points are independent from each other and they can be represented easily by the following Bayesian network:



This network has the interesting property that $P(x^{(i)} \mid \theta) = \theta$. Hence, predicting θ directly corresponds to inference in this network.

As the true parameter is unknown, one idea is to integrate it out and compute the marginal data likelihood. With the prior $P(\theta) = 1$ for $0 < \theta < 1$ and the likelihood $P(x_1, x_2, \ldots, x_n \mid \theta) = \theta^k (1-\theta)^{n-k}$ with $k = \sum_{i=1}^{n} X_i$ and binary variables, the marginal likelihood is

$$P(x_1, x_2, \ldots, x_n) = \int_0^1 P(x_1, x_2, \ldots, x_n \mid \theta)\, P(\theta)\, \mathrm{d}\theta = \frac{1}{n+1} \binom{n}{k}^{-1}$$

and, by Bayes rule, the posterior is

$$P(\theta \mid x_1, x_2, \ldots, x_n) = (n+1) \binom{n}{k} \theta^k (1-\theta)^{n-k}.$$

To now perform prediction, it can be considered to compute the probability of the variable x_{n+1}, treating the realizations as a chain. Observing that $P(x_{n+1} \mid \theta, x_1, x_2, \ldots, x_n) = P(x_{n+1} \mid \theta)$ simplifies the integrals down to

$$P(x_{n+1} \mid x_1, x_2, \ldots, x_n) = \frac{k+1}{n+2}.$$

All of these calculations are possible as the chosen prior was *conjugate* to the likelihood. That is, the posterior lies in the same distribution family as the prior.

Similarly to the above Bayesian network for a single parameter and random variable, Bayesian networks can be constructed for multiple parameters and variables to perform learning as inference. The core idea is that the separate data points are independent given the parameters, hence making inference tractable. The other observation is that the estimation only relies on *sufficient statistics* (these are the count for multinomials, for example).

6.1.5 Summary

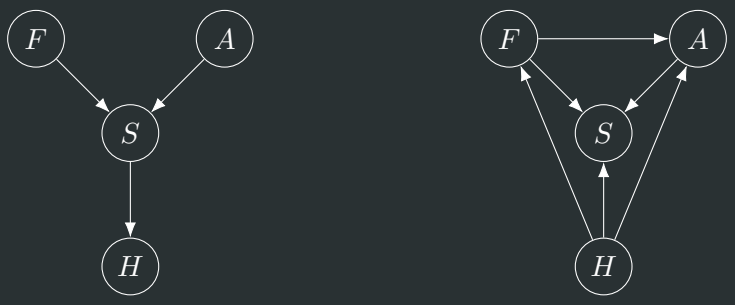
Two approaches for parameter estimation with a fixed structure have been considered: maximum likelihood and Bayesian estimation. While both are asymptotically equivalent and consistent (i.e., they converge to the same "true" value), Bayesian estimation provides smoother result regularized by the prior. And if the prior is chosen to be close to the optimal value, it converges faster.

6.2 Structure Learning / Model Selection

In the previous section, it was assumed that the structure of the Bayesian network (the random variables and dependencies between them) where given. However, this is often not the case. In essence, the goal is to find, given a P, a G which is an I-map of P. To do some, a notion of minimal I-maps and equivalence of I-maps is necessary.

6.2.1 Minimal I-Maps and Perfect Maps (P-Maps)

A straightforward approach for defining an I-map to be *minimal* is to call an I-map minimal if it is as simple as possible, i.e., the removal of an edge would make the graph not be an I-map anymore. However, this has the major caveat that even graphs with lots of edges might be "minimal" in this sense although other configuration have less edges. Consider, for instance, the following networks (left is the true model):



Although both models entail the independencies required to be an I-map, the left model is obviously simpler than the right. Hence, the above definition of a minimal I-map (removing an edge results in the graph not being an I-map anymore) is insufficient to capture simplicity and minimalism.

6.2.2 Perfect Maps (P-Maps) and I-Equivalence

An alternative approach is to not just try to find an I-map but to find a *perfect map* (P-map) with $I(G) = I(P)$. This eliminates the problem of defining minimalism and simplicity. Instead, a notion of equivalence between maps must be created. Straightforwardly, two models G_1 and G_2 are said to be *I-equivalent* if $I(G_1) = I(G_2)$. This definition leads to a theorem for I-equivalence of two Bayesian networks:

Theorem Two Bayesian networks G_1 and G_2 are I-equivalent if and only if they have the same skeleton and same immoralities (v-structures with disconnected parents). But why would one even struggle to find accurate structures? This stems from two factors: first, both networks with too many and networks with too few edges induce wrong assumptions on the structure. Second, networks with too few connections (and thus too many independencies) cannot be compensated by parameter estimation (as the relevant parameters simply do not exist); similarly, networks with too many edges have too many parameters where the number grows super-exponentially with the number of variables, making learning ambiguous and slow.

6.2.3 Obtaining a P-Map

To obtain a P-map, in a first step the basic independencies of P (skeleton and immoralities) have to be extracted. This yields the equivalence class the Bayesian network has to live in. In a second step, every BN structure has to be obtained that lies in this equivalence class.

Obtaining the skeleton, i.e., figuring out whether there is an edge between two random variables, means to check whether the local Markov assumption holds for a pair of random variables X, Y. That is, if there exists a $\mathcal{U} \subseteq \mathcal{X} \setminus \{X, Y\}$ with $|\mathcal{U}|$ such that $X \perp Y \,|\, \mathcal{U}$, then there is no edge between X and Y. Here, d is the maximal number of parents which is a hyper-parameter reducing the search space.

In the second step, the immoralities have to be identified, i.e., for a triple $X - Y - Z$ without $X - Z$, when should it be an immorality $X \to Y \leftarrow Z$? Essentially, this is the case if $X \perp Y$.

Both steps are summarized in algorithm 5. Note that all of this assumes that there is some way to determine whether $X \perp Y \,|\, \mathcal{U}$ holds. There are numerous approaches to this which are discussed in subsubsection 6.2.4.

From this process, a partially directed acyclic graph emerges as the identification of immoralities already prescribes some directions. Take, for instance, the following graph:

Here the only immorality $X \to Z \leftarrow Y$ was identified. However, from this it follows that the edge $Z - W$ must have the direction $Z \to W$ as the reverse would create even more immoralities ($X \to Z \leftarrow W$ and $Y \to Z \leftarrow W$) which have not been observed. There exist polynomial algorithms to identify these forced directions.

Algorithm 5: Bayesian Network P-Map Identification

Input: Random variables $\mathcal{X} = X_1, X_2, \ldots, X_n$, max. number of parents $d < n$
Output: Equivalence class for P-maps
// Skeleton Identification:
foreach $X, Y \in \mathcal{X}$ **do**
  $e \leftarrow \mathfrak{t}$;
  foreach $\mathcal{U} \subseteq \mathcal{X} \setminus \{X, Y\}$ *with* $|\mathcal{U}| \leq d$ **do**
    if $X \perp Y \,|\, \mathcal{U}$ **then**
      $e \leftarrow \mathfrak{f}$;
      break ;
  if $e = \mathfrak{t}$ **then**
    Add edge between X and Y. ;
// Immorality Identification:
foreach *Triple* $X - Y - Z$ *in the skeleton* **do**
  $i \leftarrow \mathfrak{t}$;
  foreach $\mathcal{U} \subseteq \mathcal{X} \setminus \{X, Y\}$ *with* $Z \in \mathcal{U}$ **do**
    if $X \perp Y \,|\, U$ **then**
      $i \leftarrow \mathfrak{f}$;
      break ;
  if $i = \mathfrak{t}$ **then**
    Add immorality $X \to Y \leftarrow Z$. ;

6.2.4 Learning Approaches

To learn the structure of a network, two distinctive approaches exist: constraint- and score-based learning. These will be presented in the following two sections.

Constraint-Based

Constraint-based learning is based on algorithm 5 and tackles the identification of the independencies $X \perp Y \,|\, \mathcal{U}$. Its approach is to use statistical tests with the null hypothesis that the independence holds. One such test is the χ^2-independency-test. A more direct measure is the mutual information

$$\hat{I}(X; Y \,|\, \mathcal{U}) = \sum_{X,Y}^{\text{marg.}} \hat{P}(X, Y \,|\, \mathcal{U}) \log \frac{\hat{P}(X, Y \,|\, \mathcal{U})}{\hat{P}(X)\, \hat{P}(Y \,|\, \mathcal{U})}$$

where $\hat{\cdot}$ denotes that the quantities are approximations (i.e., $\hat{P}$ is an estimated probability based on counting the examples).

Score-Based

Score-based learning depends on a score function to assess a model's accuracy and searches for a structure with maximum score. A straightforward approach is to use the (log-)likelihood

$$\log P(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)} \mid G) = N \sum_{i=1}^{n} \big(\hat{I}\big(X_i; \mathrm{Pa}_G(X_i)\big) - \hat{H}(X_i)\big)$$

where $\hat{I}$ and $\hat{H}$ are the approximate mutual information and entropy, respectively. Note that the data is used in the computation of the mutual information and entropy as the probabilities are based on counting the samples.

While this approach is rather simple, it has the disadvantage that adding edges always increases the log-likelihood. Hence, a fully connected network is optimal w.r.t. to the log-likelihood and it overfits badly. A practical approach is to just pick a model that scores well and use that for inferring the structure, but this yields the problem that for a small sample size, lots of models perform well but the answers of these models is often useless.

An alternative approach is to use a Bayesian score by placing a prior on the parameters. While this methods yields lots of integrals that are hard to evaluate, placing a Dirichlet prior on the parameters yields a good and simple-to-compute approximation, the *Bayesian information criterion* (BIC). For a Dirichlet prior and a large dataset (i.e., $N \lg 0$), the integral

$$P(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)} \mid G) = \int P(P(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)} \mid G), \Theta)\, P(\Theta \mid G)\, \mathrm{d}\Theta$$

for the marginal likelihood can be approximated as

$$\begin{aligned}\log P(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)} \mid G) &= \log P(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)} \mid G, \Theta) - \frac{\log M}{2} \dim G + \mathcal{O}(1) \\ &= N \sum_{i=1}^{n} \big(\hat{I}\big(X_i; \mathrm{Pa}_G(X_i)\big) - \hat{H}(X_i)\big) - \frac{\log N}{2} \dim G + \mathcal{O}(1),\end{aligned}$$

where $\dim G$ is the number of parameters. The the first term is simply the data log-likelihood and assesses the fit to the data while the second term penalizes model complexity (quantified by the number of parameters) and avoid fitting noise. As the amount of data N grows, the former term becomes prevalent and hence the BIC score is consistent (i.e., converges to the "true" data and overrides the Dirichlet prior).

6.2.5 Structure Search as Optimization

Using a scoring function (subsubsection 6.2.4), structure learning can be framed as an optimization problem trying to maximize a score. For this, the optimization algorithm needs the following ingredients:

- Training data
- Scoring function (LL, MDL, BIC, ...)
- Set of possible structures

Learning Trees (Complete Data)

If it is known that a tree structure has to be selected and the data is complete, the optimal model can found with time complexity $\mathcal{O}(n^2 \log n)$ by finding the spanning tree with the maximum score. However, if some variables are hidden or incomplete, the problem becomes hard again (and can be approximately solved using EM).

Heuristic (Local) Search

Most of the time, heuristics are used for finding a reasonable model. They leverage a set of application simple actions (e.g., add/remove/flip and edge, ...) and traverse the search space looking for high-scoring models. Some search techniques are greedy hill-climbing, best first search, simulated annealing, etc.

One of the most used approaches is *local search* that starts with some network (e.g., empty, best tree, random, ...) and at each iteration evaluates all possible changes based on the set of applicable cations and chooses the one that improves the current score the most. The algorithm terminates once no modification can improve the score anymore and a local maximum was found. After a local change, only parts that have changed have to be re-evaluated if the data is complete, but for incomplete data, everything has to be re-calculated after re-estimating the parameters.

While this approach is simple, it also exhibits some major drawbacks: first, it is prone to get stuck in local maxima. Second, it can get stuck on plateaus where no change makes a difference. As usual, standard heuristics can be used for circumvent some of these issues (e.g., random restarts, TABU search, simulated annealing, ...).

Structural EM

In practice, applying local search with incomplete data using EM is costly as every evaluation of the scoring function first requires to re-fit the parameters and therefore invoke EM. This costs time as it also has to be run on poor candidates...in practice, only a few candidates are evaluated in depth. This introduces structural EM, an EM algorithm specially designed for this problem.

Structural EM (SEM) is based on the idea that instead of optimizing the real score, optimize an alternative surrogate score that is decomposable. This alternative score must have that property that maximizing it improves the real score, too. The E-step works as usual. But in the M-step, the maximization is done in a combined parameter-structure-space, leading to a *parametric* and *structural* M-step, one of which is executed at time.

7 Dynamic Bayesian Networks

So far, only *static* Bayesian networks have been considered that cannot evolve. But the world around us is rather dynamic and changes with time, so incorporating this into the theory of probabilistic graphical models would be good. This chapter covers some basic topics with networks changing dynamically, highlighting the most common one, the hidden Markov model that has lots of nice properties and theoretical development that can all be leveraged when formulating a problem in this framework.

A *hidden Markov model* (HMM) is a graphical model with T *states* S_t, $t = 1, 2, \dots, T$ and as many *observations* O_t, $t = 1, 2, \dots, T$; see Figure 7.1. The terminology of states and observations stems from its applications in robotics and similar fields where the observations are usually sensor data and the states are the actual, non-observable/latent, physical states (e.g., the acceleration). According to the Bayesian network, the HMM factors as

$$P(S_1, S_2, \dots, S_T, O_1, O_2, \dots, O_T) = P(S_1) \prod_{t=1}^{T-1} P(S_{t+1} \mid S_t) \prod_{t=1}^{T} P(O_t \mid S_t) = \pi_{S_1} \prod_{t=1}^{T-1} a_{S_t S_{t+1}} \prod_{t=1}^{T} b_{S_t O_t}$$

where $\pi_{S_1} \coloneqq P(S_1)$, $a_{S_t S_{t+1}} \coloneqq P(S_{t+1} \mid S_t)$, $b_{S_t O_t} \coloneqq P(O_t \mid S_t)$ are called the *initial*, *transition*, and *observation* distributions, respectively. By the local Markov assumption, the "future is independent of the past given the present", i.e., the next state only depends on the current state and not on the state beforehand[1].

7.1 Inference

Performing inference in an HMM has three flavors:

- *Decoding:* Compute the probability of an observation sequence.
- *Best State Sequence:* Given observations, compute the most likely (hidden) states.
- *Parameter Estimation:* Given observations and a set of possible models, which fits the best?

These three flavors are covered in the upcoming sections. Let, for all of the following, S and O be the set of all state and observation variables for brevity. Also, let Θ be the parameters of the network.

[1] Note that there are other variants of HMMs, called *k-th order HMMs*, that can watch up to k steps into the past. So the HMM covered here has $k = 1$. However, any k-th order HMM can be reduced to a first-order HMM, so the theory stays the same (cf. higher-order ordinary differential equations).

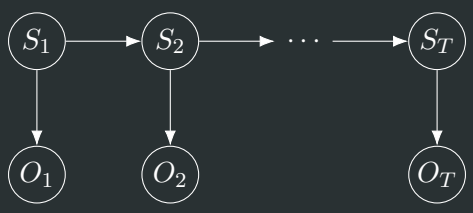


Figure 7.1: Hidden Markov model with states S_t and observations O_t.

7.1.1 Decoding

Decoding is concerned with computing the following probabilities:

$$P(O \mid S, \Theta) = \prod_{t=1}^{T} b_t \qquad P(S \mid \Theta) = \pi \prod_{t=1}^{T-1} a_t$$

$$P(O, S \mid \Theta) = P(O \mid S, \Theta)\, P(S \mid \Theta) \qquad P(O \mid \Theta) = \sum_{S}^{\text{marg.}} P(O, S \mid \Theta)$$

While of of these are trivial expressions, computing $P(O \mid \Theta)$ has complexity $\mathcal{O}(N^T \cdot 2T)$ due to the exponential number of entries in the CPT!

However, it is possible to leverage dynamic programming and defining recursive relations to compute $P(O \mid \Theta)$ efficiently:

$$\begin{aligned} \alpha_\ell(t) &:= P(O_1, O_2, \ldots, O_t,\, S_t = \ell \mid \Theta) & \beta_s(t) &:= P(O_{t+1}, O_{t+2}, \ldots, O_T \mid S_t = s, \Theta) \\ \alpha_\ell(1) &= \pi_\ell\, b_{\ell O_1} & \beta_s(T) &= 1 \\ \alpha_\ell(t+1) &= \sum_s \alpha_s(t)\, a_{s\ell}\, b_{\ell O_{t+1}} & \beta_s(t) &= \sum_\ell a_{s\ell}\, b_{\ell O_{t+1}}\, \beta_\ell(t+1) \end{aligned}$$

The left column is a forward pass for computing $\alpha_\ell(T) = P(O_1, O_2, \ldots, O_T,\, S_T = \ell \mid \Theta)$ and the right column is a backward pass for computing $\beta_s(1) = P(O_2, O_3, \ldots, O_T \mid S_1 = s \mid \Theta)$. These quantities can then be used in three different fashions for computing $P(O \mid \Theta)$:

$$P(O \mid \Theta) = \sum_s \alpha_s(T) = \sum_s \pi_s\, b_{sO_1}\, \beta_s(1) = \sum_s \alpha_s(t)\, \beta_s(t)$$

Where the latter is true for all $t = 1, 2, \ldots, T$.

Proof of Forward Pass Showing that the recursive relation for $\alpha_\ell(t)$ is straightforward by invoking the chain rule and exploiting the HMM's independencies multiple times (omitting the explicit dependence on Θ for brevity):

$$\begin{aligned} \alpha_\ell(t+1) &= P(O_1, O_2, \ldots, O_t, O_{t+1},\, S_{t+1} = \ell) \\ &= P(O_1, O_2, \ldots, O_t, O_{t+1} \mid S_{t+1} = \ell)\, P(S_{t+1} = \ell) \\ &= P(O_1, O_2, \ldots, O_t \mid S_{t+1} = \ell)\, P(O_{t+1} \mid S_{t+1} = \ell)\, P(S_{t+1} = \ell) \\ &= P(O_1, O_2, \ldots, O_t,\, S_{t+1} = \ell)\, P(O_{t+1} \mid S_{t+1} = \ell) \\ &= \sum_s P(O_1, O_2, \ldots, O_t,\, S_t = s, S_{t+1} = \ell)\, P(O_{t+1} \mid S_{t+1} = \ell) \\ &= \sum_s P(O_1, O_2, \ldots, O_t,\, S_{t+1} = \ell \mid S_t = s)\, P(S_t = s)\, P(O_{t+1} \mid S_{t+1} = \ell) \\ &= \sum_s P(O_1, O_2, \ldots, O_t,\, \mid S_t = s)\, P(S_{t+1} = \ell \mid S_t = s)\, P(S_t = s)\, P(O_{t+1} \mid S_{t+1} = \ell) \\ &= \sum_s P(O_1, O_2, \ldots, O_t,\, S_t = s)\, P(S_{t+1} = \ell \mid S_t = s)\, P(O_{t+1} \mid S_{t+1} = \ell) \\ &= \sum_s \alpha_s(t)\, a_{s\ell}\, b_{\ell O_{t+1}} \end{aligned}$$

By marginalizing out S_1 of $\alpha_{S_1}(T)$, this is equivalent to $P(O \mid \Theta)$. □

Proof of Backward Pass Analogous to the forward pass: invoke the chain rule multiple times and exploit the HMM's independencies. Showing the computation for $P(O \mid \Theta)$ is also straightforward:

$$\begin{aligned}\sum_s \pi_s \, b_{sO_1} \, \beta_s(1) &= \sum_s P(S_1 = s) \, P(O_1 \mid S_1 = s) \, P(O_2, O_3, \ldots, O_T \mid S_1 = s) \\ &= \sum_s P(S_1 = s) \, P(O_1, O_2, \ldots, O_T \mid S_1 = s) \\ &= \sum_s P(O_1, O_2, \ldots, O_T, \, S_1 = s) \\ &= P(O_1, O_2, \ldots, O_T)\end{aligned}$$

□

Proof of Combined Pass Showing the combined pass formulation is just plugging in of distributions and leveraging independencies, too:

$$\begin{aligned}\sum_s \alpha_s(t) \, \beta_s(t) &= \sum_s P(O_1, O_2, \ldots, O_t, \, S_t = s) \, P(O_{t+1}, O_{t+2}, \ldots, O_T \mid S_t = s) \\ &= \sum_s P(O_1, O_2, \ldots, O_t \mid S_t = s) \, P(O_{t+1}, O_{t+2}, \ldots, O_T \mid S_t = s) \, P(S_t = s) \\ &= \sum_s P(O_1, O_2, \ldots, O_T \mid S_t = s) \, P(S_t = s) \\ &= \sum_s P(O_1, O_2, \ldots, O_T, \, S_t = s) \\ &= P(O_1, O_2, \ldots, O_T)\end{aligned}$$

□

7.1.2 Best State Sequence

To find the most probable/best state sequence for a given observation sequence, two basic approaches exist: first, to get the states that are most likely individually; second, get the most likely sequence.

The former is relatively straightforward using the α's and β's defined before:

$$\gamma_s(t) = P(S_t = s \mid O, \Theta) = \frac{P(S_t = s, O \mid \Theta)}{P(O \mid \Theta)} = \frac{\alpha_s(t) \, \beta_s(t)}{\sum_s \alpha_s(t) \, \beta_s(t)}$$

Subsequently, this quantity can be maximized, $s_t^* = \arg\max_s \, \gamma_s(t)$.

To find the state sequence $X^* = \arg\max_X \, P(X \mid O, \Theta)$, the *Viterbi algorithm* has been developed. Let

$$\delta_\ell(t) = \max_{s_1, s_2, \ldots, s_{t-1}} \, P(s_1, s_2, \ldots, s_{t-1}, \, O_1, O_2, \ldots, O_{t-1}, \, S_t = \ell)$$

be the maximum probability (state sequence) for given observations landing in state ℓ at time t. Like before for decoding, this quantity can be computed recursively,

$$\delta_\ell(t+1) = \max_s \, \delta_s(t) \, a_{s\ell} \, b_{\ell O_{t+1}},$$

with $\delta_\ell(1) = \pi_\ell \, b_{\ell O_1}$ to kick off the iteration. The argument states are stored in ψ_ℓ,

$$\psi_\ell(t+1) = \arg\max_s \, \delta_s(t) \, a_{s\ell} \, b_{\ell O_{t+1}}.$$

After iterating, the most likely state sequence can be computed using the following:

$$s_T^* = \arg\max_s \ \delta_s(T) \qquad s_t^* = \psi_s(t+1) \qquad P(s^*) = \arg\max_s \ \delta_s(T)$$

This allows computing the most likely state sequence efficiently.

7.1.3 Parameter Estimation (Using EM)

Parameter estimation in a HMM is concerned with finding the parameters Θ given an observation sequence. This is essentially done using an EM algorithm like it was introduced before in subsection 6.1.2, but using the probabilities α and β. The estimated $\hat{\pi}$, $\hat{a}$, and $\hat{b}$ are then given as

$$\hat{\pi}_{S_1} = \gamma_{S_1}(1) \qquad \hat{a}_{S_t S_{t+1}} = \frac{\sum_{t=1}^{T-1} p_t(S_t, S_{t+1})}{\sum_{t=1}^{T-1} \gamma_{S_t}(t)} \qquad \hat{b}_{S_t O_t} = \frac{\sum_{t=1;o_t=O_t}^{T} \gamma_{S_t}(t)}{\sum_{t=1}^{T} \gamma_{S_t}(t)}$$

where $p_t(S_t, S_{t+1})$ and $\gamma_{S_t}(t)$ are the probability of traversing from a state S_t to a state S_{t+1} and the probability of being in a state S_t, respectively:

$$p_t(S_t, S_{t+1}) = \frac{\alpha_{S_t}(t)\, a_{S_t S_{t+1}}\, b_{S_{t+1} O_{t+1}}\, \beta_{S_{t+1}}(t+1)}{\sum_s \alpha_s(t)\, \beta_s(t)} \qquad \gamma_{S_t}(t) = \sum_s p_t(S_t, s)$$

Note that the data is incorporated in computing the p's and γ's (this is effectively the E-step). By iterating these two sets of equations, the algorithm will converge (although possibly to a local maximum). This is known as the *Baum-Welch algorithm*.

7.2 State Estimation (Kalman Filter)

So far, only discrete states that can be enumerated have been considered. In lots of applications such as robotics and other dynamical systems, however, the states are continuous. Also, an agent is able to perform actions u in the environment but cannot directly observe the states x but only observations z that somehow depend on x. State estimation is concerned with recovering the states x. In this section first the general framework of a *Bayes filter* is discussed and subsequently a special case, the Kalman filter is covered. The *Kalman filter* is an approach of applying the built theory to linear Gaussian dynamical systems (LGDS's) where the local distributions are Gaussian and the mean propagation is linear.

7.2.1 Bayes Filter

A *Bayes filter* used a stream of sensor-action pairs $(\boldsymbol{u}_t, \boldsymbol{z}_{t+1})$, a sensor model $P(\boldsymbol{z}_t \mid \boldsymbol{x}_t)$, an action model $P(\boldsymbol{x}_{t+1} \mid \boldsymbol{u}_t, \boldsymbol{x}_t)$, and a prior on the system state $P(\boldsymbol{x})$. It seeks for the belief $Bel(\boldsymbol{x}_t) = P(\boldsymbol{x}_t) \mid \boldsymbol{u}_1, \boldsymbol{z}_2, \boldsymbol{u}_2, \boldsymbol{z}_3, \ldots, \boldsymbol{u}_{t-1}, \boldsymbol{z}_t$ on the system state given all prior actions and observations with the assumption of a hidden Markov model on the states and observations[2]. After some massaging of the probability distributions, the belief can be expressed as

$$Bel(\boldsymbol{x}_t) \propto P(\boldsymbol{z}_t \mid \boldsymbol{x}_t) \int P(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{u}_{t-1}) Bel(\boldsymbol{x}_{t-1})\, \mathrm{d}\boldsymbol{x}_{t-1}\,.$$

For some systems like LGDS's, this integral can be solved in closed form, leading to the Kalman filter discussed in the next section. Another filtering systems used often is a particle filter which is based on using a Monte Carlo approximate of the integral.

[2] Some underlying assumptions are that the world does not change, i.e., the distributions are time-invariant, the noise is independent, the model is perfect, and that no approximation errors occur.

7.2.2 Discrete-Time Kalman Filter

The discrete-time Kalman filter tackles dynamical systems of the form

$$\begin{aligned} \boldsymbol{x}_1 &\sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) \\ \boldsymbol{x}_{t+1} &\sim \mathcal{N}\big(\boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B}_t \boldsymbol{u}_t,\, \boldsymbol{R}_t\big) \\ \boldsymbol{y}_t &\sim \mathcal{N}\big(\boldsymbol{C}_t \boldsymbol{x}_t,\, \boldsymbol{Q}_t\big) \end{aligned}$$

where $\boldsymbol{x}_{1:T}$, $\boldsymbol{y}_{1:T}$, and $\boldsymbol{u}_{1:T-1}$ are the states, observations, and actions, respectively. The matrices $\boldsymbol{A}_t$, $\boldsymbol{B}_t$, $\boldsymbol{C}_t$, $\boldsymbol{R}_t$, and $\boldsymbol{Q}_t$ are the dynamics, control, observation, state covariance, and observation covariance matrices, respectively. $\boldsymbol{\mu}_0$ and $\boldsymbol{\Sigma}_0$ are the mean and covariance of the initial state. All of these parameters are assumed to be known for state estimation.

As the integrals of the Bayesian filter can be solved in closed form for LGDS's, the Kalman filter equations can also be stated in closed form. The state estimation consists of two steps: *prediction* and *correction*. In the prediction step, the state is predicted from the prior knowledge:

$$\bar{\boldsymbol{\mu}}_t = \boldsymbol{A}_t \boldsymbol{\mu}_{t-1} + \boldsymbol{B}_t \boldsymbol{u}_t \qquad\qquad \bar{\boldsymbol{\Sigma}}_t = \boldsymbol{A}_t \boldsymbol{\sigma}_{t-1} \boldsymbol{A}_t^\top + \boldsymbol{R}_t$$

In the correction step, a new observation $\boldsymbol{y}_t$ is incorporated to correct the previous prediction:

$$\boldsymbol{\mu}_t = \bar{\boldsymbol{\mu}}_t + \boldsymbol{K}_t(\boldsymbol{y}_t - \boldsymbol{C}_t \bar{\boldsymbol{\mu}}_t) \qquad\qquad \boldsymbol{\Sigma}_t = (\boldsymbol{I} - \boldsymbol{K}_t \boldsymbol{C}_t)\bar{\boldsymbol{\Sigma}}_t$$

Here, $\boldsymbol{K}_t = \bar{\boldsymbol{\Sigma}}_t \boldsymbol{C}_t^\top \left(\boldsymbol{C}_t \bar{\boldsymbol{\Sigma}}_t \boldsymbol{C}_t^\top + \boldsymbol{Q}_t\right)^{-1}$ is the *Kalman gain*. The estimations $\boldsymbol{\mu}_{1:T}$ are the expected states with covariance $\boldsymbol{\Sigma}_{1:T}$ given the observations $\boldsymbol{y}_{1:T}$.

7.3 General Dynamic Bayesian Networks

So far, only HMMs and LGDS's, a special kind of HMM, were considered. For general dynamical Bayesian network, the theory is not that evolved. The basic ideas are based on unrolling the network for a specific number of time steps an then performing various inference tasks in it:

- Filtering: Computes the expected states given the observations until t.
- Prediction: Computes the expected next state.
- Smoothing: Computes the expected states given all observations until T.
- MLE: Computes the most likely state sequence.

However, the naive method of unrolling the BN has some major drawbacks. Foremost, the inference cost increases with t. One approach to solving this is to roll up the network again and using variable elimination to sum out the previous states.

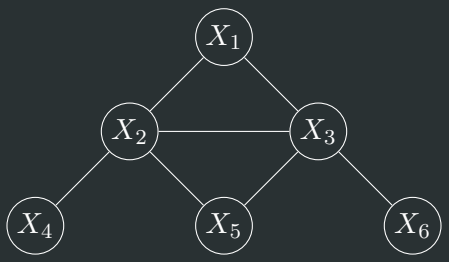
In general, exact inference is intractable and approximate methods need to be used (e.g., factored belief states).

8 Approximate Inference

As known from chapter 4, exact inference is general NP-hard and intractable even for simple cases in the setting of dynamic BNs. Hence, it is desirable to get tractable approximate inference.

8.1 Message Passing in Factor Graphs

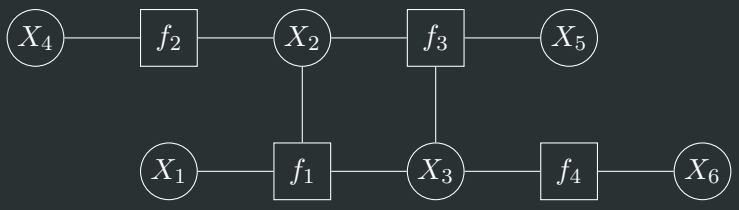
A *factor graph* is a representation of the structure of a distribution where random variables are connected via *factor nodes* that represent the factors of the distribution. The connection to Markov networks (and subsequently Bayesian networks) is best explained with an example: consider the Markov network



which factorizes as

$$P(X_1, X_2, X_3, X_4, X_5, X_6) = \underbrace{P(X_1, X_2, X_3)}_{f_1} \underbrace{P(X_2, X_4)}_{f_2} \underbrace{P(X_2, X_3, X_5)}_{f_3} \underbrace{P(X_3, X_6)}_{f_4}.$$

The corresponding factor graph then is:



Note that this factor graph has a cycle! For most parts of this section, it is required that the factor graph is acyclic.

A rather simple algorithm for performing inference in factor graphs is *sum-product belief propagation*. Similar to inference in junction tree (subsection 5.4.1), it consists of sending message between the graph's nodes. All nodes hold a belief b (with b_i for the i-th variable node and b_α for the α-th factor node) that is used to compute the respective marginals (over the node's variable's realization x_i or the factor's realization set of variables $\boldsymbol{x}_\alpha$) by normalization. Both variable and factor nodes send messages $\mu_{i\to\alpha}$ and $\mu_{\alpha\to i}$ to their neighbors. First, the messages are sent from the leaves to the root and subsequently the messages are propagated back from the root to the leaves (cf. junction tree inference).

The messages are constructed as follows (from variables to factors on the top, and vice versa on the bottom; $\mathcal{N}(\cdot)$ are the neighbors of the node/factor):

$$\mu_{i\to\alpha}(x_i) = \prod_{\alpha\in\mathcal{N}(i)\setminus\{\alpha\}} \mu_{\alpha\to i}(x_i)$$
$$\mu_{\alpha\to i}(x_i) = \sum_{\boldsymbol{x}_\alpha;\, x_{\alpha,i}=x_i} f_\alpha(\boldsymbol{x}_\alpha) \prod_{j\in\mathcal{N}(\alpha)\setminus\{i\}} \mu_{j\to\alpha}(x_{\alpha,i})$$

Subsequently (after propagating the messages), the beliefs can be computed as follows:

$$b_i(x_i) = \prod_{\alpha\in\mathcal{N}(i)} \mu_{\alpha\to i}(x_i) \qquad\qquad b_\alpha(\boldsymbol{x}_\alpha) = f_\alpha(\boldsymbol{x}_\alpha) \prod_{i\in\mathcal{N}(\alpha)} \mu_{i\to\alpha}(x_{\alpha,i})$$

The complete pseudocode is given in algorithm 6.

The sum-product algorithm can be seen as an instance of dynamic programming combined with variable elimination: the big problem of inference is broken down into small sub-problems characterized by the different parts of the graph. Additionally, the sum-product algorithm can also be used to find most probable explanations by replacing the sums with $\max$'s.

Algorithm 6: Sum-Product Algorithm

Input: acyclic factor graph
Output: exact marginals for each variable and factor variables
// Initialize messages to a uniform distribution:
$\mu_{i\to\alpha}(x_i) \leftarrow 1$
$\mu_{\alpha\to i}(x_i) \leftarrow 1$
Choose a root node.
// Send messages from leaves to root:
$\mu_{i\to\alpha}(x_i) \leftarrow \prod_{\alpha\in\mathcal{N}(i)\setminus\{\alpha\}} \mu_{\alpha\to i}(x_i)$
$\mu_{\alpha\to i}(x_i) \leftarrow \sum_{\boldsymbol{x}_\alpha;\, x_{\alpha,i}=x_i} f_\alpha(\boldsymbol{x}_\alpha) \prod_{j\in\mathcal{N}(\alpha)\setminus\{i\}} \mu_{j\to\alpha}(x_{\alpha,i})$
// Compute beliefs:
$b_i(x_i) \leftarrow \prod_{\alpha\in\mathcal{N}(i)} \mu_{\alpha\to i}(x_i)$
$b_\alpha(\boldsymbol{x}_\alpha) \leftarrow f_\alpha(\boldsymbol{x}_\alpha) \prod_{i\in\mathcal{N}(\alpha)} \mu_{i\to\alpha}(x_{\alpha,i})$
// Normalize belifs and return the marginals:
$P_i(X_i = x_i) \propto b_i(x_i)$
$P_\alpha(\boldsymbol{X}_\alpha = \boldsymbol{x}_\alpha) \propto b_\alpha(\boldsymbol{x}_\alpha)$

8.1.1 Loopy Belief Propagation

So far, only acyclic factor graphs have been considered. However, most factor graphs have cycles (loops). Hence, the messages are no longer independent and could travel in circles! And due to these cycles, dynamic programming cannot help anymore and the problem of computing the exact marginals is #P-complete. In turns out, though, that just ignoring this fact and letting run message passing until convergence works fine in most cases.

However, in some cases, the messages might loop around and drive the beliefs to deterministic distributions as the beliefs are increased in every turn (as the messages are treated to come from independent parts of the graph). But usually, cycles are long or include weak correlations such that the influence is weak.

Some approaches for dealing with loops is to just stop after a certain number of iterations or stop when the beliefs do not change anymore which can also be accelerated using simulated annealing or similar techniques. Usually, if the algorithm converges, it produces a good approximation. Some exemplary implementations are Turbo Codes and Lifted Inference.

8.2 Sampling

Another method of approximate inference is *sampling*. A *sample* is an instantiation $\boldsymbol{x}$ of random variables $\boldsymbol{X}$ generated from a distribution $P(\boldsymbol{X})$. Also, it is desired to also incorporate evidence $\boldsymbol{E} = \boldsymbol{e}$ to sample from the distribution $P(\boldsymbol{X} \mid \boldsymbol{E} = \boldsymbol{e})$. To compute the expected value of a random variable, multiple samples can be generated and averaged which (according to the strong law of large numbers) will converge to the real expectation as the number of samples increases. Similarly, the probability of an outcome can be estimated by counting[1].

For a binary random variable $X \in \{0, 1\}$ with $P(X = 0) = \theta$, a sample can be generated straightforwardly by $x = \mathbb{1}[\epsilon > \theta]$ where $\epsilon \sim \mathcal{U}(0, 1)$ is sampled from a uniform continuous distribution. Random number generators for this range are assumed to exist from now on, and they do in most programming languages (usually as *pseudo* random number generators). For a Bayesian network with a huge number of random variables and complicated distributions, sampling is not that simple. The following sections will cover some methods for sampling from these Bayesian networks.

8.2.1 Forward Sampling

Forward sampling is a naive straightforward methods that works well when no evidence has to be considered. It starts from a node with no parents, generates a sample, and propagates this sample down the Bayesian network, instantiating each CPT into a simple distribution. With evidence, some samples have to be rejected as soon as the evidence is violated. The pseudocode is given in algorithm 7.

While forward sampling has the advantages to be really simple and to produce true independent samples, incorporating the evidence in this fashion has some major drawbacks: if the probability $P(\boldsymbol{E} = \boldsymbol{e})$ of the evidence is low, most of the samples will be rejected. Hence, lots of useless samples will be generated, making the whole procedure extremely slow.

Algorithm 7: Forward Sampling

Input: Bayesian network with variables $X_1, X_2, \ldots, X_n$, number of samples T, evidence $\boldsymbol{E} = \boldsymbol{e}$
Output: Set of samples $\boldsymbol{x}^{(1:T)}$

1 **for** $t = 1, 2, \ldots, T$ **do**
  // Initialize empty sample:
2 $\boldsymbol{x}^{(t)} \leftarrow$ *Empty*
3 **foreach** X_i *in a topological ordering* **do**
4 $x_i^{(t)} \leftarrow$ Sample from $P\big(X_i \mid \mathrm{Pa}(X_i) = \boldsymbol{x}^{(t)}\big)$
5 **if** $X_i \in \boldsymbol{E}$ *and* $x_i^{(t)} \neq e_i$ **then**
6 Reject sample and start over (go to line 2).

[1] Note that this is only really applicable for discrete random variables as a specific value of a continuous variable has measure zero.

8.2.2 Gibbs Sampling

Another more efficient method for sampling with evidence is *Gibbs sampling*, a *Markov chain Monte Carlo* (MCMC) method. Compared to forward sampling, the samples are, however, not independent anymore and form a *Markov chain* (hence the name MCMC). The samples are taken from a surrogate distribution P' that converges to the true distribution P over time (with guaranteed convergence if all probabilities are positive).

The basic idea is to sample only one variable at a time, keeping all other variables fixed,

$$x_i^{(t+1)} \sim P(\cdot \mid \boldsymbol{X}_{i+1:\tilde{n}} = \boldsymbol{x}_{i+1:\tilde{n}}^{(t)}, \boldsymbol{X}_{1:i-1} = \boldsymbol{x}_{1:i-1}^{(t)}, \boldsymbol{E} = \boldsymbol{e}), \tag{8.1}$$

where the evidence variables $\boldsymbol{E}$ are supposed to be the last $(n - \tilde{n})$ random variable such that $\boldsymbol{X} = \boldsymbol{X}_{1:\tilde{n}} \cup \boldsymbol{E}$. By generating iterative many times, the distribution of the samples converges to $P(\boldsymbol{X} \mid \boldsymbol{E} = \boldsymbol{e})$! However, the conditional (8.1) seems to be hard to compute. Recall from chapter 5 that a random variable X_i is independent of all other random variables given its Markov blanket

$$M(X_i) = \mathrm{Pa}(X_i) \cup \mathrm{Ch}(X_i) \cup \left(\bigcup_{X_j \in \mathrm{Ch}(X_i)} \mathrm{Pa}(X_j) \right).$$

Hence, the distribution reduces to

$$P\big(X_i \,\big|\, \boldsymbol{X} \setminus \{X_i\}\big) = P\big(X_i \,\big|\, \mathrm{Pa}(X_i)\big) \prod_{X_j \in \mathrm{Ch}(X_i)} P\big(X_j \,\big|\, \mathrm{Pa}(X_j)\big),$$

making it straightforward to compute when the Markov blanket is given as in (8.1). The pseudocode is given in algorithm 8.

To answer probabilities queries $P(X_i \mid \boldsymbol{E} = \boldsymbol{e})$ using Gibbs sampling, it is either possible to use counting of the samples (like in forward sampling), but it is also possible to leverage the surrogate probabilities directly:

$$\hat{P}(X_i \mid \boldsymbol{E} = \boldsymbol{e}) = \frac{1}{T} \sum_{t=1}^{T} P(X_i \mid \boldsymbol{X}_{i+1:\tilde{n}} = \boldsymbol{x}_{i+1:\tilde{n}}^{(t)}, \boldsymbol{X}_{1:i-1} = \boldsymbol{x}_{1:i-1}^{(t)}, \boldsymbol{E} = \boldsymbol{e})$$

This could yield a more accurate approximate of the probability.

Algorithm 8: Gibbs Sampling

Input: Bayesian network with variables $X_1, X_2, \ldots, X_n$, number of samples T, evidence $\boldsymbol{E} = \boldsymbol{e}$
Output: Set of samples $\boldsymbol{x}^{(1:T)}$

// Initialize zeroth sample somehow, e.g., uniformly:
$\boldsymbol{x}^{(0)} \leftarrow \mathit{Random}$
for $t = 1, 2, \ldots, T$ **do**
  for $t = 1, 2, \ldots, n$ **do**
    // Generate sample levaraging the Markov blanket $M(X_i)$ to compute the CPT:
    $x_i^{(t+1)} \leftarrow$ Sample from $P(\cdot \mid \boldsymbol{X}_{i+1:\tilde{n}} = \boldsymbol{x}_{i+1:\tilde{n}}^{(t)}, \boldsymbol{X}_{1:i-1} = \boldsymbol{x}_{1:i-1}^{(t)}, \boldsymbol{E} = \boldsymbol{e})$

Burn-In

One problem of Gibbs sampling is that the first samples do not follow P as the initialization is random. This is known as the *burn-in phase* and can be solved by throwing away the first K samples. However, K cannot be determined in a principled fashion and is set by intuition. Alternative approaches are to draw the first samples from an approximate $P(\boldsymbol{X} \mid \boldsymbol{E} = \boldsymbol{e})$ found by loopy belief propagation (subsection 8.1.1), for example.

Convergence: Irreducibility, Aperiodicity, and Ergodicity

The convergence of Gibbs sampling to a stationary distribution π^* is guaranteed under three conditions: the sample chain must be irreducible, aperiodic, and ergodic. Let $p_{ij} = P(\boldsymbol{x}^{(i)} \to \boldsymbol{x}^{(j)})$ be the *transition kernel* describing the probability of transition from $\boldsymbol{x}^{(i)}$ to $\boldsymbol{x}^{(j)}$ in the Markov chain. Then irreducibility, aperiodicity, and ergodicity are defined as follows:

- *Irreducibility:* A Markov chain is *irreducible* if every state can be reached from every other state (not necessarily in one step), i.e., $\forall i, j : \exists k : p_{ij}^{(k)}$ where k is the number of steps.
- *Aperiodicity:* Let $d(i) \coloneqq \gcd\{k > 0 : \text{it is possible to go from state } i \text{ to state } i \text{ in } k steps\}$, where $\gcd$ is the greatest common divisor. Then a Markov chain is *aperiodic* if $\forall i : d(i) = 1$, i.e., it is only possible to go back to the same state irregularly.
- *Ergodicity:* *Ergodic* states i, j are recurrent and aperiodic. States i, j are *recurrent* if the chain returns to it with probability 1, i.e., $\sum_n p_{ij}^{(n)} \to \infty$. An extra condition is that the expected recurrence time is finite which is trivial in finite chains.

The required ergodicity requires $\forall i, j : p_{ij} > 0$, i.e., it is possible to reach every state from every other state in one step with some probability. This generally holds in networks where all probabilities are positive. Consider a network containing (among others) two random variables X and Y that are correlated such that $X = 0$ iff $Y = 0$. Then once $X = 0$ or $Y = 0$ is reached, Gibbs sampling is force to set the other to zero, too. Hence, it will never be possible to change their values and parts of the sampling space remain uncovered.

However, it can take quite some time as the samples are dependent (autocorrelation) and variance is extremely high in high-dimensional settings. Hence, speeding up convergence has to tackle these problems: reducing the dependence between samples and reducing the variance. This will be covered in the upcoming two sections.

Reducing Autocorrelation

To reduce autocorrelation between samples, two major techniques have been developed: *skipping samples* and *randomizing the variable order*. While both of these are straightforward to implement and quite intuitive, they have different pros and cons. Of course, skipping samples is inefficient and samples are wasted. Additionally, it increases the variance. Randomizing the variable order, on the other hand, can even decrease the variance! However, it is a tad harder to implement.

Reducing Variance

Reducing the variance can again be tackled by two major techniques: *blocking* and *Rao-Blackwellization*. The former bundles highly correlated variables into *blocks* and samples them in one step, greatly reducing the variance and convergence speed. However, it comes at the cost that the domain that the blocks' domains grow exponentially... Raw-Blackwellization, on the other hand, simply reduces the number of variables to sample and samples only a subset. Of course, this reduces the variance of the samples.

Multi-Chain Gibbs Sampling

In *multi-chain Gibbs sampling*, M chains of size K are generated which generate independent samples and independent surrogate distributions P'_m. The probability $P(X_i \mid \boldsymbol{E} = \boldsymbol{e})$ can then be estimated as the average $\bar{P}(X_i \mid \boldsymbol{E} = \boldsymbol{e}) = \sum_{m=1}^{M} P'_m / M$.

8.2.3 Likelihood Weighting

Likelihood weighting is a forward sampling-based method for sampling that builds on clamping the evidence and weighting the samples by the evidence likelihood, working well for likely evidence. Pseudocode for this process is given in algorithm 9. After generating the samples and weights, the posterior marginals can be estimated. Like Gibbs sampling, likelihood weighting converges to the exact distribution. However, convergence may be slow, too and the sample variance is high.

Algorithm 9: Likelihood Weighting Sampling

Input: Bayesian network with variables $X_1, X_2, \ldots, X_n$, number of samples T, evidence $\boldsymbol{E} = e$
Output: Set of samples $\boldsymbol{x}^{(1:T)}$ and weights $w^{(1:T)}$

for $t = 1, 2, \ldots, T$ **do**
    // Initialize empty sample and sample weight:
    $\boldsymbol{x}^{(t)} \leftarrow \textit{Empty}$
    $w^{(t)} \leftarrow 1$
    foreach X_i *in a topological ordering* **do**
        if $X_i \in \boldsymbol{E}$ *and* $x_i^{(t)} \neq e_i$ **then**
            $x_i^{(t)} \leftarrow e_i$
            $w^{(t)} \leftarrow w^{(t)} P\big(E_i = e_i \mid \mathrm{Pa}(X_i) = \boldsymbol{x}^{(t)}\big)$
        else
            $x_i^{(t)} \leftarrow$ Sample from $P\big(X_i \mid \mathrm{Pa}(X_i) = \boldsymbol{x}^{(t)}\big)$

9 Tractable Probabilistic Models

In the current era of *deep learning*, the trend goes to stacking many layers in neural networks instead of relying on shallow representations as they turn out to be much more powerful. But, opposed to PGMs, they usually have no probabilistic semantics and learning requires extensive computation. Borrowing the idea of deep representations, *probabilistic circuits* have been developed. These circuits are computational graphs for joint distributions.

This chapter only touches upon tractable probabilistic models, circuits, and sum-product networks. For a more thorough treatment, see section "Probabilistic Circuits" in chapter "Probabilistic Graphical Models" in the summary of "Deep Learning: Architectures and Methods"[1].

9.1 Sum-Product Networks

Sum-product networks (SPNs) are a special kind of probabilistic circuits (and arithmetic circuits (ACs)) whose nodes consists of sum- and product-nodes. Additionally, the edges are weighted an the numbers flowing across them are multiplied with the weight before feeding into the next node. A neat property of SPNs is that inference is linear in network size!

To compute a full joint distribution, the inputs of the SPN are set to the respective values and then the prescribed calculation is carried out. To compute a marginal, the variables to marginalize are all set to 1 and then the prescribed calculation is carried out. To perform MAP, the sums are replaced by $\max$'s in the forward pass and by $\arg\max$'s in the backward pass to get the MPE.

Learning an SPN from data can be done in various fashions. The most simplistic one is to start with a dense SPN and find the structure by learning the parameters (where zero weights indicate the absence of a connection). However, this can cause vanishing gradients by the many multiplications. To directly learn the (Tree-) SPN's structure, testing for independencies similar to learning Bayesian networks (subsubsection 6.2.4) is also a fruitful choice. These tests then identify product nodes while subsequently clustering algorithms like k-means can be used for identifying the sum nodes (which can be understood as mixture models).

Another major advantage of SPNs is that they can be compiled into simple and library-free code that can be executed on mobile devices. This is useful for real-time applications and similar. As they encode multivariate distributions as polynomials of univariate distributions, they can also be used in symbolic methods for inference.

[1] `https://fabian.damken.net/summaries/cs/elective/iws/dlam/dlam-summary.pdf`

10 Deep Generative Models

While data is relatively cheap in the era of big data, the majority of the available data is unlabeled and providing labels is time intensive, costly, requires expertise, etc. Hence, it is desirable to be able to learn without supervision. (Deep) *generative models* are concerned with modeling the distribution of the data itself instead of relying on a discriminator function. The found distribution can then also be used to generate new data points (e.g., images). A major challenge in generative models is the *curse of dimensionality:* in high dimensions, the data is usually sparse and an exponential amount of data is needed to fill the space.

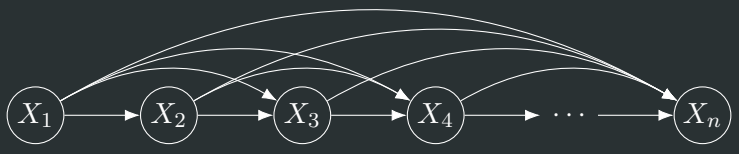
In this chapter, multiple variants of deep generative models are covered, categorized into likelihood-based and likelihood-free methods.

10.1 Likelihood-Based

Likelihood-based methods share the property that they learn by maximizing the (log-) likelihood of a model.

10.1.1 Autoregressive Generative Models

Autoregressive generative models are fully observed graphical models of the form



factorizing as

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid X_1, X_2, \ldots, X_{i-1}).$$

Of course, the CPTs for this model would again have exponentially many entries in the number of variables. Later on, it will be introduced to parameterize the distributions with a neural network and approximating the distribution.

To learn the parameters, it is possible to directly maximize the log-likelihood over the dataset $\mathcal{D}$:

$$\arg\max_{\theta} \log P(\mathcal{D} \mid \theta) = \arg\max_{\theta} \sum_{\boldsymbol{x} \in \mathcal{D}} \log P(\mathcal{D} \mid \theta) = \arg\max_{\theta} \sum_{\boldsymbol{x} \in \mathcal{D}} \sum_{i=1}^{n} \log P(x_i \mid x_1, x_2, \ldots, x_{i-1}).$$

The constraint on tractable factors allows exact computation of the likelihood! The conditionals can be evaluated in parallel during training, making it faster. To sample from the model, forward sampling can be used with the usual caveats (i.e., being slow with evidence).

As the number of entries in the CPTs of this model rise exponentially with the number of RVs, alternative parametrizations are desired. Most of the time, neural networks are used:

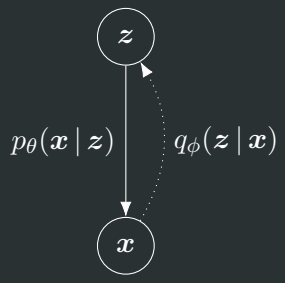


Figure 10.1: Graphical model of a variational auto-encoder with the latent states $\boldsymbol{z}$, the decoder network $p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$ and the encoder network $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$.

- Fully-Visible Sigmoid Belief Network (FVSBN)
- Neural Autoregressive Density Estimator (NADE): one hidden layer NN
- Masked Auto-Encoder Distribution Estimator (MADE): multilayer NN

Also other variants based on recurrent and convolutional neural networks have been proposed.

10.1.2 Variational Auto-Encoders

Variational autoencoders (VAEs) are models using latent variables and inference networks to infer the latent distribution (see Figure 10.1 for the graphical model). Its goal is to maximize the marginal log-likelihood $\log p_\theta(\mathcal{D})$ of the dataset. But this quantity is intractable! The key idea of VAEs is to approximate the posterior $p_\theta(\boldsymbol{z} \mid \boldsymbol{x})$ with a simpler distribution $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ represented by a neural network. Inference is then cast as optimization over θ and ϕ.

By massaging the quantity $\log p_\theta(\boldsymbol{x})$ a bit, one arrives at the following representation:

$$\log p_\theta(\boldsymbol{x}) = \underbrace{\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log \frac{p_\theta(\boldsymbol{x}, \boldsymbol{z})}{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})}\right]}_{\mathrm{ELBO}(\boldsymbol{x};\theta,\phi)} + D_{\mathrm{KL}}\big(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \,\big\|\, p_\theta(\boldsymbol{z} \mid \boldsymbol{x})\big).$$

The last term (the KL-divergence) is still intractable, but as it is non-negative, it can be discarded, yielding the *evidence lower bound* (ELBO), the left term. By optimizing the ELBO, the posterior approximate gets better and better and finally, the model can be used to generate new data using forward sampling.

Note that the ELBO can be decomposed as

$$\mathrm{ELBO}(\boldsymbol{x}; \theta, \phi) = \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_\theta(\boldsymbol{x} \mid \boldsymbol{z})\big] - D_{\mathrm{KL}}\big(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \,\big\|\, p_\theta(\boldsymbol{z})\big),$$

where the first term is the negative reconstruction error and the second is a penalty for diverging from the prior $p_\theta(\boldsymbol{z})$.

But while VAEs are quite successful, there are still various open research questions, e.g., better optimization techniques, mor expressive approximations, alternative loss functions, ...

10.1.3 Normalizing Flows

Normalizing flows leverage invertible neural networks to find a mapping $\boldsymbol{f}_\theta : \mathbb{R}^n \to \mathbb{R}^n$ between a latent space $\boldsymbol{z}$ and $\boldsymbol{x}$ that is invertible and deterministic such that $\boldsymbol{x} = \boldsymbol{f}_\theta(\boldsymbol{z})$ and $\boldsymbol{z} = \boldsymbol{f}_\theta^{-1}(\boldsymbol{x})$. One example for such

functions is a linear map. Then, change of variables

$$p_X(\boldsymbol{x}) = p_Z(\boldsymbol{z}) \left(\det \frac{\partial \boldsymbol{f}_\theta^{-1}}{\partial \boldsymbol{X}} \bigg|_{\boldsymbol{X}=\boldsymbol{x}} \right)$$

is used to define the distributions. In normalizing flows, multiple of these transformations are composed with each other to build complex structures and distributions.

Learning is again done by maximizing the log-likelihood and exact likelihood-evaluation is possible by design. For sampling, forward sampling can be applied straightforwardly. Also, the latent representation can be inferred using the inverse transformation.

10.2 Likelihood-Free

Opposed to likelihood-based models, *likelihood-free* methods do not rely on the computation of a likelihood for learning.

10.2.1 Generative Adversarial Networks

Generate adversarial networks (GANs) feature two neural distributions: a generator trying to generate fake images and a discriminator trying to distinguish fake and real images. By training both in an alternating minimax fashion, the generator learns to generate better and better images while the discriminator gets better at differentiating them.

10.3 Applications in Scientific Discovery

Generative models have various applications, e.g., causal discovery and inference, model-based reinforcement learning, scientific discovery. But there are still various open questions, e.g., evaluation metrics for sampling and bias and fairness in generative models.