Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.438 Algorithms for Inference
Fall 2020

1 Overview and Introduction

This course focuses on how to address inference in complex engineering settings. While driven by applications, the course is *not* about these applications. Rather, it emphasizes a common foundation and conceptual framework for inference-related questions arising in various fields not limited to machine learning, signal processing, artificial intelligence, computer vision, control, and communication.

Our focus is on the general problem of *inference*: learning about the generically hidden state of the world that we care about from available observations. This course is part of a two course sequence: 6.437 and 6.438:

- **6.437** "Inference and Information" is about fundamental principles and concepts of inference. It is offered in the Spring.
- **6.438** "Algorithms for Inference" is about how to perform inference in a computationally efficient manner by exploiting structure. It is offered (now) in the Fall.

A key theme is that there is a fundamental relationship between the degree of structure inherent in a given problem, and how efficiently inference can ultimately be performed. We will explore this relationship using the powerful language of probabilistic graphical models.

Graphical models combine probability theory with graph theory into a powerful framework for multivariate statistical modeling by providing an economic representation of a joint distribution using local relationships between variables.

Remark: graphical models vs networks. Nowadays, we see a lot of settings where the model/data itself is a graph or a network (e.g., social networks). These setups are different from the graphical models we will study in this class. In graphical models, we represent random variables as nodes, and we use edges (i.e., connections between nodes) to represent the relationships between the random variables. In other words, only the nodes are observed; edges are there to represent the structure in the joint probability distribution of the random variables. In contrast, in network models, the edges themselves correspond to the random variables that are observed, which makes it different from graphical models. Although these two family of models are related to each other, our focus in this course will be on graphical models, not network models.

1.1 Origins of graphical models

Graphical models were independently developed many decades ago in several different areas. The *statistical physics* community used undirected graphs to represent a distribution over a large system of interacting particles [Gibbs, 1902]. In *genetics*, directed graphs were used to model inheritance in natural species [Wright, 1921]. Also, the *statistics* community used graphs to represent interactions in multi-dimensional contingency tables [Bartlett, 1935]. Graphical models have subsequently found widespread application and proven enormously successful in many problem domains in diverse fields. Moreover, they have served as a unifying framework for many classes of algorithms once thought to be unrelated. Even today, the study of graphical models is a very active research field—we will cover the foundations as well as contemporary approaches/perspectives on problems of statistical inference and learning.

1.2 Examples

To motivate the course, we begin with some highly successful engineering applications of graphical models and simple, efficient inference algorithms based on the important concept of message-passing.

1.2.1 Navigation

Consider control and navigation of robots, autonomous vehicles (e.g., self-driving cars) and spacecraft (e.g., lunar landings, guidance of space probes) with noisy observations from various sensors, depicted in Fig. 1. Abstractly, these scenarios can be viewed as ones where noisy observations of a hidden state are obtained. Accurately inferring the hidden state allows us to control and achieve a desired trajectory for the vehicle. Formally, such scenarios are often well-modeled by the Guassian graphical model shown in Fig. 2. An efficient inference algorithm for this graphical model is the Kalman filter, a sequential message-passing algorithm developed in the early 1960's that ultimately made possible landing on the moon and the space shuttle program.

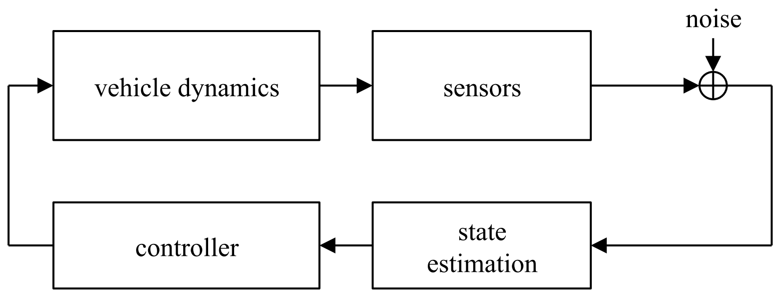

Figure 1: Navigation feedback control in the presence of noisy observations.

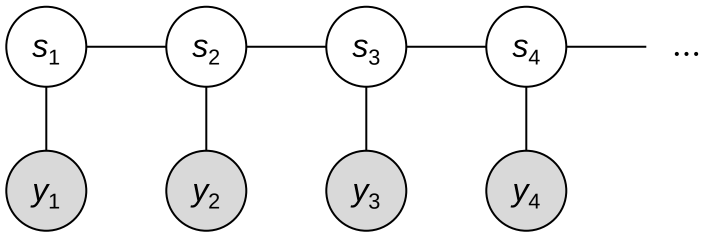

Figure 2: The Gaussian graphical model corresponding to a linear dynamical system, as used to model vehicle dynamics in autonomous and semi-autonomous control and navigation applications. In this case, the associated efficient inference algorithm is the Kalman filter and its generalizations.

1.2.2 Automatic speech recognition

Consider the task of decoding the audio of a person speaking, which plays an important role in a wide range of conspicuous applications, such as smartphone digital assistants. The most widely used approach to such speech recognition is to take the audio input, segment it into 10ms (or other small) time intervals, and then capture appropriate 'signature' or 'structure' (in the form of frequency response) of each of these time segments (the so-called cepstral coefficient vector or the "features"). Speech has structure that is captured through correlation in time, i.e., what one says now and soon after are correlated. A succinct way to represent this correlation is via an undirected graphical model called a Hidden Markov model, shown in Fig. 3. This model, which is ubiquitous in the speech recognition community, is very closely related to the graphical model described in the navigation example above. The difference is that the hidden states in this speech example are discrete rather than continuous. In this case, the Viterbi algorithm is the natural efficient inference algorithm for this scenario, together with the corresponding Baum-Welch learning algorithm. Both make use of message-passing to achieve their efficiency.

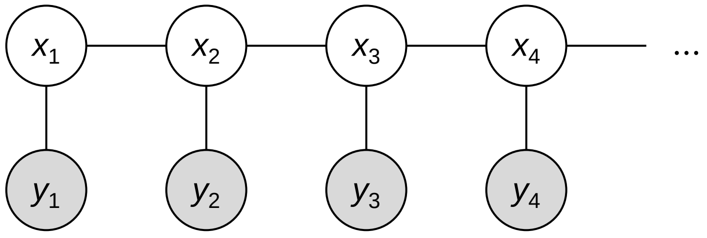

Figure 3: Hidden Markov model for automatic speech recognization. In this case, efficient inference takes the form of the Viterbi algorithm, and efficient modeling takes the form of the Baum-Welch algorithm.

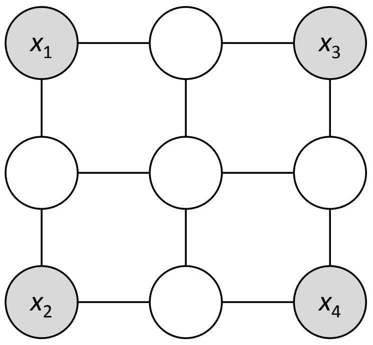

Figure 4: A simple Markov random field model for images in computer vision applications. This particular example is an instance of what is referred to as a two-dimensional Ising model. The shaded nodes represent observed pixel values from an available low-resolution image, while the remaining nodes represent pixels in the high-resolution image we might wish to determine. In such applications, loopy belief propagation is a popular algorithm for performing efficient inference.

1.2.3 Computer Vision and Image Processing

Many high- and low-level computer vision tasks can be viewed as problems of inference. Consider, for example, enhancing the resolution of an image, i.e., building a higher resolution version of a low-resolution image. This is an inference problem as we wish to predict or infer from the low-resolution image (observation) detail in the corresponding high-resolution image. Or, as another example, consider recovering an image from a noise-corrupted version of it. In this case, the denoising task can be viewed as an inference problem.

In these applications, the structure in images that makes such inference effective is frequently expressed in the form of an undirected graphical model, also known as a Markov random field (MRF). A simple but popular example of such a model is depicted in Fig. 4, and expresses that the behavior of pixels is governed by nearest-neighbor interactions, and that as a result, among other properties, nearby pixels are more likely to be similar than different. In this case, message-passing in the form of loopy belief propagation is one popular approach to performing efficient inference with such models.

1.2.4 Causal Relationships

In many real-life applications, there are causal relationships between random variables, which makes it quite natural to use directed graphs that can represent causality. Sewall Wright (1920) developed the foundations of directed graphical models when he was studying the heredity of features. Fig. 5 is an example of such a graphical model representing causal relationships. By figuring out the relationships between different features in the family tree represented as a directed graph, he studied the heredity of such features.

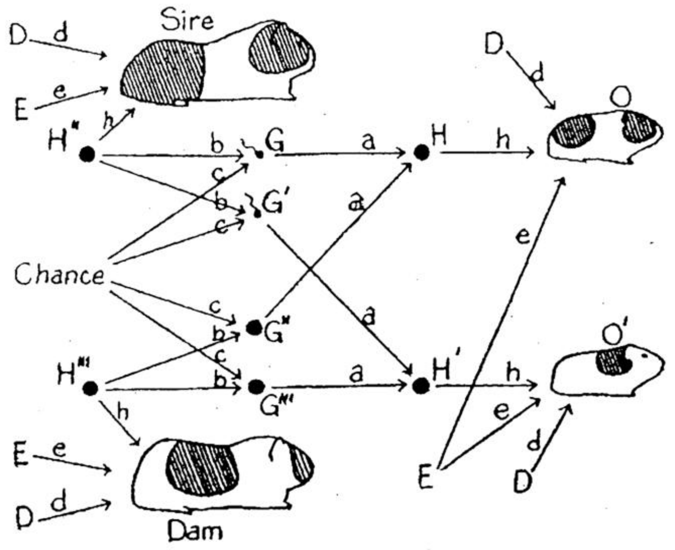

Figure 5: Directed graphical models: Sewall Wright's first path diagram (1920)

Note a key difference between the use of undirected and directed graphical models apparent in the examples above: While undirected graphical models as in Fig. 4 are mainly used for inference (i.e., identifying unobserved/latent variables from observed variables), directed graphical models as in Fig. 5 also play an important role in data interpretation through the identification of causal relationships. In this class we will develop and discuss algorithms for both of these problems: *inference* and *learning*. The former is concerned with inferring the latent state of the world for example by computing marginal/conditional distributions or calculating the most probable configurations (see Section 1.3) assuming the underlying model is known. The latter is concerned with learning the underlying model, either just the parameters or the graph itself, from data (see Section 1.4). In the heredity example, given the graph structure (i.e., the family tree), learning the parameters provides insights into how features are inherited. In other applications, we often do not have the luxury of knowing the graph structure. For example, when studying gene regulation, the critical problem is to learn the regulatory relationships between genes.

1.2.5 Reliable Communication (optional reading)

In a link in a digital communication network, we want to reliably send a message, encoded as k information bits, over a noisy channel. Because the channel may corrupt the message, we add redundancy to our message by encoding each message as an n-bit code sequence or *codeword*, where $n > k$. The receiver's task is to recover or *decode* the original message bits from the received, corrupted codeword, which can be viewed as a problem of inference. This scenario is depicted in Fig. 6. The structure of coded bits plays an important role in this inference problem.

A very simple example of a scheme to encode message bits as codeword bits is

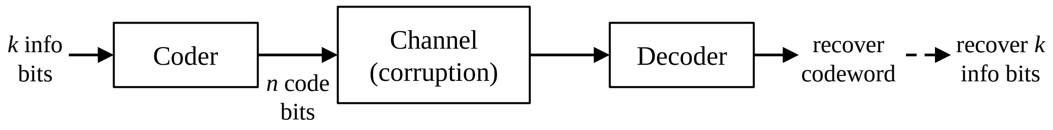

Figure 6: Basic digital communication system.

the so-called $(7, 4)$ Hamming code, whose structure is represented by the graphical model depicted in Fig. 7. Specifically, each 4-bit message is first mapped to a 7-bit codeword before transmission over the noisy channel. Note that there are $2^4 = 16$ different 7-bit codewords (among $2^7 = 128$ possible 7-bit sequences), each codeword corresponding to one of 16 possible 4-bit messages. The 16 possible codewords can be described by means of bits. These constraints are represented via the graphical model in Fig. 7, an example of a factor graph. Since this code requires 7 bits to be transmitted to convey 4 bits, we say the effective code rate is $R = 4/7$ bits per channel use.

The task of the receiver, as explained above, is to decode the 4-bit message that was sent based on the observations about the 7 received, possibly corrupted coded bits. The goal of the decoding algorithm is to infer the 4 message bits using the constraints in the Hamming code and based on a model for the noise in the channel. Given the graphical model representation, efficient decoding can be done using

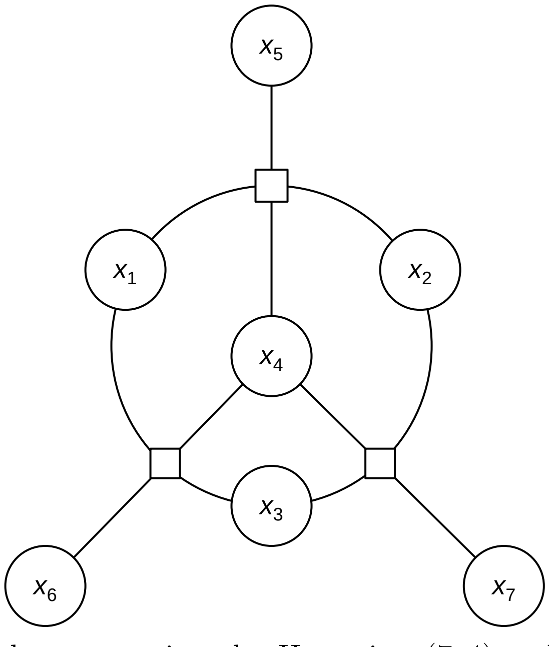

Figure 7: Factor graph representing the Hamming $(7, 4)$ code for protecting a set of transmitted bits. The corresponding efficient inference algorithm for decoding the transmitted bits from the corrupted received bits is referred to as loopy belief propagation.

a message-passing algorithm referred to as loopy belief propagation. For the simple $(7,4)$ Hamming code such an efficient decoder is not critical. However, the performance required in many applications requires codewords that are on the order of thousands of symbols long, corresponding to encoding thousands of bits at a time. In such cases, the existence efficient decoding algorithms based on message-passing has ultimately made such codes feasible to use in practice.

1.3 Inference, Complexity, and Graphs

We begin with a few broader perspectives on inference and the role played by graphical models in designing inference algorithms. We comment in advance that model learning is also an important part of the broader framework, and although we don't say much about it at this point (see Section 1.4), we will return to it in some detail later in the subject.

1.3.1 Inference problems

Consider a collection of random variables $\mathsf{x} = (\mathsf{x}_1, \dots, \mathsf{x}_N)$ and let the observations about them be represented by random variables $\mathsf{y} = (\mathsf{y}_1, \dots, \mathsf{y}_N)$. Let each of these random variables $\mathsf{x}_i, 1 \le i \le N$, take on a value in $\mathcal{X}$ (e.g., $\mathcal{X} = \{0,1\}$ or $\mathbb{R}$) and each observation variable $\mathsf{y}_i, 1 \le i \le N$ take on a value in $\mathcal{Y}$. Given observation $\mathsf{y} = \mathbf{y}$, the goal is to say something meaningful about possible realizations of $\mathsf{x} = \mathbf{x}$ for $\mathbf{x} \in \mathcal{X}^N$. One can consider there to be effectively two primary computation problems of interest, as we now describe.

Calculating (Posterior) Beliefs and Marginalization. Here the goal is to perform computations of the form

$$p_{\mathsf{x}|\mathsf{y}}(\mathbf{x}|\mathbf{y}) = \frac{p_{\mathsf{x},\mathsf{y}}(\mathbf{x},\mathbf{y})}{p_{\mathsf{y}}(\mathbf{y})} = \frac{p_{\mathsf{x},\mathsf{y}}(\mathbf{x},\mathbf{y})}{\sum_{\mathbf{x}' \in \mathcal{X}^N} p_{\mathsf{x},\mathsf{y}}(\mathbf{x}',\mathbf{y})}. \tag{1}$$

The denominator (i.e., normalization) in (1) is referred to as the *partition function*.

Evidently, computing the partition function is equivalent to performing *marginalization*, and for large alphabets (as typically arises in high-dimensions), is prohibitively expensive unless there is structure to exploit.

Let us further examine the complexity of the underlying basic marginalization task. For example, suppose $\mathcal{X}$ is a discrete set. Then computing

$$p_{\mathsf{x}_1}(x_1) = \sum_{x_2 \in \mathcal{X}} p_{\mathsf{x}_1,\mathsf{x}_2}(x_1, x_2) \tag{2}$$

requires roughly $|\mathcal{X}|$ operations for a fixed x_1. There are $|\mathcal{X}|$ many values of x_1, so overall, the number of operations needed scales as $|\mathcal{X}|^2$.

Now suppose we want to marginalize out M of the N components of x to generate an $(N-M)$-dimensional marginal. Then each summation will require on the order of $|\mathcal{X}|^M$ computations, and this must be done for each of the $|\mathcal{X}|^{N-M}$ possible values of the remaining variables, resulting in a total complexity on the order of $|\mathcal{X}|^N$, which we observe is *exponential* in the number of variables N, and thus typically prohibitive for even only moderately large values of N.

Note that this behavior should not be surprising. Indeed, without structure to exploit, the joint distribution of a collection of N variables each over alphabet $|\mathcal{X}|$ requires a table of $|\mathcal{X}|^N$, so just reading out all the values of the joint distribution has exponential complexity.

Calculating Most Probable Configurations (MPCs). There the goal is to perform the computation

$$\hat{\mathbf{x}} \in \underset{\mathbf{x}\in\mathcal{X}^N}{\arg\max}\, p_{\mathsf{x}|\mathsf{y}}(\mathbf{x}|\mathbf{y}). \tag{3}$$

Therefore,

$$\hat{\mathbf{x}} \in \underset{\mathbf{x}\in\mathcal{X}^N}{\arg\max}\, p_{\mathsf{x}|\mathsf{y}}(\mathbf{x}|\mathbf{y}) = \underset{\mathbf{x}\in\mathcal{X}^N}{\arg\max}\, \frac{p_{\mathsf{x},\mathsf{y}}(\mathbf{x},\mathbf{y})}{p_{\mathsf{y}}(\mathbf{y})} = \underset{\mathbf{x}\in\mathcal{X}^N}{\arg\max}\, p_{\mathsf{x},\mathsf{y}}(\mathbf{x},\mathbf{y}). \tag{4}$$

Without any additional structure, the above optimization problem obviously requires searching over all $|\mathcal{X}|^N$ entries in the table representing the joint distribution $p_{\mathsf{x}|\mathsf{y}}(\cdot|\mathbf{y})$, so has complexity that is exponential in N.

1.3.2 Structure

To illustrate the dramatic impact structure can have, consider the following somewhat extreme example. Specifically, suppose now that the N random variables of interest are mutually independent, so that

$$p_{\mathsf{x}_1,\mathsf{x}_2,\ldots,\mathsf{x}_N}(x_1,x_2,\ldots,x_N) = p_{\mathsf{x}_1}(x_1)\,p_{\mathsf{x}_2}(x_2)\cdots p_{\mathsf{x}_N}(x_N). \tag{5}$$

Then the marginalization inherent in posterior belief calculation can be done separately for each variable, and each has complexity $|\mathcal{X}|$, so the total complexity is of order $N|\mathcal{X}|$, which is *linear* in N.

Similarly, most probable configurations can be obtained by separately finding the assignment of each variable that maximizes its own probability. As there are N variables, the overall computational complexity is also linear in N, i.e., of order $N|\mathcal{X}|$.

Evidently, independence structure dramatically reduces the complexity of basic inference tasks. As we will develop in this subject, more generally more sophisticated forms of independence and related structure in distributions can be exploited to likewise reduce the complexity of these basic inference tasks, and often to similarly surprising degree.

1.3.3 Causality and do-operations

In this class, we will study another operation called the *do-operation*, which is motivated by causal inference. For illustration, consider two random variables x and y with a causal relationship $\mathsf{x} \to \mathsf{y}$. For example, let $\mathsf{x} = \mathbb{1}_{\mathcal{X}}$ and $\mathsf{y} = \mathbb{1}_{\mathcal{Y}}$ where $\mathcal{X}$ and $\mathcal{Y}$ are the following events:

$$\begin{aligned} \mathcal{X} &= \{\text{A gets a chocolate truffle}\}, \\ \mathcal{Y} &= \{\text{A is happy}\}. \end{aligned}$$

Here, $\mathsf{x} \to \mathsf{y}$ because eating delicious chocolates makes A happy. The do-operation represents an intervention to the value of a random variable by setting it to a particular value. By setting x to a particular $x \in \{0, 1\}$ (e.g., by giving A chocolates), then the distribution of y changes (e.g., A will become happy); we denote the resulting distribution by

$$p(\mathsf{y} \mid \mathrm{do}(\mathsf{x} = x)).$$

In this case, we have $p(\mathsf{y} \mid \mathrm{do}(\mathsf{x} = x)) = p(\mathsf{y} \mid \mathsf{x} = x)$. It is important to note that this equality does not hold in general; having $\mathsf{x} \to \mathsf{y}$ is one of the special cases for it to hold. In particular, consider the intervention on y, where A is made happy/unhappy through a different intervention. Then, the distribution of x does not change because the chocolate has nothing to do with A's happiness. Thus, we have $p(\mathsf{x} \mid \mathrm{do}(\mathsf{y} = y)) = p(\mathsf{x})$.

Hence when intervening on x in $\mathsf{x} \to \mathsf{y}$ the do-operation is equivalent to conditioning, while when intervening on y in $\mathsf{x} \to \mathsf{y}$ the do-operation is equivalent to marginalizing. Understanding the do-operation is becoming increasingly important, because interventional data is becoming more prevalent in many different applications, including recommender systems, personalized ads, gene knockout experiments, etc. In addition, we will also analyze under what assumptions causal claims like predicting the effect of a particular intervention through the do-operation can be performed based on purely observational data.

1.4 Hierarchy of learning problems

There are different kinds of learning problems, and we can classify those into the following hierarchy, in increasing order of complexity:

1. Given the factorization structure, learn (i.e., estimate) the parameters of the factors.
2. Given the relevant variables and their alphabets, learn the factorization (or graph) structure.
3. Given the variables, learn their alphabets, and, in turn, their (factorized) joint distribution.

4. Learn the number of random variables in the system, their alphabets, and their joint distribution.

In this class, we will focus on the first two problems, and we will see that the two problems are closely related.

1.5 Graphical models

As explained above, graphical models are an extremely natural way to capture the kinds of independence and factorization structure in phenomena that can be exploited to increase the efficiency of inference. While there is an extraordinary range of types of graphical models in the broader literature, in this class we will focus on the most basic ones, which serve as the key building blocks for the discipline. In particular, we will develop (to varying degrees):

1. directed acyclic graphs (DAG's), also referred to as "Bayesian networks," which among other applications are widely used to capture causal relationships in phenomena;

2. undirected graphs, also referred to as "Markov random fields," which naturally represent many types of conditional independencies; and

3. factor graphs, which can capture a wide range of types of factorization structure in joint distributions, and related algebraic constraints.

In general, a graph $\mathcal{G}$ consists of a collection of nodes $\mathcal{V}$ and edges $\mathcal{E}$. In general, variables are associated with nodes, edges capture constraints between variables. In directed graphs, each edge has an orientation, while in the undirected graph they do not. Factor graphs have two types of nodes, only some of which represent random variables.

Ultimately, in the development of inference algorithms we will also find the graphical model convenient for expressing the underlying distributed computation involved. In particular, we will find it natural to associate processors with various nodes, with edges capturing the interaction between processors.