Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
6.438 Algorithms for Inference
Fall 2014

1 Course Overview

This course is about performing inference in complex engineering settings, providing a mathematical take on an engineering subject. While driven by applications, the course is *not* about these applications. Rather, it primarily provides a common foundation 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 fundamental structures and algorithms for inference. It is offered (now) in the fall.

A key theme is that constraints imposed on system design are what make engineering challenging and interesting. To this end, **6.438** "Algorithms for Inference" is about recognizing structure in inference problems that lead to efficient algorithms.

1.1 Examples

To motivate the course, we begin with some highly successful applications of graphical models and simple, efficient inference algorithms.

1. *Navigation.* Consider control and navigation of spacecraft (e.g., lunar landings, guidance of shuttles) with noisy observations from various sensors, depicted in Figure 1. Abstractly, these scenarios can be viewed as a setup where noisy observations of a hidden state are obtained. Accurately inferring the hidden state allows us to control and achieve a desired trajectory for spacecraft. Formally, such scenarios are well modeled by an undirected Guassian graphical model shown in Figure 2. An efficient inference algorithm for this graphical model is the Kalman filter, developed in the early 1960's.

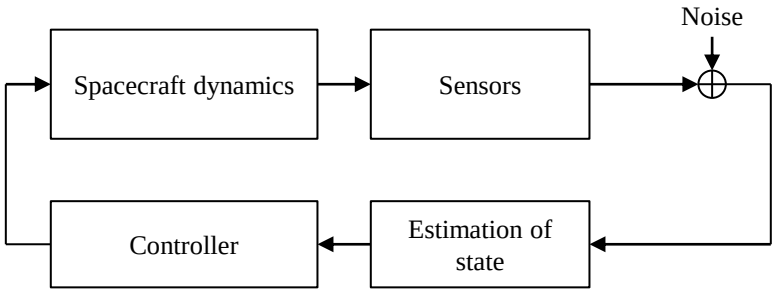

Figure 1: Navigation feedback control in the presence of noisy observations.

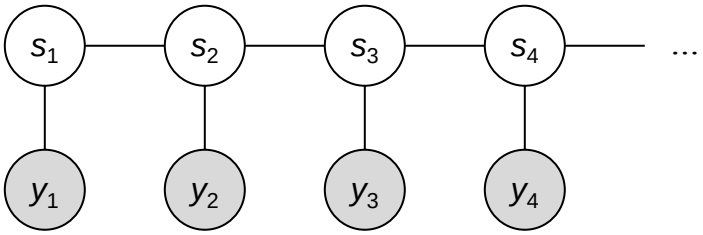

Figure 2: Undirected Gaussian graphical model for a linear dynamical system, used to model control and navigation of spacecraft. Efficient inference algorithm: Kalman filtering.

2. *Error correcting codes.* In a communication system, we want to 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 bits. A diagram showing this system is in Figure 3. The decoding task is an inference problem. The structure of coded bits plays an important role in inference.

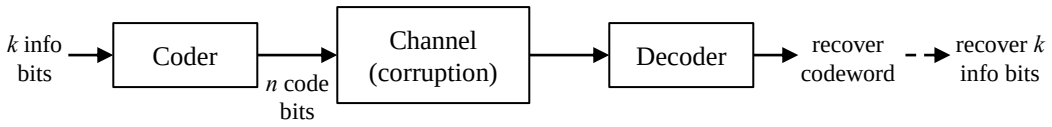

Figure 3: Simple digital communication system.

An example of a scheme to encode message bits as codeword bits is the Hamming $(7, 4)$ code, represented as a graphical model in Figure 4. Here, 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 constraints on the codeword bits. These constraints are represented via the graphical model in Figure 4, an example of a factor graph. Note that this code requires 7 bits to be transmitted to convey 4 bits, so the effective code rate is $4/7$.

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 some noise model for the channel. Given the graphical model representation, efficient decoding can be done using the loopy belief propagation algorithm, even for large codeword lengths (e.g., one million transmitted bits rather than 7).

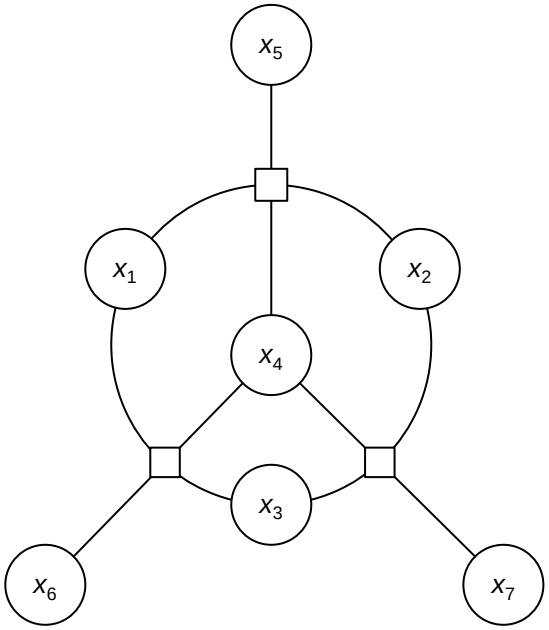

Figure 4: Factor graph representing the Hamming $(7, 4)$ code. Efficient inference algorithm: loopy belief propagation.

3. *Voice recogniziation.* Consider the task of recognizing the voice of specific individuals based on an audio signal. One way to do this is to take the audio input, segment it into 10ms (or some small) time interval, and then capture appropriate 'signature' or 'structure' (in the form of frequency response) of each of these time sgements (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 Figure 5. This model is similar to the graphical model considered for the navigation example earlier. The difference is that the hidden states in this speech example are discrete rather than continuous. The Viterbi algorithm provides an efficient, message-passing implementation for inference in such setting.

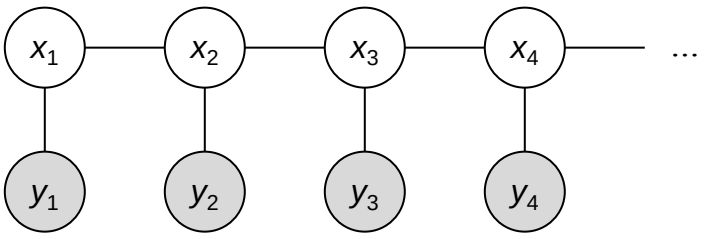

Figure 5: Hidden Markov model for voice recognization. Efficient inference algorithm: Viterbi.

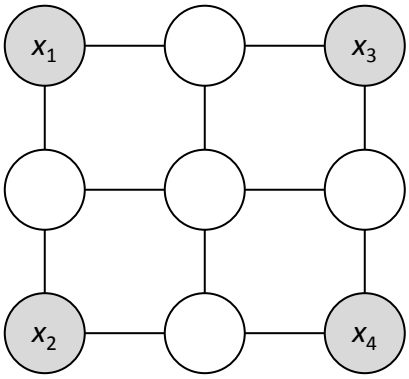

Figure 6: Markov random field for image processing. Efficient inference algorithm: loopy belief propagation.

4. *Computer vision/image processing.* Suppose we have a low resolution image and want to come up with a higher resolution version of it. This is an inference problem as we wish to predict or infer from a low resolution image (observation) how a high resolution image may appear (inference). Similarly, if an image is corrupted by noise, then denoising the image can be viewed as an inference problem. The key observation here is that images have structural properties. Specifically, in many real-world images, nearby pixels look similar. Such a similarity constraint can be captured by a nearest-neighbor graphical model of image. This is also known as a Markov random field (MRF). An example of an MRF is shown in Figure 6. The loopy belief propagation algorithm provides an efficient inference solution for such scenarios.

1.2 Inference, Complexity, and Graphs

Here we provide the key topics that will be the focus of this course: inference problems of interest in graphical models and the role played by the structure of graphical models in designing inference algorithms.

Inference problems. Consider a collection of random variables $\mathbf{x} = (x_1, \ldots, x_N)$ and let the observations about them be represented by random variables $\mathbf{y} = (y_1, \ldots, y_N)$. Let each of these random variables $x_i, 1 \leq i \leq N$, take on a value in $\mathcal{X}$ (e.g., $\mathcal{X} = \{0, 1\}$ or $\mathbb{R}$) and each observation variable $y_i, 1 \leq i \leq N$ take on a value in $\mathcal{Y}$. Given observation $\mathbf{y} = \mathbf{y}$, the goal is to say something meaningful about possible realizations of $\mathbf{x} = \mathbf{x}$ for $\mathbf{x} \in \mathcal{X}^N$. There are two primary computation problems of interest:

1. Calculating (posterior) beliefs:

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

   The denominator in equation (1) is also called the *partition function*. In general, computing the partition function is expensive. This is also called the "marginalization" operation. For example, suppose $\mathcal{X}$ is a discrete set. Then,

$$p_{x_1}(x_1) = \sum_{x_2 \in \mathcal{X}} p_{x_1,x_2}(x_1, x_2). \tag{2}$$

   But this requires $|\mathcal{X}|$ number of operations for a fixed x_1. There are $|\mathcal{X}|$ many values of x_1, so overall, the number of operations needed scales as $|\mathcal{X}|^2$. In general, if we are thinking of N variables, then this starts scaling like $|\mathcal{X}|^N$. This is not surprising since, without any additional structure, a distribution over N variables with each variable taking on values in $\mathcal{X}$ requires storing a table of size $|\mathcal{X}|^N$, where each entry contains the probability of a particular realization. So without structure, computing posterior has complexity exponential in the number of variables N.

2. Calculating the most probable configuration also called the maximum a posteriori (MAP) estimate:

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

   Therefore,

$$\begin{aligned} \hat{\mathbf{x}} &\in \arg\max_{\mathbf{x} \in \mathcal{X}^N} p_{\mathbf{x}|\mathbf{y}}(\mathbf{x}|\mathbf{y}) \\ &= \arg\max_{\mathbf{x} \in \mathcal{X}^N} \frac{p_{\mathbf{x},\mathbf{y}}(\mathbf{x},\mathbf{y})}{p_{\mathbf{y}}(\mathbf{y})} \\ &= \arg\max_{\mathbf{x} \in \mathcal{X}^N} p_{\mathbf{x},\mathbf{y}}(\mathbf{x},\mathbf{y}). \end{aligned} \tag{4}$$

   Without any additional structure, the above optimization problem requires searching over the entire space $\mathcal{X}^N$, resulting in an exponential complexity in the number of variables N.

Structure. Suppose now that the N random variables from before are independent, meaning that we have the following factorization:

$$p_{x_1,x_2,\ldots,x_N}(x_1, x_2, \ldots, x_N) = p_{x_1}(x_1) p_{x_2}(x_2) \cdots p_{x_N}(x_N). \tag{5}$$

Then posterior belief calculation can be done separately for each variable. Computing the posterior belief of a particular variable has complexity $|\mathcal{X}|$. Similarly, MAP estimation can be done by finding each variable's assignment that maximizes its own probability. As there are N variables, the computational complexity of MAP estimation is thus $N \cdot |\mathcal{X}|$.

Thus, independence or some form of factorization enables efficient computation of both posterior beliefs (marginalization) and MAP estimation. By exploiting factorizations of joint probability distributions and representing these factorizations via graphical models, we will achieve huge computational efficiency gains.

Graphical models. In this course, we will examine the following types of graphical models:

1. Directed acyclic graphs (DAG's), also called "Bayesian networks"
2. Undirected graphs, also called "Markov Random Fields"
3. Factor graphs which more explicitly capture algebraic constraints

Recall that a graph constains a collection of nodes and edges. Each edge, in the directed case, has an orientation. The nodes represent variables, and edges represent constraints between variables. Later on, we will see that for some inference algorithms, we can also think of each node as a processor and each edge as a communication link.

6.438 Algorithms for Inference

Fall 2014