Dynamic Bayesian Networks[*]

Kevin P. Murphy
www.ai.mit.edu/∼murphyk

12 November 2002

1 Introduction

Chapter **??** introduced hidden Markov models (HMMs), and Chapter **??** introduced state space models (SSMs), both of which are popular, but somewhat inflexible, models of sequential data. In this chapter, we consider more complex models. The key generalization is to represent the hidden state in terms of a set of random variables, instead of a single random variable.[1] Similarly we can represent the observations in a factorized or distributed manner. We can then use graphical models to represent conditional indepencies between these variables, both within and across positions in the sequence.

Sequential data comes in two main forms: temporal (time-series) data, which is generated sequentially by some causal process, and sequence data (e.g., bio-sequences or natural language), where we are more agnostic about the generating mechanism. For modelling time-series data, it is natural to use directed graphical models, which can capture the fact that time flows forward. Arcs within a time-slice can be directed or undirected, since they model "instantaneous" correlation. If all arcs are directed, both within and between slices, the model is called a dynamic Bayesian network (DBN). (The term "dynamic" means we are modelling a dynamic system, and does not mean the graph structure changes over time.) DBNs are quite popular because they are easy to interpret and learn: because the graph is directed, the conditional probability distribution (CPD) of each node can be estimated independently. In this chapter, we will focus on DBNs.

For modelling atemporal sequence data, it is possible to use directed or undirected graphical models. Much of our discussion of offline inference in temporal models is also applicable to atemporal models. The online inference methods we discuss can be applied to temporal models, but can also be used for sequential learning of static models, which is useful if the data is non-stationary or too large for batch methods.

We will not discuss learning (parameter estimation and model selection) in this chapter, since the techniques are just simple extensions of the methods for learning static models. Also, we defer discussion of sequential decision making (control/ reinforcement learning problems) to Chapter **??**.

2 Representation

In an HMM, the hidden state is represented in terms of a single discrete random variable, which can take on M possible values, $Q_t \in \{1, \ldots, M\}$. In an SSM, the hidden state is represented in terms of a single vector-valued random variable, $X_t \in \mathbb{R}^M$. In a DBN, the hidden state is represented in terms of a set of N_h random variables, $Q_t^{(i)}$, $i \in \{1, \ldots, N_h\}$, each of which can be discrete or continuous. Similarly, the observation can be represented in terms of N_o random varables, each of which can be discrete or continuous.

In an HMM/SSM, we have to define the transition model, $P(Q_t|Q_{t-1})$, the observation model, $P(Y_t|Q_t)$, and the initial state distribution, $P(Q_1)$. (These distributions may be conditioned on an input (control signal) U_t if present, e.g., the transition model would become $P(Q_t|Q_{t-1}, U_{t-1})$.) In a DBN, Q_t, Y_t and U_t represent sets of variables, so

---

[*] To appear in *Probabilistic Graphical Models*, M. Jordan

[1] It is always possible to combine continuous-valued variables into a single vector-valued variable, but this obscures any conditional independencies that may exist between the components, defeating the purpose of using a graphical model. See Section 2.9.

we define the corresponding conditional distributions using a two-slice temporal Bayes net (2TBN), which we shall denote by $B_{\rightarrow}$. The transition and observation models are then defined as a product of the CPDs in the 2TBN:

$$P(Z_t|Z_{t-1}) = \prod_{i=1}^{N} P(Z_t^{(i)}|\text{Pa}(Z_t^{(i)}))$$

where $Z_t^{(i)}$ is the i'th node in slice t (which may be hidden or observed; hence $N = N_h + N_o$), and Pa$(Z_t^{(i)})$ are the parents of $Z_t^{(i)}$, which may be in the same or previous time-slice (assuming we restrict ourselves to first-order Markov models). We can represent the unconditional initial state distribution, $P(Z_1^{(1:N)})$, using a standard (one-slice) Bayes net, which we shall denote by B_1. Together, B_1 and $B_{\rightarrow}$ define the DBN. The joint distribution for a sequence of length T can be obtained by "unrolling" the network until we have T slices, and then multiplying together all of the CPDs:

$$P(Z_{1:T}^{(1:N)}) = \prod_{i=1}^{N} P_{B_1}(Z_1^{(i)}|\text{Pa}(Z_t^{(i)})) \times \prod_{t=2}^{T}\prod_{i=1}^{N} P_{B_{\rightarrow}}(Z_t^{(i)}|\text{Pa}(Z_t^{(i)}))$$

For example, Figure 1 shows a 2TBN for a standard HMM/SSM, and an unrolled version for a sequence of length $T = 4$. In this case, $Z_t^{(1)} = Q_t$, $Z_t^{(2)} = Y_t$, Pa$(Q_t) = Q_{t-1}$ and Pa$(Y_t) = Q_t$, so the joint becomes

$$P(Q_{1:T}, Y_{1:T}) = P(Q_1)P(Y_1|Q_1) \times \prod_{t=2}^{T} P(Q_t|Q_{t-1})P(Y_t|Q_t)$$

The parameters for slices $t = 2, 3, \ldots$ are assumed to be the same (tied across time), as shown in Figure 2. This allows us to model unbounded amounts of data with a finite number of parameters.

In the following sections, we will present a series of more complex examples which should illustrate the large variety of models which we can define within this framework.

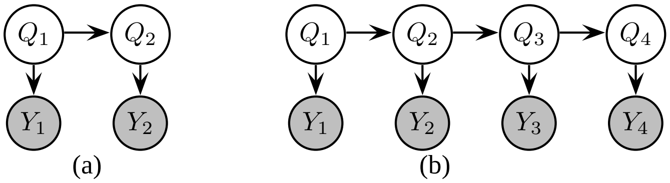


Figure 1: (a) A 2TBN for an HMM/SSM. (b) The model unrolled for $T = 4$ slices.

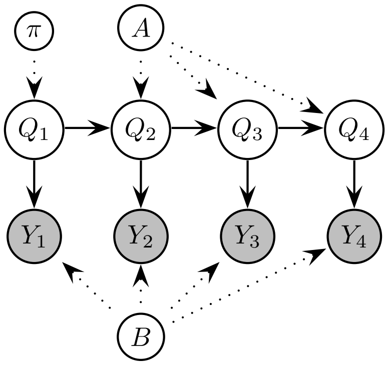


Figure 2: An HMM in which we explicitly represent the parameters (nodes with outgoing dotted arcs) and the fact that they are tied across time-slices. The parameters are $P(Q_1 = i) = \pi(i)$, $P(Q_t = j|Q_{t-1} = i) = A(i,j)$, and $P(Y_t = j|Q_t = i) = B(i,j)$. If the CPD for Y_t is a Gaussian, we would replace the B node with the mean and covariance parameters.

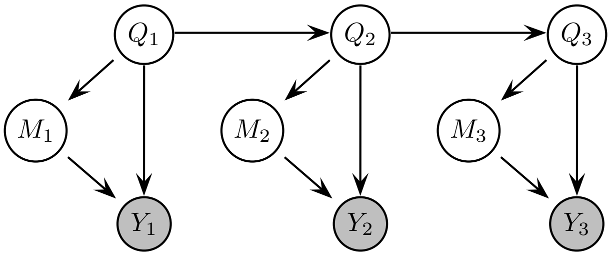


Figure 3: An HMM with mixture of Gaussians output.

2.1 HMMs with mixture-of-Gaussians output

In many applications, it is common to represent $P(Y_t|Q_t = i)$ using a mixture of Gaussians for each state i. We can explicitly model the mixture variable as shown in Figure 3. (M_t and Y_t are examples of transient nodes, since they do not have any children in the next time slice; by contrast, Q_t is a persistent node.) The CPDs for the Y_t and M_t nodes are as follows:

$$\begin{aligned} P(Y_t = y_t|Q_t = i, M_t = m) &= \mathcal{N}(y_t; \mu_{i,m}, \Sigma_{i,m}) \\ P(M_t = m|Q_t = i) &= C(i, m) \end{aligned}$$

The i'th row of C encodes the mixture weights for state i.

In many applications, it is common that the observations are high-dimensional vectors (e.g., in speech recognition, Y_t is often a vector of cepstral coefficients and their derivatives, $Y_t \in \mathbb{R}^{39}$), so estimating a full covariance matrix for each value of Q_t and M_t requires a lot of data. An alternative is to assume there is a single global pool of Gaussians, and each state corresponds to a different mixture over this pool. This is called a semi-continuous or tied-mixture HMM. In this case, there is no arc from Q_t to Y_t, so the CPD for Y_t becomes

$$P(Y_t = y_t|M_t = m) = \mathcal{N}(y_t; \mu_m, \Sigma_m)$$

The effective observation model becomes

$$\begin{aligned} P(Y_t|Q_t = i) &= \frac{\sum_m P(Y_t, M_t = m, Q_t = i)}{P(Q_t = i)} \\ &= \frac{\sum_m P(Q_t = i)P(M_t = m|Q_t = i)P(Y_t|M_t = m)}{P(Q_t = i)} \\ &= \sum_m P(M_t = m|Q_t = i)\mathcal{N}(y_t; \mu_m, \Sigma_m) \end{aligned}$$

Hence all information about Y_t gets to Q_t via the "bottleneck" M_t. The advantages of doing this are not only reducing the number of parameters, but also reducing the number of Gaussian density calculations, which speeds up inference dramatically. (We discuss inference in Section 4.) For instance, the system can calculate $\mathcal{N}(y_t; \mu_m, \Sigma_m)$ for each m, and then reuse these in a variety of different models by simple reweighting: typically $P(M_t = m|Q_t = i)$ is non-zero for only a small number of m's. The $M_t \rightarrow Y_t$ arc is like vector quantization, and the $Q_t \rightarrow M_t$ arc is like a dynamic reweighting of the codebook.

2.2 Auto-regressive HMMs

The standard HMM assumption that the observations are conditionally independent given the hidden state (i.e., $Y_t \perp Y_{t'}|Q_t$) is quite strong, and can be relaxed at little extra cost as shown in Figure 4. This model is sometimes called an auto-regressive HMM. This model reduces the effect of the Q_t "bottleneck", by allowing Y_t to be predicted by Y_{t-1} as well as Q_t; this results in models with higher likelihood (although not necessarily better classificatiom performance).

If Y_t is discrete, its CPD can be represented as a 3-dimensional table, $P(Y_t|Q_t, Y_{t-1})$. If Y_t is continuous, one possibility for its CPD is a conditional linear Gaussian:

$$P(Y_t = y_t|Q_t = i, Y_{t-1} = y_{t-1}) = \mathcal{N}(y_t; W_i y_{t-1} + \mu_i, \Sigma_i)$$

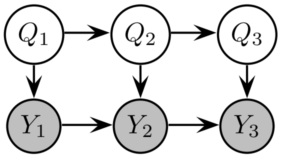


Figure 4: An auto-regressive HMM.

where W_i is the regression matrix given that Q_t is in state i. This model has a variety of different names: correlation HMM, conditionally Gaussian HMM, switching regression model, switching Markov model, etc. See Exercise **??** for some extensions.

2.3 Mixed-memory Markov models

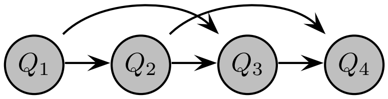


Figure 5: A trigram (second-order Markov) model, which defines $P(Q_t|Q_{t-1}, Q_{t-2})$.

One of the simplest approaches to modelling sequential data is to construct a Markov model of order k. These are often called n-gram models (where $n = k + 1$), e.g., standard first-order Markov models are bigram models ($n = 2$), second-order Markov models are trigram models ($n = 3$), etc. See Figure 5 for an example.

When Q_t is a discrete random variable with many possible values (e.g., if Q_t represents words), then there might not be enough data to reliably estimate $P(Q_t = k|Q_{t-1} = j, Q_{t-2} = i)$. A common approach is to create a mixture of lower-order Markov models:

$$P(Q_t|Q_{t-1}, Q_{t-2}) = \\ \alpha_3(Q_{t-1}, Q_{t-2}) f(Q_t|Q_{t-1}, Q_{t-2}) + \alpha_2(Q_{t-1}, Q_{t-2}) f(Q_t|Q_{t-1}) + \alpha_1(Q_{t-1}, Q_{t-2}) f(Q_t)$$

where the α coefficients may optionally depend on the history, and $f(\cdot)$ is an arbitrary (conditional) probability distribution. This is the basis of the method called deleted interpolation.

We can explicitly model the latent mixture variable implicit in the above equation as shown in Figure 6. Here $S_t = 1$ means use a unigram, $S_t = 2$ means use bi- and unigrams, and $S_t = 3$ means use tri-, bi- and uni-grams. S_t is called a switching parent, since it determines which of the other parents links are active, i.e., the effective topology of the graph changes depending on the value of the switch. Since S_t is hidden, the net effect is to use a mixture of all of these.

A very similar approach, called mixed-memory Markov models, is to always use mixtures of bigrams at different lags, e.g., if $S_t = 1$, we use $P(Q_t|Q_{t-1})$, and if $S_t = 2$, we use $P(Q_t|Q_{t-2})$. This can be achieved by simply changing the "guard condition" on the Q_{t-1} to Q_t link to be $S_t = 1$ instead of $S_t \geq 2$; similarly, the guard on the Q_{t-2} to Q_t link becomes $S_t = 2$ instead of $S_t \geq 3$. Hence S_t acts like the input to a multiplexer, choosing one of the parents to feed into Q_t. Using the terminology of [BZ02], S_t is the switching parent, and the other parents are the conditional parents. The overall effect is to define the CPD as follows:

$$P(Q_t|Q_{t-1}, \ldots, Q_{t-n}) = \sum_{i=1}^{n} P(Q_t|Q_{t-i}) P(S_t = i)$$

2.4 Factorial HMMs

Figure 7 shows a factorial HMM, which has a single output (observation) variable but has a distributed representation for the hidden state. Note that, although all the hidden chains are a priori independent, once we condition on the evidence, they all become correlated. Intuitively, this is because all the chains "compete" to explain each observation (the explaining away phenomenon); more formally, this can be seen by noticing that all the chains become connected in the moral graph, because they share a common observed child. This fact means exact inference in this model takes

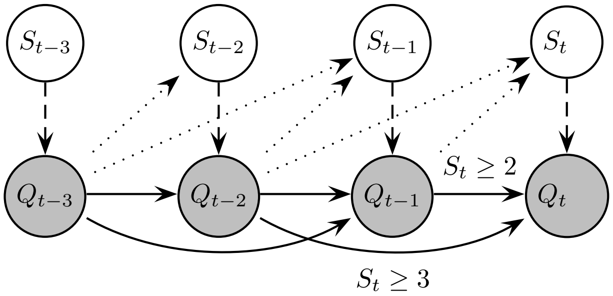


Figure 6: A mixed-memory Markov model. The dashed arc from S_t to Q_t means S_t is a switching parent: it is used to decide which of the other parents to use, either Q_{t-1} or Q_{t-2} or both. The conditions under which each arc is active are shown for the Q_t node only, to reduce clutter. In deleted interpolation, the $Q_{t-1} \rightarrow Q_t$ arc is active for states $S_t \in \{2,3\}$, and the $Q_{t-2} \rightarrow Q_t$ arc is active for states $S_t \in \{3\}$. (If $S_t = 1$, then $Q_t \perp (Q_{t-1}, Q_{t-2})$.) In mixed memory models, the $Q_{t-1} \rightarrow Q_t$ arc is active iff $S_t = 1$, and the $Q_{t-2} \rightarrow Q_t$ arc is active iff $S_t = 2$. The dotted arcs from Q_{t-1} and Q_{t-2} to S_t are optional, and reflect the fact that the coefficients can depend on the identity of the words in the window. See text for details. Based on Figure 15 of [Bil01].

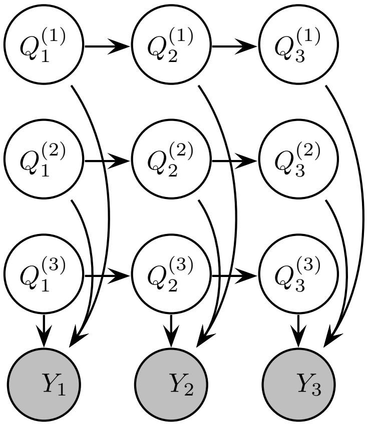


Figure 7: A factorial HMM with 3 hidden chains.

$O(M^{N_h})$ operations per time step. Unfortunately this is true for most DBNs, even ones which do not have densely connected nodes such as Y_t, as we discuss in Section 4.

A factorial HMM, like any discrete-state DBN, can be converted to a regular HMM by creating a single "mega" variable, Q_t, whose state space is the Cartesian product of each of the discrete hidden variables. However, this is a bad idea: the resulting "flat" representation is hard to interpret, inference in the flat model will be exponentially slower (see Section 4.2), and learning will be harder because there will be exponentially many more parameters.

This last statement needs some explanation. Although it is true that the entries of the transition matrix of the flat HMM have constraints between them (so the number of free parameters remains the same in both the flat and the factored representation), it is hard to exploit these constraints either in inference or learning. For example, if $N_h = 2$, the flat transition matrix can be computed as follows:

$$P(Q_t^{(1)} = j_1, Q_t^{(2)} = j_2 | Q_{t-1}^{(1)} = i_1, Q_{t-1}^{(2)} = i_2) = P(Q_t^{(1)} = j_1 | Q_{t-1}^{(1)} = i_1) \times P(Q_t^{(2)} = j_2 | Q_{t-1}^{(2)} = i_2)$$

Hence the free parameters, which are the pairwise transition matrices on the right hand side of this equation, get combined in a way which is hard to disentangle: see Figure 8. By contrast, If all CPDs are linear-Gaussian, so the resulting joint distribution is a multivariate Gaussian, sparse graphical structure corresponds to sparse matrices in the traditional sense of having many zeros: see Section 2.9.

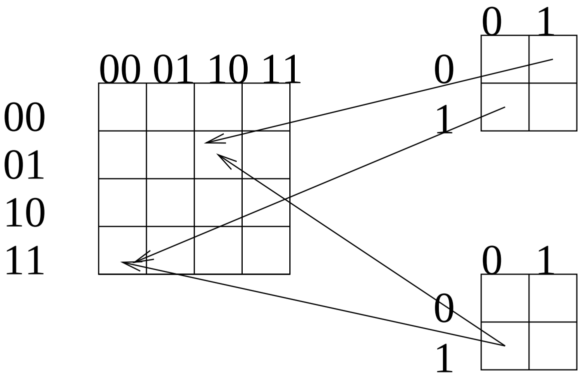


Figure 8: Computing the joint 4×4 transition matrix as the product of two 2×2 transition matrices. The cell of the 4×4 matrix in row 01, column 10 represents $P(X_t^1 = 1, X_t^2 = 0 | X_{t-1}^1 = 0, X_{t-1}^1 = 1)$ which is computed as $P(X_t^1 = 1 | X_{t-1}^1 = 0) \times P(X_t^2 = 0 | X_{t-1}^1 = 1)$, the product of the two incoming arrows.

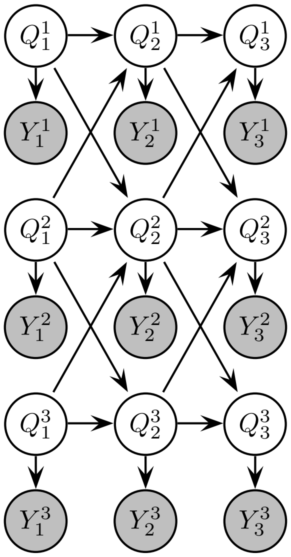


Figure 9: A coupled HMM with 3 chains.

2.5 Coupled HMMs

Figure 9 shows another example of a DBN, called a coupled HMM, in which the hidden variables are assumed to interact locally with their neighbors. Also, each hidden node has its own "private" observation.

This model has many applications. For example, in audio-visual speech recognition, one chain might represent the speech signal, and the other chain the visual signal (e.g., obtained from a lip tracking system). Another example concerns monitoring the state of a freeway using magnetic loop detectors distributed down the middle lane. Let $Y_t^{(i)}$ represent the measurement of sensor i at time t, and $Q_t^{(i)}$ represent the hidden state of location i at time t. It is natural to assume the hidden state at location i depends on the previous hidden state at i, plus the previous hidden states at locations immediately upstream and downstream. This can easily be modelled using a coupled HMM.

2.6 Variable-duration (semi-Markov) HMMs

The self-arc on a state in an HMM defines a geometric distribution over waiting times. Specifically, the probability we remain in state i for exactly d steps is $p_i(d) = (1 - p)p^{d-1}$, where $p = A(i, i)$ is the self-loop probability. One way to generalize this is to replace each HMM state with n new states, for some n to be determined, each with the same

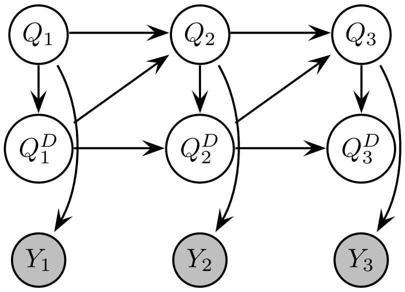
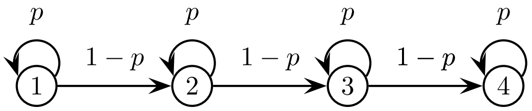


Figure 10: A variable-duration HMM. Q_t represents the state, and Q_t^D represents how long we have been in that state (duration).

emission probabilities as the original state. For example, consider this model.



Obviously the smallest sequence this can generate is of length $n = 4$. Any path of length l through the model has probability $p^{l-n}(1-p)^n$; the number of possible paths is $\binom{l-1}{n-1}$, so the total probability of a path of length l is

$$p(l) = \binom{l-1}{n-1} p^{l-n}(1-p)^n$$

This is the negative binomial distribution. By adjusting n and p, we can model a wide range of waiting times.

A more general approach is to explicitely model the duration of each state. This is called a semi-Markov model, because to predict the next state, it is not sufficient to condition on the past state: we also need to know how long we've been in that state. $p_i(d)$ can be represented as a table (a non-parametric approach) or as some kind of parametric function. If $p_i(d)$ is a geometric distribution, the model becomes equivalent to a standard HMM.

A first attempt at modelling this as a DBN is shown in Figure 10. We explicitly add Q_t^D, the remaining duration of state Q_t, to the state-space. (Even though Q_t is constant for a long period, we copy its value across every time slice, to ensure a regular structure.) When we first enter state i, Q_t^D is set to a value from from $q_i(\cdot)$; it then deterministically counts down to 0. When $Q_t^D = 0$, the state is free to change, and Q_t^D is set to the duration of the new state. We can encode this behavior by defining the CPDs as follows:

$$\begin{aligned} P(Q_t = j | Q_{t-1} = i, Q_{t-1}^D = d) &= \begin{cases} \delta(i,j) & \text{if } d > 0 \text{ (remain in same state)} \\ A(i,j) & \text{if } d = 0 \text{ (transition)} \end{cases} \\ P(Q_t^D = d' | Q_{t-1}^D = d, Q_t = k) &= \begin{cases} p_k(d') & \text{if } d = 0 \text{ (reset)} \\ \delta(d', d-1) & \text{if } d > 0 \text{ (decrement)} \end{cases} \end{aligned}$$

Since we have expanded the state space, inference in a variable-duration HMM is slower than in a regular HMM. The naive approach to inference in this DBN takes $O(TD^2M^2)$ time, where D is the maximum number of steps we can spend in any state, and M is the number of HMM states. However, we can exploit the fact that the CPD for Q^D is deterministic to reduce this to $O(TDM^2)$.

To facilitate future generalizations, we introduce deterministic "finish" nodes, that "turn on" when the duration counter reaches 0. This is a signal that Q_t can change state, and that a new duration should be chosen. See Figure 11. We define the CPDs as follows:

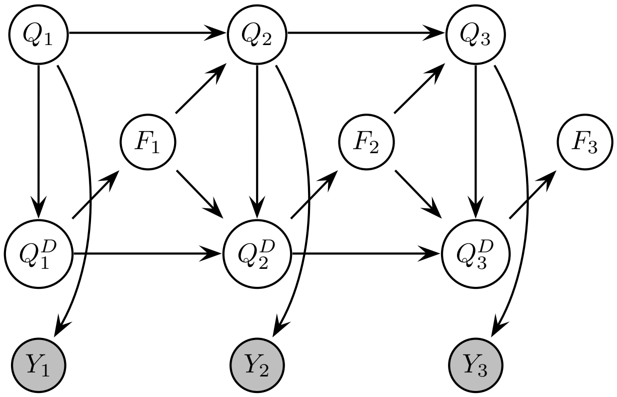


Figure 11: A variable-duration HMM with explicit finish nodes. Q_t represents the state, and Q_t^D represents how long we have been in that state (duration), and F_t is a binary indicator variable that turns to indicate that Q_t^D has finished.

$$\begin{aligned}
P(Q_t = j | Q_{t-1} = i, F_{t-1} = f) &= \begin{cases} \delta(i,j) & \text{if } f = 0 \text{ (remain in same state)} \\ A(i,j) & \text{if } f = 1 \text{ (transition)} \end{cases} \\
P(Q_t^D = d' | Q_{t-1}^D = d, Q_t = k, F_{t-1} = 1) &= p_k(d') \\
P(Q_t^D = d' | Q_{t-1}^D = d, Q_t = k, F_{t-1} = 0) &= \begin{cases} \delta(d', d-1) & \text{if } d > 0 \\ \text{undefined} & \text{if } d = 0 \end{cases} \\
P(F_t = 1 | Q_t^D = d) &= \delta(d, 0)
\end{aligned}$$

Note that $P(Q_t^D = d' | Q_{t-1}^D = d, Q_t = k, F_{t-1} = 0)$ is undefined if $d = 0$, since if $Q_{t-1}^D = 0$, then $F_{t-1} = 1$, by construction.

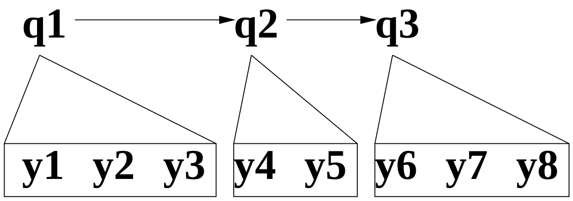


Figure 12: A time-series is divided into 3 segments of different lengths.

The overall effect of using a variable duration HMM is illustrated in Figure 12: the sequence is divided into a series of segments of different lengths; the observations within each segment are assumed to be iid, conditioned on the state. (We will generalize this in Section 2.7.) The assignment of values to the nodes in Figure 11 corresponding to this example is as follows:

Q_t	q_1	q_1	q_1	q_2	q_2	q_3	q_3	q_3
D_t	2	1	0	1	0	2	1	0
F_t	0	0	1	0	1	0	0	1
Y_t	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8

We see that the places where F turns on represent segment boundaries.

2.7 Segment models

The basic idea of a segment model is that each HMM state can generate a sequence of observations, as in Figure 12, instead of just a single observation, The difference from a variable-duration HMM is that we do not assume the observations within each segment are conditionally independent; instead we can use an arbitrary model for their joint

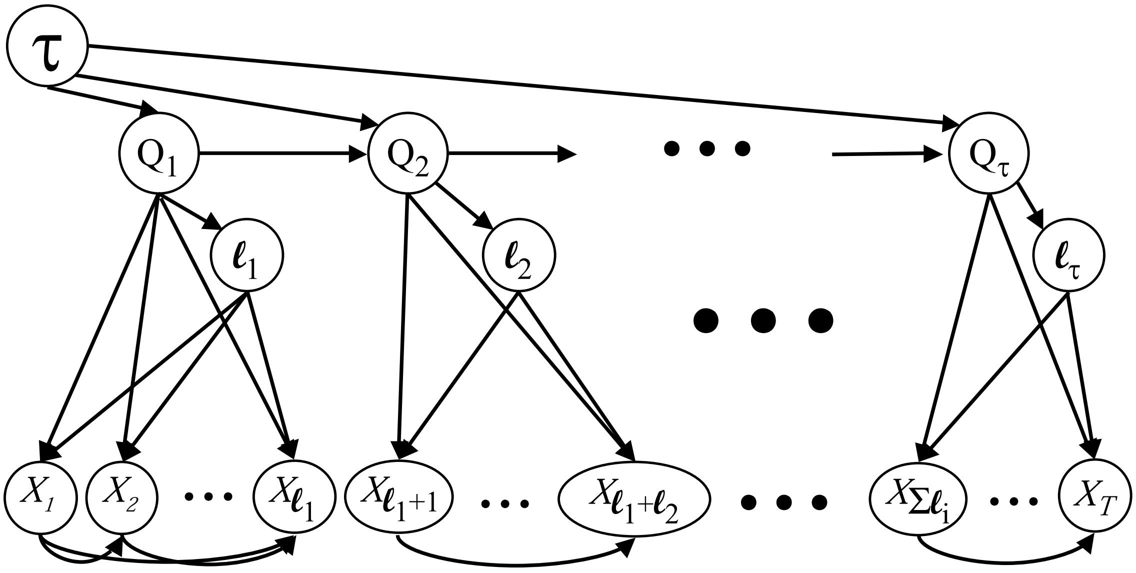


Figure 13: A schematic depiction of a segment model. The X_t nodes are observed, the rest are hidden. The X_t's within a segment need not be fully connected. Also, there may be dependencies between the observables in adjacent segments (not shown). This is not a valid DBN since the l_i's are random variables, and hence the structure is not fixed. Thanks to Jeff Bilmes for this Figure.

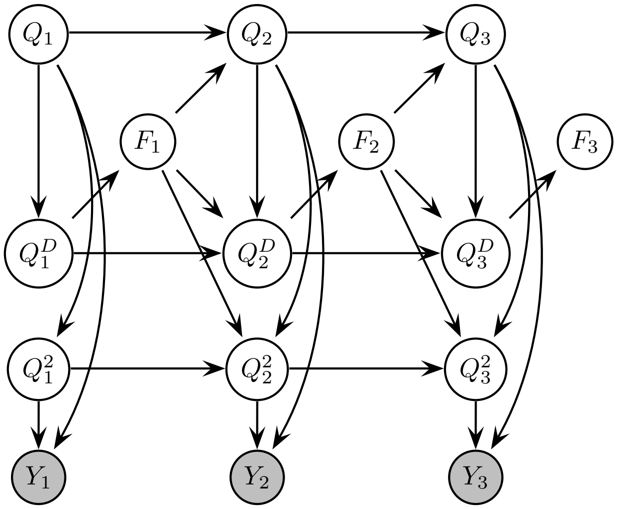


Figure 14: A segment model where each segment is modelled by an HMM.

distribution. The likelihood is given by

$$P(y_{1:T}) = \sum_{\tau} \sum_{q_{1:\tau}} \sum_{l_{1:\tau}} \prod_{i=1}^{l_i} P(\tau) P(q_i|q_{i-1}, \tau) P(l_i|q_i) P(y_{t_0(i):t_1(i)}|q_i, l_i)$$

where τ is the number of segments, l_i is the length of the i'th segment (that satisfies the constraint $\sum_{i=1}^{\tau} l_i = T$), and $t_0(i) = \sum_{j=1}^{i-1} l_j + 1$ and $t_1(i) = t_0(i) + l_i - 1$ are the start and ending times of segment i.

A first attempt to represent this as a graphical model is shown in Figure 13. This is not a fully specified graphical model, since the L_i's are random variables, and hence the topology is variable. To specify a segment model as a DBN, we must make some assumptions about the form of $P(y_{t:t+l}|q, l)$. Let us consider a particular segment and renumber so that $t_0 = 1$ and $t_1 = l$. If we assume the observations are conditionally independent,

$$P(y_{1:l}|Q_t = k, l) = \prod_{t=1}^{l} P(y_t|Q_t = k)$$

we recover the variable-duration HMM in Figure 11. (If $p(l|q)$ is a geometric distribution, this becomes a regular HMM.) Note that the number of segments is equal to the number of transitions of the Q_t nodes, or equivalently, the number of times F_t turns on. By considering all possible assignments to Q_t and Q_t^D (and hence to F_t), we consider all possible segmentations of the data.

The next simplest segment model is to model each segment $P(y_{1:l}|Q_t = k, l)$ using an HMM, i.e.,

$$P(y_{1:l}|Q_t = k, l) = \sum_{q_{1:l}} \pi_k(q_1) P(y_1|Q_t = k, Q_1^2 = q_1) \prod_{\tau=2}^{l} A_k(q_{\tau-1}, q_\tau) P(y_t|Q_t = k, Q_\tau^2 = q_\tau)$$

where Q_t^2 represents the state of the HMM within this particular segment. We can model this as a DBN as shown in Figure 14. (Naturally we could allow arcs between the Y_t arcs, as in Section 2.2). The CPDs for Q_t, Q_t^D and F_t are the same as in Figure 11. The CPD for Y_t gets modified because it must condition on both Q_t and the state within the segment-level HMM, Q_t^2. The CPD for Q_t^2 is as follows:

$$P(Q_t^2 = j|Q_{t-1}^2 = i, Q_t = k, F_{t-1} = f) \quad = \quad \begin{cases} \pi_k^2(j) & \text{if } f = 0 \text{ (reset)} \\ A_k^2(i,j) & \text{if } f = 1 \text{ (transition)} \end{cases}$$

It is straightforward to use other models to define the segment likelihood $P(y_{1:l}|Q_t = k, l)$, e.g., a second-order Markov model, or a state-space model (Section 2.9).

2.8 Hierarchical HMMs (HHMMs)

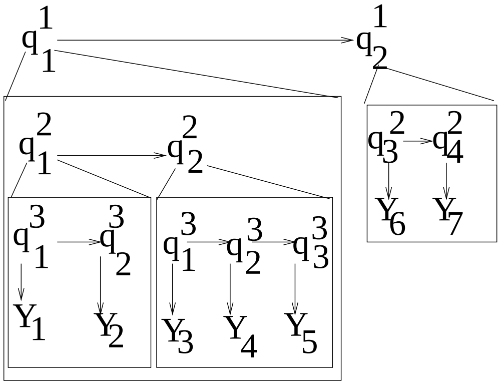


Figure 15: A time-series is hierarchically divided into segments. The transition/observation distributions of the sub-HMMs may be conditioned on their immediate context, or their entire surrounding context. In a context free system, information is not allowed to flow across segment boundaries, but this restriction can be lifted.

The Hierarchical HMM (HHMM) is a generalization of the segment HMM. It allows segment HMMs to be composed of segment sub-HMMs in a hierarchical fashion. See Figure 15 for an illustration of the basic idea. In addition, instead of specifying the length of each segment with a random variable, the length is defined implicitly by the amount of time it takes the sub-HMM to enter one of its end states. (The transition to the end state could be triggered by some environmental condition.) Entering an end-state terminates that segment, and returns control to the calling state, which is then free to change.

The calling context is memorized on a depth-limited stack. Hence an HHMM is less powerful than stochastic context-free grammars (SCFGs) and recursive transition networks (RTNs), both of which can handle recursion to an unbounded depth. However, HHMMs are sufficiently expressive for many practical problems, which often only involve tail-recursion (i.e., self transitions to an abstract state). Furthermore, HHMMs can be made much more computationally efficient than SCFGs. In particular, the inference algorithm for SCFGs, which is called the inside-outside algorithm, takes $O(T^3)$, whereas we can use any of the techniques we discuss in Section 3 to do inference in an HHMM in $O(T)$ time.

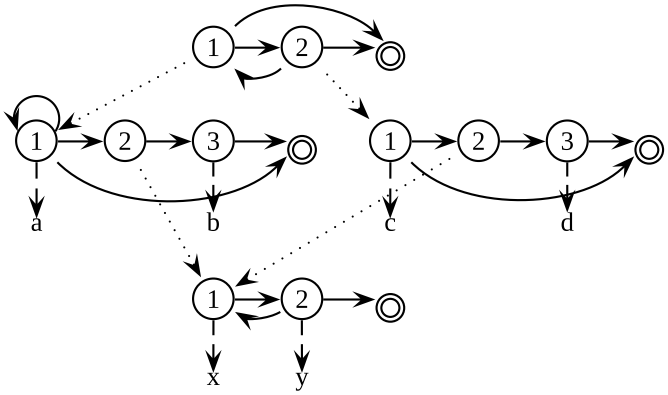


Figure 16: State transition diagram for a three-level HHMM representing the regular expression $(a^+|a^+(xy)^+b|c|c(xy)^+d)^+$. Solid arcs represent horizontal transitions between states; dotted arcs represent vertical transitions, i.e., calling a sub-HMM. Double-ringed states are accepting (end) states. Dashed arcs are emissions from production (concrete) states. In this example, each production state emits a unique symbol, but in general, it can have a distribution over output symbols.

Figure 16 shows the state transition diagram of an example HHMM which models the regular expression

$$\left(a^+ \;\mid\; a^+(xy)^+b \;\mid\; c \;\mid\; c(xy)^+d\right)^+ .^2$$

The key difference from using a flat HMM to model this is that the model for the sub-expression $(xy)^+$ is re-used in both the $a^+(xy)^+b$ and $c(xy)^+d$ subexpressions. In this small example, the benefits of re-use are minimal, but in general a hierarchical model with reuse needs substantially fewer parameters.

The state of the HHMM can be defined by specifying the value of the state variables at each level of the hierarchy, call them $Q_t^{(1)}, \ldots, Q_t^{(D)}$, where D is the depth. States which emit single observations are called "production states", and those that emit strings (by calling sub-HMMs) are termed "abstract states". For example, consider the states in the middle row on the left of Figure 16: state $(1, 1, -)$ is concrete and emits a, state $(1, 2, -)$ is abstract, and state $(1, 3, -)$ is concrete and emits b. The "leaf" states are all concrete, by definition: state $(1, 2, 1)$ emits x and $(1, 2, 2)$ emits y. Note that states $(1, 2, 1)$ and $(2, 2, 1)$ are the same (we could denote them by $(-, 2, 1)$), since the bottom-level HMM is reused in both top-level contexts.

The DBN that corresponds to the above example is shown in Figure 17. Let us see how the assignment of values to the nodes in the DBN correspond to different parses (paths through the automaton). Consider the string $aaxybc$; there are two possible parses, $(aaxyb)(c)$ or $(a)(axyb)(c)$, corresponding to whether the two a's belong together (i.e., both are generated from $a^+(xy)^+b$), or are separate (i.e., one is from a^+ and one is from $a^+(xy)^+b$). In addition, the c could be the beginning of the subexpression $c(xy)^+d$, or a single c; we assume we know this is a complete string (rather than a substring), hence the c is terminal (so $F_T^2 = 1$). The assignments to the nodes in the DBN for the joint aa hypothesis are as follows ($-$ represents a don't care value):

---

[2] This notation means the model must produce one or more *a*'s, *or* one or more *a*'s followed by one or more *x*'s and *y*'s followed by a single *b*, *or* a *c*, *or* a *c* followed by one or more *x*'s and *y*'s followed by a *d*. Having created one of these strings, it is free to create another one. An HHMM cannot make a horizontal transition before it makes a vertical one; hence it cannot produce the empty string. For example, in Figure 16, it is not possible to transition directly from state 1 to 2.

Q_t^1	1	1	1	1	1	2
Q_t^2	1	1	2	2	3	1
F_t^2	**0**	0	0	0	1	1
Q_t^3	-	-	1	2	-	-
F_t^3	-	-	0	1	1	-
Y_t	a	a	x	y	b	c

The assignments to the nodes in the DBN for the separate aa hypothesis are the same as above, except $F_1^2 = 1$ (shown in bold), representing the presence of a segmentation boundary. (This corresponds to the following path through the state transition diagram: from state $(1, 1, -)$ to its end state, returning to the root state $(1, -, -)$, and then re-entering $(1, 1, -)$.)

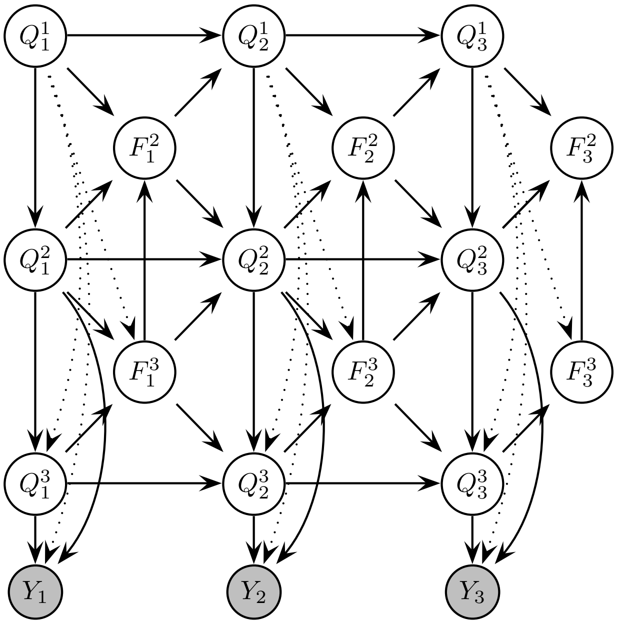


Figure 17: A 3-level HHMM represented as a DBN. Q_t^d is the state at time t, level d; $F_t^d = 1$ if the HMM at level d has finished (entered its exit state), otherwise $F_t^d = 0$. Shaded nodes are observed; the remaining nodes are hidden. If the length of the string is known to be T, we can set $F_T^d = 1$ for all d, to ensure all sub-models have terminated. If the dotted arcs from level 1 to the bottom are omitted, it represent the fact that the bottom-level HMM is shared amongst different top-level HMMs, as in the example in Figure 16.

We leave the details of defining the CPDs to Exercise 7.3.

2.8.1 Applications of HHMMs

The most obvious application of HHMMs is to speech recognition. It is natural to think of words as composed of phones, which produce subphones, which produce the acoustic signal. In practice, the mapping from words to phones is fixed by a phonetic dictionary, and the mapping from phones to acoustics is fixed by assuming a 3-state left-to-right HMM with mixture of Gaussian output. Hence the whole hierarchy can be "collapsed" into a flat HMM, with appropriate parameter tying to represent the fact that bottom levels of the hierarchy can be reused in different higher level contexts. However, if one were interested in learning the hierarchical decomposition, an HHMM might be useful. See Chapter **??** for more on speech recognition.

Another application of HHMMs is to map learning by mobile robots. It is natural to think of an indoor office-like environment as composed of floors, which are composed of corridors and rooms, which are composed of finer-grained locations. For this application, it is necessary to modify the model somewhat. Firstly, we typically condition all state transitions on the action performed by the robot. Secondly, and more subtly, when we leave one abstract state (e.g., corridor), the new concrete state (e.g., location within corridor) depends on the old concrete state, i.e., not all the old context is “popped off the stack” upon exiting. (In other words, the corridor models are not context free.) Instead, we condition the new starting state on some function of the old state. We leave the details as an exercise.

2.9 State-space models

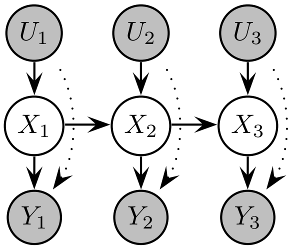


Figure 18: An HMM/SSM with input. The dotted arcs are optional, and reflect the fact that the input may affect the output directly.

The graph structure for an SSM looks identical to that for an HMM, since it makes the same conditional independence assumptions. Typically one also includes an input node, as in Figure 18. In an SSM, all the nodes are continuous, and all the CPDs are linear-Gaussian, i.e.,

$$\begin{aligned} P(X_t = x_t | X_{t-1} = x_{t-1}, U_t = u) &= \mathcal{N}(x_t; Ax_{t-1} + Bu, Q) \\ P(Y_t = y | X_t = x, U_t = u) &= \mathcal{N}(y; Cx + Du, R) \end{aligned}$$

We do not need to define $P(U_t)$, since this is an exogoneous input variable that is always observed.

In the case of SSMs, unlike for HMMs, there is a one-to-one correspondence between sparse graphical structure and sparse parametric structure. In particular, $A(i,j) > 0$ iff there is a directed arc from $X_{t-1}^{(i)}$ to $X_t^{(j)}$, and similarly for the other regression matrices, B, C and D. For the covariance matrices, we make use of the fact that, in an *undirected* Gaussian graphical model, zeros in the precision (inverse covariance) matrix correspond to absent arcs. Hence $Q^{-1}(i,j) > 0$ iff there is an undirected arc from $X_t^{(i)}$ to $X_t^{(j)}$ within the same time slice. For example, consider a factorial SSM, as in Figure 7, but where all the CPDs are linear-Gaussian, and there are two chains. The corresponding joint transition function is

$$P(X_t^1, X_t^1 | X_{t-1}^1, X_{t-1}^1) = \mathcal{N}\left(\begin{pmatrix} X_t^1 \\ X_t^2 \end{pmatrix}; \begin{pmatrix} A^1 & 0 \\ 0 & A^2 \end{pmatrix} \begin{pmatrix} X_{t-1}^1 \\ X_{t-1}^2 \end{pmatrix}, \begin{pmatrix} Q_1^{-1} & 0 \\ 0 & Q_2^{-1} \end{pmatrix} \right)$$

Contrast this with the discrete case shown in Figure 8.

For a more complex example, consider a vector auto-regressive process of order 2:

$$X_t = A_1 X_{t-1} + A_2 X_{t-2} + \epsilon_t$$

where $\epsilon_t \sim \mathcal{N}(0, \Sigma)$. (We ignore the U_t and Y_t variables for simplicity.) Suppose the parameters are as follows.

$$A_1 = \begin{pmatrix} \frac{3}{5} & 0 & \frac{1}{5} & 0 & 0 \\ 0 & \frac{3}{5} & 0 & -\frac{1}{5} & 0 \\ \frac{2}{5} & \frac{1}{3} & \frac{3}{5} & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{2} & \frac{1}{5} \\ 0 & 0 & \frac{1}{5} & 0 & \frac{2}{5} \end{pmatrix}, \ A_2 = \begin{pmatrix} 0 & 0 & -\frac{1}{5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{5} & 0 & \frac{1}{3} \\ 0 & 0 & 0 & 0 & -\frac{1}{5} \end{pmatrix}$$

and

$$\Sigma = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & 0 & 0 \\ \frac{1}{2} & 1 & -\frac{1}{3} & 0 & 0 \\ \frac{1}{3} & -\frac{1}{3} & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \ \Sigma^{-1} = \begin{pmatrix} 2.13 & -1.47 & -1.2 & 0 & 0 \\ -1.47 & 2.13 & 1.2 & 0 & 0 \\ -1.2 & 1.2 & 1.8 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Then the resulting dynamic chain graph is illustrated in Figure 19.

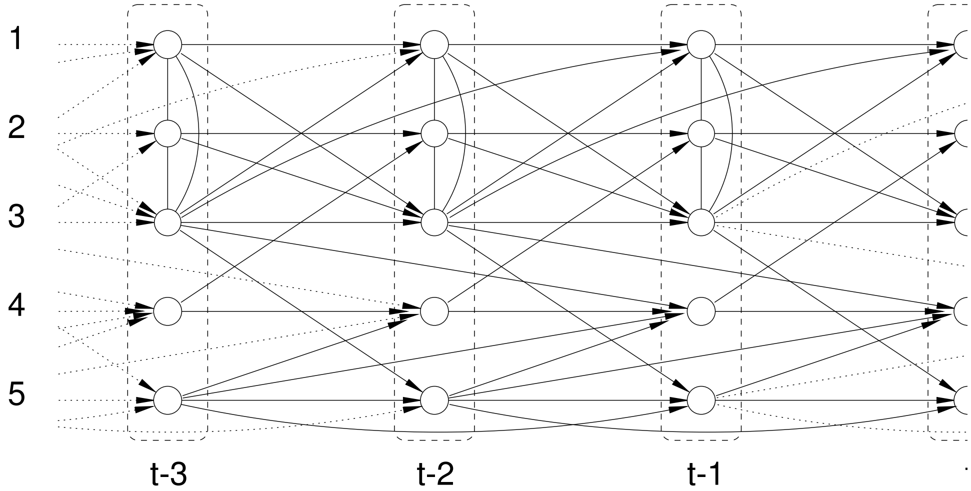


Figure 19: A VAR(2) process represented as a dynamic chain graph. From [DE00].

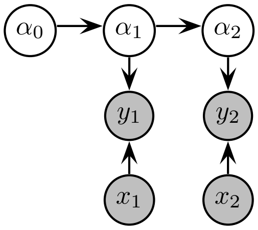


Figure 20: A DBN for the recursive least squares problem.

2.10 Joint state-parameter estimation

The problem of online parameter estimation can be represented by adding the parameters to the state space.

For example, suppose we want to recursively (i.e., sequentially) estimate the coefficients, α, in a linear regression model, where we assume the noise level, R, is known. This can be modelled as shown in Figure 20. The CPDs are as follows:

$$\begin{aligned} P(\alpha_0) &= \mathcal{N}(\alpha_0; 0, \infty I) \\ P(y_t|x_t, \alpha_t) &= \mathcal{N}(y_t; x_t' \alpha_t, R) \\ P(\alpha_t|\alpha_{t-1}) &= \mathcal{N}(\alpha_t; \alpha_{t-1}, 0) \end{aligned}$$

(We do not need to specify the CPD for x_t, since in linear regression, we assume the input is always known, so it suffices to use a conditional likelihood model.) The infinite variance for α_0 is an uninformative prior. The zero variance for α_t reflects the fact that the parameter is constant. If we made the variance non-zero, we could allow slow changes in the parameters, and hence track non-stationarities.

We can perform exact inference in this model using the Kalman filter, where α_t is the hidden state, and we use a time-varying output matrix $C_t = x_t'$; we set the transition matrix to $A = I$, the transition noise to $K = 0$, and the observation noise to R. This is just the recursive least squares algorithm discussed in Section 2.10.

Now consider a non-linear regression model:

$$y_t = \sum_{j=1}^{k} a_{j,t} \phi(||x_t - \mu_{j,t}||) + b_t + \beta_t' x_t + \epsilon_t$$

where $\epsilon_t \sim \mathcal{N}(0, \sigma_t)$ is a noise term and $\phi(||x_t - \mu_{j,t}||)$ is e.g., a radial basis function (RBF); if $k = 0$, this reduces to a linear regression model. We can write this in vector-matrix form as follows:

$$y_t = D(\mu_t, x_t)\alpha_t + n_t$$

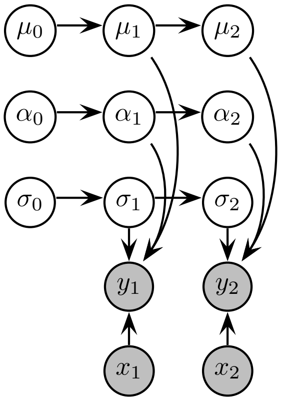


Figure 21: A DBN for sequential Bayesian non-linear regression.

where $\alpha_t = (a_t, b_t, \beta_t)$ are all the weights. The resulting DBN is shown in Figure 21. If we allow the parameters to change over time (modelled by a random walk), the CPDs are

$$\begin{aligned} P(\mu_t|\mu_{t-1}) &= \mathcal{N}(\mu_t; \mu_{t-1}, \delta_\mu I) \\ P(\alpha_t|\alpha_{t-1}) &= \mathcal{N}(\alpha_t; \alpha_{t-1}, \delta_\alpha I) \\ P(\log\sigma_t|\log\sigma_{t-1}) &= \mathcal{N}(\log\sigma_t; \log\sigma_{t-1}, \delta_\sigma I) \\ P(y_t|x_t, \alpha_t, \mu_t) &= \mathcal{N}(y_t; D(\mu_t, x_t)\alpha_t, \sigma_t) \end{aligned}$$

(I have omitted the priors at time 1 for simplicity.)

Although this model is linear in the α parameters, it is not linear in σ_t or μ_t. Hence it is not possible to perform exact inference (i.e., Bayesian parameter learning) in this model. However, it is possible to sample σ_t and μ_t, and then apply a conditional Kalman filter to α_t: see Section 5.4.3. Other approaches to inference in non-linear models will be discussed in Section 5. Methods for selecting the number of basis functions online will be discussed in Section 5.5.1.

2.11 Switching SSMs

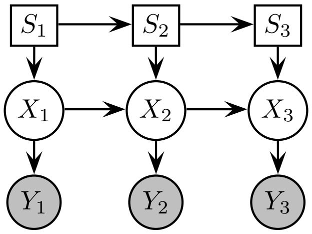


Figure 22: A switching state-space model. In this figure, square nodes are discrete, round nodes are continuous. We have omitted the input U for simplicity.

In Figure 22 we show a switching SSM, also called a switching linear dynamical system (LDS), a jump-Markov model, a jump-linear system, a conditional dynamic linear model (DLM), etc. The basic idea is that the model can switch between different kinds of dynamical “modes” or “regimes”. (The resulting piece-wise linearity is one way to approximate non-linear dynamics.) The dynamics of the modes themselves are governed by a discrete-state Markov chain. Hence the CPDs are as follows:

$$
\begin{aligned}
P(X_t = x_t | X_{t-1} = x_{t-1}, S_t = i) &= \mathcal{N}(x_t; A_i x_{t-1}, Q_i) \\
P(Y_t = y | X_t = x) &= \mathcal{N}(y; Cx, R) \\
P(S_t = j | S_{t-1} = i) &= M(i, j)
\end{aligned}
$$

Since this model has both discrete and continuous hidden variables, it is sometimes called a hybrid DBN. Exact inference in such model is very difficult, as we discuss in Section 5.2.4. Contrast this with an auto-regressive HMM (Section 2.2) in which all the hidden variables are discrete, making exact inference easy.

In addition to switching X_t, it is possible to switch Y_t. This can be used to represent $P(Y_t|X_t)$ as a mixture, which can approximate non-Gaussian observation densities. For example, in order to be robust to outliers, it is common to assume $P(Y_t|X_t)$ is a mixture of a Gaussian distribution and a uniform distribution (which can be approximated using a Gaussian with very large covariance). An application of switching SSM to fault diagnosis is discussed in Exercise **??**. We will see some other applications of this below.

2.11.1 Data association

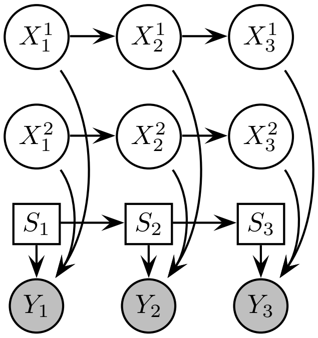


Figure 23: A DBN for the data association problem. The observation Y_t comes from one of two independent sources, as determined by the multiplexer node S_t. Square nodes are discrete, round nodes are continuous.

Another important application of switching SSMs is for modelling the data association (correspondence) problem. In this problem, we are interested in tracking the state of several "objects" $X_t^{(1)}, \ldots, X_t^{(D}$. At each time step, we get an observation, Y_t, but we do not which (if any) of the objects caused it.

The data association problem is extremely common. For example, consider visually tracking a person as they move around a cluttered room. At each frame, we must decide which body part (if any) caused which pixel. Another example concerns tracking missiles using radar. At each scan, we must decide which missile (if any) caused which "blip" on the screen.

A simple model for this is shown in Figure 23. S_t is a latent variable which specifies the identity of the source of the observation Y_t. It is a switching parent, and is used to "select" which of the hidden chains to "pass through" to the output, i.e., the CPD for $P(Y_t|X_t^{1:D}, S_t)$ is a (Gaussian) multiplexer:

$$P(Y_t = y | S_t = i, X_t^{(1)}, \ldots, X_t^{(D)}) = \mathcal{N}(y; W_i X_t^{(i)} + \mu_i, R_i)$$

Because of its military importance, a lot of effort has been expended on the problem of tracking multiple targets in clutter. See Section 5.5 for further discussion.

3 Inference

Having seen a large variety of different DBNs, we now discuss how to perform inference in such models. There are a variety of inference problems we might be interested in (see Figure 24 for a summary):

Filtering Computing $P(X_t|y_{1:t})$, i.e., monitoring (tracking) the state over time.

Prediction Computing $P(X_{t+h}|y_{1:t})$ for some horizon $h > 0$ into the future.

Fixed-lag smoothing Computing $P(X_{t-l}|y_{1:t})$, i.e., estimating what happened $l > 0$ steps in the past given all the evidence up to the present (hindsight).

Fixed-interval smoothing (offline) Computing $P(X_t|y_{1:T})$. This is used as a subroutine for offline training.

Viterbi decoding Computing $\arg\max_{x_{1:t}} P(x_{1:t}|y_{1:t})$, i.e., figuring out the most likely cause/ explanation of the observed data.

Classification Computing $P(y_{1:t}) = \sum_{x_{1:t}} P(x_{1:t}, y_{1:t})$. This can be used to compute the likelihood of a sequence under different models.

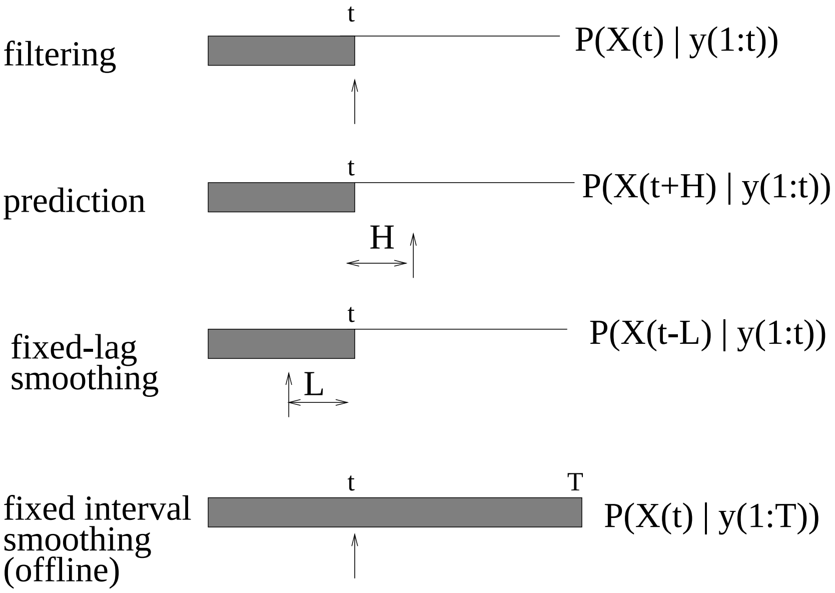


Figure 24: The main kinds of inference for DBNs. The shaded region is the interval for which we have data. The arrow represents the time step at which we want to perform inference. t is the current time, and T is the sequence length. h is a prediction horizon and l is a time lag.

We will focus on online filtering and offline smoothing; the other problems can be solved by similar methods.[3]

It will be helpful to distinguish models in which all the hidden nodes are discrete from models in which some (or all) hidden nodes are continuous, since they require different solution techniques (just as we saw that inference in an HMM required different techniques than inference in an SSM).

4 Exact inference in discrete-state models

In this section, we discuss exact filtering and smoothing algorithms for DBNs in which all the hidden variables are discrete. (Continuous nodes that are observed do not cause a problem, no matter what their distribution, since they only affect the inference by means of the conditional likelihood, c.f., the $O_t(i,i) = P(y_t|X_t = i)$ term for HMMs.)

4.1 Forwards-backwards algorithm

The simplest inference method for a discrete-state DBN is to convert it to an HMM, and then to apply the forwards-backwards algorithm. If there are N_h hidden variables per slice, and each such variable can have up to M values,

---

[3] For example, the quantity needed to classify a sequence, $P(y_{1:t})$, is computed as a by-product of filtering. Also, any filtering algorithm can be converted to a Viterbi algorithm by replacing the “sum” operator with the “max” operator. Finally, any offline smoothing algorithm can be made online by simply using a sliding window.

the resulting HMM will have $S = M^{N_h}$ states. Provided S is not too large, this is the method of choice, since the forwards-backwards algorithm is exact, and is very simple to implement. Unfortunately, in general the number of states will be too large, since it is exponential in the number of hidden variables; hence we must look for more efficient methods.

4.2 Unrolled junction tree

The second simplest inference method is to unroll the DBN for T slices (where T is the length of the sequence) and then to apply any static Bayes net inference algorithm. For filtering, it is natural to apply the variable elimination algorithm (Chapter **??**). This need only keep two slices in memory at a time: starting with slice 1, we add slice 2, then marginalize out slice 1, then add slice 3, etc. For smoothing, it is more efficient to use the junction tree algorithm, since most of the work in computing $P(Q_T|y_{1:T})$ can be reused when computing $P(Q_{T-1}|y_{1:T})$, etc. Later we will see how to modify the junction tree algorithm to only work with two slices at a time (which is essential for online filtering), but in this section, we consider the more naive approach of treating the network as a single large unrolled graph.

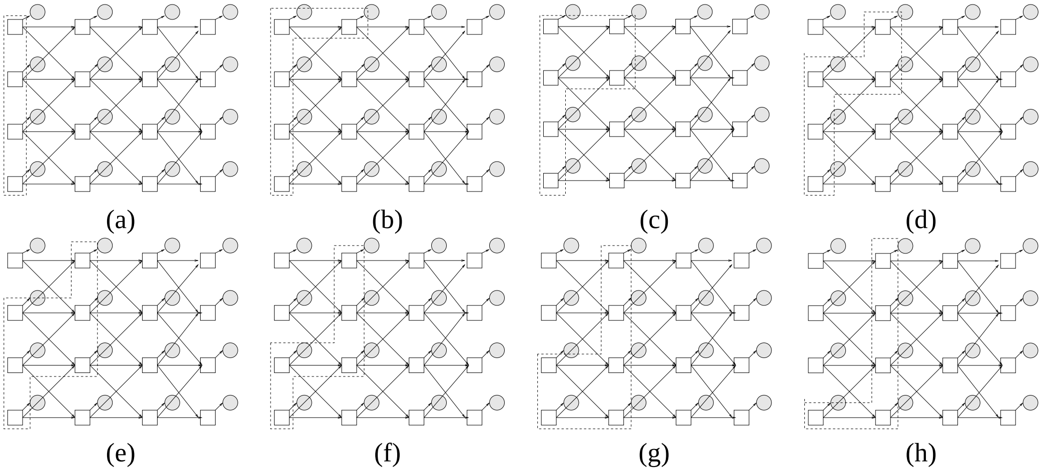


Figure 25: Some of the (non maximal) cliques in the junction tree constructed from a coupled HMM with 4 chains. The junction tree will contain these cliques connected together in a chain; the cliques containing $(Q_t^{(i)}, Y_t^{(i)})$ (not shown) "hang off" this "backbone" c.f., Figure 33. (a) The clique initially contains $Q_{t-1}^{(1:4)}$; then we update it as follows: (b) add $Q_t^{(1)}$; (c) add $Q_t^{(2)}$; (d) remove $Q_{t-1}^{(1)}$; (e) add $Q_t^{(3)}$; (f) remove $Q_{t-1}^{(2)}$; (g) add $Q_t^{(4)}$; (h) remove $Q_{t-1}^{(3)}$; (last step not shown) remove $Q_{t-1}^{(4)}$ to yield $Q_t^{(1:4)}$. It is possible to combine consecutive add/remove steps, resulting in just the maximal cliques shown in Figures c, e and g. This is what the standard junction tree algorithm does.

Unfortunately, when we create a junction tree from an unrolled DBN, the cliques tend to be very large, often making exact inference intractable. In particular, for every time slice (except possibly near the beginning and end of the sequence), there will usually be a clique that contains all the nodes within that slice that have inter-slice connections. (We will be more precise below). See Figure 25 for an example. The intuitive reason for this is illustrated in Figure 26: even if nodes within a slice are not directly correlated, they will eventually become correlated by virtue of sharing common ancestors in the past.

4.2.1 Constrained elimination orderings

The size of the maximum clique (or largest factor in the variable elimination algorithm) depends on the elimination ordering. Let us define $m(\pi)$ to be the size of the maximum clique induced by ordering π. (A related quantity is the width of an ordering, defined as $w(\pi) = m(\pi) - 1$.) The max clique size of a graph, $m(G)$, is defined as $m(\pi^*(G))$, where $\pi^*(G)$ is the optimal elimination ordering for G. Exact inference takes time and space exponential in $m(\pi)$, as we saw in Chapter **??**.

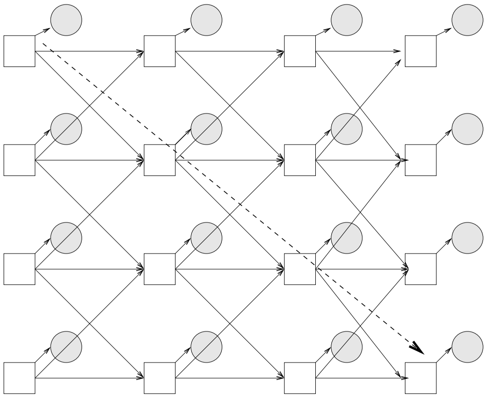


Figure 26: A coupled HMM with 4 chains. Even though chain 1 is not directly connected to chain 4, they become correlated once we unroll the DBN, as indicated by the dotted line. That inference in this model is intractable should not be surprising, since the graph is very similar to a 2D lattice MRF.

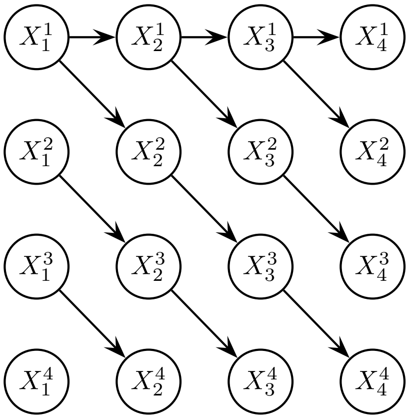


Figure 27: A DBN in which all the nodes in the last slice become connected when we eliminate nodes in a slice-by-slice order π, implying $m(\pi) = 4 + 1 = 5$, even though, using an unconstrained order that exploits the tree structure, $m(\pi^*) = 2$.

A natural ordering is to work left-to-right, i.e., we eliminate all nodes in slice t (in some optimal order) before any in slice $t + 1$. Unfortunately, such a constrained ordering is not always optimal. For example, Figure 27 shows a DBN where $m(\pi^*) = 2$ (since the graph is a set of trees). However, if we unroll the graph for $T \geq 4$ slices, and eliminate all the nodes in slice t before eliminating any nodes in slice $t + 1$, we find $m(\pi) = 5$. This is a consequence of the following theorem:

Theorem (Constrained elimination ordering) [RTL76]. Let $A_1, \ldots, A_n$ be an elimination sequence triangulating the (moral) graph G, and let A_i, A_k be two non-neighbors in G, $i < k$. Then the elimination sequence introduces the fill-in $A_i - A_k$ iff there is a path $A_i - A_j - \ldots - A_k$ such that all intermediate nodes A_j are eliminated before A_i.

To apply this theorem to the example in Figure 27, let $A_i = X_t^{(i)}$ and $A_k = X_t^{(k)}$. Suppose we first eliminate all the $X_\tau^{(i)}$ for all $\tau < t$; these become the A_j's. Since A_i is connected to A_k via some (undirected) path through the past A_j's (if $t \geq 4$), we will add a fill-in between them. Hence all the nodes in slice t will become connected.

To precisely characterize the consequences of using a slice-by-slice elimination ordering, let us first make the following definition.

Definition. Let the set of *temporal arcs* between slices $t - 1$ and t be denoted by $E^{\text{tmp}}(t) = \{(u, v) \in E | u \in V_{t-1}, v \in V_t\}$, where V_t are the nodes in slice t. The *incoming interface* $I_t^{\leftarrow}$ is defined as all nodes v s.t. v, or one of

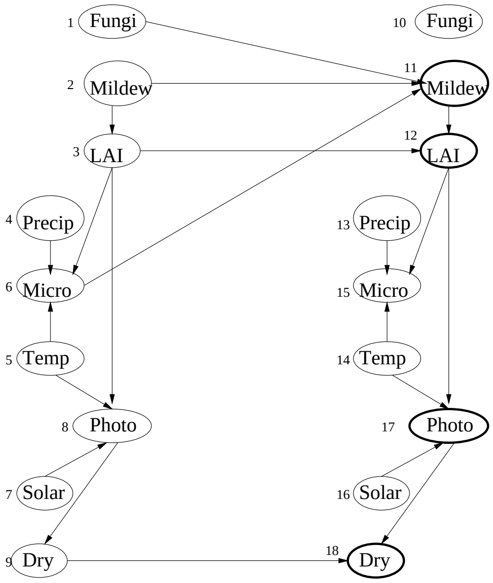


Figure 28: The Mildew DBN, designed for foreacasting the gross yield of wheat based on climatic data, observations of leaf area index (LAI) and extension of mildew, and knowledge of amount of fungicides used and time of usage [Kja95]. The nodes in the incoming interface are shown in bold outline. Note how the transient node photo is in the incoming interface, because it is the parent of the persistent node dry, which has an incoming temporal arc.

its children, has a parent in slice $t-1$, i.e., $I_t^{\leftarrow} = \{v \in V_t | (u,v) \in E^{\text{tmp}}(t)$ or $\exists w \in \text{ch}(v) : (u,w) \in E^{\text{tmp}}(t), u \in V_{t-1}\}$. See Figure 28 for an example. (We call it the *incoming* interface to distinguish it from the *outgoing* interface, which we shall define shortly.)

Now we can put tight lower and upper bounds on the complexity of inference using a slice-by-slice elimination ordering.

Theorem. Let G be a DBN with N nodes per slice, unrolled for $T > 1$ slices, which forms a single connected component.[4] If π is any slice-by-slice elimination ordering for G, then $|I^{\leftarrow}|+1 \leq m(\pi) \leq |I^{\leftarrow}|+N$, and the bounds are tight.
Proof. When we eliminate the nodes in slice t, the only variables that are involved are $V_t \cup I_{t+1}^{\leftarrow}$. Hence $m(\pi) \leq |V_t \cup I_t^{\leftarrow}| = N + |I^{\leftarrow}|$. Figure 29 shows a DBN where $I^{\leftarrow} = N$ and $m(\pi^*) = 2N$; since $m(\pi) \leq m(\pi^*)$, we see that the upper bound is tight. To establish the lower bound, note that since the graph is a single connected component, at least one of the nodes in the incoming interface must have a parent in the previous slice; hence there must be some stage in the elimination process which contains this "old" node plus all the interface nodes, so $|I^{\leftarrow}|+1 \leq m(\pi)$. To see that the bound is tight, consider a Markov chain: this has $|I^{\leftarrow}| = 1$, so a max clique size of 2 can be achieved. ■

Hence the constrained junction tree algorithm takes $\Omega(M^{|I^{\leftarrow}|+1})$ operations per time step (assuming all nodes in the incoming interface can have M possible values). Note that this may be exponentially faster than the time required by the forwards-backwards algorithm, which performs $O(M^{2N_h})$ operations per time step; nevertheless, it is still exponential in $|I^{\leftarrow}|$.

4.2.2 Unconstrained elimination orderings

What happens if we use an unconstrained elimination ordering? In principle we can do better. However, in practice, this is rarely the case. Recall that finding the optimal elimination ordering is NP-hard. Hence most elimination orderings are computed using local search algorithms. Empirically it has been found by several researchers that

---

[4] Figure 30 shows a counter-example to the theorem when the DBN is not a single connected component.

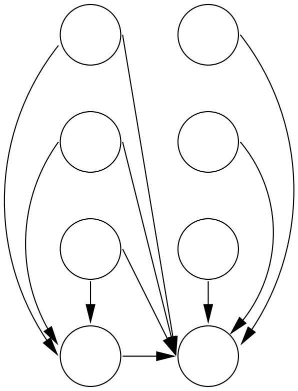


Figure 29: A worst case DBN from a complexity point of view, since we must create a clique which contains all nodes in both slices. Hence the max clique size is $2N$.

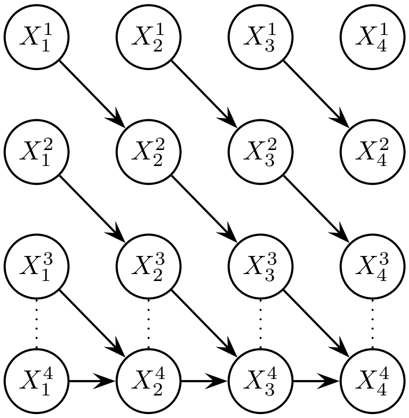


Figure 30: A DBN in which the nodes in the last slice do *not* become connected even when we use a slice-by-slice elimination ordering; hence $m(\pi^*) = 3 < |I^{\leftarrow}| + 1 = 4$, contrary to the theorem. This is because the nodes in the last slice do not belong to one connected component. Contrast this with Figure 27. Dotted lines represent moralization arcs.

providing a constrained space to search in often improves the quality of the local optima that are found. For example, an unconstrained algorithm might choose to eliminate the first node in every slice, then the second, etc., creating wide "horizontal" cliques that span many time slices, as opposed to narrow "vertical" cliques that are temporally localized. Such horizontal orderings usually have larger clique sizes than vertical orderings (for large enough T), and hence would not be chosen by a good search algorithm. However, a greedy search algorithm (e.g., based on the min-fill heuristic) may easily choose such a sub-optimal ordering.

An additional problem with unconstrained orderings is the following. We don't want to have to compute a new elimination ordering every time we change the length of the sequence, because it is too slow. Instead we would like to triangulate a fixed size DBN, identify the cliques in the resulting junction tree, and then reuse them for all sequence lengths by copying the repeating structure. In order to guarantee that such repeating structure occurs in the junction tree, we must use a temporally constrained elimination ordering. Even then, the book-keeping involved with dynamically changing the structure of the junction tree becomes quite tricky.

4.3 The frontier algorithm

An alternative approach, which turns out to be simpler and better suited to online inference, is to think explicitly in terms of time slices. The forwards-backwards algorithm for HMMs works because conditioning on Q_t d-separates the past from the future. This idea can be generalized to DBNs by noting that all the nodes in a slice d-separate the past

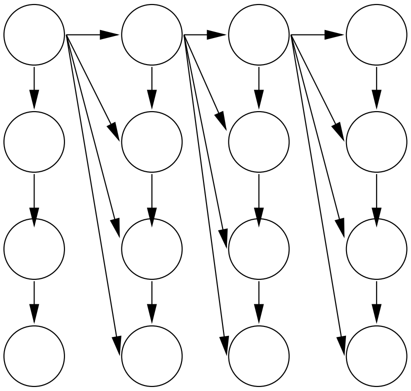


Figure 31: This graph has a max clique size of 3, an outgoing interface of size 1, and an incoming interface of size 4. Based on Figure 2 of [Dar01].

from the future. We will call this set the "frontier" and denote it by Z_t. Hence the belief state, $P(Z_t|y_{1:t})$, can act as a sufficient statistic for filtering; for smoothing, we can define an analogous quantity for the backwards direction.

The filtering problem reduces to computing $P(Z_t|y_{1:t})$ from $P(Z_{t-1}|y_{1:t-1})$. In an HMM, this can be done with a single matrix multiply, but we wish to avoid constructing, yet alone multiplying by, an $O(S \times S)$ matrix, where $S = M^{N_h}$. The following rules will advance the frontier one node at a time such that at each step the frontier d-separates all the nodes to its left from the nodes to its right: (1) a node can be added to the frontier if all its parents are already in the frontier, and (2) a node can be removed from the frontier as soon as all its children have been added. See Figure 25 for an example. Adding a node means multiplying its CPD onto the frontier distribution, and removing a node means marginalizing it out of the frontier distribution. We leave the details as an exercise, since we are about to develop a more efficient algorithm based on similar ideas.

4.4 The interface algorithm

The frontier distribution, $P(Z_t|y_{1:t})$, has size $O(M^{N_h})$. Fortunately, it is easy to see that can exclude from the frontier all nodes that do not have any children in the next slice. We will prove this below. First let us make the following definition:

Definition. The *outgoing interface* $I_t^{\rightarrow}$ is the set of nodes which have children in the next slice, i.e., $I_t^{\rightarrow} \overset{\text{def}}{=} \{u \in V_t | (u,v) \in E^{\text{tmp}}(t+1), v \in V_{t+1}\}$.

Now we prove that the outgoing interface d-separates the past from the future, and hence can be used as a sufficient statistic for inference.

Theorem. $(V_{1:t-1} \cup (V_t \setminus I_t^{\rightarrow})) \perp V_{t+1:T} | I_t^{\rightarrow}$, i.e., the outgoing interface d-separates the past from the future, where the past is all the nodes in slices prior to t, plus the non-interface nodes in slice t, and the future is all nodes in slices after t.

Proof. Let I be a node in the outgoing interface, connected to a node P in the past and a node F in the future (which must be a child of I, by definition). If P is a parent, the graph looks like this: $P \rightarrow I \rightarrow F$. If P is a child, the graph looks like this: $P \leftarrow I \rightarrow F$. Either way, we have $P \perp F | I$, since I is never at the bottom of a v-structure. Since all paths between any node in the past and any node in the future are blocked by some node in the outgoing interface, the result follows. ■

Figure 31 shows a DBN for which $|I^{\rightarrow}| = 1 < |I^{\leftarrow}| = 4$, but Figure 28 shows a DBN for which $|I^{\rightarrow}| = 5 > |I^{\leftarrow}| = 4$. Although the outgoing interface is not necessarily smaller than the incoming interface, in practice it often is. (If all temporal arcs are persistence arcs (arcs of the form $Q_{t-1}^{(i)} \rightarrow Q_t^{(i)}$), then it is easy to see that the outgoing interface is never larger than the incoming interface.) In addition, the outgoing interface lends itself to

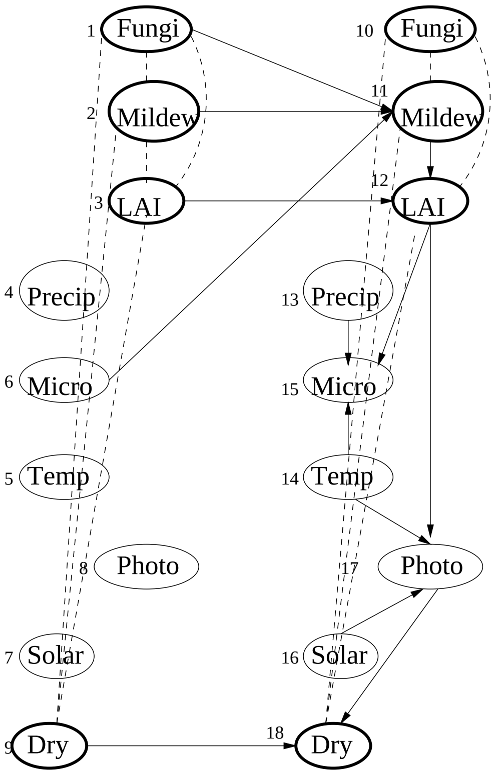


Figure 32: The Mildew 1.5DBN. Nodes in the outgoing interface are shown in bold outline. Nodes which are not in the outgoing interface of slice 1 are disconnected, and can be removed. Dashed arcs are added to ensure the outgoing interfaces are fully connected.

simpler algorithms, which are better suited to online inference, as we will see below.

The filtering problem now reduces to computing $P(I_t^{\to}|y_{1:t})$ from $P(I_{t-1}^{\to}|y_{1:t-1})$. To do this, we will use a modified junction tree on a modified 2TBN; this will allow us to exploit any sparse structure that may exist within a slice (something the frontier algorithm does not do). We will call the modified 2TBN a 1.5DBN, since it contains "one-and-a-half" slices. Specifically, the 1.5DBN contains all the nodes in slice 2, but only the outgoing interface nodes from slice 1. See Figure 32 for an example.

Let us denote the 1.5DBN by $G_{1.5}$. To convert this to a modified junction tree, we proceed as follows. First we moralize $G_{1.5}$ to create $M(G_{1.5})$. Then we add undirected edges between all nodes in $I_{t-1}^{\to}$ and between all nodes in $I_t^{\to}$ to create $M^+(G_{1.5})$; this ensures there will be cliques that are large enough to contain $P(I_{t-1}^{\to}|y_{1:t-1})$ and $P(I_t^{\to}|y_{1:t})$. Then we triangulate using an unconstrained elimination ordering to create $T(M^+(G_{1.5}))$. Finally we create a junction tree from the maximal cliques of $T(M^+(G_{1.5}))$. If this contains all the nodes in slice t and the interface nodes in slice $t-1$, we will call it J_t. (J_1 just contains slice 1, of course.) Let in-clq denote the clique in J_t which contains $I_{t-1}^{\to}$, and out-clq denote the clique which contains $I_t^{\to}$. See Figure 33 for an example.

Having created a junction tree, we can define the smoothing algorithm as in Figure 34. The forwards operator, $(\alpha_t, J_t) = \text{fwd}(\alpha_{t-1}, y_t)$, does one step of Bayesian updating, i.e., it computes $\alpha_t \overset{\text{def}}{=} P(I_t^{\to}|y_{1:t})$ from $\alpha_{t-1} = P(I_{t-1}^{\to}|y_{1:t-1})$. It also returns the potentials ϕ of all the cliques and separators in J_t (these will needed in the backwards smoothing pass). The $\downarrow$ operator represents (sum or max) marginalization; thus $\phi_{out} \downarrow I_t^{\to}$ means marginalize the clique potential on out-clq onto the forwards interface.

Similarly, the backwards operator, $(\gamma_{t-1}, J_t) = \text{back}(\gamma_t, \alpha_t, J_t)$, computes the analog of the equation (12.45) for HMMs:

$$\xi(q_{t-1}, q_t) = \frac{\alpha(q_{t-1})P(y_t|q_t)A(q_{t-1}, q_t)\gamma(q_t)}{\alpha(q_t)}$$

since $\alpha(q_{t-1})P(y_t|q_t)A(q_{t-1}, q_t)$ is included in J_t, and the back operator multiplies by $\gamma(q_t)/\alpha(q_t)$. (Exercise 7.5

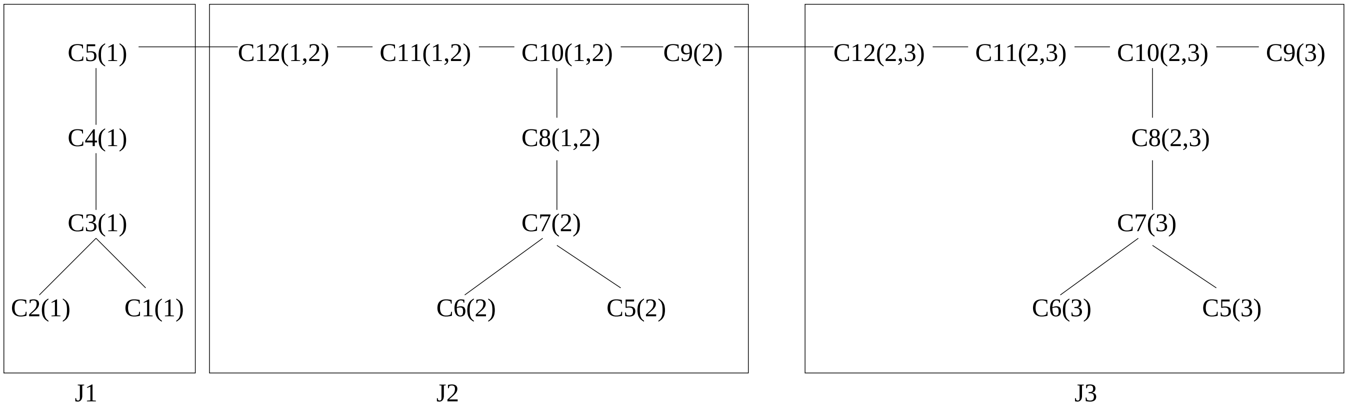


Figure 33: The junction tree constructed from 3 slices of the DBN in Figure 32. The notation $Cn(t, t+1)$ means clique n containing nodes from slices t and $t+1$. Thus $C12(2,3)$ is the same as $C12(1,2)$, except all the node numbers are shifted up by N, the number of nodes per slice. (Cliques in J_t, $t > 1$, start with number 5 because cliques 1 to 4 correspond to disconnected nodes in slice 1 of the 1.5DBN, and can be ignored.) For J_t, $t > 1$, the incoming clique is C12 and the outgoing clique is C9; for J_1, the incoming/outgoing clique is C5. α_t represents the potential on the separator between J_t and J_{t+1} on the forwards pass; γ_t represents this potential on the backwards pass.

```
function $(\alpha_t, J_t) = \text{Fwd}(\alpha_{t-1}, y_t)$
Initialize $\phi(J_t)$ using the CPDs from slice 2 and the evidence $y_t$
$\phi_{in} = \phi_{in} * \alpha_{t-1}$
Collect to out-clq
$\alpha_t = \phi_{out} \downarrow I_t^{\rightarrow}$

function $(\alpha_t, J_t) = \text{Fwd1}(y_1)$
Initialize $\phi(J_t)$ using the CPDs from slice 1 and the evidence $y_1$
Collect to out-clq
$\alpha_t = \phi_{out} \downarrow I_t^{\rightarrow}$

function $(\gamma_{t-1}, J_t) = \text{Back}(\gamma_t, \alpha_t, J_t)$
$\phi_{out} = \phi_{out} * \frac{\gamma_t}{\alpha_t}$
Distribute from out-clq
$\gamma_{t-1} = \phi_{in} \downarrow I_{t-1}^{\rightarrow}$

function $(J_1) = \text{Back1}(\gamma_1, \alpha_1, J_1)$
$\phi_{out} = \phi_{out} * \frac{\gamma_1}{\alpha_1}$
Distribute from out-clq

function $(J_1, \ldots, J_t) = \text{smooth}(y_{1:T})$
$(\alpha_1, J_1) = \text{fwd1}(y_1)$
For $t = 2, \ldots, T$,
    $(\alpha_t, J_t) = \text{fwd}(\alpha_{t-1}, y_t)$
$\gamma_T = \alpha_t$
For $t = T, \ldots, 2$
    $(\gamma_{t-1}, J_t) = \text{back}(\gamma_t, \alpha_t, J_t)$
$J_1 = \text{back1}(\gamma_1, \alpha_1, J_1)$
```

Figure 34: Pseudo-code for the generalized $\alpha - \gamma$ algorithm.

asks you to develop an $\alpha - \beta$ version of this algorithm instead.)

We can compute any single node marginal $P(Q_t^{(i)}|y_{1:T})$ by marginalizing γ_t. Similarly, we can compute any family marginal $P(Q_t^{(i)}, \text{Pa}(Q_t^{(i)})|y_{1:T})$ by marginalizing the appropriate potential in J_t.

4.5 Conditionally tractable substructure

The $O(M^{|I^{\rightarrow}|})$ complexity of filtering is a purely graph-theoretic (worst-case) result. It sometimes happens that the CPDs encode conditional independencies that are not evident in the graph structure. This can lead to significant speedups. For example, consider Figure 35. This is a schematic for two "processes" (here represented by single nodes), B and C, both of which only interact with each other via an intermediate layer, R.

The outgoing interface is $\{R, B, C\}$; since B and C represent whole subprocesses, this might be quite large. Now suppose we remove the dotted arcs, so R becomes a "root", with only outgoing arcs to itself and B and C. Again, the interface is $\{R, B, C\}$. Finally, suppose that R is in fact a static node, i.e., $P(R_t|R_{t-1}) = \delta(R_t, R_{t-1})$. (For example, R might be a fixed parameter.) In this case, the model can be simplified as shown in Figure 36. This model enjoys the property that, conditioned on R, the interface factorizes:

$$P(R, B_t, C_t|y_{1:t}) = P(B_t|R, y_{1:t})P(C_t|R, y_{1:t})P(R|y_{1:t}) = P(B_t|R, y_{1:t}^B)P(C_t|R, y_{1:t}^C)P(R|y_{1:t})$$

where $y_t = (y_t^B, y_t^C)$. This follows since $B_t \perp y_{1:t}^C | R$ and $C_t \perp y_{1:t}^B | R$, i.e., evidence which is local to a process does not influence other processes: the R node acts like a barrier. Hence we can recursively update each subprocess separately:

$$\begin{aligned} P(B_t|R, y_{1:t}^B) &= P(B_t|R, y_{1:t-1}^B, y_t^B) \\ &\propto P(y_t^B|B_t) \sum_b P(B_t|B_{t-1} = b, R) P(B_{t-1} = b|R, y_{1:t-1}^B) \end{aligned}$$

The distribution over the root variable can be computed at any time using

$$P(R|y_{1:t}) \propto \sum_b P(R)P(B_t = b|R, y_{1:t})$$

The reason we cannot apply the same factoring trick to the model in Figure 35 is that $P(B_t|R_t, y_{1:t}) \neq P(B_t|R_t, y_{1:t}^B)$, since there is a path (e.g., via R_{t-1}) connecting Y_t^C to B_t. If we could condition on the whole chain $R_{1:t}$ instead of just on R_t, we could factor the problem. Of course, we cannot condition on all possible values of $R_{1:t}$, since there are M^t of them; however, we can *sample* representative instantiations. Given these samples, we can update the processes exactly. This is an example of Rao-Blackwellised particle filtering: see Section 5.4.3.

5 Approximate filtering

So far, we have discussed how to perform exact filtering in discrete-state models. Although this is always possible, it might be too slow, because the belief state, $P(X_t|y_{1:t})$ (or, more accurately, $P(I_t^{\rightarrow}|y_{1:t})$) is too large. Hence we must resort to approximations.

What about models with continuous, or mixed discrete-continuous, hidden state? It turns out that for nearly all such models, exact (i.e., closed-form, fixed-sized) representations of the belief state do not exist. The exception is models in which the transition and observation models are conjugate. Linear-Gaussian (state-space) models have this property: the belief state, $P(X_t|y_{1:t})$, can always be exactly represented by a Gaussian.[5] However, in general, we must approximate the belief state. In the following sections, we review a number of different algorithms, which use different representations for the approximate belief state. Most of these methods are only suitable for restricted classes of DBNs. At present, discretization and sampling (Section 5.4) are the only methods that can, in theory, handle DBNs with arbitrary topology and arbitrary CPDs. Unfortunately, discretization is not feasible for large networks/nodes, and to make sampling work well in practice, it is usually necessary to make additional restrictions/ assumptions and to use model-specific heuristics. Hence the picture is somewhat less satisfying than in Section 4.

---

[5] To see this, note that since the previous belief state, $P(X_{t-1}|y_{1:t-1})$, is Gaussian (by assumption), then so is $P(X_t|y_{1:t-1})$, since X_t is a linear transformation of X_{t-1} plus additive Gaussian noise. Combinining a Gaussian prior, $P(X_t|y_{1:t-1})$, with a Gaussian likelihood, $P(y_t|X_t)$, yields a Gaussian posterior, $P(X_t|y_{1:t})$. In other words, linear-Gaussian models are closed under Bayesian updating.

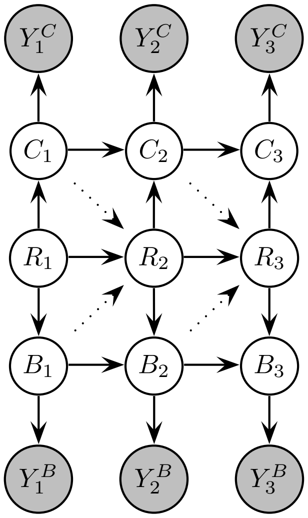


Figure 35: Two “processes”, here represented by single nodes, B and C, only interact with each other via an “interface” layer, R.

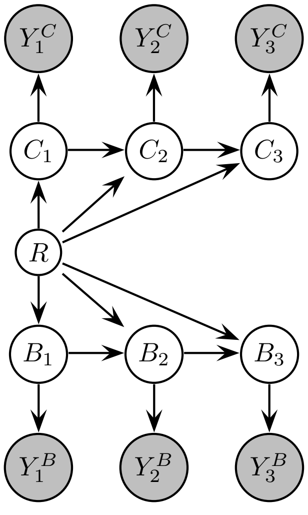


Figure 36: Conditioned on the static root, the interface is fully factored.

5.1 Belief state = discrete distribution

In this section, we consider problems where the belief state, $P(X_t|y_{1:t})$, is a discrete probability distribution, as in a histogram. This might arise if we discretized all the continuous hidden variables, or if all the variables were all discrete to begin with. Although in principle we can use the exact methods discussed in Section 4, in practice they might be too slow. Hence we develop faster, but approximate, algorithms.

5.1.1 Assumed density filtering (ADF)/ BK algorithm

Basic idea Assumed density filtering (ADF) assumes the belief state is a member of some restricted family $\mathcal{F}$. Figure 37 illustrates the basic idea. We start with a prior, $\tilde{\alpha}_{t-1} \in \mathcal{F}$, and perform one step of exact Bayesian updating to get $\widehat{\alpha}_t$. Usually $\widehat{\alpha}_t \not\in \mathcal{F}$, so we approximate it by choosing the "closest" approximation in the family: $\tilde{\alpha}_t = \arg\min_{q \in \mathcal{F}} D(\hat{\alpha}_t||q)$. If $\mathcal{F}$ is in the exponential family, this equation can be solved by moment matching.

For a DBN, it is natural to let $\mathcal{F}$ be the set of factorized distributions, i.e., we approximate the joint distribution α_t by a product of marginals over clusters of variables:

$$\alpha_t \approx \tilde{\alpha}_t = \prod_{i=1}^{C} P(Q_t^{(c_i)}|y_{1:t})$$

where $\cup_i c_i = I^{\rightarrow}$ is a partition of the interface into clusters (which may overlap). This is often called the "Boyen-Koller" (BK) approximation. The best approximate distribution in this family can be found by moment matching, which in the discrete case means computing the marginals on each cluster. We discuss how to do this efficiently below.

$$\begin{tikzcd}
 & \widehat{\alpha}_t \arrow[d, "P"'] & \widehat{\alpha}_{t+1} \arrow[d, "P"'] & \text{exact} \\
\widetilde{\alpha}_{t-1} \arrow[ur, "U"] & \widetilde{\alpha}_t \arrow[ur, "U"] & \widetilde{\alpha}_{t+1} & \text{approx}
\end{tikzcd}$$

Figure 37: The ADF algorithm as a series of update (U) and projection (P) steps.

Error analysis It can be shown that the error remains bounded over time, in the following sense:

$$E\ D(\alpha_t||\tilde{\alpha}_t) \leq \frac{\epsilon_t}{\gamma^*}$$

where the expectation is take over the possible observation sequences, γ^* is the mixing rate of the DBN, and ϵ_t is the additional error incurred by projection, over and above any error inherited from the factored prior:

$$\epsilon_t = D(\alpha_t||\tilde{\alpha}_t) - D(\alpha_t||\hat{\alpha}_t).$$

The intuition is that, even though projection introduces an error at every time step, the stochastic nature of the transitions, and the informative nature of the observations, reduces the error sufficiently to stop it building up.

The accuracy of the BK algorithm depends on the clusters that we use to approximate the belief state. Exact inference corresponds to using a single cluster, containing all the interface nodes. The most aggressive approximation corresponds to using one cluster per variable (a "fully factorized" approximation).

The best case for BK is when the observations are perfect (noiseless), or when the transition matrix is maximally stochastic (i.e., $P(X_t|X_{t-1})$ is independent of X_{t-1}), since in either case, errors in the prior are irrelevant. Conversely, the worst case for BK is when the observations are absent or uninformative, and the transition matrix is deterministic, since in this case, the error grows over time. (This is consistent with the theorem, since deterministic processes can have infinite mixing time.) In Section 5.4, we will see that the best case for BK is the worst case for particle filtering (when we propose from the transition prior), and vice versa.

Implementation We can implement the BK algorithm by slightly modifying the interface algorithm (Section 4.4). Specifically, we construct the junction tree for the 1.5DBN as before, but instead of adding undirected edges between all nodes in $I^{\rightarrow}_{t-1}$ and $I^{\rightarrow}_{t}$, we only add them between all nodes in each cluster. This usually results in much smaller cliques. We leave the details to exercise 7.8.

5.1.2 Beam search

A completely different kind of approximation is to assume that there are only a small number of likely hypotheses. From each such likely prior state, we try to find the next most probable set of states. We can either keep the k most likely states, or keep enough states to ensure we cover enough of the probability mass. This method is widely used in speech recogntion, where we only keep track of the most probably interpretations of the sentence, and also in fault diagnosis, where we only keep track of the most probable faults.

5.2 Belief state = Gaussian

In this section, we consider algorithms that approximate $P(X_t|y_{1:t})$ by a single Gaussian. (Many of these could be generalized to use any member of the exponential family.)

5.2.1 Kalman filter (KF)

Recall the form of the state-space model from Chapter **??**:

$$\begin{aligned} x_t &= Ax_{t-1} + v_t \\ y_t &= Cx_t + w_t \end{aligned}$$

where $v_t \sim \mathcal{N}(0, Q)$ is the process noise, $w_t \sim \mathcal{N}(0, R)$ is the observation noise, and the variables satisfy the conditional independence assumptions implicit in Figure 1. (In this and following sections we adopt the convention that capital letters denote matrices, and lower case letters denote random variables or values of random variables. To simplify notation, we omit the input variable u_t.) In this case, the belief state is Gaussian and can be represented exactly by its first two moments: $\hat{x}_{t|t} \overset{\text{def}}{=} E[x_t|y_{1:t}]$ and $P_{t|t} \overset{\text{def}}{=} E[(x_t - \hat{x}_t)(x_t - \hat{x}_t)^T|y_{1:t}]$. This belief state can be updated recursively using the Kalman filter, as we saw in Chapter **??**. For completeness, we restate the equations here. First we perform the time update (prediction step):

$$\begin{aligned} x_{t|t-1} &= Ax_{t-1|t-1} \\ P_{t|t-1} &= AP_{t-1|t-1}A^T + Q \\ y_{t|t-1} &= Cx_{t|t-1} \end{aligned}$$

Then we perform the measurement update (correction step):

$$\begin{aligned} \tilde{y}_t &= y_t - \hat{y}_{t|t-1} \text{ (error)} \\ P_{\tilde{y}_t} &= CP_{t|t-1}C^T + R \text{ (covariance of error)} \\ P_{x_t y_t} &= P_{t|t}C^T \text{ (cross covariance)} \\ K_t &= P_{x_t y_t} P_{\tilde{y}_t}^{-1} \text{ (Kalman gain matrix)} \\ x_{t|t} &= x_{t|t-1} + K_t(y_t - y_{t|t-1}) \\ P_{t|t} &= P_{t|t-1} - K_t P_{\tilde{y}_t} K_t^T \end{aligned}$$

We can generalize the Kalman filter equations to apply to any linear-Gaussian DBN by using the junction tree algorithm discussed in Section 4, and modifying the definition of summation, multiplication and division so that these operations can be applied to Gaussian potentials instead of discrete ones.

5.2.2 Extended Kalman filter (EKF)

The extended Kalman filter (EKF) can be applied to models of the form

$$\begin{aligned} x_t &= f(x_{t-1}) + v_t \\ y_t &= g(x_t) + w_t \end{aligned}$$

where f and g are arbitrary, differentiable functions. The basic idea is to linearize f and g about the previous state estimate using a second order Taylor expansion, and then to apply the standard Kalman filter equations. (The noise variance in the equations (Q and R) is not changed, i.e., the additional error due to linearization is not modeled.) Thus we approximate the stationary nonlinear dynamical system with a non-stationary linear dynamical system. That is, at time t, we approximate the model by

$$\begin{aligned} x_t &= f(\hat{x}_{t-1|t-1}) + A_{\hat{x}_{t|t-1}}(x_{t-1} - \hat{x}_{t-1|t-1}) + v_t \\ y_t &= g(\hat{x}_{t|t-1}) + C_{\hat{x}_{t|t-1}}(x_t - \hat{x}_{t|t-1}) + w_t \end{aligned}$$

where $\hat{x}_{t|t-1} = f(\hat{x}_{t-1|t-1})$ and

$$\begin{aligned} A_{\hat{x}} &\overset{\text{def}}{=} \left.\frac{\partial f}{\partial x}\right|_{\hat{x}} \\ C_{\hat{x}} &\overset{\text{def}}{=} \left.\frac{\partial g}{\partial x}\right|_{\hat{x}} \end{aligned}$$

We then use these state-dependent matrices in the standard KF equations.

It is possible to improve performance by repeatedly re-linearizing the equations around $\hat{x}_{t|t}$ instead of $\hat{x}_{t|t-1}$; this is called the iterated EKF, and yields better results, especially in the case of highly nonlinear measurement models.

5.2.3 Unscented Kalman filter (UKF)

The unscented Kalman filter (UKF) propagates a deterministically chosen set of points through the non-linear functions f and g, and fits a Gaussian to the resulting transformed points. (This is called an unscented transform, and is closely related to Gaussian quadrature methods for the numerical evaluation of integrals.) The resulting Gaussian approximation is accurate to at least second order, whereas the EKF is only a first order approximation. Furthermore, the UKF does not require the analytic evaluation of any derivatives, making it simpler and more widely applicable than the EKF. In general, both algorithms perform $O(d^3)$ operations per step (where $X_t \in \mathbb{R}^d$).

Before explaining the UKF, we first explain the unscented transform. Assume $P(x) = \mathcal{N}(x; \hat{x}, P_x)$, and consider estimating $P(y)$, where $y = f(x)$ for some nonlinear function f. The unscented transform does this as follows. First we create a matrix X of $2d+1$ sigma vectors x_i, given by

$$\begin{aligned} x_0 &= \hat{x} \\ x_i &= \hat{x} + (\sqrt{(d+\lambda)P_x})_i, \quad i = 1, \ldots, d \\ x_i &= \hat{x} - (\sqrt{(d+\lambda)P_x})_i, \quad i = d+1, \ldots, 2d \end{aligned}$$

where $\lambda = \alpha^2(d+\kappa) - d$ is a scaling parameter. (In general, the optimal values of α, β and κ are problem dependent, but when $d = 1$, they are $\alpha = 1$, $\beta = 0$, $\kappa = 2$.) $(\sqrt{(d+\lambda)P_x})_i$ is the i'th column of the matrix square root. (These points are just ± 1 standard deviation around the mean.) These sigma vectors are propagated through the nonlinear function to yield $y_i = f(x_i)$, and the mean and covariance for y are computed as follows:

$$\begin{aligned} \hat{y} &= \sum_{i=0}^{d} W_i^{(m)} y_i \\ P_y &= \sum_{i=0}^{d} W_i^{(c)} (y_i - \hat{y})(y_i - \hat{y})^T \end{aligned}$$

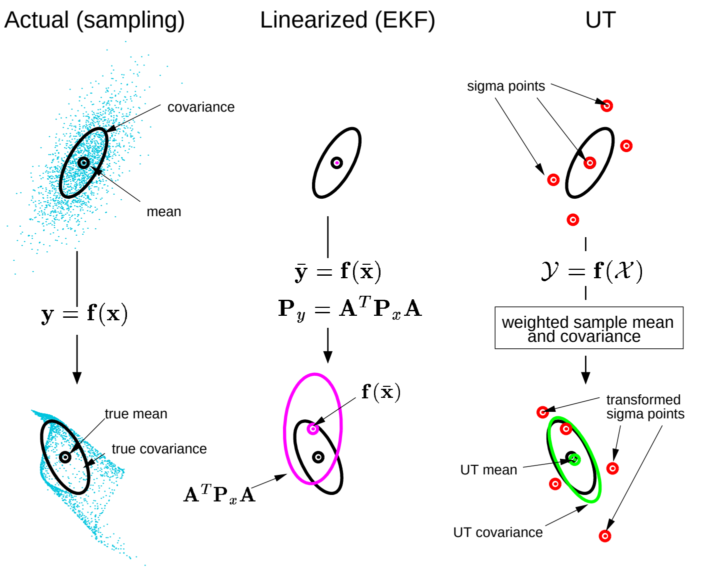

Actual (sampling)
Linearized (EKF)
UT
covariance
mean
sigma points
$\mathbf{y} = \mathbf{f}(\mathbf{x})$
$\bar{\mathbf{y}} = \mathbf{f}(\bar{\mathbf{x}})$
$\mathbf{P}_y = \mathbf{A}^T\mathbf{P}_x\mathbf{A}$
$\mathcal{Y} = \mathbf{f}(\mathcal{X})$
weighted sample mean and covariance
true mean
true covariance
$\mathbf{f}(\bar{\mathbf{x}})$
$\mathbf{A}^T\mathbf{P}_x\mathbf{A}$
transformed sigma points
UT mean
UT covariance

Figure 38: An example of the unscented transform in two dimensions. Thanks to Rudolph van der Merwe for this Figure.

where

$$\begin{aligned} W_0^{(m)} &= \lambda/(d+\lambda) \\ W_0^{(c)} &= \lambda/(d+\lambda) + (1-\alpha^2+\beta) \\ W_i^{(m)} &= W_i^{(c)} = 1/(2(d+\lambda)) \end{aligned}$$

See Figure 38 for an example.

The UKF algorithm is simply two applications of the unscented tranform, one to compute $P(X_t|y_{1:t-1})$ and the other to compute $P(X_t|y_{1:t})$. In the case of zero mean additive Gaussian noise, the algorithm is as follows. First we create the vector

$$X_{t-1} = \left[\hat{x}_{t-1|t-1},\ \ \hat{x}_{t-1|t-1} + \gamma\sqrt{P_{t-1|t-1}},\ \ \hat{x}_{t-1|t-1} - \gamma\sqrt{P_{t-1|t-1}}\ \right]$$

where $\gamma = \sqrt{d+\lambda}$. The time update becomes:

$$\begin{aligned} X_{t|t-1} &= f(X_{t-1}) \\ \hat{x}_{t|t-1} &= \sum_{i=0}^{2d} W_i^{(m)} X_{i,t|t-1} \\ \hat{P}_{t|t-1} &= \sum_{i=0}^{2d} W_i^{(c)} (X_{i,t|t-1} - \hat{x}_{t|t-1})(X_{i,t|t-1} - \hat{x}_{t|t-1})^T + Q \\ \hat{Y}_{t|t-1} &= g(X_{t|t-1}) \\ \hat{y}_{t|t-1} &= \sum_{i=0}^{2d} W_i^{(m)} Y_{i,t|t-1} \end{aligned}$$

The measurement update becomes

$$\begin{aligned}
\hat{P}_{\tilde{y}_t} &= \sum_{i=0}^{2d} W_i^{(c)} (Y_{i,t|t-1} - \hat{y}_{t|t-1})(Y_{i,t|t-1} - \hat{y}_{t|t-1})^T + R \\
\hat{P}_{x_t,y_y} &= \sum_{i=0}^{2d} W_i^{(c)} (X_{i,t|t-1} - \hat{x}_{t|t-1})(Y_{i,t|t-1} - \hat{y}_{t|t-1})^T \\
K_t &= P_{x_t,y_t} P_{\tilde{y}_t}^{-1} \\
\hat{x}_{t|t} &= \hat{x}_{t|t-1} + K_t (y_t - \hat{y}_{t|t-1}) \\
P_{t|t} &= P_{t|t-1} - K_t P_{\tilde{y}_t} K_t^T
\end{aligned}$$

We can see that the UKF is a simple modification to the standard KF, but which can handle nonlinearities with greater accuracy than the EKF, and without the need to compute derivatives.

5.2.4 Assumed density filter (ADF)/ moment matching filter

We already encountered ADF in Section 5.1.1, where we assumed the belief state was representable as a product of discrete marginals. In this section, we assume the belief state is represented by a Gaussian. The goal is to find the parameters of this Gaussian.

For example, consider a system with linear-Gaussian dynamics but non-linear, non-Gaussian observations (e.g., a mixture of Gaussians, which could not be handled by the EKF or UKF). We start with an approximate Gaussian prior:

$$P(x_{t-1}|y_{1:t-1}) \approx \tilde{\alpha}(x_{t-1|t-1}) = \mathcal{N}(x_{t-1}; \tilde{x}_{t-1|t-1}, \tilde{P}_{t-1|t-1}).$$

This is propagated through the linear-Gaussian dynamics to get a Gaussian one-step-ahead prediction density:

$$P(x_t|y_{1:t-1}) \approx \tilde{\alpha}(x_{t|t-1}) = \mathcal{N}(x_t; A\tilde{x}_{t-1|t-1}, A\tilde{P}_{t-1|t-1} + Q).$$

The exact update step becomes

$$\begin{aligned}
P(x_t|y_{1:t}) &= \frac{P(y_t|x_t)P(x_t|y_{1:t-1})}{\int_{x_t} P(y_t|x_t)P(x_t|y_{1:t-1})} \\
&\approx \frac{P(y_t|x_t)\tilde{\alpha}(x_{t|t-1})}{\int_{x_t} P(y_t|x_t)\tilde{\alpha}(x_{t|t-1})} \\
&\stackrel{\text{def}}{=} \widehat{\alpha}(x_{t|t})
\end{aligned}$$

The goal is to compute the closest Gaussian approximation to this posterior. To simplify notation, let us rewrite the above as

$$\widehat{\alpha}(x_{t|t}) = \hat{p}(\theta) = \frac{l(\theta) q^{old}(\theta)}{\int_\theta l(\theta) q^{old}(\theta)}$$

where $\theta = x_t$, $l(\theta) = P(y_t|x_t)$ is the likelihood, and $q^{old}(\theta) = \tilde{\alpha}(x_{t|t-1})$ is the Gaussian prior. The goal is to minimize $D(\hat{p}(\theta)||q^{new}(\theta))$ subject to the constraint that $q^{new}(\theta) = \mathcal{N}(\theta; m_\theta^{new}, V_\theta^{new})$ is Gaussian, where $m_\theta^{new} = E_{q^{new}}[\theta]$ and $V_\theta^{new} = E_{q^{new}}[\theta\theta^T] - E_{q^{new}}[\theta]E_{q^{new}}[\theta]^T$. Taking derivatives wrt m_θ^{new} and V_θ^{new} gives the expectation constraints

$$\begin{aligned}
E_{q^{new}}[\theta] &= E_{\hat{p}}[\theta] = \int_\theta \hat{p}(\theta)\theta \\
E_{q^{new}}[\theta\theta^T] &= E_{\hat{p}}[\theta\theta^T] = \int_\theta \hat{p}(\theta)\theta\theta^T
\end{aligned}$$

We can compute the expectations wrt $\hat{p}$ using the following relations (obtained from integration by parts):

$$\begin{aligned}
Z(m_\theta^{old}, V_\theta^{old}) &= \int_\theta l(\theta) q^{old}(\theta) \\
d_m &= \nabla_{m_\theta^{old}} \log Z(m_\theta^{old}, V_\theta^{old}) \\
d_V &= \nabla_{V_\theta^{old}} \log Z(m_\theta^{old}, V_\theta^{old}) \\
E_{\hat{p}}[\theta] &= m_\theta^{old} + V_\theta^{old} d_m \\
E_{\hat{p}}[\theta\theta^T] - E_{\hat{p}}[\theta] E_{\hat{p}}[\theta]^T &= V_\theta^{old} - V_\theta^{old} \left(d_m d_m^T - 2 d_V \right) V_\theta^{old}
\end{aligned}$$

These equations hold for any likelihood term $l(\theta)$, which could be e.g., a mixture of Gaussians. In general, solving the integral in the definition of Z might require further approximations.

5.3 Belief state = mixture of Gaussians

A natural extension is to allow the belief state to be represented by a mixture of Gaussians. The goal is to take the old belief state, $P(X_{t-1}|y_{1:t-1})$, represented by a mixture of M Gaussians, to pass it through an arbitrary transition model to get the prior $P(X_t|y_{1:t-1})$, to do Bayesian updating with this prior and an arbitrary observation model, $P(Y_t|X_t)$, and then to approximate the result with another mixture of Gaussians. Unfortunately, an efficient, general algorithm to do this has not yet been found.

An important special case for which this problem can be (approximately) solved is filtering in a switching SSM (Section 2.11). The old belief, $P(X_{t-1}, S_{t-1}|y_{1:t-1})$, is naturally represented as a mixture of M Gaussians, one for each value of the switch, S_{t-1}. For each value of the new switch, $S_t = j$, we have a linear dyamical system with parameters θ_j. We apply the standard Kalman filter update to each mode i of the prior, using parameters θ_j. Hence the posterior $P(X_t, S_t = j|y_{1:t}, S_{t-1} = i)$ is a mixture of M^2 Gaussians. The posterior needs to be reduced back to a mixture of M Gaussians, otherwise the size of the belief state will grow exponentially with time. There are a variety of heuristic ways to do this:

- Drop mixture components of low weight.
- Sample mixture components according to their weight.
- Repeatedly merge the most similar pair of mixture components.
- Minimize the divergence between the M^2-component and the M-component mixture using nonlinear optimization.
- For each value of $S_t = j$, use moment matching (Section 5.2.4) to approximate the mixture $\sum_i P(X_t, S_t = j|S_{t-1} = i, y_{1:t})$ by a single Gaussian. This is sometimes called the second-order generalized pseudo Bayesian (GPB2) algorithm, and we explain it in more detail in Section 5.3.1 below.

5.3.1 GPB2 algorithm

The basic idea behind GBP2 is shown in Figure 39. For each mode of the prior, $P(X_{t-1}, S_{t-1} = i|y_{1:t-1})$, and for each value of the current switch $S_t = j$, we compute $P(X_t, S_t = j, S_{t-1} = i|y_{1:t})$ using a Kalman filter with parameters θ_j. We then collapse the different S_{t-1} components of $P(X_t, S_t = j, S_{t-1} = i|y_{1:t})$ for each j by moment matching. Specifically, if

$$P(X_t, S_t = j, S_{t-1} = i|y_{1:t}) = p_{ij} \mathcal{N}(X_t; \mu_{ij}, \Sigma_{ij})$$

then the new distribution is given by

$$P(X_t, S_t = j|y_{1:t}) = p_j \mathcal{N}(X_t; \mu_j, \Sigma_j)$$

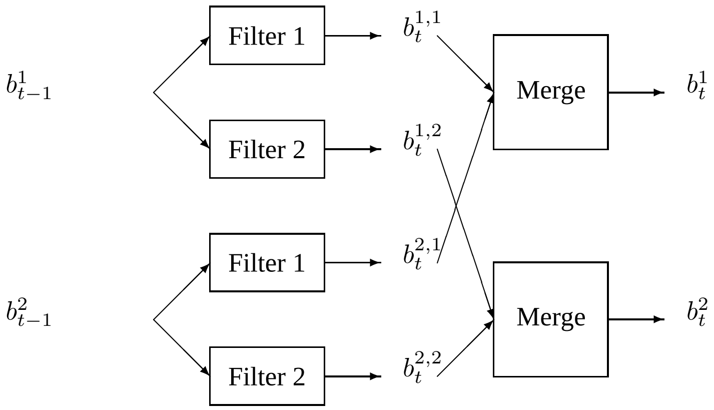


Figure 39: The GPB2 algorithm for the case of a binary switch. $b_{t-1}^i = P(X_{t-1}, S_{t-1} = i|y_{1:t-1})$ is the prior, $b_t^{i,j} = P(X_t, S_{t-1} = i, S_t = j|y_{1:t})$, is the joint posterior, and $b_t^j = P(X_t, S_t = y|y_{1:t})$ is the marginal posterior. The filter box implements the Kalman filter equations. The merge box implements the moment matching equations.

where

$$\begin{aligned}
p_j &= \sum_i p_{ij} \\
p_{j|i} &= \frac{p_{ij}}{\sum_j p_{ij}} \\
\mu_j &= \sum_i \mu_{ij} p_{j|i} \\
\Sigma_j &= \sum_i \Sigma_{ij} p_{j|i} + \sum_i (\mu_{ij} - \mu_j)(\mu_{ij} - \mu_j)^T p_{j|i}
\end{aligned}$$

In the junction tree literature, this is called “weak marginalization”. It can be applied to any conditionally Gaussian model, not just switching SSMs.

The GPB2 algorithm requires running M^2 Kalman filters at each step. A cheaper alternative, known as interacting multiple models (IMM), can be obtained by first collapsing the prior to a single Gaussian (by moment matching), and then updating it using M different Kalman filters, one per value of S_t: see Figure 40. Unfortunately, it is hard to extend IMM to the smoothing case, unlike GPB2, a smoothing version of which is discussed in Section 6.1.3.

5.3.2 Viterbi approximation

If there are a large number of discrete variables, it may be too slow to perform M^2 or even M KF updates, as required by GPB2 and IMM. Instead, one can enumerate the discrete values in a priori order of probability. (Computing their posterior probability is as expensive as an exact update step.) This makes sense for a DBN for fault diagnosis, where there is a natural ordering on the discrete values: one fault is much more likely than two faults, etc. However, it would not be applicable to the data association DBN in Figure 23, where there is no such ordering.

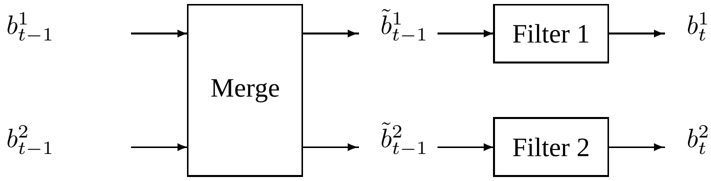


Figure 40: The IMM (interacting multiple models) algorithm for the case of a binary switch.

5.4 Belief state = set of samples (particle filtering)

The basic idea behind particle filtering[6] is to approximate the belief state by a set of weighted particles or samples:

$$P(X_t|y_{1:t}) \approx \sum_{i=1}^{N_s} w_t^i \delta(X_t, X_t^i)$$

(In this section, X_t^i means the i'th sample of X_t, and $X_{t,i}$ means the i'th component of X_t.) This is a non-parametric approach, and hence can handle non-linearities, multi-modal distributions, etc. The advantage over discretization is that the method is adaptive, placing more particles (corresponding to a finer discretization) in places where the probability density is higher.

Given a prior of this form, we can compute the posterior using importance sampling. In importance sampling, we assume the target distribution, $\pi(x)$, is hard to sample from; instead, we sample from a proposal or importance distribution $q(x)$, and weight the sample according to $w^i \propto \pi(x)/q(x)$. (After we have finished sampling, we can normalize all the weights so $\sum_i w^i = 1$). We can use this to sample paths with weights

$$w_t^i \propto \frac{P(x^i_{1:t}|y_{1:t})}{q(x^i_{1:t}|y_{1:t})}$$

The probability of a sample path, $P(x^i_{1:t}|y_{1:t})$, can be computed recursively using Bayes rule. Typically we will want the proposal distribution to be recursive also, i.e., $q(x_{1:t}|y_{1:t}) = q(x_t|x_{1:t-1}, y_{1:t})q(x_{1:t-1}|y_{1:t-1})$. In this case we have

$$\begin{aligned} w_t^i &\propto \frac{P(y_t|x_t^i)P(x_t^i|x_{t-1}^i)P(x^i_{1:t-1}|y_{1:t-1})}{q(x_t^i|x^i_{1:t-1}, y_{1:t})q(x^i_{1:t-1}|y_{1:t-1})} \\ &= \frac{P(y_t|x_t^i)P(x_t^i|x_{t-1}^i)}{q(x_t^i|x^i_{1:t-1}, y_{1:t})} w_{t-1}^i \\ &\stackrel{\text{def}}{=} \hat{w}_t^i \times w_{t-1}^i \end{aligned}$$

where we have defined $\hat{w}_t^i$ to be the incremental weight.

For filtering, we usually only care about the posterior marginal $P(X_t|y_{1:t})$, as opposed to the full posterior $P(X_{1:t}|y_{1:t})$. Hence we use the following proposal: $q(x_t|x^i_{1:t-1}, y_{1:t}) = q(x_t|x^i_{t-1}, y_t)$. This means we only need to

---

[6] Particle filtering is also known as sequential Monte Carlo, sequential importance sampling with resampling (SISR), the bootstrap filter, the condensation algorithm, survival of the fittest, etc.

```
function $[\{x_t^i, w_t^i\}_{i=1}^{N_s}] = \text{PF}(\{x_{t-1}^i, w_{t-1}^i\}_{i=1}^{N_s}, y_t)$
for $i = 1 : N_s$
    Sample $x_t^i \sim q(\cdot|x_{t-1}^i, y_t)$
    Compute $\hat{w}_t^i$ from Equation 1
    $w_t^i = \hat{w}_t^i \times w_{t-1}^i$
Compute $w_t = \sum_{i=1}^{N_s} w_t^i$
Normalize $w_t^i := w_t^i / w_t$
Compute $N_{eff}$ from Equation 2
if $N_{eff} <$ threshold
  $\pi = \text{resample}(\{w_t^i\}_{i=1}^{N_s})$
  $x_t^{\cdot} = x_t^{\pi}$
  $w_t^i = 1/N_s$
```

Figure 41: Pseudo-code for a generic particle filter. The resample step samples indices with replacement according to their weight; the resulting set of sampled indices is called π. The line $x_t^{\cdot} = x_t^{\pi}$ simply duplicates or removes particles according to the chosen indices.

store x_t in each particle, instead of the whole trajectory, $x_{1:t}$. With this proposal, the weights simplify to

$$\hat{w}_t^i = \frac{P(y_t|x_t^i)P(x_t^i|x_{t-1}^i)}{q(x_t^i|x_{t-1}^i, y_t)} \tag{1}$$

5.4.1 The resampling step

The algorithm described so far is known as sequential importance sampling (SIS). A well known problem with SIS is that, over time, one of the normalized importance weights tends to 1, while the others tend to zero (even if we use the optimal proposal distribution: see Section 5.4.2). Hence a large number of samples are effectively wasted, since their weight is negligible. This is called particle "impoverishment".

An estimate of the "effective" number of samples is given by

$$N_{eff} = \frac{1}{\sum_{i=1}^{N_s} (w_t^i)^2} \tag{2}$$

If this drops below some threshold, we can sample with replacement from the current belief state. Essentially this throws out particles with low weight and replicates those with high weight. This is called resampling, and can be done in $O(N_s)$ time using a variety of methods. After resampling, the weights are reset to the uniform distribution: the past weights are reflected in the frequency with which particles are sampled, and do not need to be kept.

Particle filtering is just sequential importance sampling with resampling. The resampling step was the key innovation in the 1990s; SIS itself has been around since at least the 1950s. The overall algorithm is sketched in Figure 41. Its simplicity and generality is one reason for the algorithm's widespread popularity.

Although resampling kills off unlikely particles, it also reduces the diversity of the population (which is why we don't do it at every time step; if we did, then $w_t^i = \hat{w}_t^i$). This a particularly severe problem is the system is highly deterministic (e.g., if the state space contains static parameters). A simple solution is to apply a kernel around each particle and then resample from the kernel. An alternative is to use an MCMC step, that samples from $P(X_t|y_{1:t})$, thus introducing diversity without affecting the correctness of the algorithm.

5.4.2 The proposal distribution

By far the most common proposal is to sample from the transition prior: $q(x_t|x_{t-1}^i, y_t) = P(x_t|x_{t-1}^i)$. In this case, the weights simplify to $\hat{w}_t^i = P(y_t|x_t^i)$. This version of particle filtering is sometimes called the condensation algorithm: see Figure 42.

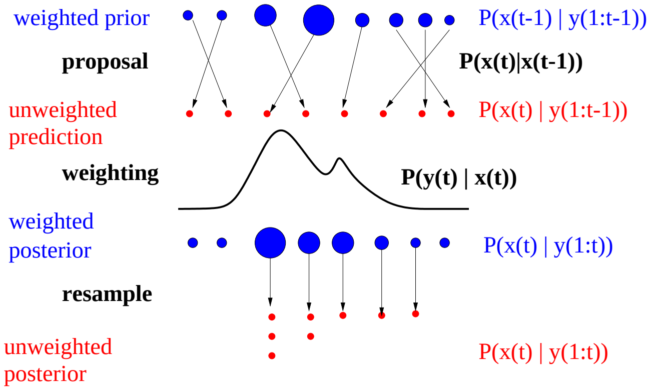


Figure 42: Particle filtering, where the proposal is the transition prior, $q(x_t|x^i_{t-1}, y_t) = P(x_t|x^i_{t-1})$, and hence the incremental weight is the likelihood, $\hat{w}^i_t = P(y_t|x^i_t)$. In this example, we assume we start with a set of (unequally) weighted particles, representing the prior, and transform them to a set of (equally) weighted particles representing the posterior. Notice how the resampling step decreases diversity of the population, and makes all the weights uniform.

For predicting the future, sampling from the transition prior is adequate, since there is no future evidence. But for monitoring/ filtering, it is not very efficient, since it amounts to "guess until you hit". For example, if the transitions are highly stochastic, sampling from the prior will result in particles being proposed all over the state-space; if the observations are highly informative, most of the particles will get "killed off" (i.e., assigned low weight). In such a case, it makes more sense to first look at the evidence, y_t, and then propose:

$$q(x_t|x^i_{t-1}, y_t) = P(x_t|x^i_{t-1}, y_t) \propto P(y_t|x_t)P(x_t|x^i_{t-1})$$

In fact, one can prove this is the optimal proposal distribution, in the sense of minimizing the variance of the weights. (High variance distributions are wasteful, since they correspond to the case where some weights are high and some are low; the low weight particles are "wasted".)

If we use the optimal proposal, the incremental weight is given by the one-step-ahead likelihood:

$$\hat{w}^i_t = P(y_t|x^i_{t-1}) = \int_{x_t} P(y_t|x_t)P(x_t|x^i_{t-1})$$

In general this may be intractable to compute, but if the observation model, $P(Y_t|X_t)$, is linear-Gaussian, and the process noise is Gaussian, i.e.,

$$\begin{aligned} P(X_t|x^i_{t-1}) &= \mathcal{N}(X_t; f_t(x^i_{t-1}), Q_t) \\ P(Y_t|X_t) &= \mathcal{N}(y_t; C_t X_t, R_t) \end{aligned}$$

then one can use the standard Kalman filter rules[7] to show that

$$\begin{aligned} P(X_t|x^i_{t-1}, y_t) &= \mathcal{N}(X_t; x_{t|t}, P_{t|t}) \\ \hat{w}^i_t = P(y_t|x^i_{t-1}) &= \mathcal{N}(y_t; C_t f_t(x_{t-1}), Q_t + C_t R_t C'_t) \end{aligned}$$

where

$$\begin{aligned} P^{-1}_{t|t} &= Q^{-1}_t + C^T_t R^{-1}_t C_t \\ x_{t|t} &= \Sigma_t \left(Q^{-1}_t f_t(x^i_{t-1}) + H'_t R^{-1}_t y_t \right) \end{aligned}$$

If the model does not satisfy these requirements, one can still use a Gaussian approximation to $P(X_t|x^i_{t-1}, y_t)$ as the proposal. For example, if we store a mean and covariance in each particle, we can compute $P(X_t|x^i_{t-1}, y_t) \approx$

---

[7] Specifically, we use the information form, since the old belief state is a delta function: $x_{t-1|t-1} = x^i_{t-1}$, $P_{t-1|t-1} = 0$. Hence $x_{t|t-1} = f(x^i_{t-1})$ and $P_{t|t-1} = Q_t$. By Equation 15.43, $P^{-1}_{t|t} = P^{-1}_{t|t-1} + C^T_t R^{-1}_t C_t$, and by Equation 15.54, $P^{-1}_{t|t}\hat{x}_{t|t} = P^{-1}_{t|t-1}\hat{x}_{t|t-1} + C^T_t R^{-1}_t y_t$.

```
function $[\{s_t^i, \mu_t^i, \Sigma_t^i, w_t^i\}] =$ RBPF-SSSM-prior$(\{s_{t-1}^i, \mu_{t-1}^i, \Sigma_{t-1}^i, w_{t-1}^i\}, y_t)$
for $i = 1 : N_s$
    Sample $s_t^i \sim P(S_t | s_{t-1}^i)$
    $(\mu_t^i, \Sigma_t^i, \hat{w}_t^i) =$ KF$(\mu_{t-1}^i, \Sigma_{t-1}^i, y_t, \theta_{s_t^i})$
    $w_t^i = \hat{w}_t^i \times w_{t-1}^i$
Compute $w_t = \sum_{i=1}^{N_s} w_t^i$
Normalize $w_t^i := w_t^i / w_t$
Compute $N_{eff}$ from Equation 2
if $N_{eff} <$ threshold
   $\pi =$ resample$(\{w_t^i\}_{i=1}^{N_s})$
   $s_t^{\cdot} = s_t^{\pi}, \mu_t^{\cdot} = \mu_t^{\pi}, \Sigma_t^{\cdot} = \Sigma_t^{\pi}$
   $w_t^i = 1/N_s$
```

Figure 43: Pseudo-code for Rao-Blackwellised particle filtering applied to a switching SSM, where we sample from the prior.

$\mathcal{N}(X_t; \hat{x}_{t|t}^i, P_{t|t}^i)$ using the EKF/UKF. We can then sample from this, and copy $P_{t|t}^i$ to the newly sampled particle. This is called the extended/ unscented particle filter. The sampling step can overcome bias introduced by the deterministic approximation. Not surprisingly, using a clever proposal dramatically improves performance.

If the process noise is non-Gaussian, but the observation model is (conditional) linear-Gaussian, we can propose from the likelihood and weight by the transition prior.

5.4.3 Rao-Blackwellised Particle Filtering (RBPF)

The Rao-Blackwell theorem shows how to improve upon any given estimator under every convex loss function. At its core is the following well-known identity:

$$\text{Var}[\tau(X,R)] = \text{Var}[E(\tau(X,R)|R)] + E[\text{Var}(\tau(X,R)|R)]$$

where $\tau(X,R)$ is some estimator of X and R. Hence $\text{Var}[E(\tau(X,R)|R)] \leq \text{Var}[\tau(X,R)]$, so $\tau'(X,R) = E(\tau(X,R)|R)$ is a lower variance estimator. So if we can sample R and compute the expectation of X given R analytically, we will need less samples (for a given accuracy). Of course, less samples does not necessarily mean less time: it depends on whether we can compute the conditional expectation efficiently or not.

We now give a simple but useful example of this idea. Consider a switching SSM. If we knew $S_{1:t}$, we could compute $P(X_t | y_{1:t}, s_{1:t})$ exactly using a Kalman filter. Since $S_{1:t}$ is unknown, we can sample it, and then, for each such sample, integrate out X_t analytically using the Kalman filter. Now we are only sampling in a small discrete space, instead of a large hybrid space, so the performance is much better, both in theory and practice.

Algorithmically, what this means is that each particle contains a sampled value for S_t, which implicitly represents a whole trajectory, $S_{1:t}$, plus the sufficient statistics for the Kalman filter conditioned on this trajectory, $\mu_t^i = E[X_t | y_{1:t}, s_{1:t}^i]$ and $\Sigma_t^i = \text{Cov}[X_t | y_{1:t}, s_{1:t}^i]$. If we propose from the transition prior for S_t, we can compute the weight using the marginal likelihood $P(y_t | s_{1:t}^i)$:

$$\begin{aligned}
\hat{w}_t^i &= \frac{P(y_t | s_t, s_{1:t-1}^i) P(s_t | s_{t-1}^i)}{P(s_t | s_{t-1}^i)} \\
&= \int_{x_t} P(y_t | x_t, s_t) P(x_t | s_t, s_{1:t-1}^i) \\
&= \mathcal{N}(y_t; C_{s_t} A_{s_t} \mu_{t-1}^i, C_{s_t}(A_{s_t} \Sigma_{t-1}^i A_{s_t}^T + Q_{s_t}) C_{s_t}^T + R_{s_t})
\end{aligned}$$

This term is just a byproduct of applying the Kalman filtering equations to the sufficient statistics $\mu_{t-1}^i, \Sigma_{t-1}^i$ with the parameter set $\theta_{s_t} = (A_{s_t}, C_{s_t}, Q_{s_t}, R_{s_t})$. The overall algorithm is shown in Figure 43. Note that we can use this algorithm even if S_t is not discrete.

```
function $[\{s_t^i, \mu_t^i, \Sigma_t^i, w_t^i\}]$ = RBPF-SSSM-opt($\{s_{t-1}^i, \mu_{t-1}^i, \Sigma_{t-1}^i, w_{t-1}^i\}, y_t$)
for $i = 1 : N_s$
    for each $s$
        $(\mu^s, \Sigma^s, L(s))$ = KF($\mu_{t-1}^i, \Sigma_{t-1}^i, y_t, \theta_s$)
        $q(s) = L(s) \times P(S_t = s|s_{t-1}^i)$
    $\hat{w}^s = \sum_s q(s)$
    Normalize $q(s) := q(s)/\hat{w}^s$
    Sample $s \sim q(\cdot)$
    $s_t^i = s$, $\mu_t^i = \mu^s$, $\Sigma_t^i = \Sigma^s$, $\hat{w}_t^i = \hat{w}^s$
    $w_t^i = \hat{w}_t^i \times w_{t-1}^i$
...
```

Figure 44: Pseudo-code for RBPF applied to a switching SSM, where we sample from the optimal proposal. The third return argument from the KF routine is $L(s) = P(y_t|S_t = s, s_{1:t-1}^i, y_{1:t-1})$. . . . means the code continues as in Figure 43.

```
function $[x_t^i, \hat{w}_t^i]$ = LW($x_{t-1}^i, y_t$)
$\hat{w}_t^i = 1$
$x_t^i$ = empty vector of length $N$
for each node $i$ in topological order
    Let $u$ be the value of Pa($X_t^i$) in $(x_{t-1}^i, x_t^i)$
    If $X_t^i$ not in $y_t$
        Sample $x_t^i \sim P(X_t^i|\text{Pa}(X_t^i) = u)$
    else
        $x_t^i$ = the value of $X_t^i$ in $y_t$
        $\hat{w}_t^i = \hat{w}_t^i \times P(X_t^i = x_t^i|\text{Pa}(X_t^i) = u)$
```

Figure 45: Pseudo-code for likelihood weighting.

If the number of values of S_t is sufficiently small, and the transition model for S_t is not very informative, it might be beneficial to use the optimal proposal distribution, which is given by

$$P(S_t = s|s_{1:t-1}^i, y_{1:t}) \propto P(y_t|S_t = s, s_{1:t-1}^i, y_{1:t-1})P(S_t = s|s_{t-1}^i)$$

As usual, the incremental weight is just the normalizing constant for the optimal proposal:

$$\hat{w}_t^i = \sum_s P(y_t|S_t = s, s_{1:t-1}^i, y_{1:t-1})P(S_t = s|s_{t-1}^i)$$

The modified algorithm is shown in Figure 44. This is more expensive than sampling from the transition prior, since for each particle i, we must loop over all states s in order to compute the proposal distribution. However, this may require fewer particles, making it faster overall.

5.4.4 Particle filtering for DBNs

So far, we have assumed it is easy to sample from the transition model, $P(X_t|X_{t-1}^i)$, and to compute the weight $P(y_t|x_t^i)$. But what if the transition and observation models are represented by a complex DBN, instead of a simple parametric function? To apply particle filtering to a DBN, we can use the likelihood weighting (LW) routine in Figure 45 to sample x_t^i and compute $\hat{w}_t^i$, given x_t^{i-1} and y_t. The proposal distribution that LW corresponds to depends on which nodes of the DBN are observed. In the simplest case of an HMM, where the observation is at a leaf node, LW samples from the transition prior, $P(X_t^i|x_{t-1}^i)$, and the computes the weight as $w = P(y_t|x_t^i)$. (We discuss how to improve this below.)

In general, some of the evidence might occur at arbitrary locations within the DBN slice. In this case, the proposal is $q(x_t, y_t) = \prod_j P(x_{t,j}|\text{Pa}(X_{t,j}))$, and the weight is $w(x_t, y_t) = \prod_j P(y_{t,j}|\text{Pa}(Y_{t,j}))$, where $x_{t,j}$ is the (value of the) j'th hidden node at time t, and $y_{t,j}$ is the (value of the) j'th observed node at time t, and the parents of both $X_{t,j}$ and $Y_{t,j}$ may contain evidence. This is consistent, since

$$P(x_t, y_t) = \prod_j P(x_{t,j}|\text{Pa}(X_{t,j})) \times \prod_j P(y_{t,j}|\text{Pa}(Y_{t,j})) = q(x_t, y_t)w(x_t, y_t)$$

Improving the proposal distribution Since the evidence usually occurs at the leaves, likelihood weighting effectively samples from the transition prior without looking at the evidence y_t. A general way to take the evidence into account while sampling in a Bayes net is called "evidence reversal". This means applying the rules of "arc reversal" until all the evidence nodes become parents instead of leaves.

To reverse an arc from $X \to Y$, we must add Y's unique parents, Y_p, to X, and add X's unique parents, X_p, to Y (both nodes may also share common parents, C): see Figure 46. The CPDs in the new network are given by

$$\begin{aligned}
P(Y|Y_p, X_p, C) &= \sum_x P(Y, x|Y_p, X_p, C) \\
&= \sum_x P(Y|x, Y_p, C)P(x|X_p, C) \\
P(X|Y,\ Y_p, X_p, C) &= \frac{P(Y|X,\ Y_p, X_p, C)P(X|Y_p, X_p, C)}{P(Y|Y_p, X_p, C)} \\
&= \frac{P(Y|X, Y_p, C)P(X|X_p, C)}{P(Y|Y_p, X_p, C)}
\end{aligned}$$

Note that X_p, Y_p and C could represent sets of variables. Hence the new CPDs could be much larger than before the arc reversal.

In the case of an HMM, the arc reversal operation is shown in Figure 47. Applying the above rules with $X = X_t$, $Y = Y_t$, $X_p = X_{t-1}$, $Y_p = C = \emptyset$ yields the new CPDs:

$$\begin{aligned}
P(Y_t|X_{t-1}) &= \sum_{x_t} P(Y_t|x_t)P(x_t|X_{t-1}) \\
P(X_t|X_{t-1}, Y_t) &= \frac{P(Y_t|X_t)P(X_t|X_{t-1})}{P(Y_t|X_{t-1})}
\end{aligned}$$

For a general DBN, we may apply arc reversal to one or more of the leaf nodes, in order to improve the efficiency of the proposal distribution.

An alternative to using likelihood weighting combined with arc reversal is to create a junction tree from the original 2TBN. The evidence, E, consists of all of the nodes in slice $t-1$ (the values are stored in the old particle) plus a subset of the nodes in slice t (the new observations). One can sample values for the remaining nodes, H, from the optimal distribution $P(H|E)$ using the following two pass algorithm. First collect to the root, as usual, but then, in the distribute phase, draw a random sample from $P(X_{C_i \setminus S_i}|x_{S_i}, E)$ for each clique C_i, where S_i is the separator nearer to the root. Although this samples from the optimal distribution, it is not usually possible to implement the necessary operations for manipulating the potentials if some of the variables are continuous and non-Gaussian. (Of course, arc reversal cannot be applied in this case either.)

Combining particle filtering with a factored belief state It is possible to combine particle filtering with the Boyen-Koller algorithm (see Section 5.1.1), i.e., to approximate the belief state by

$$\hat{P}(X_t|y_{1:t}) \approx \prod_{c=1}^{C} \frac{1}{N_c} \sum_{i=1}^{N_c} \delta(X_{t,c}, x_{t,c}^i)$$

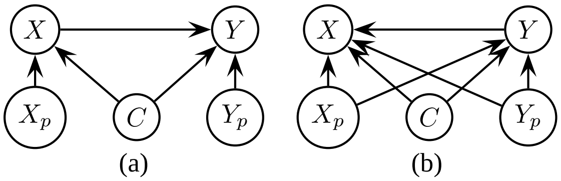


Figure 46: Arc reversal. (a) The original network. (b) The network after reversing the $X \to Y$ arc. The modified network encodes the same probability distribution as the original network.

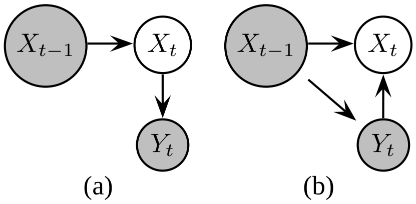


Figure 47: A DBN (a) before and (b) after evidence reversal.

where C is the number of clusters, and N_c is the number of particles in each cluster (assumed uniformly weighted). By applying particle filtering to a set of smaller state-spaces (each cluster), one can reduce the variance, at a cost of increasing the bias by using a factored representation. However, it now becomes much more complicated to propagate each particle.

5.5 Belief state = variable sized

In some applications, the size of the state-space is not fixed in advance. For instance, when we are tracking multiple objects, each measurement could have been caused by any of them, the background, or perhaps a new object that has entered the "scene". This problem arises in visual tracking, tracking missiles with radar, and also in mobile robotics, in particular in the SLAM (simultaneous localization and mapping) problem.

A standard heuristic way to detect the presence of new objects is if an observation arises in a location that was not expected. If we have a prior over each object's position, we can compute a distribution over the expected measurement position:

$$P(Y_t|y_{1:t-1}) = \int_x P(Y_t|X_t = x)P(X_t = x|y_{1:t-1})$$

If these densities are Gaussians, this is often represented as a confidence ellipse or validation gate. If an observation falls inside this ellipse, we assume it was generated by the object. If there is more than one observation inside the ellipse, or if the observation falls inside more than one ellipse, we can either assign the observation to the nearest target (using Mahalanobis distance), or compute the likelihood of all possible joint assignments of observations to targets, and pick the most likely one (Note that the nearest neighbor rule might assign the same measurement to multiple objects, which leads to inaccuracies.)

If an observation does not fall inside the validation gate of any existing object, it could either be due to a new object, or due to background clutter. Hence we consider both hypotheses, and add the new object to a provisional list. Once the object on the provisional list receives a minimum number of measurements inside its validation gate, it is added to the state space.

It is also possible for an object to be removed from the state space if it has not been updated recently (e.g., it left the scene). Hence in general we must allow the state space to grow and shrink dynamically.

A more rigorous approach to onference in such variable-sized state-spaces models is to use particle filtering. This is easy since each particle can have a different dimensionality. We give a detailed example below, in the context of online model selection.

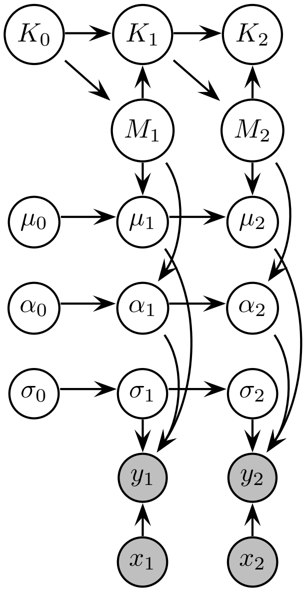


Figure 48: A DBN for sequential Bayesian model selection.

5.5.1 Non-linear regression with online model selection

We discussed a model for online parameter estimation for non-linear regression in Section 2.10. Now we will let the number of basis functions (and hence the size of the state space) vary over time. Let K_t be the number of bases at time t, and let M_t be a random variable with five possible values: B=birth, D=death, S=split, M=merge and N=no-change. These specify the ways in which K_t can increase/decrease by 1 at each step. Birth means we create a new basis function; its position can depend on the previous basis function centers, μ_{t-1}, as well as the current data point, x_t; death means we remove a basis function at random, split means we split one center into two, and merge means we combine two centers into one. We enforce that $0 \le K_t \le K_{max}$ at all times. The DBN is shown in Figure 48, and the CPDs are as follows.

$$\begin{aligned}
P(M_t = (B, D, S, M, N)|K_{t-1} = k) &= \begin{cases} (\frac{1}{2}, 0, 0, 0, \frac{1}{2}) & \text{if } k = 0 \\ (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, 0, \frac{1}{4}) & \text{if } k = 1 \\ (0, \frac{1}{3}, 0, \frac{1}{3}, \frac{1}{3}) & \text{if } k = K_{max} \\ (\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}) & \text{otherwise} \end{cases} \\
P(K_t = k'|K_{t-1} = k, M_t = m) &= \begin{cases} \delta(k', k+1) & \text{if } m = B \text{ or } m = S \\ \delta(k', k-1) & \text{if } m = D \text{ or } m = M \\ \delta(k', k) & \text{if } m = N \end{cases} \\
P(\mu_t|\mu_{t-1}, M_t = m) &= \begin{cases} \mathcal{N}(\mu_t; \mu_{t-1}, \delta_\mu I) & \text{if } m = N \\ \mathcal{N}(\mu_t; \text{birth}(\mu_{t-1}), \cdot) & \text{if } m = B \\ \mathcal{N}(\mu_t; \text{death}(\mu_{t-1}), \cdot) & \text{if } m = D \\ \mathcal{N}(\mu_t; \text{split}(\mu_{t-1}), \cdot) & \text{if } m = S \\ \mathcal{N}(\mu_t; \text{merge}(\mu_{t-1}), \cdot) & \text{if } m = M \end{cases} \\
P(\alpha_t|\alpha_{t-1}, M_t = m) &= \begin{cases} \mathcal{N}(\alpha_t; \alpha_{t-1}, \delta_\alpha I) & \text{if } m = N \\ \mathcal{N}(\alpha_t; \text{grow}(\alpha_{t-1}), \cdot) & \text{if } m = B \text{ or } m = S \\ \mathcal{N}(\alpha_t; \text{shrink}(\alpha_{t-1}), \cdot) & \text{if } m = D \text{ or } m = M \end{cases} \\
P(\log \sigma_t|\log \sigma_{t-1}) &= \mathcal{N}(\log \sigma_t; \log \sigma_{t-1}, \delta_\sigma I) \\
P(y_t|x_t, \alpha_t, \mu_t) &= \mathcal{N}(y_t; D(\mu_t, x_t)\alpha_t, \sigma_t)
\end{aligned}$$

The function birth(μ_{t-1}) takes in the vector of RBF centers and adds a new one to the end, according to some heuristic. In principle, this heuristic could depend on x_t as well (not shown). Call the new center μ_{birth}. The confidence associated with μ_{birth} is yet another free parameter; suppose it is Q_{birth}; then the full CPD would be

$$P(\mu_t|\mu_{t-1}, M_t = B) = \mathcal{N}\left(\mu_t; \begin{pmatrix}\mu'_{t-1} & \mu'_{\text{birth}}\end{pmatrix}', \begin{pmatrix}0 & 0\\ 0 & Q_{\text{birth}}\end{pmatrix}\right)$$

The 0 terms in the covariance matrix do not mean we are certain about the locations of the other centers, only that we have not introduced any extra noise, i.e.,

$$\text{Cov}(\mu_t|y_{1:t}, M_t = B) = \begin{pmatrix}\text{Cov}(\mu_{t-1}|y_{1:t-1}) & 0\\ 0 & Q_{\text{birth}}\end{pmatrix}$$

Of course, we could relax this assumption. If we were full-blooded Bayesians, we would now add priors to all of our parameters (such as Q_{birth}, δ_α, etc.); these priors would in turn have their own (hyper-)parameters; we would continue in this way until the resulting model is insenstive to the parameters we choose. This is called hierarchical Bayesian modelling.

Despite the apparent complexity, it is actually quite easy to apply particle filtering to this model. [AdFD00] suggest a more complex scheme, which combines particle filtering with reversible jump MCMC [Gre98]. (Regular MCMC can only be applied if the state-space has a fixed size.) We can use the simpler method of particle filtering because we included M_t in the state-space.

6 Approximate smoothing

There are two main kinds of approximate smoothing algorithms: those that make a single forwards-backwards pass over the data (two-filter smoothers), and those that make multiple passes. Exact inference algorithms only need to make a single forwards-backwards pass, but in general, approximate inference can achieve better results if it is allowed to make multiple passes. We will briefly discuss a variety of algorithms below.

6.1 Two-filter smoothing

As for filtering, we can classify two-filter smoothers based on which representation they choose for the forwards and backwards messages. By forwards message we mean $\alpha_t = P(X_t|y_{1:t})$, the filtered estimate; by backwards message we mean either $\gamma_t = P(X_t|y_{1:T})$,the smoothed estimate, or $\beta_t = P(y_{t+1:T}|X_t)$, the conditional likelihood. (If the dynamics are invertible, we can define a more natural backwards filtered estimate, $P(X_t|y_{t+1:T})$. However, in general we wish to avoid the invertibility assumption. The backwards messages can be computed by modifying the forward recursive update equations.

6.1.1 Belief state = discrete distribution

We have already discussed how to perform an exact backwards step, either in β or γ form, for discrete-state DBNs in Section 4. In Section 5.1, we discussed some approximate filtering methods if the exact ones are too slow. We now discuss how to extend these to the smoothing case.

ADF/BK algorithm It is fairly easy to modify the interface algorithm to use a factored representation, as we saw in Section 5.1.1. The same modifications can be used in the backwards pass in the obvious way.

Beam search The standard approximation is to compute β only for those states i which were considered at time t of the forwards pass, i.e., for which $\hat{\alpha}_t(i) > 0$, where $\hat{\alpha}_t$ is the pruned version of α_t.

6.1.2 Belief state = Gaussian distribution

Kalman smoother We saw the β and γ versions of the Kalman smoother in Chapter **??**. For completness, we restate the equations here. First we compute the following predicted quantities (or we could pass them in from the filtering stage):

$$\begin{aligned} x_{t+1|t} &= A_{t+1} x_{t|t} \\ P_{t+1|t} &= A_{t+1} P_{t|t} A_{t+1}^T + Q_{t+1} \end{aligned}$$

Then we compute the smoother gain matrix.

$$J_t = P_{t|t} A_{t+1}^T P_{t+1|t}^{-1}$$

Finally, we compute our estimates of the mean, variance, and cross variance $P_{t,t-1|T} = \mathrm{Cov}[X_{t-1}, X_t | y_{1:T}]$.

$$\begin{aligned} x_{t|T} &= x_{t|t} + J_t \left(x_{t+1|T} - x_{t+1|t} \right) \\ P_{t|T} &= P_{t|t} + J_t \left(P_{t+1|T} - P_{t+1|t} \right) J_t' \\ P_{t-1,t|T} &= J_{t-1} P_{t|T} \end{aligned}$$

As for Kalman filtering, We can generalize the Kalman smoother equations to apply to any linear-Gaussian DBN by using the junction tree algorithm discussed in Section 4, but modifying the definition of summation, multiplication and division, so that these operations can be applied to Gaussian potentials instead of discrete ones.

Extended Kalman smoother As we saw in Section 5.2.2, the EKF linearizes the system and then applies the standard KF equations. Similarly, the extended Kalman smoother just applies the above smoothing equations to the same linearized system. Note that the system is linearized about the filtered estimate $\hat{x}_{t|t}$. Although it is possible to relinearize about the backwards estimates, this requires that the dynamics be invertible, which will not generally be the case.

Unscented Kalman smoother It is possible to apply the unscented transform to either the β or γ version of the backwards pass, to get an unscented Kalman smoother. We leave the details as an exercise.

ADF/Moment matching smoother We will discuss the moment matching smoother in the context of switching SSMs in Section 6.1.3 and in the context of expectation propagation in Section 6.2.

6.1.3 Belief state = mixture of Gaussians

The GPB2 filter corresponds to a collect operation in the following junction tree, where we use weak marginalization to marginalize out discrete nodes from a mixture of Gaussians:

$$(Z_{t-1}, Z_t) - [Z_t] - (Z_t, Z_{t+1}) - [Z_{t+1}] - \cdots$$

where $Z_t = (S_t, X_t)$. (We ignore observed nodes for simplicity.) The cliques are in round brackets and the separators are in the square brackets. (Smaller cliques are possible.)

Hence we can create a GPB2 smoother simply by running the backwards pass in the same tree. No additional assumptions are necessary.

6.1.4 Belief state = set of particles

One approach to particle smoothing is to sample complete trajectories, and then reweight them given all the evidence. Another is to sample in both the forwards and backwards direction; unfortunately, this takes $O(T{N_s}^2)$ time, and is therefore generally considered intractable. A much more popular sampling technique for the offline case is Gibbs sampling, discussed in Section 6.4.

6.2 Expectation propagation (EP)

Expectation propagation generalizes assumed density filtering (Section 5.2.4) to the batch context. It is less sensitive to the order in which the Bayesian updates are performed because it can go back and re-optimize each term of the overall likelihood in the context of the other terms. We explain this in more detail below.

Recall that in ADF, the exact posterior is given by

$$\hat{p}(\theta) = \frac{l(\theta)q^{old}(\theta)}{Z}$$

where $q^{old}(\theta) \in \mathcal{F}$ is the prior (a member of the tractable family of distributions $\mathcal{F}$), $l(\theta)$ is the likelihood, and $Z = \int_\theta l(\theta)q^{old}(\theta)$. We then pick the best tractable approximation to this by computing $q^{new}(\theta) = \arg_{q\in\mathcal{F}} \min D(\hat{p}(\theta)||q(\theta))$. This can be viewed as updating the prior and then projecting the posterior, as in Figure 37. Another way to view it is as computing an approximation to the likelihood, $\tilde{l}(\theta)$, that is conjugate to the prior, such that when $\tilde{l}(\theta)$ is combined with $q^{old}(\theta)$, the resulting posterior will be in $\mathcal{F}$. This approximate likelihood can be inferred from

$$\tilde{l}(\theta) = \frac{\hat{p}(\theta)Z}{q^{old}(\theta)}$$

This makes it simple to remove the old likelihood from the joint, and then use a newly approximated likelihood, to get a better approximation.

A simple example of this algorithm applied to a switching SSM is shown in Figure 49. Here the ADF operations correspond to moment-matching for a mixture of Gaussians, as discussed in Section 5.3.1. The first forwards-backwards pass corresponds exactly to the GPB2 smoother. In subsequent iterations, we recompute each clique potential $P(Z_{t-1}, Z_t|y_{1:T})$ by absorbing α messages from the left and β messages from the right, and combining with the local potentials, $\psi_t(z_{t-1}, z_t) = P(y_t|z_t)P(z_t|z_{t-1})$. From this, we can compute the separator potentials, $q(z_t) \approx P(X_t, S_t|y_{1:T})$, using weak marginalization. From this, we can infer the equivalent tractable messages by division: $\alpha(z_t) \propto q(z_t)/\beta(z_t)$ and $\beta(z_t) \propto q(z_t)/\alpha(z_t)$, which can be propagated to future/past slices. Typically 2-3 passes is enough to improve performance.

Another application of EP is to iterate the BK algorithm. This allows the algorithm to recover from incorrect independence assumptions it made in earlier passes. Typically 2-3 passes is enough to improve performance.

6.3 Variational methods

Variational methods are discussed in Chapter **??**, and can be applied to DBNs in a straightforward way. Some of these methods, such as structured variational approximations and variational EM, use the exact inference routines in Section 4 as subroutines.

As an example, we discuss how to apply a structured variational approximation to the switching SSM in Section 2.11.1. In this model, the observation switches between one of M chains, each of which evolves linearly and independently. For simplicity, we assume the observation noise is tied, $R_i = R$, and that $\mu_i = 0$.

Since exact inference is intractable in this model, we choose a tractable approximation of the following form:

$$Q(\{S_t, X_t\}) = \frac{1}{Z_Q}\left[\psi(S_1)\prod_{t=2}^{T}\psi(S_{t-1}, S_t)\right]\prod_{m=1}^{M}\psi(X_1^{(m)})\prod_{t=2}^{T}\psi(X_{t-1}^{(m)}, X_t^{(m)})$$

where the ψ are unnormalized potentials and Z_Q is the normalizing constant. This corresponds to the graphical model in Figure 50.

The potentials for the discrete switching process are defined as follows.

$$\begin{aligned}\psi(S_1 = m) &= P(S_1 = m)q_1^{(m)} \\ \psi(S_{t-1}, S_t = m) &= P(S_t = m|S_{t-1})q_t^{(m)}\end{aligned}$$

where the $q_t^{(m)}$ are variational parameters of the Q distribution (we will discuss how to compute them shortly). These parameters play the same role as the observation likelihood, $P(Y_t|S_t = m)$, does in a regular HMM.

```
for $t = 1 : T$
  $\alpha_t(z_t) = 1$
  $\beta_t(z_t) = 1$
  if $t == 1$
    $\psi_1(z_1) = P(y_1|z_1)P(z_1)$
  else
    $\psi_t(z_{t-1}, z_t) = P(y_t|z_t)P(z_t|z_{t-1})$
while not converged
    for $t = 1 : T$
      if $t == 1$
        $\hat{p}(z_1) = \psi_1(z_1)\beta_1(z_1)$
      else
        $\hat{p}(z_{t-1}, z_t) = \alpha_{t-1}(z_{t-1})\psi_t(z_{t-1}, z_t)\beta_t(z_t)$
      $q(z_t) = \text{Collapse}_{s_{t-1}} \int_{x_{t-1}} \hat{p}(z_{t-1}, z_t)$
      $\alpha_t(z_t) = \frac{q(z_t)}{\beta_t(z_t)}$
    for $t = T : 1$
      $\hat{p}(z_{t-1}, z_t) = \alpha_{t-1}(z_{t-1})\psi_t(z_{t-1}, z_t)\beta_t(z_t)$
      $q(z_{t-1}) = \text{Collapse}_{s_t} \int_{x_t} \hat{p}(z_{t-1}, z_t)$
      $\beta_{t-1}(z_{t-1}) = \frac{q(z_{t-1})}{\alpha_{t-1}(z_{t-1})}$
```

Figure 49: Pseudo-code for EP applied to a switching SSM, based on [HZ02]. $Z_t = (X_t, S_t)$. Collapse means weak marginalization (moment matching). Note that the messages are equivalent to the separator potentials,

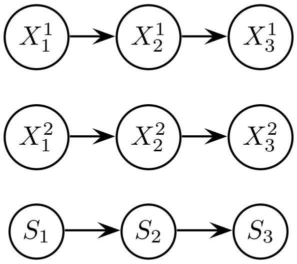


Figure 50: A structured variational approximation to the switching SSM in Figure 23. We remove all the vertical links, replacing them with variational parameters, but leave the horizontal (temporal) links intact.

The potentials for each chain are defined as follows.

$$\begin{aligned}\psi(X_1 = m) &= P(X_1)\left[P(Y_1|X_1^{(m)}, S_1 = m)\right]^{h_1^{(m)}}\\ \psi(X_{t-1}, X_t) &= P(X_t^{(m)}|X_{t-1}^{(m)})\left[P(Y_t|X_t^{(m)}, S_t = m)\right]^{h_t^{(m)}}\end{aligned}$$

where the $h_t^{(m)}$ are more variational parameters of Q, called the responsibilities. The vector $h_t^{(m)}$ is like a soft version of the switch variable S_t: if $h_t^{(m)} = 0$, it means $P(S_t = m) = 0$, so Y_t will not influence $X_t^{(m)}$; if $h_t^{(m)} = 1$, it means $P(S_t = m) = 1$, so Y_t will only influence $X_t^{(m)}$; for values in between, Y_t will have an intermediate effect on $X_t^{(m)}$.

We can compute locally optimal variational parameters by minimizing the KL divergence between Q and P:

$$D(Q||P) = \sum_{\{S_t\}} \int_{\{X_t\}} Q(\{S_t, X_t\}) \log\left[\frac{Q(\{S_t, X_t\})}{P(\{S_t, X_t\}|\{Y_t\})}\right] = H(Q) - < P >$$

where $H(Q)$ is the entropy of Q and $< P >$ is the expected value of P wrt Q. We would like to minimize $D(P||Q)$, but that is intractable to compute, because we would have to take expectations wrt P, which is a mixture of M^T Gaussians.

ADF/EP performs this optimization for a single time step. Hence the difference between EP and variational methods is that EP locally optimizes $D(P||Q)$, whereas variational methods (try to) globally optimize $D(Q||P)$.

Taking derivatives of $D(Q||P)$ wrt the variational parameters yields the following set of coupled equations:

$$\begin{aligned} h_t^{(m)} &= Q(S_t = m) \\ q_t^{(m)} &= \exp\left\{-\tfrac{1}{2}\left\langle (Y_t - C^{(m)} X_t^{(m)'}) R^{-1} (Y_t - C^{(m)} X_t^{(m)}) \right\rangle\right\} \\ &= \exp\left\{-\tfrac{1}{2} Y_t' R^{-1} Y_t + Y_t' R^{-1} C^{(m)} < X_t^{(m)} > -\tfrac{1}{2}\mathrm{tr}\left[C^{(m)'} R^{-1} C^{(m)} < X_t^{(m)} X_t^{(m)'} >\right]\right\} \end{aligned}$$

The $Q(S_t = m)$ term can be computed using the forwards backwards algorithm on an HMM where the observation probabilities are given by $q_t^{(m)}$. The $< X_t^{(m)} >$ and $< X_t^{(m)} X_t^{(m)'} >$ terms, where the expectation is wrt Q, can be computed by running one Kalman smoother per chain, weighting the data by the responsibilities $h_t^{(m)}$. (Weighting the data by $h_t^{(m)}$ is equivalent to using unweighted data and a time-varying observation noise of $R_t^{(m)} = R/h_t^{(m)}$.) The above equations can be iterated until convergence.

In practice, it is essential to anneal the fixed point equations, to avoid getting stuck in bad local optima. This can be done by minimizing $\mathcal{T} H(Q) - < P >$, where $\mathcal{T}$ represents a temperature, and gradually reducing the temperature to $\mathcal{T} = 1$. This ensures a certain level of entropy in the system, and is called deterministic annealing.

Some simple experiments on this model suggest that the variational approximation with deterministic annealing performs about as well as the ADF algorithm (uniterated EP); without annealing, it performs much worse.

6.4 Gibbs sampling

MCMC methods in general, and Gibbs sampling particular, are discussed in Chapter **??**, and can be applied to DBNs in a straightforward way. Unfortunately, Gibbs sampling mixes very slowly for time series, because the data are correlated. It is therefore crucial to use Rao-Blackwellisation.

An example of where this is possible is in switching SSMs. (The advantage of using Gibbs sampling in this context, as opposed to EP or variational methods, is that the algorithm will provably eventually converge to the right answer.) The basic algorithm is as follows:

1. Randomly initialise $s_{1:T}^0$.
2. For $i = 1, 2, \ldots$ until convergence
   (a) For $t = 1, \ldots, T$, sample $s_t^i \sim P(S_t | y_{1:T}, s_{-t}^i)$
   (b) Compute $x_{0:T}^i = \mathrm{E}\left[x_{0:T} | y_{1:t}^i, s_{1:T}^i\right]$

where $s_{-t}^i \overset{\text{def}}{=} \left(s_1^i, \ldots, s_{t-1}^i, s_{t+1}^{i-1}, \ldots, s_T^{i-1}\right)$ contains new values of S_t to the left of t, and old values to the right. The final estimate is then obtained by averaging the samples $s_{1:T}^i$ and $x_{1:T}^i$, possibly discarding an initial segment corresponding to the burn-in period of the Markov chain.

The main issue is how to efficiently compute the sampling distribution,

$$P(s_t | y_{1:T}, s_{-t}) \propto P(s_t | s_{-t}) P(y_{1:T} | s_{1:T})$$

The first term is just

$$P(s_t | s_{-t}) = P(s_t | s_{t-1}) P(s_{t+1} | s_t)$$

The second term is given by

$$\begin{aligned} P(y_{1:T} | s_{1:T}) &= \sum_i P(y_{t+1:T} | y_{1:t}, s_{1:T}, X_t = i) P(y_{1:t}, X_t = i | s_{1:T}) \\ &= \sum_i P(y_{t+1:T} | s_{t+1:T}, X_t = i) P(y_{1:t}, X_t = i | s_{1:t}) \\ &= P(y_{1:t} | s_{1:t}) \sum_i P(y_{t+1:T} | s_{t+1:T}, X_t = i) P(X_t = i | y_{1:t}, s_{1:t}) \\ &= P(y_{1:t-1} | s_{1:t-1}) P(y_t | y_{1:t-1}, s_{1:t}) \sum_i P(y_{t+1:T} | s_{t+1:T}, X_t = i) P(X_t = i | y_{1:t}, s_{1:t}) \end{aligned}$$

Hence, dropping terms that are independent of s_t,

$$P(s_t|y_{1:T}, s_{-t}) \propto P(s_t|s_{t-1})P(s_{t+1}|s_t)P(y_t|y_{1:t-1}, s_{1:t}) \sum_i P(y_{t+1:T}|s_{t+1:T}, X_t = i)P(X_t = i|y_{1:t}, s_{1:t})$$

The key insight is that $P(y_{t+1:T}|s_{t+1:T}, X_t = i)$ is independent of $s_{1:t}$. Hence we can fix $s_{1:t}$ and compute $P(y_{t+1:T}|s_{t+1:T}, X_t = i)$ in a backwards pass using the β form of the backwards Kalman filter (which does not assume invertible dynamics). In the forwards pass, we can sample $s_{1:t}$ in order, computing $P(X_t = i|y_{1:t}, s_{1:t})$ and $P(y_t|y_{1:t-1}, s_{1:t})$ terms as we go.

The relative performance (in terms of accuracy vs. time) of Rao-Blackwellised Gibbs sampling, Rao-Blackwellised particle smoothing, expectation propagation, and structured variational approximations, all of which can be applied to switching SSMs, is unknown at this time.

7 Exercises

7.1 Buried Markov Models

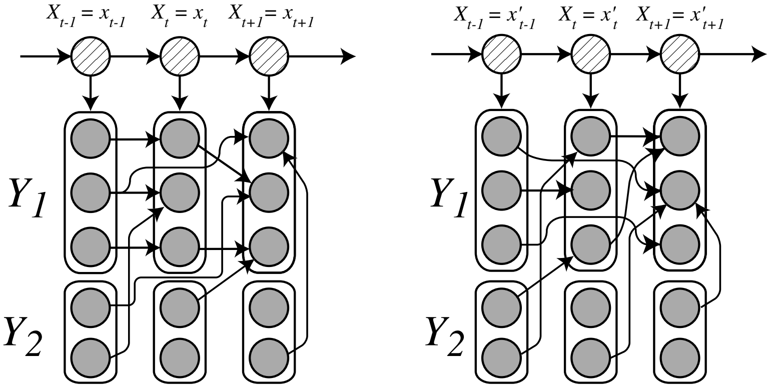


Figure 51: A buried Markov model. Depending on the value of the hidden variables, Q_t, the effective graph structure between the components of the observed variables (i.e., the non-zero elements of the regression matrix), can change. Two different instantiations are shown. Thanks to Jeff Bilmes for this figure.

"Buried" Markov models generalize auto-regressive HMMs (Section 2.2) by allowing non-linear dependencies between the observable nodes. Furthermore, the nature of the dependencies can change depending on the value of Q_t: see Figure 51 for an example. Such a model is called a dynamic Bayesian "multi net", since it is a mixture of different networks. Discuss what changes (if any) need to be made to the forwards-backwards algorithm to do inference in such models.

7.2 2D HMMs

Consider defining a likelihood function for 2D data such as scanned text. It is natural to model each row of the image using an HMM, and to model the connection between the rows using another HMM. This is called a pseudo-2D or embedded HMM. The basic idea is shown in Figure 52, where we there are 2 rows, each of length 3.

1. What is the relationship between this model and the segment HMMs discussed in Section 2.7?
2. Derive an efficient message-passing inference algorithm for pseudo 2D HMMs. Hint: first compute the likelihood of each row, $P(y_{i1:iC}|X_i)$, then use these in the top-level HMM; in the backwards pass, use the top-down information to do full smoothing within each row-HMM.
3. Derive a true 2D "HMM", by modifying the coupled HMM in Section 2.5, so each node depends on its nearest neighbors above and to its left. (Such a model is sometimes called a "causal" MRF.)

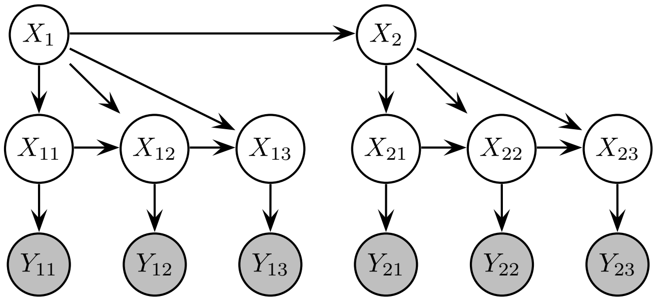


Figure 52: An embedded HMM. Here we have 2 sequences of length 3, both of which are modelled by sub-HMMs.

4. What is the complexity of exact inference in a true 2D HMM?
5. What are the advantages and disadvantages of a causal MRF compared to a traditional undirected MRF?

7.3 Hierarchical HMMs

This exercise asks you to define the CPDs for the DBN representation of an HHMM. We consider the bottom, middle and top layers of the hierarchy separately (since they have different local topology), as well as the first, middle and last time slices.

1. Give the definition for $P(Q_t^D = j|Q_{t-1}^D = i, F_{t-1}^D = f, Q_t^{1:D-1} = k)$ for $t \geq 2$. D is the bottom level of the hierarchy, so $Q_t^{1:D-1} = k$ is an integer encoding the entire stack state. (Not all possible joint values of this vector may be possible, depending on whether the automaton state transition diagram is dense or sparse.)
2. Give the definition for $P(F_t^D = 1|Q_t^{1:D-1} = k, Q_t^D = i)$ for $t \geq 2$. (Assume that each sub-HMM has a unique end/exit state.)
3. Give the definition for $P(Q_t^d = j|Q_{t-1}^d = i, F_{t-1}^{d+1} = b, F_{t-1}^d = f, Q_t^{1:d-1} = k)$ for $t \geq 2$ and $2 \leq d < D$. (The difference from the bottom level is the extra input F_{t-1}^{d+1} representing the signal from below.)
4. Give the definition for $P(F_t^d = 1|Q_t^d = i, Q_t^{1:d-1} = k, F_t^{d+1} = b)$ for $t \geq 2$ and $2 \leq d < D$.
5. Give the definition for $P(Q_t^d = j|Q_{t-1}^d = i, F_{t-1}^{d+1} = b, F_{t-1}^d = f)$ for $t \geq 2$ and $d = 1$ (the top level).
6. Give the definition for $P(F_t^d = 1|Q_t^d = i, F_t^{d+1} = b)$ for $t \geq 2$ and $d = 1$ (the top level).
7. Give the definition for $P(Q_1^1 = j)$ and $P(Q_1^d = j|Q_1^{1:d-1} = k)$ for $d = 2, \ldots, D$.
8. Unlike the automaton representation, the DBN never actually enters an end state (i.e., Q_t^d can never taken on the value "end"); if it did, what should the corresponding observation be? (Unlike an HMM, a DBN must generate an observation at every time step.) Instead, Q_t^d causes F_t^d to turn on, and then enters a new (non-terminal) state at time $t+1$, chosen from its initial state distribution. This means that the DBN and HHMM transition matrices are not identical. What is the relationship between them?

7.4 Fault diagnosis

One important application of switching SSMs is for diagnosing faults. For example, consider the "two tank" system in Figure 53, a benchmark problem in the fault diagnosis community (although one typically considers n tanks, for $n \gg 2$). This is a nonlinear system, since flow = pressure / resistance (or flow = pressure × conductance). More problematically, the values of the resistances can slowly drift, or change discontinuously due to burst pipes. Also, the sensors can fail intermittently and give erroneous results. Show how to represent this model as a DBN.

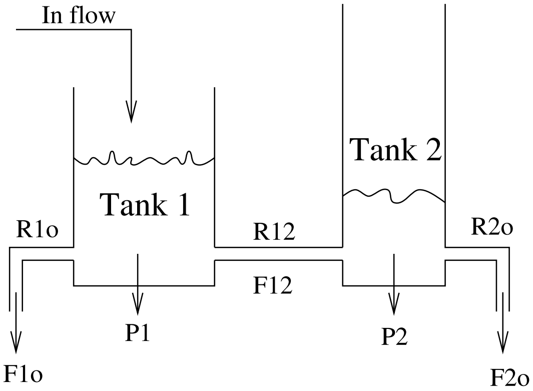


Figure 53: The two-tank system. The goal is to infer when pipes are blocked or have burst, or sensors have broken, from (noisy) observations of the flow out of tank 1, $F1o$, out of tank 2, $F2o$, or between tanks 1 and 2, $F12$. $R1o$ is a hidden variable representing the resistance of the pipe out of tank 1, $P1$ is a hidden variable representing the pressure in tank 1, etc. From Figure 11 of [KL01].

7.5 Generalized $\alpha - \beta$ algorithm

Derive a generalized $\alpha - \beta$ algorithm based on the generalized $\alpha - \gamma$ algorithm in Section 4.4. The forwards pass will be identical, but the backwards pass should compute the analog of the equation (12.44) for HMMs:

$$\xi(q_{t-1}, q_t) = \frac{\alpha(q_{t-1})P(y_t|q_t)A(q_{t-1}, q_t)\beta(q_t)}{P(y)}$$

Discuss the relative merits of the two algorithms.

7.6 Log-space smoothing

This exercise develops a version of the forwards-backwards algorithm that computes $P(X_t|y_{1:T})$ in $O(S \log T)$ working space instead of $O(ST)$ space (ignoring the space required to represent the input and output to the algorithm). This is useful for learning models from long sequence, since the sufficient statistics, $\sum_t P(X_t|y_{1:T})$, can be stored in constant space.

1. Suppose, for simplicity, that T is odd, and let the halfway point be $h = (T+1)/2$. Show that $P(X_t|y_{1:T})$ can be compute for $t = 1 : h$ given just α_1, β_{h+1} and $y_{1:h}$.
2. Show a similar result for the second half of the sequence.
3. Given the results above, a divide-and-conquer algorithm can be constructed by first running forward along the sequence and then backwards from the end, storing just the required messages at the middle and the ends. Then the algorithm is called recursively on the left and right halfs. Write out the algorithm in detail.
4. Show that the space complexity is $O(S \log T)$, where S is the size of the α or β message. What is the time complexity?
5. How does the complexity change if we divide into $k > 2$ pieces? What k yields the minimal space usage? What k should one use if one is willing to wait at most twice as long as the standard algorithm?

7.7 Constant-space smoothing

1. Given an $O(T^2)$ algorithm that computes $P(X_t|y_{1:T})$ for all t in constant space (ignoring the space required to represent the input and output).

2. Suppose we could invert the forwards operator, so that $\alpha_t = \text{Fwd}^{-1})(\alpha_{t+1}, y_{t+1})$. Show how we could use this to compute $P(X_t|y_{1:T})$ for all t in $O(1)$ space and $O(T)$ time. Hint: Express $\beta_{t-l+1|t}$ as a product of matrices times $\beta_{t+1|t} = 1$; similarly express $\beta_{t-l+2|t+1}$ as a product of matrices times $\beta_{t+2|t+1} = 1$; then find a relationship between them.
3. Explain why in general the forwards (and backwards) operators are not invertible. Give a counterexample.

7.8 BK algorithm

Modify the forwards operator from Figure 34 so it takes as input a factored prior α_{t-1}, and returns as output a factored posterior α_t.

8 Historical remarks and bibliography

The term "DBN" was first coined in [DK89]. HMMs with semi-tied mixtures of Gaussians are discussed in [Gal99]. Mixed-memory Markov models are discussed in [SJ99]. ngram models for language are described in [Jel97, Goo01]; variable length Markov models are described in [RST96, McC95]. For continuous data, ngram models correspond to standard regression models, either linear or nonlinear. For a recent tree-based approach to regression, see [MCH02]. Factorial HMMs are discussed in [GJ97]. Coupled HMMs are discussed in [SJ95, Bra96]; the AVSR application is described in [NLP$^+$02] (see also [NY00]), and the traffic application in [KCB00]. Variable-duration HMMs are described in [Rab89]. Segment models are described in [ODK96]. Hierarchical HMMs were introduced in [FST98], where they proposed an $O(T^3)$ inference algorithm. The $O(T)$ inference algorithm was introduced in [MP01]. Applications of HHMMs to robot navigation appear in [TRM01]. HHMMs are very closely related to abstract HMMs [BVW01] and cascaded finite automata [Moh96, PR97].

State-space models date back to the '60s. See e.g., [DK01] for a recent account of applications of SSMs to time-series analysis, and [WH97] for a more Bayesian approach. Dynamic chain graphs are described in [DE00]. The example of sequential nonlinear regression is from [AdFD00]. Switching SSMs date back to the '60s, and are commonly used in econometrics and control. Data association is described in many places, e.g., [BSF88].

Exact inference for DBNs using unrolled junction trees was first described in [Kja92]; see [Kja95] for a more detailed account. Some of the complexity results are from [Dar01]. The frontier algorithm is from [Zwe96]; a special case is described in Appendix B of [GJ97]. The interface algorithm is from [Mur02]. The example of conditionally tractable substructure is from [TDW02]. The idea of sampling the root nodes to get a factored belief state was first proposed in [Mur00] in the context of map learning, and was extended in [MTKW02].

The idea of classifying filtering algorithms based on the type of belief state representation is from [Min02]. The Boyen-Koller algorithm is described in [BK98b]; the smoothing extension is in [BK98a]. A method for computing the top N most probable paths in an HMM is described in [NG01]; the generalization to any Bayes net is in [Nil98]. The KF and EKF algorithms date back to the '60s: see e.g., [AM79]. (For a derivation of the KF equations from a Bayes net perspective, see [NT92, Mur98a].) The unscented Kalman filter was first described in [JU97]; the presentation in Section 5.2.3 is closely based on [WdM01]. ADF is a classical technique; the presentation in Section 5.2.4 is based on [Min01]. Various approaches to filtering in switching SSMs are discussed in [SM80, TSM85]. The GPB2 and IMM filters are described in [BSL93]; the smoothing step is discussed in [Kim94, Mur98b]. Exact inference in conditionally Gaussian models (even tree-structured ones) is NP-hard, as proved in [LP01].

See [AMGC02, LC98, Dou98] for good tutorials on particle filtering, and [DdFG01] for a collection of recent papers. Rao-Blackwellisation in general is discussed in [CR96]. The application to switching SSMs is discussed in [AK77, CL00, DGK99]. The unscented particle filter is described in [vdMDdFW00]. Likelihood weighting is described in [FC89, SP89], arc reversal in [Sha86], and the idea of combining them in [FC89, KKR95, CB97]. Using a junction tree for sampling is described in [Daw92]. Combining particle filtering and BK is described in [NPP02].

Multi-object tracking is discussed in [BSF88, BS90, CH96]. SLAM is discussed in e.g., [DNCDW01]. The online model selection example is based on [AdFD00].

Particle smoothing is discussed in [GDW00, IB98]. Expectation propagation was introduced in [Min01]. The application to switching SSMs is from [HZ02]; the iterated BK algorithm (another instance of expectation propagation) is discussed in [MW01]. The structured variational approximation for switching SSMs is from [GH98]; [PRM00] discuss a similar approximation for the case where the dynamics switch. The Rao-Blackwellised Gibbs sampler for switching SSMs is from [CK96]. [KN98] is a whole book devoted to deterministic and MCMC methods for switching SSMs.

Buried Markov models are discussed in [Bil00]. Pseudo-2D HMMs are discussed in [KA94, NI00, Hoe01], and real 2D HMMs are discussed in [LNG00]. The representation of an HHMM as a DBN was introduced in [MP01]. The fault diagnosis example is from [KL01]. The generalized $\alpha - \beta - \gamma$ algorithm is from [Mur02]. The log-space smoothing algorithm was introduced in in [BMR97] and applied to a speech recognition task in [ZP00]. The $O(1)$-space, $O(T^2)$-time smoothing algorithm is classical, but can also be found in [Dar01]. The discussion of $O(1)$-space, $O(T)$-time smoothing is based in part on [RN02].

References

[AdFD00] C. Andrieu, N. de Freitas, and A. Doucet. Sequential Bayesian estimation and model selection for dynamic kernel machines. Technical report, Cambridge Univ., 2000.

[AK77] H. Akashi and H. Kumamoto. Random sampling approach to state estimation in switching environments. *Automatica*, 13:429–434, 1977.

[AM79] B. Anderson and J. Moore. *Optimal Filtering*. Prentice-Hall, 1979.

[AMGC02] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking. *IEEE Trans. on Signal Processing*, 50(2):174–189, February 2002.

[Bil00] J. Bilmes. Dynamic Bayesian multinets. In *UAI*, 2000.

[Bil01] J. A. Bilmes. Graphical models and automatic speech recognition. Technical Report UWEETR-2001-0005, Univ. Washington, Dept. of Elec. Eng., 2001.

[BK98a] X. Boyen and D. Koller. Approximate learning of dynamic models. In *NIPS-11*, 1998.

[BK98b] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *UAI*, 1998.

[BMR97] J. Binder, K. Murphy, and S. Russell. Space-efficient inference in dynamic probabilistic networks. In *IJCAI*, 1997.

[Bra96] M. Brand. Coupled hidden Markov models for modeling interacting processes. Technical Report 405, MIT Lab for Perceptual Computing, 1996.

[BS90] Y. Bar-Shalom, editor. *Multitarget-multisensor tracking : advanced applications*. Artech House, 1990.

[BSF88] Y. Bar-Shalom and T. Fortmann. *Tracking and data association*. Academic Press, 1988.

[BSL93] Y. Bar-Shalom and X. Li. *Estimation and Tracking: Principles, Techniques and Software*. Artech House, 1993.

[BVW01] H. Bui, S. Venkatesh, and G. West. Tracking and surveillance in wide-area spatial environments using the Abstract Hidden Markov Model. *Intl. J. of Pattern Rec. and AI*, 2001.

[BZ02] J. Bilmes and G. Zweig. The graphical models toolkit: An open source software system for speech and time-series processing. In *Intl. Conf. on Acoustics, Speech and Signal Proc.*, 2002.

[CB97] A. Y. W. Cheuk and C. Boutilier. Structured arc reversal and simulation of dynamic probabilistic networks. In *UAI*, 1997.

[CH96] I.J. Cox and S.L. Hingorani. An Efficient Implementation of Reid's Multiple Hypothesis Tracking Algorithm and its Evaluation for the Purpose of Visual Tracking. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 18(2):138–150, 1996.

[CK96] C. Carter and R. Kohn. Markov chain Monte Carlo in conditionally Gaussian state space models. *Biometrika*, 83:589–601, 1996.

[CL00] R. Chen and S. Liu. Mixture Kalman filters. *J. Royal Stat. Soc. B*, 2000.

[CR96] G Casella and C P Robert. Rao-Blackwellisation of sampling schemes. *Biometrika*, 83(1):81–94, 1996.

[Dar01] A. Darwiche. Constant Space Reasoning in Dynamic Bayesian Networks. *Intl. J. of Approximate Reasoning*, 26:161–178, 2001.

[Daw92] A. P. Dawid. Applications of a general propagation algorithm for probabilistic expert systems. *Statistics and Computing*, 2:25–36, 1992.

[DdFG01] A. Doucet, N. de Freitas, and N. J. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer Verlag, 2001.

[DE00] R. Dahlhaus and M. Eichler. Causality and graphical models for time series. In P. Green, N. Hjort, and S. Richardson, editors, *Highly structured stochastic systems*. Oxford University Press, 2000.

[DGK99] A. Doucet, N. Gordon, and V. Krishnamurthy. Particle Filters for State Estimation of Jump Markov Linear Systems. Technical report, Cambridge Univ. Engineering Dept., 1999.

[DK89] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Artificial Intelligence*, 93(1–2):1–27, 1989.

[DK01] J. Durbin and S. J. Koopman. *Time Series Analysis by State Space Methods*. Oxford University Press, 2001.

[DNCDW01] M. G. Dissanayake, P. Newman, S. Clark, and H. F. Durrant-Whyte. A Solution to the Simultaneous Localization and Map Building (SLAM) Problem. *IEEE Trans. Robotics and Automation*, 17(3):229–241, 2001.

[Dou98] A Doucet. On sequential simulation-based methods for Bayesian filtering. Technical report CUED/F-INFENG/TR 310, Department of Engineering, Cambridge University, 1998.

[FC89] R. Fung and K. Chang. Weighting and integrating evidence for stochastic simulation in Bayesian networks. In *UAI*, 1989.

[FST98] S. Fine, Y. Singer, and N. Tishby. The hierarchical Hidden Markov Model: Analysis and applications. *Machine Learning*, 32:41, 1998.

[Gal99] M. J. F. Gales. Semi-tied covariance matrices for hidden Markov models. *IEEE Trans. on Speech and Audio Processing*, 7(3):272–281, 1999.

[GDW00] S. Godsill, A. Doucet, and M. West. Methodology for Monte Carlo Smoothing with Application to Time-Varying Autoregressions. In *Proc. Intl. Symp. on Frontiers of Time Series Modelling*, 2000.

[GH98] Z. Ghahramani and G. Hinton. Variational learning for switching state-space models. *Neural Computation*, 12(4):963–996, 1998.

[GJ97] Z. Ghahramani and M. Jordan. Factorial hidden Markov models. *Machine Learning*, 29:245–273, 1997.

[Goo01] J. Goodman. A bit of progress in language modelling. *Computer Speech and Language*, pages 403–434, October 2001.

[Gre98] P. Green. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82:711–732, 1998.

[Hoe01] J. Hoey. Hierarchical unsupervised learning of facial expression categories. In *ICCV Workshop on Detection and Recognition of Events in Video*, 2001.

[HZ02] T. Heskes and O. Zoeter. Expectation propagation for approximate inference in dynamic Bayesian networks. In *UAI*, 2002.

[IB98] M. Isard and A. Blake. A smoothing filter for condensation. In *Proc. European Conf. on Computer Vision*, volume 1, pages 767–781, 1998.

[Jel97] F. Jelinek. *Statistical methods for speech recognition*. MIT Press, 1997.

[JU97] S. Julier and J. Uhlmann. A new extension of the Kalman filter to nonlinear systems. In *Proc. of AeroSense: The 11th Intl. Symp. on Aerospace/Defence Sensing, Simulation and Controls*, 1997.

[KA94] S. Kuo and O. Agazzi. Keyword spotting in poorly printed documents using pseudo 2-D Hidden Markov Models. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 16:842–848, 1994.

[KCB00] J. Kwon, B. Coifman, and P. Bickel. Day-to-day travel time trends and travel time prediction from loop detector data. *Transportation Research Record*, 1554, 2000.

[Kim94] C-J. Kim. Dynamic linear models with Markov-switching. *J. of Econometrics*, 60:1–22, 1994.

[Kja92] U. Kjaerulff. A computational scheme for reasoning in dynamic probabilistic networks. In *UAI-8*, 1992.

[Kja95] U. Kjaerulff. dHugin: A computational system for dynamic time-sliced Bayesian networks. *Intl. J. of Forecasting*, 11:89–111, 1995.

[KKR95] K. Kanazawa, D. Koller, and S. Russell. Stochastic simulation algorithms for dynamic probabilistic networks. In *UAI*, 1995.

[KL01] D. Koller and U. Lerner. Sampling in Factored Dynamic Systems. In A. Doucet, N. de Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

[KN98] C-J. Kim and C. Nelson. *State-Space Models with Regime-Switching: Classical and Gibbs-Sampling Approaches with Applications*. MIT Press, 1998.

[LC98] J. Liu and R. Chen. Sequential Monte Carlo methods for dynamic systems. *JASA*, 93:1032–1044, 1998.

[LNG00] J. Li, Amir Najmi, and Robert M. Gray. Image classification by a two-dimensional hidden markov model. *IEEE Transactions on Signal Processing*, 48(2):517–533, February 2000. citeseer.nj.nec.com/article/li98image.html.

[LP01] U. Lerner and R. Parr. Inference in hybrid networks: Theoretical limits and practical algorithms. In *UAI*, 2001.

[McC95] A. McCallum. *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, Univ. Rochester, 1995.

[MCH02] Christopher Meek, David Maxwell Chickering, and David Heckerman. Autoregressive tree models for time-series analysis. In *Proceedings of the Second International SIAM Conference on Data Mining*, pages 229–244, Arlington, VA, April 2002. SIAM.

[Min01] T. Minka. *A family of algorithms for approximate Bayesian inference*. PhD thesis, MIT, 2001.

[Min02] T. Minka. Bayesian inference in dynamic models: an overview. Technical report, CMU, 2002.

[Moh96] M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 20(1):1–33, 1996.

[MP01] K. Murphy and M. Paskin. Linear time inference in hierarchical HMMs. In *NIPS*, 2001.

[MTKW02] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. In *AAAI*, 2002.

[Mur98a] K. P. Murphy. Filtering and smoothing in linear dynamical systems using the junction tree algorithm. Technical report, U.C. Berkeley, Dept. Comp. Sci, 1998.

[Mur98b] K. P. Murphy. Switching Kalman filters. Technical report, DEC/Compaq Cambridge Research Labs, 1998.

[Mur00] K. P. Murphy. Bayesian map learning in dynamic environments. In *NIPS-12*, pages 1015–1021, 2000.

[Mur02] K. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, Dept. Computer Science, UC Berkeley, 2002.

[MW01] K. Murphy and Y. Weiss. The Factored Frontier Algorithm for Approximate Inference in DBNs. In *UAI*, 2001.

[NG01] D. Nilsson and J. Goldberger. Sequentially finding the N-Best List in Hidden Markov Models. In *IJCAI*, pages 1280–1285, 2001.

[NI00] A. Nefian and M. Hayes III. Maximum likelihood training of the embedded HMM for face detection and recognition. In *IEEE Intl. Conf. on Image Processing*, 2000.

[Nil98] D. Nilsson. An efficient algorithm for finding the M most probable configurations in a probabilistic expert system. *Statistics and Computing*, 8:159–173, 1998.

[NLP$^+$02] A. Nefian, L. Liang, X. Pi, X. Liu, and K. Murphy. Dynamic Bayesian Networks for Audio-Visual Speech Recognition. *J. Applied Signal Processing*, 2002.

[NPP02] B. Ng, L. Peshkin, and A. Pfeffer. Factored particles for scalable monitoring. In *UAI*, 2002.

[NT92] S. Normand and D. Trichtler. Parameter updating in a bayes network. *JASA*, 87(420):1109–1115, 1992.

[NY00] H. J. Nock and S. J. Young. Loosely coupled HMMs for ASR. In *Intl. Conf. on Acoustics, Speech and Signal Proc.*, 2000.

[ODK96] M. Ostendorf, V. Digalakis, and O. Kimball. From HMMs to segment models: a unified view of stochastic modeling for speech recognition. *IEEE Trans. on Speech and Audio Processing*, 4(5):360–378, 1996.

[PR97] F. Pereira and M. Riley. Speech recognition by composition of weighted finite automata. In *Finite-State Language Processing*, pages 431–453. MIT Press, 1997.

[PRM00] V. Pavlovic, J. M. Rehg, and J. MacCormick. Learning switching linear models of human motion. In *NIPS-13*, 2000.

[Rab89] L. R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proc. of the IEEE*, 77(2):257–286, 1989.

[RN02] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2002. 2nd edition, in preparation.

[RST96] Dana Ron, Yoram Singer, and Naftali Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, 25, 1996.

[RTL76] D. Rose, R. Tarjan, and G. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. on Computing*, 5:266–283, 1976.

[Sha86] R. Shachter. Evaluating influence diagrams. *Operations Research*, 33(6):871–882, 1986.

[SJ95] L. Saul and M. Jordan. Boltzmann chains and hidden Markov models. In *NIPS-7*, 1995.

[SJ99] L. Saul and M. Jordan. Mixed memory markov models: Decomposing complex stochastic processes as mixture of simpler ones. *Machine Learning*, 37(1):75–87, 1999.

[SM80] A. Smith and U. Makov. Bayesian detection and estimation of jumps in linear systems. In O. Jacobs, M. Davis, M. Dempster, C. Harris, and P. Parks, editors, *Analysis and optimization of stochastic systems*. 1980.

[SP89] R. D. Shachter and M. A. Peot. Simulation approaches to general probabilistic inference on belief networks. In *UAI*, volume 5, 1989.

[TDW02] M. Takikawa, B. D'Ambrosio, and E. Wright. Real-time inference with large-scale temporal bayes nets. In *UAI*, 2002.

[TRM01] G. Theocharous, K. Rohanimanesh, and S. Mahadevan. Learning Hierarchical Partially Observed Markov Decision Process Models for Robot Navigation. In *ICRA*, 2001.

[TSM85] D. M. Titterington, A. F. M. Smith, and U. E. Makov. *Statistical analysis of finite mixture distributions*. Wiley, 1985.

[vdMDdFW00] R. van der Merwe, A. Doucet, N. de Freitas, and E. Wan. The unscented particle filter. In *NIPS-13*, 2000.

[WdM01] E. A. Wan and R. Van der Merwe. The Unscented Kalman Filter. In S. Haykin, editor, *Kalman Filtering and Neural Networks*. Wiley, 2001.

[WH97] Mike West and Jeff Harrison. *Bayesian forecasting and dynamic models*. Springer, 1997.

[ZP00] G. Zweig and M. Padmanabhan. Exact alpha-beta computation in logarithmic space with application to map word graph construction. In *Proc. Intl. Conf. Spoken Lang. Proc.*, 2000.

[Zwe96] G. Zweig. A forward-backward algorithm for inference in Bayesian networks and an empirical comparison with HMMs. Master's thesis, Dept. Comp. Sci., U.C. Berkeley, 1996.