Hierarchical HMMs

Kevin P. Murphy
Computer Science Department
University of California
Berkeley, CA 94720-1776
murphyk@cs.berkeley.edu

23 November 2001
Last updated: 14 March 2002

Abstract

The hierarchical hidden Markov model (HHMM) is a generalization of the hidden Markov model (HMM) that models sequences with structure at many length/time scales (Fine, Singer & Tishby, 1998). Unfortunately, the original inference algorithm is rather complicated, and takes $O(T^3)$ time, where T is the length of the sequence, making it impractical for many domains. In this paper, we show how HHMMs are a special kind of dynamic Bayesian network (DBN), and thereby derive a much simpler inference algorithm, which only takes $O(T)$ time. We explain how to do parameter and structure learning, and discuss related models and applications.

1 Introduction

Many domains naturally have structure at many length/time scales, e.g., natural language, XML, DNA sequences [HIM+00], handwriting [FST98], maps of the environment [TRM01, BVW01], etc. The need for hierarchy in order to learn long distance dependencies was discussed in [EB96]; the need for hierarchy in order to efficiently solve planning problems has also been recognized for a long time, both in classical AI planning and in the reinforcement learning/MDP community. Also, for some problems (e.g., plan recognition [BVW00a] and visual action recogntion [IB00, ME01]), we want to represent the state of the world at multiple levels of abstraction.

The hierarchical hidden Markov model (HHMM) [FST98] is a generalization of the hidden Markov model (HMM) that has hierarchical structure. HHMMs are less expressive than stochastic context free grammars (SCFGs), since they only allows hierarchies of bounded depth, but they are more efficient and easier to learn. Unfortunately, the original inference algorithm described in [FST98] is somewhat complicated, and takes $O(T^3 Q^D)$ time, where T is the length of the sequence, D is the depth of the hierarchy, and Q is the (maximum) number of states at each level of the hierarchy. In this paper, we show how to represent an HHMM as a dynamic Bayesian network (DBN), and thereby derive a much simpler and faster inference algorithm[1], which takes at most $O(TQ^{2D})$ time; empirically, we fine it takes only $O(TDQ^{\lceil 1.5D \rceil})$ time. Furthermore, by drawing the connection between HHMMs and DBNs, we enable the application of approximate inference techniques such as belief propagation, which can perform inference in $O(TD^2Q^2)$ time.

We will describe HHMMs in Section 2. The original $O(T^3)$ inference algorithm was derived by analogy with the Inside-Outside algorithm for SCFGs; this will briefly be explained in Section 3. The main contribution of the paper is in Section 4, where we show how to represent an HHMM as a DBN. In Section 5, we discuss how to do efficient inference in this DBN. In Section 6, we discuss how to estimate the parameters of an HHMM, and briefly discuss how to do structure learning. In Section 7, we discuss related work and applications.

[1] By inference, we mean offline smoothing, i.e., conditioning on a fixed-length observation sequence. This is needed as a subroutine for EM. Once the model has been learned, it will typically be used for online inference (filtering). The [FST98] paper did not consider this, but it is straightforward in the DBN formulation.

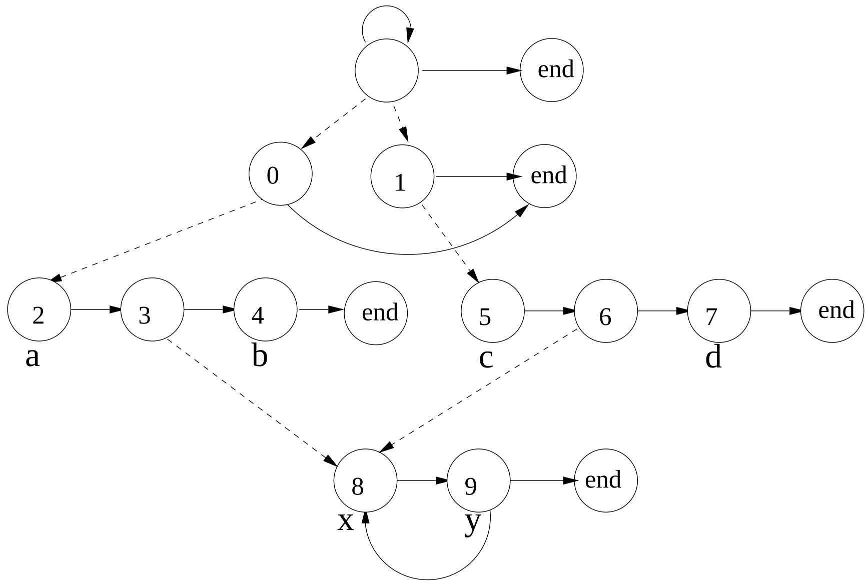


Figure 1: A 3-level hierarchical FSA representing the regular expression $a(xy)^+b|c(xy)^+d$. Solid lines represent horizontal transitions, dotted lines represent vertical transitions. Letters below a production state represent the symbol that is emitted. The unnumbered root node is considered level 0, and could be omitted if we fully interconnected states 0 and 1.

2 Hierarchical HMMs

2.1 HMMs

HMMs model sequential data using a stochastic finite state automaton (FSA) whose state is hidden. The hidden state is represented in terms of a single discrete random variable, Q_t, which stochastically generates (emits) an observation, O_t, at each time step t, and then undergoes a transition to a new state determined by the transition matrix of a (stationary) Markov chain, $A(i,j) = P(Q_t = j|Q_{t-1} = i)$. (For simplicity of exposition, we shall assume all observations are discrete symbols, but (H)HMMs can easily be generalized to handle continuous observations, as we discuss in Sections 4.7 and 4.8.)

2.2 HHMMs

In an HHMM, the states of the stochastic automaton can emit single observations or strings of observations. Those that emit single symbols are called "production states", and those that emit strings are termed "abstract states". The strings emitted by abstract states are themselves governed by sub-HHMMs, which can be called recursively. When the sub-HHMM is finished, control is returned to wherever it was called from; the calling context is memorized using a depth-limited stack.

We illustrate the generative process with Figure 1, which shows the state transition diagram of an example HHMM which models the regular expression $a(xy)^+b|c(xy)^+d)$. We start in the root state. Since this is an abstract state, it calls its (unique) sub-HMM; it can enter via state 0 or state 1; this is called a "vertical transition". Suppose it enters state 0, which is itself abstract. State 0 then enters state 2. Since this is a production state, it emits the symbol "a". It is then forced to make a horizontal transition to state 3. Since this is an abstract state, it calls the HMM at level 3. In state 8, we emit "x", make a horizontal transition to state 9, and then emit "y". Suppose at this point we make a horizontal transition to the end state at level 3. This returns control to wherever we were called from — in this case, state 3. State 3 then makes a horizontal transition to state 4, which emits "b". State 4 is then forced to make a horizontal transition to the end state, which returns control to state 0. State 0 then enters its end state, and returns control to the

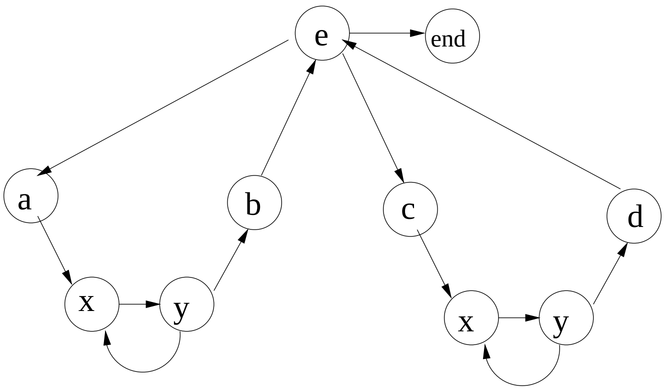


Figure 2: A flattened FSA equivalent to Figure 1. "e" represents the empty string (a silent emission). Note how abstract states have been "compiled out", but shared substructure had to be copied.

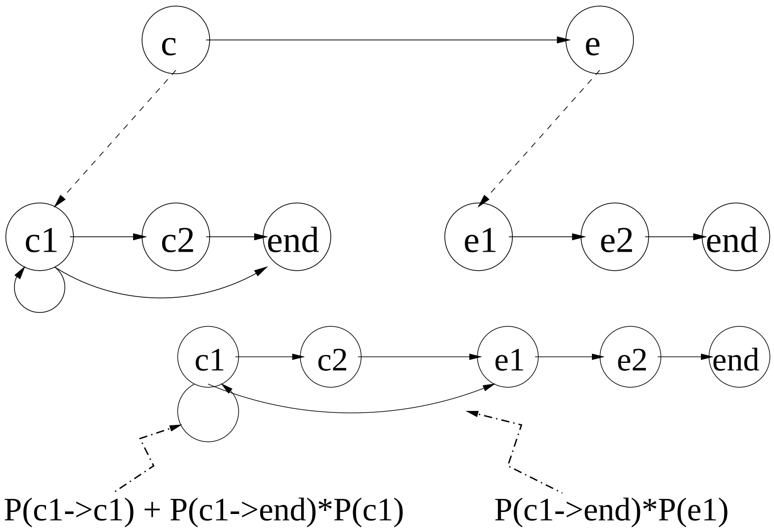


Figure 3: (a) An HHMM and (b) its flat HMM equivalent.

root. The root can then either re-enter level 1, or enter its end state, which terminates the process.[2]

Note that an abstract state can only "call" a single sub-HMM, although it may enter this sub-HMM via any state. (This is why the root in Figure 1 cannot directly call the "ab" model or the "cd" model, but must go via states 0 or 1.) However, a given sub-HMM may be called from many different parent states.

2.3 Converting an HHMM to an HMM

Any HHMM can be converted to an HMM by creating an HMM state for every possible HHMM stack configuration $Q_t^{1:D}$: see Figure 2 for an example. The resulting state-space may be smaller, because it does not contain abstract states, but may also be larger, because shared sub-structure (such as the sub-expression $(xy)^+$) must be duplicated.

Computing the parameters of the resulting flat HMM is not always trivial. The probability of an i to j transition in the flat model is the sum over paths from i to j in the hierarchical model which only pass through abstract (non-emitting) states. For example, in Figure 3, the probability of a self-transition from state c_1 to c_1 is given by

$$P(c_1 \to c_1) + P(c_1 \to \text{end} \to c \to c_1) = A^c(c_1, c_1) + A^c(c_1, \text{end}) \times \pi^c(c_1)$$

[2]Code for this example is available in BNT. See `BNT/examples/dynamic/HHMM/abcd_hhmm`.

Although it is always possible to convert an HHMM to an HMM, there are several disadvantages:

- A flat HMM cannot provide a multi-scale intrepretation of the data.
- Flattening loses modularity, since the parameters of the sub-HMMs get combined in a complex way.
- Training HMMs separately and combining requires segmented data.
- A flat HMM may have more parameters. It is the ability to reuse sub-models in different contexts that makes HHMMs more powerful than standard HMMs, since the parameters of such shared sub-models only need to be learned once.

3 Cubic-time inference

The inference algorithm for HHMMs presented in [FST98] runs in $O(T^3)$ time and is based on the Inside-Outside algorithm [LY90], an exact inference algorithm for stochastic context-free grammars (SCFGs) which we now describe.

In an SCFG, sequences of observations are generated by a set of stochastic production rules. Each production rule stochastically rewrites a non-terminal symbol N^i into either a symbol of the alphabet ($N^i \xrightarrow{p} a$) or a pair of nonterminal symbols ($N^i \xrightarrow{p} N^j N^k$). Observation strings are generated by starting with the distinguished "start" nonterminal N^0, and continually re-writing all non-terminals using stochastic production rules until, finally, only symbols of the alphabet remain.

The Inside-Outside algorithm computes $P(N^i \to N^j N^k | O_{t:t+k})$, where $O_{t:t+k} = O_t, O_{t+1}, \ldots, O_{t+k}$ is a subsequence. This can then be used to compute the expected sufficient statistics needed by the EM algorithm to learn the parameters of the model.

If there are N non-terminals in the grammar and the training sequence is of length T, then the Inside-Outside algorithm requires $O(N^3 T^3)$ time. To see why, note that we must compute $P(N^i \to N^j N^k | O_{t:t+\tau})$ for all end points t and $t + \tau$, and for all midpoints τ' such that N^j generates $O_{t:t+\tau'}$ and N^k generates $O_{t+\tau'+1:t+\tau}$ — the three degrees for freedom t,τ and τ' gives rise to the T^3 term. The N^3 term arises because we must consider all N^i, N^j and N^k.

The inference algorithm for HHMMs presented in [FST98] is based upon a straightforward adaptation of the Inside-Outside algorithm. The algorithm computes $P(\text{in state } q_i^d \text{ at time } t | O_{t:t+\tau})$ by assuming that sub-state q_j^{d+1} generates $O_{t:t+\tau'}$, that a transition to state k occurs, and that q_k^{d+1} generates $O_{t+\tau'+1:t+\tau}$. Hence the algorithm takes $O(NT^3)$ time, where N is the total number of states.

We can always "flatten" an HHMM into a regular HMM (possibly with extra states), and hence do inference in $O(N^2 T)$. Unfortunately, this flat model does not represent the original hierarchical structure, and therefore is certainly incapable of learning it. Hence it is not possible to use inference in the flat model to compute the expected sufficient statistics needed by EM for parameter estimation in the original, hierarchical model, since some of the relevant quantities have been marginalized out (see Section 6). In the next section, we show how to represent the HHMM as a DBN, and thereby get the best of both worlds: low time complexity without losing hierarchical structure.

4 Representing the HHMM as a DBN

We can represent the HHMM as a dynamic Bayesian network (DBN) as shown in Figure 4. (We assume for simplicity that all production states are at the bottom of the hierarchy; this restriction is lifted in Section 4.9.) The state of the HMM at level d and time t is represented by Q_t^d. The state of the whole HHMM is encoded by the vector $\vec{Q}_t = (Q_t^1, \ldots, Q_t^D)$; intuitively, this encodes the contents of the stack, that specifies the complete "path" to take from the root to the leaf state.

F_t^d is an indicator variable that is "on" if the HMM at level d and time t has just "finished" (i.e., is about to enter an end state), otherwise it is off. Note that if $F_t^d = 1$, then $F_t^{d'} = 1$ for all $d' > d$; hence the number of F nodes that are "off" represents the effective height of the "context stack", i.e., which level of the hierarchy we are currently on.

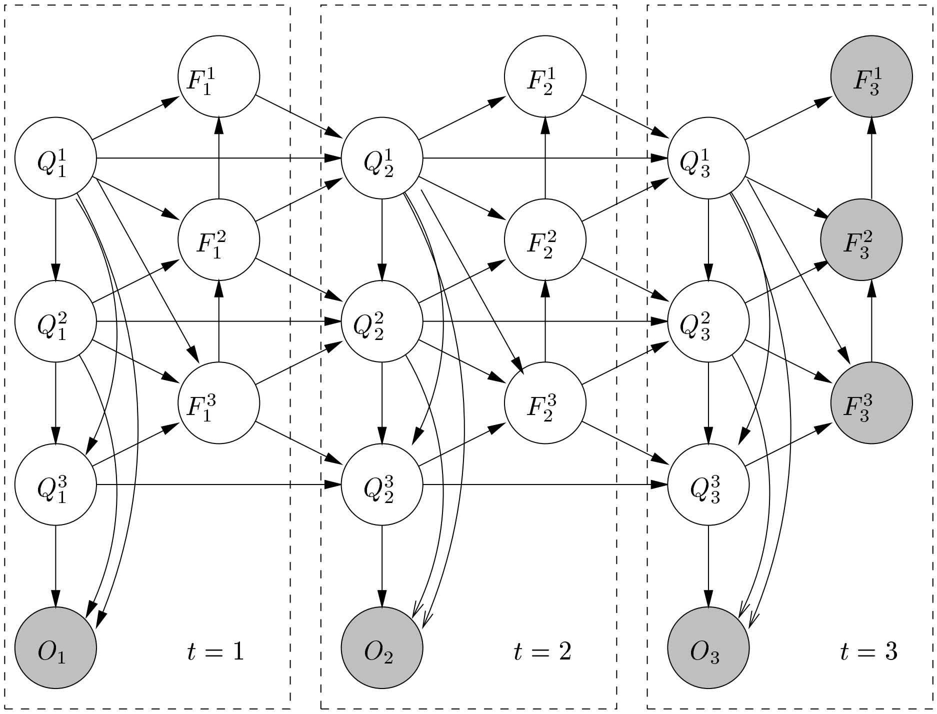


Figure 4: An HHMM represented as a DBN. Q_t^d is the state at time t, level d; $F_t^d = 1$ if the HMM at level d has finished (entered its exit state), otherwise $F_t^d = 0$. Shaded nodes are observed; the remaining nodes are hidden. We may optionally clamp $F_T^d = 1$, where T is the length of the observation sequence, to ensure all models have finished by the end of the sequence.

The downward going arcs between the Q variables represent the fact that a state "calls" a sub-state. The upward going arcs between the F variables enforce the fact that a higher-level HMM can only change state when the lower-level one is finished. This ensures proper nesting of the parse trees, and is the key difference between an HHMM and a hidden Markov decision tree [JGS96].

We will define the conditional probability distributions (CPDs) of each of the node types below, which will complete the definition of the model. 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.

4.1 Bottom level: $d = D$, $t = 2 : T - 1$

Consider the bottom level of the hierarchy. Q^D follows a Markov chain with parameters determined by its position in the automaton, which is encoded by the vector of higher-up state variables $Q_t^{1:D-1} = Q_t^1, \ldots, Q_t^{D-1}$, which we will represent by the integer k.[3] When Q^D enters its end state, it will "turn on" F^D, to mean it is finished; this will be a signal that higher-level HMMs can now change state. In addition, it will be a signal that the next value of Q^D should be drawn from its prior distribution (representing a vertical transition), instead of its transition matrix (representing a horizontal transition). Formally, we can write this as follows:

$$P(Q_t^D = j | Q_{t-1}^D = i, F_{t-1}^D = f, Q_t^{1:D-1} = k) = \begin{cases} \tilde{A}_k^D(i,j) & \text{if } f = 0 \\ \pi_k^D(j) & \text{if } f = 1 \end{cases}$$

where we have assumed $i, j \neq \text{end}$. A_k^D is the transition matrix for level D given that the parent variables are in state k, and $\tilde{A}_k^D$ is just a rescaled version (see Section 4.6 for details on the rescaling). Similarly, π_k^D is the initial distribution for level D given that the parent variables are in state k. The equation for F_D is

[3] If the topology is sparse, this distribution will be 0 for many values of k. This will be discussed in Section 4.8.

simply

$$P(F_t^D = 1 | Q_t^{1:D-1} = k, Q_t^D = i) = A_k^D(i, \mathrm{end}).$$

4.2 Intermediate levels: $d = 2 : D-1$, $t = 2 : T-1$

As before, Q^d follows a Markov chain with parameters determined by $Q^{1:d-1}$, and F^d specifies whether we should use the transition matrix or the prior. The difference is that we now also get a signal from below, F^{d+1}, specifying whether the sub-model has finished or not; if it has, we are free to change state, otherwise we must remain in the same state. Formally, we can write this as follows:

$$P(Q_t^d = j | Q_{t-1}^d = i, F_{t-1}^{d+1} = b, F_{t-1}^d = f, Q_t^{1:d-1} = k) = \begin{cases} \delta(i,j) & \text{if } b = 0 \\ \tilde{A}_k^d(i,j) & \text{if } b = 1 \text{ and } f = 0 \\ \pi_k^d(j) & \text{if } b = 1 \text{ and } f = 1 \end{cases} \quad (1)$$

F^d should "turn on" only if Q^d is "allowed" to enter a final state, the probability of which depends on the current context $Q^{1:d-1}$. Formally, we can write this as follows:

$$P(F_t^d = 1 | Q_t^d = i, Q_t^{1:d-1} = k, F_t^{d+1} = b) = \begin{cases} 0 & \text{if } b = 0 \\ A_k^d(i, \mathrm{end}) & \text{if } b = 1 \end{cases} \quad (2)$$

4.3 Top level: $d = 1$, $t = 2 : T-1$

The top level differs from the intermediate levels in that the Q node has no Q parent to specify which distribution to use. The equations are the same as above, except we eliminate the conditioning on $Q_t^{1:d-1} = k$. (Equivalently, we can imagine a dummy top layer HMM, which is always in state 1: $Q_t^0 = 1$. This is often how HHMMs are represented, so that this top-level state is the root of the overall parse tree, as in Figure 1.)

4.4 Final slice: $t = T$, $d = 1 : D$

To ensure that all sub-HMMs have reached their end states by the time we reach the end of sequence, we can clamp $F_T^d = 1$ for all d. (A similar trick was used in [Zwe97].) This is not strictly necessary, but provides useful extra information.

4.5 Initial slice: $t = 1$, $d = 1 : D$

The CPDs for the nodes in the first slice are as follows: $P(Q_1^1 = j) = \pi^1(j)$ for the top level and $P(Q_1^d = j | Q_1^{1:d-1} = k) = \pi_k^d(j)$, for $d = 2, \ldots, D$.

4.6 Handling end states

Unlike the automaton representation, the DBN never actually enters an end state (i.e., Q_t^d can never taken on the value "end"), because if it did, it would not be able to emit the symbol O_t. Instead, Q_t^d causes F_t^d to turn on, and then enters a new (non-terminal) state at time $t+1$. This means that the DBN and HHMM transition matrices are not identical. However, they satisfy the following relation

$$\tilde{A}_k^d(i,j)\left(1 - \tau_k^d(i)\right) = A_k^d(i,j)$$

where A represents the automaton transition matrix, $\tilde{A}$ represents the DBN transition matrix, and $\tau_k^d(i) \stackrel{\text{def}}{=} A_k^d(i, \mathrm{end})$ is the probability of terminating from state i. The reason is that the probability of each horizontal transition in the DBN gets multiplied by the probability that $F_t^d = 0$, which is $1 - \tau_k^d(i)$; this product should match the original probability. It is easy to see that the new matrix is stochastic (i.e., $\sum_{j=1}^Q \tilde{A}_k^d(i,j) = 1$), since the original matrix satisfies $\sum_{j=1}^Q A_k^d(i,j) + A_k^d(i, \mathrm{end}) = 1$.

4.7 Observations

If the observations are discrete symbols, we may represent $P(O_t|\vec{Q}_t)$ as a multinomial (i.e., using a table), where $\vec{Q}_t = (Q_t^1, \ldots, Q_t^D)$. If the observations are real-valued vectors, we can use a Gaussian for each value of $\vec{Q}_t$. We discuss more parsimonious representations below. Alternatively, we can violate the Markov property, and only condition O_t on the bottom of the hierarchy, Q_t^D.

4.8 Parsimonious representations of the CPDs

The number of parameters needed to represent $P(Q_t^d|Q_{t-1}^d, Q_t^{1:d-1} = k)$ as a multinomial is $O(Q^{d+1})$. If the state-transition diagram of the hierarchical automaton is sparse, many of the entries in this table will be 0. For instance, in Figure 1, there is only one well-defined transition matrix at level 3, which is shared by contexts (values) $k = 3$ and $k = 6$. Hence we can represent this CPD more parsimoniously using a decision tree [BFGK96]. The number of parameters needed to represent the tree can vary from $O(Q)$ to $O(Q^{d+1})$ if the tree is full. We can learn this CPD using a decision-tree induction algorithm. Unfortunately, it is difficult to exploit tree-structured CPDs to speed up inference.

An alternative is to represent this distribution as a mixture of smaller transition matrices at different depths. (The same trick is often to represent the transition matrices of higher order Markov chains; see e.g., [SJ99].) More specifically, let $R_t^k = Q_t^k$ if $k < d$ and $R_t^k = Q_{t-1}^d$ if $k = d$, and let Z_t^d be a latent variable which specifies which of the R variables will actually determine the output: $P(Q_t^d|R_t^{1:d}, Z_t = k) = P(Q_t^d|R_t^k)$. Thus the CPD acts like a multiplexer. This requires dQ^2 parameters to represent. We can learn this CPD using EM. Furthermore, if we use loopy belief propagation to do *approximate* inference, we can exploit the fact that the CPD is a multiplexer to compute the required messages efficiently [Mur99]: see Section 5.

A third possibility is to represent $P(Q_t^d|Q_t^{1:d-1}, Q_{t-1}^d)$ using the softmax function. Since the inputs are multi-valued categorical nodes, we need to use a distributed 1-of-Q representation for each one (see e.g., [Pfl98] for details). Hence we require dQ^2 parameters overall. We can learn this CPD using IRLS, although it is hard to use this parsimonious structure to speed up inference.

If the observation nodes are discrete, we can use any of the above methods to represent $P(O_t|Q_t^{1:D})$. Trees are a particularly good choice, since they can exploit the fact that variables can have different numbers of values depending on the level and on the value of their parents. If the size of the alphabet is too large (e.g., if O_t represents an English word), we can use feature-based parameterizations [PPL97].

If the observation nodes are continuous, we can use a mixture of a linear number of Gaussians as follows [GJ97]: $P(O_t|Q_t^{1:D}) = N(O_t; \mu_t, \Sigma)$, where $\mu_t = \sum_{d=1}^{D} W^d(:, Q_t^d)$. Each W^d is a $Y \times Q$ matrix, and $W^d(:, j)$ represents the j'th column; Y is the size of the output vector. Hence we use YDQ parameters to represent the means, instead of YQ^D. Also, we use a shared covariance matrix, with Y^2 parameters, instead of using a separate one for each instantiation of $Q_t^{1:D}$.

4.9 Allowing production states at different levels

In [FST98], the authors remark that putting all the production (concrete) states at the bottom level did not result in learning "interesting" hierarchical structure (i.e., the results were essentially the same as using a flat HMM). They therefore constructed the automaton topology by hand, and allowed production states at multiple levels of the hierarchy.

So far, the DBN representation has assumed that all the production states are at the bottom level. This can always be made the case, as illustrated in Figure 5. However, we now develop a way to directly encode in the DBN the fact that concrete states can occur at different leves in the hierarchy. We simply ensure that when Q_t^d enters a concrete state, it causes all the Q nodes below it to enter a special "uninformative" or "null" state. In addition, all the F nodes below level d will be forced on, so that level d is free to change state without waiting for its children to finish. The observation distribution will ignore null states, so the distribution simplifies from $P(O_t|Q_t^{1:D})$ to $P(O_t|Q_t^{1:d})$. It is straightforward to modify the CPD definitions to cause this behavior. We give the details below.

Let $C(k)$ be a binary indicator function that is 1 if state k is concrete, and is 0 otherwise. For example, in Figure 1, $C(2) = 1$ but $C(3) = 0$. At the bottom level, the Q CPD becomes

$$P(Q_t^D = j|Q_{t-1}^D = i, F_{t-1}^D = f, Q_t^{1:D-1} = k)$$

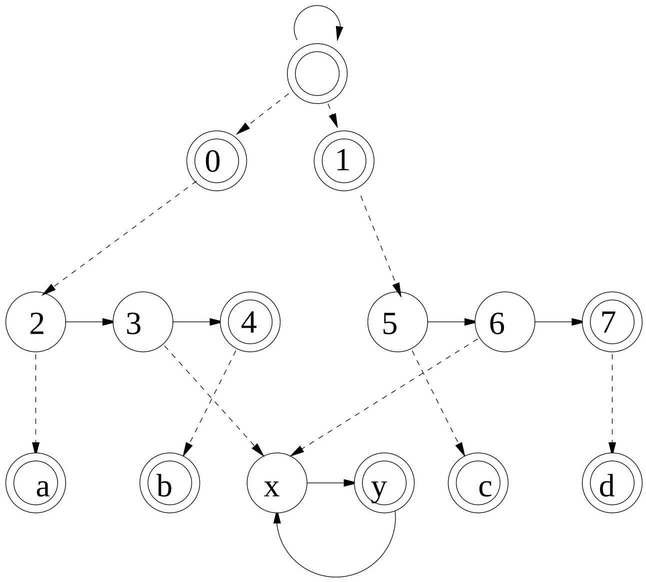


Figure 5: A hierarchical FSA, equivalent to Figure 1, where all the production states are at the bottom of the hierarchy. Double-ringed states are accepting, i.e., they can transition to the end state, which is omitted for clarity. (The state must either make a vertical transition or emit a symbol before moving to the end state.)

$$= \begin{cases} 1.0 & \text{if } j = \text{null and } C(k) = 1 \\ \tilde{A}_k^D(i,j) & \text{if } f = 0 \text{ and } C(k) = 0 \text{ and } j \neq \text{null} \\ \pi_k^D(j) & \text{if } f = 1 \text{ and } C(k) = 0 \text{ and } j \neq \text{null} \end{cases}$$

Since $j = \text{null}$ iff $C(k) = 1$, we do not strictly need to check that $j \neq \text{null}$ if we know $C(k) = 0$. The F CPD becomes

$$P(F_t^D = 1 | Q_t^{1:D-1} = k, Q_t^D = i) = \begin{cases} 1.0 & \text{if } C(k) = 1 \\ A_k^D(i, \text{end}) & \text{otherwise} \end{cases}$$

Alternatively, since we know $Q_t^D = \text{null}$ if $C(Q_t^{1:D-1}) = 1$, we can rewrite this as

$$P(F_t^D = 1 | Q_t^{1:D-1} = k, Q_t^D = i) = \begin{cases} 1.0 & \text{if } i = \text{null} \\ A_k^D(i, \text{end}) & \text{otherwise} \end{cases}$$

At the intermediate level, the Q CPD becomes

$$P(Q_t^d = j | Q_{t-1}^d = i, F_{t-1}^{d+1} = b, F_{t-1}^d = f, Q_t^{1:d-1} = k)$$
$$= \begin{cases} 1.0 & \text{if } j = \text{null and } C(k) = 1 \\ \delta(i,j) & \text{if } b = 0 \text{ and } C(k) = 0 \\ \tilde{A}_k^d(i,j) & \text{if } b = 1 \text{ and } f = 0 \text{ and } C(k) = 0 \\ \pi_k^d(j) & \text{if } b = 1 \text{ and } f = 1 \text{ and } C(k) = 0 \end{cases}$$

This CPD is illustrated in Figure 6. The F CPD becomes

$$P(F_t^d = 1 | Q_t^{1:d-1} = k, Q_t^d = i, F_t^{d+1} = b) = \begin{cases} 1.0 & \text{if } i = \text{null} \\ 0 & \text{if } b = 0 \text{ and } i \neq \text{null} \\ A_k^d(i, \text{end}) & \text{if } b = 1 \text{ and } i \neq \text{null} \end{cases}$$

The Q CPDs for the top level remain the same, since there are no parents above. The F CPDs for the top level remain the same, since the top level can never enter a null state.

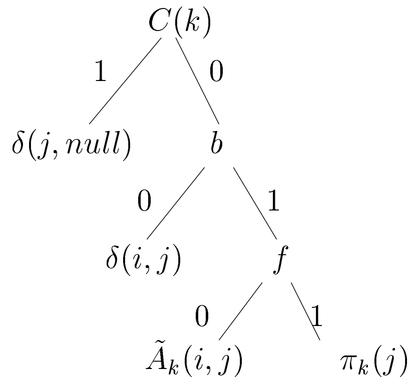


Figure 6: Decision tree representation of $P(Q_t^d = j | Q_{t-1}^d = i, F_{t-1}^{d+1} = b, F_{t-1}^d = f, Q_t^{1:d-1} = k)$.

The above solution is somewhat inefficient since it introduces an extra dummy value, "null". A more elegant solution would be to allow the state spaces for all the levels below d to collapse to a single value when Q_t^d enters a concrete state; if a variable can only be in one state, it conveys no information (and incurs no inferential cost) and can be essentially eliminated from the graph. However, it is not clear how to handle conditionally-sized state-spaces in a DBN. An alternative would be to allow the structure to change conditionally on the values of the Q nodes [Bil00]. Specifically, when Q_t^d enters a concrete state, all nodes $Q_t^{d'}$ and $F_t^{d'}$, for $d' > d$, and their incoming and outgoing links, could be deleted. (Note that knowing when Q_t^d enters a concrete state is equivalent to knowing when the sub-HMM at level d has finished.)

5 Linear-time inference

We define inference to be computing $P(S_i | O_{1:T})$ for all sets of nodes $S_i = \{i\} \cup \text{parents}(i)$ in the DBN. These "family" marginals are needed by EM, as we will see in Section 6. The simplest way to do this is to merge all the hidden nodes in each slice into a single "mega node", M_t, with $M = 2^D Q^D$ possible values. (The 2^D term arises from the binary F nodes.) We can then apply the forwards-backwards algorithm for HMMs, which takes $O(M^2 T)$ time. The resulting family marginals can be computed by marginalizing down the two-slice HMM marginals $\xi_t(i, j) = P(M_{t-1} = i, M_t = j | O_{1:T})$.

Note that converting the DBN in Figure 4 to an HMM is different from converting the original HHMM to an HMM, which "compiles out" the abstract states as discussed in Section 2. Nevertheless, converting the DBN to an HMM in this way will not be tractable for reasonably large Q or D. (Even storing the $M \times M$ transition matrix is likely to consume too much space.) Fortunately, we can do better by exploiting the structure of the model. In [Mur01], we present a way of applying the junction tree (jtree) algorithm to variable-length DBNs, based on a simplification of [Kja95]; we give a brief sketch here.

The algorithm works by performing a forwards-backwards sweep through a chain of jtrees. Each jtree is formed from a "$1\frac{1}{2}$-slice DBN"; this is a DBN that contains all the nodes in slice 1 but only the interface nodes from slice 2. The interface nodes are those nodes in slice 2 that have an incoming temporal arc, plus parents of nodes that have incoming temporal arcs. In the case of an HHMM, the interface is all the Q nodes.

The cost of doing inference in each jtree depends on the sizes of the cliques. Minimizing the maximal clique size is NP-hard, so we used a standard one-step look-ahead (greedy) algorithm [Kja90]. The resulting cliques are hard to interpret, but we can still analyze the complexity. Let $N_Q(c, D)$ be the number of Q nodes in clique c, let $N_F(c, D)$ be the number of F nodes, and let $N_c(D)$ be the number of cliques. Then the cost of inference in a jtree is proportional to

$$\sum_{c=1}^{N_c(D)} Q^{N_Q(c,D)} \times 2^{N_F(c,D)} < N_c(D) \times Q^{\max_c N_Q(c,D)} \times 2^{\max_c N_F(c,D)}$$

Empirically we find that, for a wide range of D, $N_c(D) < D+2$, $\max_c N_F(c, D) \le \lceil 0.5D \rceil$ and $\max_c N_Q(c, D) \le$

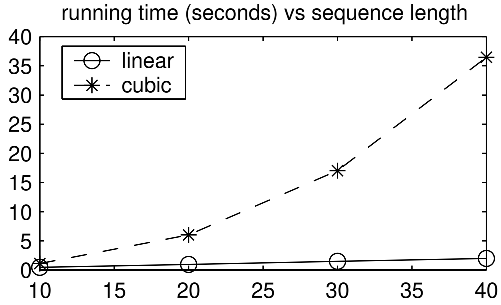


Figure 7: Running time vs. sequence length. Both algorithms were implemented in Matlab. The HHMM is the one in Figure 20.

$1 + \lceil 1.5D \rceil$. Hence a crude upper bound on the cost of inference in each jtree is $O((D+2)Q^{\lceil 1.5D \rceil} 2^{\lceil 0.5D \rceil})$, yielding an overall time and space complexity of $O(TDQ^{\lceil 1.5D \rceil})$. We remind readers that the original algorithm has $O(T^3 Q^D)$ time complexity, since there can be up to $N = Q^D$ states in the HHMM. The advantage of the new algorithm in practice is clearly illustrated in Figure 7.

We can reduce the space complexity from $O(T)$ to $O(\log T)$, at a cost of increasing the time complexity by a $O(\log T)$ factor, by using the "island algorithm" [BMR97]. We can reduce the time (and space) complexity from $O(TDQ^{\lceil 1.5D \rceil})$ to $O(TDQ^D)$ by using *approximate* DBN inference techniques such as the "factored frontier (FF) algorithm" [MW01], which is equivalent to applying "loopy belief propagation" to the DBN using a left-right scheduling of the messages.

The reason that FF takes time exponential in D is not because of the clique size (FF does not create cliques), but because of the high fan-in of the nodes. Specifically, the Q node in layer d has as parents all the Q nodes in the higher-up layers. As discussed in Section 4.8, if we use multiplexer CPDs, we can reduce the cost of computing the messages at each node from $O(Q^{d+1})$ to $O(dQ^2)$ [Mur99], yielding a complexity of $O(\sum_{d=1}^{D} dQ^2) = O(Q^2 D^2)$ per step,

Of course, all of the above algorithms can also be used for online filtering (although FF would require a fixed window length into the past, since it is an iterative algorithm). In addition, by replacing sum with max, we can do Viterbi segmentation in the usual way.

6 Learning

6.1 Parameter learning

We can compute maximum likelihood parameter estimates using EM in the usual way. We give the equations below. We assume there is a single observation sequence, $O_{1:T}$; if there is more than one, we simply sum up the expected sufficient statistics over each sequence. We assume $d > 1$; for $d = 1$, just eliminate the conditioning on $Q_t^{1:d-1} = k$.

For the horizontal transitions, we have

$$\begin{aligned} \xi(t,d,i,k,j) &\stackrel{\text{def}}{=} P(Q_{t-1}^d = i, F_{t-1}^d = 0, Q_t^{1:d-1} = k, Q_t^d = j | O_{1:T}) \\ \hat{\tilde{A}}_k^d(i,j) &= \frac{\sum_{t=2}^{T} \xi(t,d,i,k,j)}{\sum_{j'} \sum_{t=2}^{T} \xi(t,d,i,k,j')} \end{aligned}$$

For a vertical exit, we compute the probability of ending in state i at level d:

$$\begin{aligned} \xi_e(t,d,k,i) &\stackrel{\text{def}}{=} P(Q_t^{1:d-1} = k, Q_t^d = i, F_t^d = 1 | O_{1:T}) \\ \hat{\tau}_k^d(i) &= \frac{\sum_{t=1}^{T} \xi_e(t,d,k,i)}{\sum_{i'} \sum_{t=1}^{T} \xi_e(t,d,k,i')} \end{aligned}$$

For a vertical entrance, we compute the probability of starting in state i at level d:

$$
\begin{aligned}
\chi(t,d,k,i) &\stackrel{\text{def}}{=} P(F_{t-1}^d = 1, Q_t^{1:d-1} = k, Q_t^d = i | O_{1:T}) \\
\chi(1,d,k,i) &\stackrel{\text{def}}{=} P(Q_1^{1:d-1} = k, Q_1^d = i | O_{1:T}) \\
\hat{\pi}_k^d(j) &= \frac{\sum_{t=1}^T \chi(t,d,k,i)}{\sum_{i'} \sum_{t=1}^T \chi(t,d,k,i')}
\end{aligned}
$$

If we model the observation density as a multinomial, which depends on the entire state vector, we can estimate it very easily as follows:

$$
\begin{aligned}
\gamma(t,k,o) &\stackrel{\text{def}}{=} P(Q_t^{1:D} = k, O_t = o | O_{1:T}) \\
\hat{B}(o|k) &= \frac{\sum_{t=1}^T \gamma(t,k,o)}{\sum_{o'} \sum_{t=1}^T \gamma(t,k,o')}
\end{aligned}
$$

A similar equation can be derived for Gaussian distributions.

As mentioned in Section 4.1, many of these parameters are only defined for certain contexts (values of k). If the matrices are initialized to 0 for the invalid k's, they will remain 0 during the re-estimation process. Hence structural knowledge can be used to massively speed up parameter estimation.

If we don't have such prior structural knowledge, and choose to use fully connected graphs with parsimonious representations of the CPDs, as discussed in Section 4.8, the M step will involve running EM (in the case of mixture distributions) or IRLS (in the case of softmax) or a decision tree induction algorithm. This is called generalized EM, and enjoys the same convergence properties as regular EM.

Note that we have segmented data, we can learn the sub-HMMs separately, and then try to combine them together: see Sections 7.9 and 7.11 for examples.

6.2 Structure learning

By the structure of an HHMM, we mean the graph structure of the transition diagram (e.g., Figure 1), i.e., the number of levels, the number of states in each sub-model, the allowable horizontal transitions within each submodel, and the allowable vertical transitions.

The problem of determining the number of states and the set of non-zero horizontal transitions at any given level is equivalent to the problem of learning the structure of an HMM. A top-down, state-splitting approach is described in [FM00], where they use cross validation to determine if a local change to the model is beneficial. An alternative, bottom-up, data-driven approach to learning HMM structure is presented in [SO92]. Here, we start with an HMM that has one state for every position in every sequence (hence it assumes the training set consisting of a set of short strings). It then considers the benefit of merging all possible pairs of states (where the benefit is measured by the increase in the model posterior), and greedily performs the best merge. Clearly this algorithm cannot cannot be used to learn a model from a single long time series, nor from continuous observations.

A much more efficient way of learning HMM structure, which avoids discrete search, is to do MAP estimation of A_k with an entropic prior, followed by parameter pruning [Bra99]. The entropic prior has the form $P(\theta) \propto e^{-H(\theta)}$, and hence discourages parameters which are "uncertain" given the data; such parameters can be safely pruned from the model during EM. This approach assumes that we can afford to initialize the HMM with a fully connected topology and a large number of states, which might be problematic in the HHMM setting. However, it can useful be applied to the observation distributions, too; see Section 7.10 for an example application.

Now we turn to learning the vertical structure. An incoming vertical arc from state k to state i can be pruned if $\pi_k^d(i) \approx 0$, e.g., in Figure 1, $\pi_1^2(6) = 0$, which corresponds to the absent $1 \rightarrow 6$ arc. An outgoing vertical arc is unnecessary if it goes to a single state which has no other incoming arcs, e.g., in Figure 5, the arc from state 2 to state a is unnecessary, since state 2 can be made concrete, as in Figure 1. (This assumes we initialize with all production states at the bottom of the hierarchy, and gradually prune back.) The real problem is detecting shared sub-structure, i.e., when we should do parameter tying. For instance,

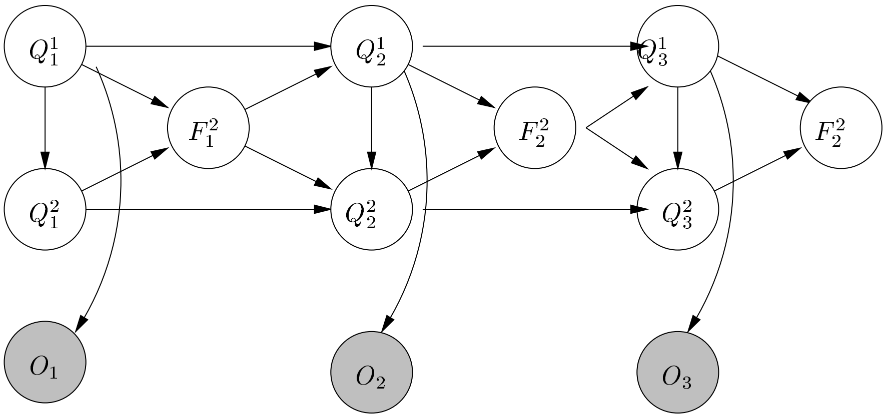


Figure 8: A variable-duration HMM modelled as a 2-level HHMM. Q_t^1 represents the state, and Q_t^2 represents how long we have been in that state.

in Figure 1, suppose initially states 3 and states 6 call separate sub-HMMs. We need some kind of similarity metric on HMMs (e.g., [JR85]) to decide if we should merge them. Alternatively, we could speculatively consider merging all possible pairs of sub-models, and seeing how much the model posterior increases as a result, as in [SO92].

7 Related work and applications

7.1 Representing the distribution over SCFG parse trees

[PW96] discusses how to create a Bayes net to represent the distribution of parse trees obtained from an SCFG. However, this is rather different from the current work, since they need to first run the inside-outside algorithm (which takes $O(T^3)$ time) to create the structure of the Bayes net. The complexity of inference will depend on the structure of the resulting graph, the elimination/triangulation ordering used, and the specific query that is being asked, but in general, it is likely to be very slow.

7.2 Hidden Markov decision trees

Hidden Markov decision trees (HMDT) [JGS96] are DBNs with a structure similar to Figure 4, but they lack the F nodes and the upward going arcs; hence they are not able to model the idea that a subprocess must finish before its parent process can change state.

7.3 Online learning of hierarchical structure

[Pfl00] is concerned with the task of online learning of hierarchical structure from discrete data ("chunking"). The proposed solution is called hierarchical sparse n-grams [Pfl01]. A sparse n-gram can represent the joint distribution over n symbols using a decision tree. What makes it hierarchical is that trees for smaller k-grams, $k < n$, can be used to estimate the frequency of longer substrings, and hence can be used to grow the larger tree. The approach to structure learning mentioned in Section 6.2 is quite different: it is an offline algorithm and relies on hidden state; furthemore, it can work with continuous data (which prohibits solutions that use "counting").

7.4 Variable-duration HMMs

A variable-duration (semi-Markov) HMM is a special case of a 2-level HHMM, as shown in Figure 8. The bottom level counts how long we have been in a certain state; when the counter expires, the F node turns on, and the parent can change state. Formally, we have

$$\pi_k^2(i) = \Pr(\text{stay } i \text{ steps in state } k)$$

and

$$A_k^2(i,j) = \begin{cases} 1 & \text{if } i > 1 \text{ and } j = i-1 \\ 1 & \text{if } i = 1 \text{ and } j = \text{end} \\ 0 & \text{otherwise} \end{cases}$$

Inference in this HHMM takes $O(TQ^2S^2)$ time, where S is the maximum number of steps we can spend in any state; this is the same complexity as the original algorithm [Rab89].

In the generative model, once we have chosen a duration (length) for a state, we can model the corresponding fixed-length subsequence in any way we choose, e.g., using a linear dynamical system. This is called a "segment model" [ODK96], and has been used for speech recognition. However, it is not widely popular because of the increased computational complexity.

7.5 Automatic speech recognition

Automatic speech recognition (ASR) uses HMMs which are hierarchical, as illustrated in Figure 9. However, the state-space topology is usually constructed by hand — the mapping from words to phones is determined by a pronunciation dictionary, and each phone is modelled by a 3 state left-right HMM. Hence the parameter tying can be manually specified, and standard HMM learning techniques suffice. Note that we assume the subphone models are independent of the word in which they occur — this is shown in Figure 10. This dramatically reduces the size of the state-space. (Nevertheless, with, say, 6000 words, 60 phones, and 3 subphones, there are still about 1 million unique states.) The sub-phone and observation nodes in Figure 10 can be represented in factored form [Zwe97, Bil98].

One potential advantage of representing ASR models explicitely as HHMMs arises when we do decoding [JM00, p.252]. One problem with Viterbi in HMMs is that it computes the most probable sequence of states, not words. The correct method is to sum out the phones and max out the words. In the HHMM, words and phones are represented as separate random variables, and hence this is well-defined; however, mixing the $\sum$, max and $\prod$ operators requires some care, because they are not commutative c.f., solving influence diagrams [JJD94]. Another advantage of the HHMM approach is that we can use trigram grammars by parameterizing the transition matrix of the *words*; in an HMM, more sophisticated methods are necessary to model language models which are more complex than bigrams.

When training an HMM where the word sequence is known, but the other hidden nodes are not, we need a way of specifying which sequence of submodels to use, but without specifying when. In traditional HMM modeling, this is solved by joining all the HMMs for each word together by silent (epsilon) transitions. This is not possible in a DBN. The solution adopted by [Zwe97] is shown in Figure 11. We introduce a deterministic indexing process Q_t^i, which increases by 1 whenever a word submodel has finished. Then we define $P(Q_t^w = w | Q_t^i = k, \vec{w}) = \delta(w, w_k)$, i.e., Q_t^w corresponds to the k'th word in the sequence if the index variable is equal to k. Since Q_t^w is a deterministic function of the observed words, $W_{1:N}$, and the word index, Q_t^i, we can "compile out" the Q^w layer for a given word sequence, reducing the model to a standard 2-layer HHMM, but with non-homogeneous transition parameters, i.e., $P(Q_t^P | Q_{t-1}^P, Q_t^i = k) = P(Q_t^P | Q_{t-1}^P, Q_t^w = w_k)$.

Overall, this implements a probabilistic mapping from $W_{1:N}$ to $O_{1:T}$, where $N < T$. The complexity of inference is $O(TNK^2)$, where K is the number of phones and subphones (assuming a dense transition matrix). (The reason it is not N^2 is because of the deterministic nature of the CPT for Q_t^i.) This is the same complexity as the asychronous input-output HMM [BB96], except they propose treating the acoustic sequence as input and the words as output.

The technique of adding an index layer can easily be extended to convert any HHMM into a hierarchical probabilistic transducer.

7.6 Embedded HMMs for face recognition

Embedded HMMs [KA94, NI00, MN01] are a special case of HHMMs originally designed for modelling 2D images (see Figure 12). There are two key differences: firstly, the hierarchy is always depth two, and

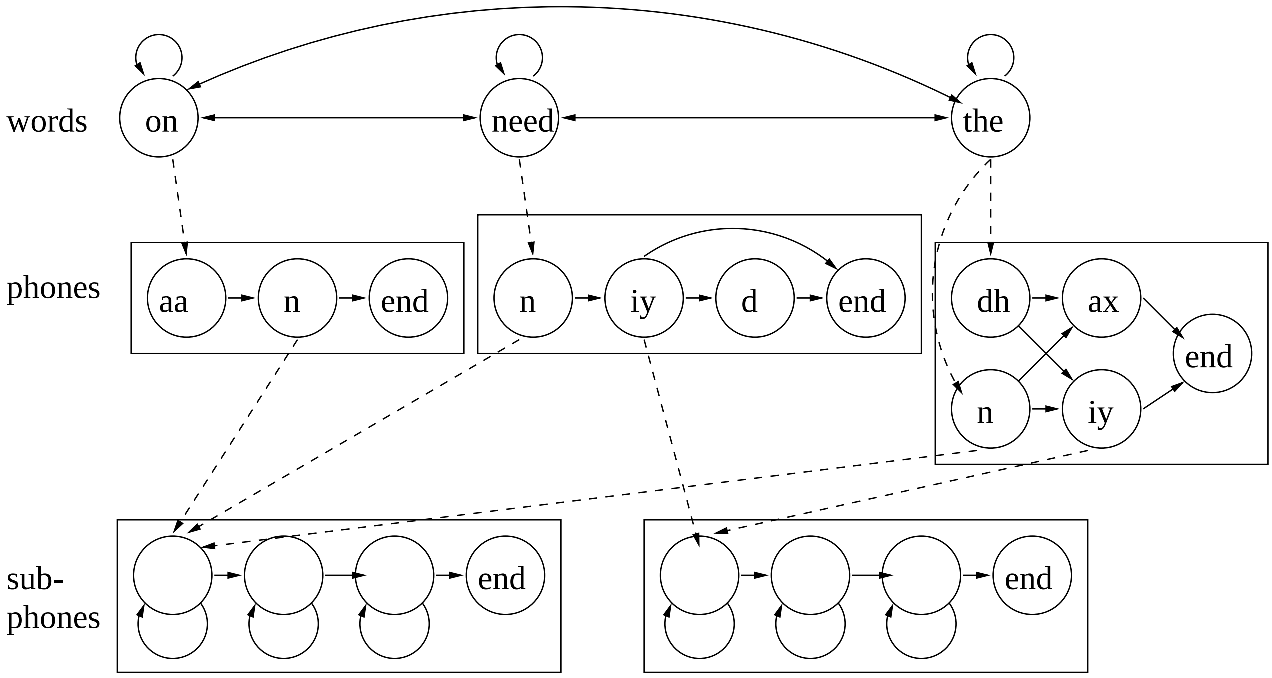


Figure 9: An example HHMM for an ASR system which can recognize 3 words (adapted from [JM00]). The phone models are shared (tied) amongst different words.

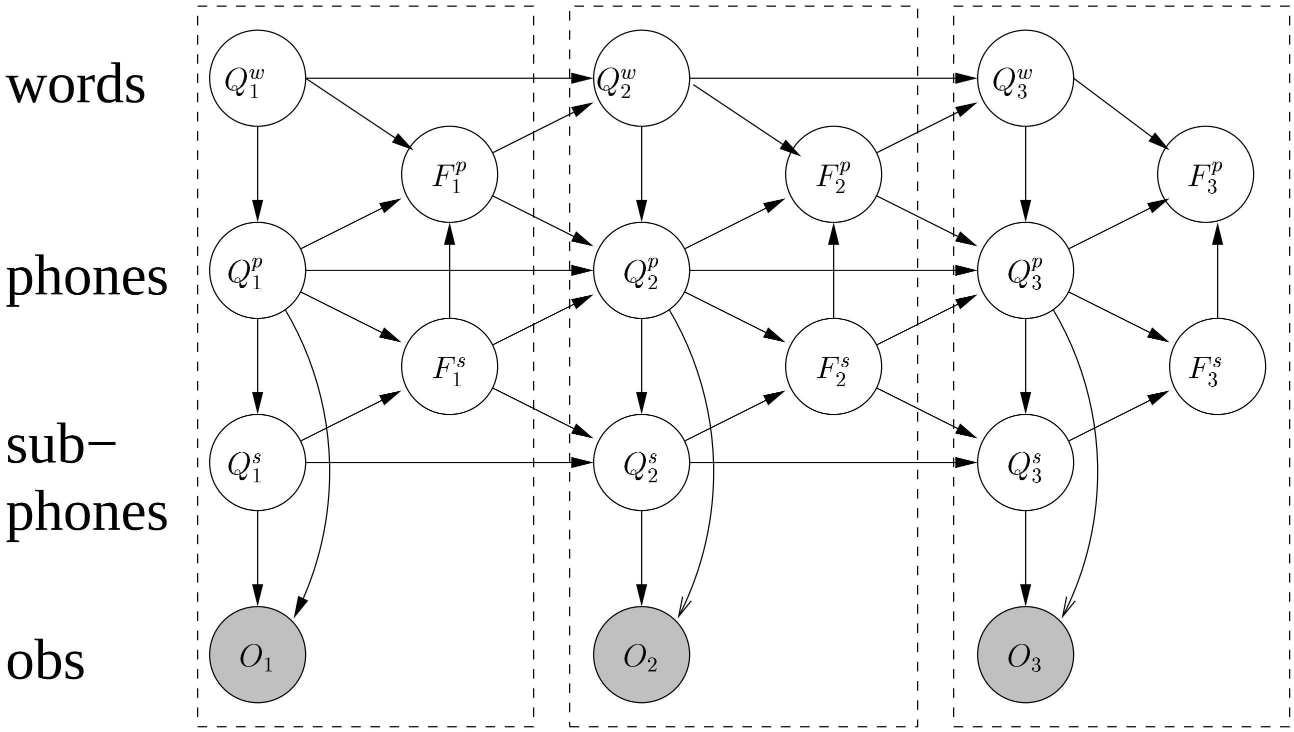


Figure 10: An HHMM-DBN suitable for ASR. The difference from the 3-level HHMM in Figure 4 is that there are no arcs from Q^w to Q^s, F^s or O, since these submodels have the same parameters regardless of what word they are used in, i.e., the joint probability over words, phones, subphones and observations is given by $P(Q^W_{1:T})P(Q^P_{1:T}|Q^W_{1:T})P(Q^S_{1:T}|Q^P_{1:T})P(O_{1:T}|Q^S_{1:T}, Q^P_{1:T})$.

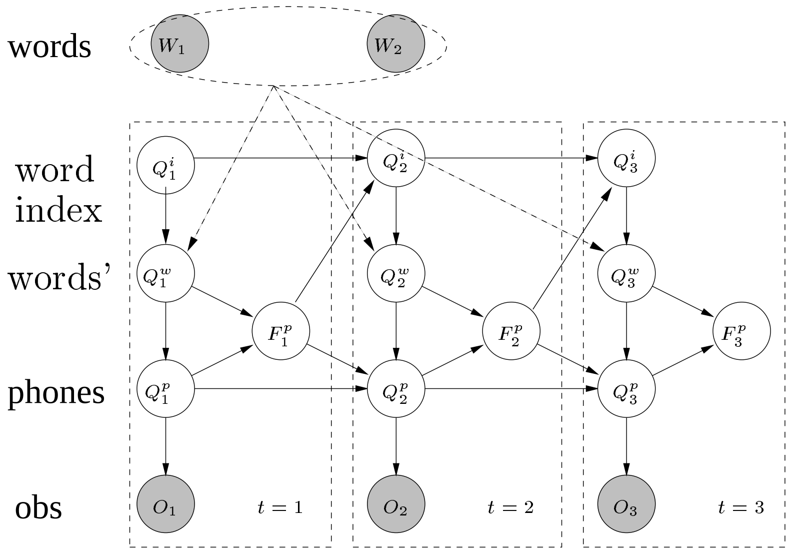
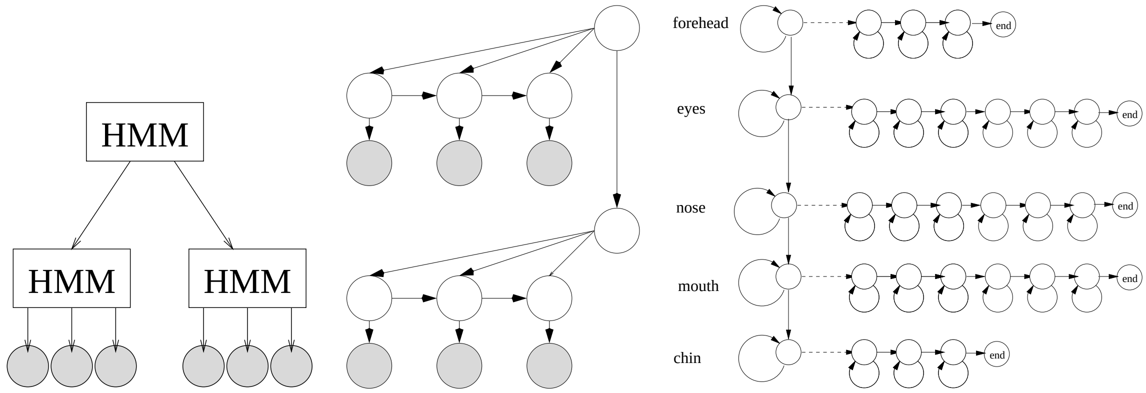


Figure 11: Supervised training of an ASR-DBN, i.e., a probabilistic transducer. The sub-phones are omitted to simplify the figure.



(a) (b) (c)

Figure 12: An embedded HMM represented (a) schematically, (b) as a DBN and (c) as an HHMM, based on [NI00]. We assume there are 2 rows and 3 columns in the image, which is clear from the DBN representation, but not from the HHMM representation. The number of states in each sub-model can be different, which is clear from the HHMM but not the DBN.

secondly, the ending "time" of the sub-HMM is known in advance, since each sub-HMM models exactly C observations, where C is the number of columns in the image. Hence the problem of deciding when to return control to the (unique) parent state is avoided.

Inference in embedded HMMs is very efficient since, conditioned on the states of the top-level HMM, the DBN reduces to a set of chains. For example, to compute the likelihood of an image, for each row r of the image and each state i in the top-level HMM, we compute $b_r(i) = P(O(r,:)|Q_r = i)$ using the forwards algorithm with the i'th set of HMM parameters, where $O(r,:)$ is the r'th row of pixels. These conditional likelihoods can then be used in the forwards algorthm at the top-level in lieu of the more conventional $b_t(i) = P(O_t|Q_t = i)$. See [MN01] for details.

[Hoe01] presents a very similar model to the embedded HMM which he calls a "hierarchical mixture of Markov chains"; this allows for any number of levels. (The ending times are pre-determined because the data is manually segmented.) He uses a similar cutset-conditionining/ Pearl hybrid inference algorithm.

7.7 Abstract HMMs for plan recognition

[BVW00a] describes the "Abstract HMM" (AHMM), which is very closely related to HHMMs. (They have 3 slightly different models in [BVW00a, BVW00b, BVW01]; we consider the version in [BVW00b].) These authors are interested in inferring what abstract policy an agent is following by observing its effects in the world.

A stochastic policy (controller) is a probabilistic mapping from (fully observed) states to actions: $\pi(s, a) = P(\text{do action } a|\text{in state } s)$. An abstract (stochastic) policy is a (probabilistic) mapping from states to lower level abstract policies, or "macro actions". An abstract policy can call a sub-policy, which runs until termination, returning control to the parent policy; the sub-policy can in turn call lower-level abstract policies, until we reach the bottom level of the hierarchy, where a policy can only produce concrete actions.

An HHMM is like an AHMM if we consider Q_t^d to represent the (abstract) policy being followed at level d and time t; Q_t^D represents the concrete action, which causes the observation. In addition, we need to add a hidden global state variable S_t, which is a parent of the O_t node, all the F_t nodes and all the Q_{t+1} nodes. S_t is hidden to us as observers, but not to the agent performing the actions (the agent is solving a fully-observed decision problem). F_t^d has as parents the current goal, Q_t^d, and the current world state, S_t, and decides if the goal has been achieved or not. The Q nodes have Q and S as parents, since the next (abstract) action depends on the policy being followed and the world state.

Note that Bui et al. assume that Q_t^d only depends on its immediate parent, Q_t^{d-1}, but not its whole context, $Q_t^{1:d-1}$. This substantially reduces the number of parameters and allows for more efficient approximate inference, as we see below. However, it may also lose some information. For example, if the bottom level controller determines how to leave a room via a door, we may be able to predict the future location of the agent (e.g., will they leave via the left or right door) more accurately if we take into account the complete calling context (e.g., their top-level goals).

[BVW00a] consider abstract policies of the "options" kind [SPS99], which is equivalent to assuming that there are no horizontal transitions within a level. (HAMs [PR97] generalize this by allowing horizontal transitions (i.e., internal state) within a controller.) This simplifies the CPTs, as we see below.

7.7.1 Policy termination in AHMMs

The termination probability is assumed to be independent of the context, $Q_t^{1:d-1}$, so the F nodes no longer have arcs from Q nodes above, and Equation 2 simplifies to

$$P(F_t^d = 1|Q_t^d = \pi, S_t = s, F_t^{d+1} = b) = \begin{cases} 0 & \text{if } b = 0 \\ \beta_\pi(s) & \text{if } b = 1 \end{cases}$$

where $\beta_\pi(s)$ is the probability of terminating in state s given that the current policy is π.

At the bottom level, concrete actions always terminate in 1 step (by definition), hence F_t^D is always on; therefore this node and its associated arcs can be removed. In addition, we can remove the horizontal links between the action nodes (from Q_{t-1}^D to Q_t^D).

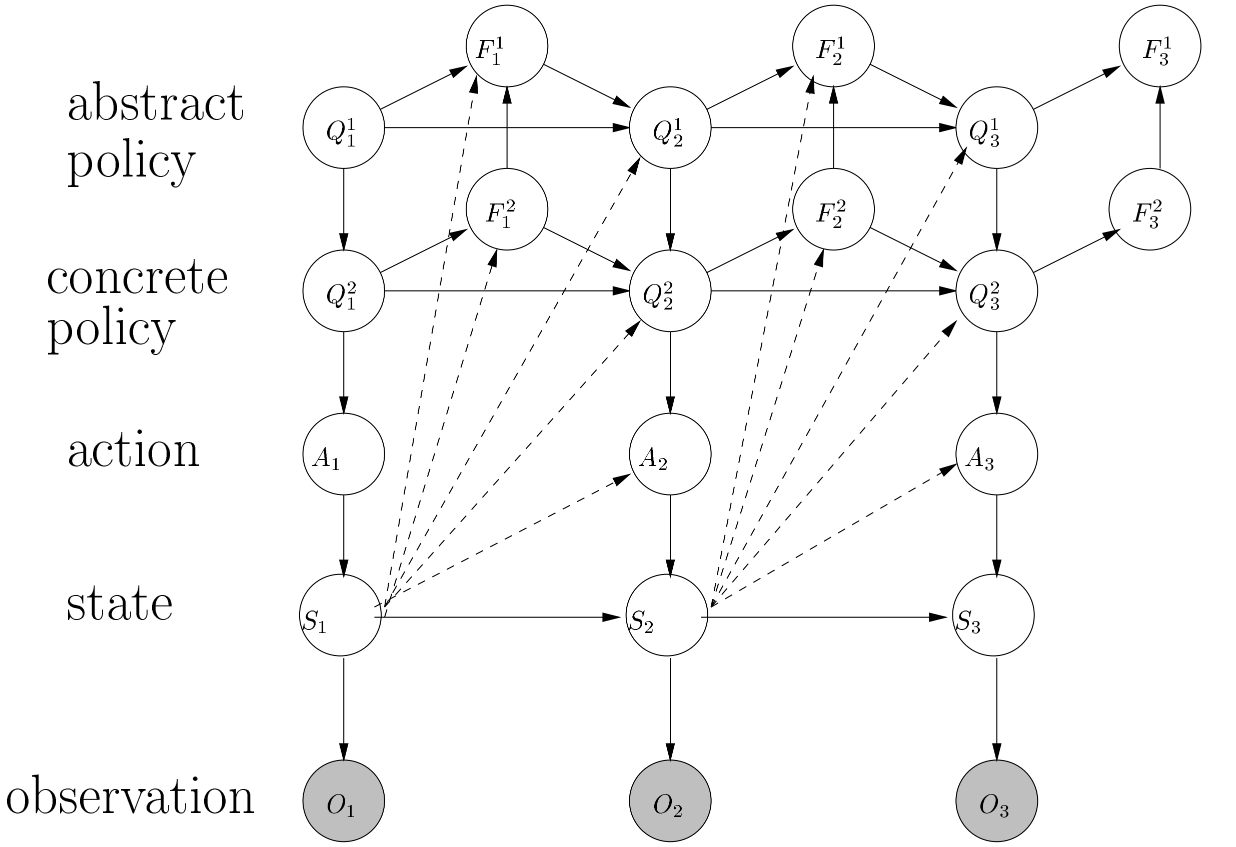


Figure 13: The DBN for a 3-level abstract HMM, from [BVW00b]. Note that we call the top layer level 1, they call it level D. Note also that we draw F_t^d above Q_t^d, they draw it below, but the topology is the same. The top level encodes the probability of choosing an abstract policy: $P(Q_t^1 = \pi' | S_{t-1} = s) = \pi_{\text{top}}(s, \pi')$. The second level encodes the probability of choosing a concrete policy: $P(Q_t^2 = \pi | S_{t-1} = s, Q_t^1 = \pi') = \pi'(s, \pi)$. The third level encodes the probability of choosing a concrete action: $P(A_t = a | S_{t-1} = s, Q_t^2 = \pi) = \pi(s, a)$, where $A_t = Q_t^3$. The last level encodes the world model: $P(S_t = s' | S_{t-1} = s, A_t = a)$. The lines from S_t are shown dotted merely to reduce clutter.

7.7.2 Policy selection in AHMMs

Since we cannot make any horizontal transitions, Equation 1 simplifies to

$$P(Q_t^d = j | Q_{t-1}^d = i, Q_t^{d-1} = \pi, S_{t-1} = s, F_{t-1}^d = f) = \begin{cases} \delta(i,j) & \text{if } f = 0 \\ \sigma_\pi(s,j) & \text{if } f = 1 \end{cases}$$

where $\sigma_\pi(s, j)$ is the probability that abstract policy π chooses to call sub-policy j in state s. (We do not need to condition on the termination status of the level below, $b = F_{t-1}^{d+1}$, since $f = 1$ only if $b = 1$.)

[BVW00a] assume that the top level policy, Q_t^0, is fixed (equivalent to assuming the existence of a fixed root node in the HHMM); hence F_t^0 is always off and can be removed. To simplify notation, we simply omit Q_t^0 and suppress conditioning on it, since it is a global constant. The final DBN is shown in Figure 13.

7.7.3 Approximate inference in AHMMs

[BVW00a, BVW00b] make the observation that, conditioned on S_t and $F_t^{1:D}$ (i.e., the termination status of all policies), the topology within a slice reduces to a chain, and hence we can represent the conditional belief state, $P(Q_t^{1:D} | F_t^{1:D}, S_t, O_{1:t})$, as a product of factors. They therefore suggest using Rao-Blackwellised

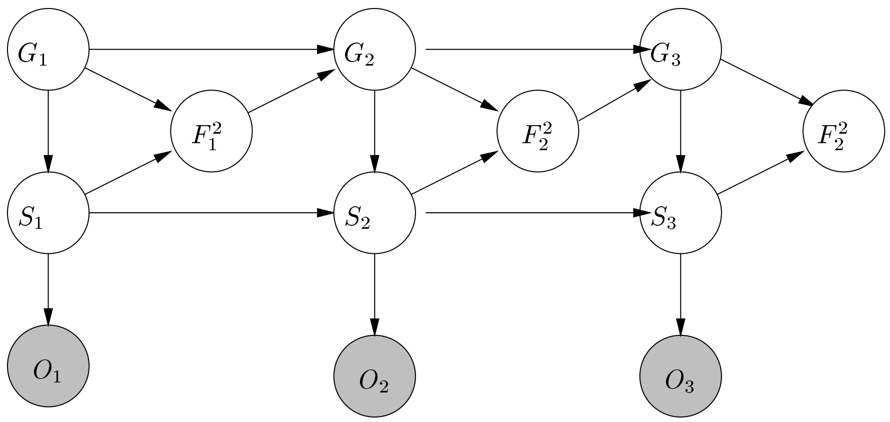


Figure 14: A 2-level HHMM. F_t turns on if state S_t statisfies goal G_t; this causes a new goal to be chosen. However, the new state may depend on the old state; hence there is no arc from F_{t-1} to S_t.

particle filtering (see e.g., [DdFMR00]) for approximate inference. In particular, one only needs to sample S_t and $F_t^{1:D}$ — there is no need to also sample $Q_t^{1:D}$.

Unfortunately, their trick does not work in our model, since, as can be seen from Figure 4, the Q nodes are not connected in a chain even when the F nodes are instantiated. Of course, we can apply regular particle filtering to our model, but we suspect an online version of factored frontier (FF) [MW01] would be more efficient. For offline smoothing (needed for EM), a deterministic approximation method such as FF is also likely to be much more efficient than particle smoothing or MCMC.

7.7.4 2-level HHMMs

In [BVW01], they consider the special-case of a 2-level model, which is shown in Figure 14. This is very similar to an AHHM/HHMM, but the semantics are slightly different. $G_t \equiv Q_t^1$ represents the goal that the agent currently has, and $S_t \equiv Q_t^2$ is the world state.

The transition matrix for G is the same as for Q^1 in an HMMM, except we never reset, since F^1 is not a parent (since the top level can never terminate):

$$P(G_t = j | G_{t-1} = i, F_{t-1}^2 = f) = \begin{cases} \delta(i,j) & \text{if } f = 0 \\ A^1(i,j) & \text{if } f = 1 \end{cases}$$

([BVW01] consider the special case where $A^1(i,j)$ is a deterministic successor-state function.)

The transition matrix for F^2 is the same as for F^D in an HMMM:

$$P(F_t^2 = 1 | G_t = g, S_t = i) = A_g^2(i, \text{end})$$

The transition matrix for S is the same as for Q^D in an HHMM, except we never "reset", since F^2 is not a parent (since the next state always depends on the previous state, even if the goal changes):

$$P(S_t = j | G_{t-1} = g, S_{t-1} = i) = A_g^2(i,j)$$

7.7.5 Application of 2-level HHMMs to people tracking

To be written.

7.8 Hierarchical POMDPs for robot map-making and navigation

[RTM00] used an HHMM to learn a multi-scale topological map. They trained separate HMMs on manually segmented data to represent corridors and intersections; they then learned how to connect these sub-models (i.e., the top-level HHMM topology) using EM (they used the original $O(T^3)$ inference algorithm).

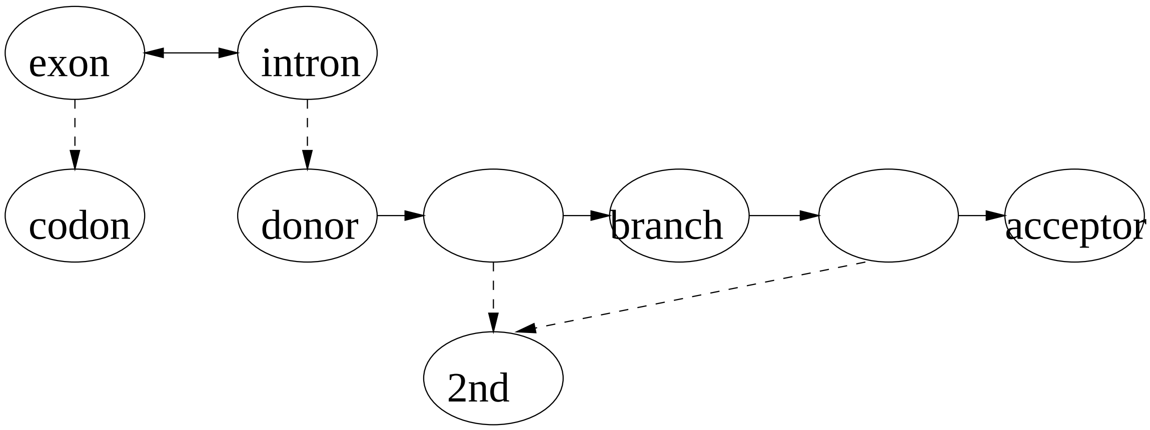


Figure 15: (H)HMM for gene recognition. 2nd refers to a second-order Markov model.

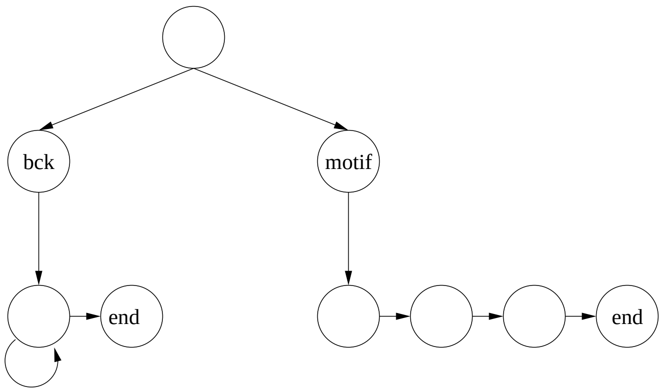


Figure 16: The state transition diagram for an HHMM for motif discovery.

[TRM01] extends this work by allowing actions nodes, as in a partially observable Markov decision process (POMDP). This is very similar to abstract HMMs described in Section 7.7, except the policies at each level of the hierarchy are not explicitely represented (the Q_t^d nodes represent the location of the robot at different levels of abstraction). They proposed the following heuristic planning method. For any given goal, find the corresponding abstract and concrete state containing it; plan to get to the abstract state by treating the top-level as an MDP, and assuming the current state is the mode of $P(Q_t^1|O_{1:t})$ (the most likely state (MLS) heuristic). To implement this plan, we treat each sub-model as an MDP, and use the MLS heuristic to devise a plan to reach the corresponding end state (or the concrete goal state). Hence the macro-actions at the top-level are implemented in terms of controllers at the bottom-level, which execute primitive actions.

7.9 Gene recognition using (H)HMMs

[HIM$^+$00] discuss building an HMM for recognizing genes (see Figure 15). The HMM contains sub-HMMs, for detecting exons and introns, which are trained separately using manually-segmented data. These separate HMMs are then manually "glued together". To learn when to transition from an exon to an intron or vice versa, the training procedure was modified as follows. When training the intron model, the subsequences corresponding to exons were replaced by a single "E" symbol. The state that emits the "E" symbol is later replaced by a transition to the exon HMM. Similarly, the exon HMM calls the intron HMM.

7.10 Motif discovery using HHMMs

A motif is a short pattern, or distribution over short patterns (profile), that has biological significance. An important problem in biology is to detect such frequently recurring patterns automatically from DNA sequence data. One approach is to model this using a 2-level HHMM, as shown in Figure 16. Here we have assumed that the motif is of length 3, and that the background can be modelled by a single state HMM. If there is more than one type of motif, we simply add more sub-HMMs to the model. Similarly, it is simple to use use more complex background models.

The main challenges are to learn, in an unsupervised way, the number of motif types, their lengths, and their distributions. This is a hard model selection (structure learning) problem. Let us consider a much easier problem: identify the identity of a single motif of known length. For example, suppose the motif is the pattern 'accca' (real motifs would be longer and have variability). In my experiment[4] I created 100 sequences of length 20, where each character was drawn uniformly at random from $\{a, c, g, t\}$. I then randomly selected a single location and inserted the motif. This was the training set.

I found that straightforward maximum likelihood estimation using EM was unable to learn the motif: there was not enough inductive bias to prevent the background state from modelling everything. So then I used a minimum entropy prior [Bra99] for the output distributions for the motif states (but not the background states). This encodes the prior knowledge that we expect the motif to be (nearly) deterministic. In other words, we expect the output distribution of each motif state to be (nearly) a delta function; we just don't know where (on which letter) to place the spike. With this approach, I was able to recover the true motif.

I am currently working with Eric Xing on extending these techniques to real biosequence data.

7.11 Information extraction/ shallow parsing

Information extraction refers to filling in a "template", such as a database record containing the name, title and location of a seminar, from a text document. One common method [FM99, FM00] is to train separate HMMs for each field in the template using manually segmented data. Each HMM has two kinds of states: target states, which model the field, and background states, which model surrounding context. Information is extracted by performing a Viterbi segmentation using each HMM separately; the parts of the document that are assigned to target states are then extracted.

The above approach has several disadvantages: (1) different HMMs may "claim" the same pieces of text; (2) it requires manually segmented data; and (3) the HMMs should really be trained discriminatively. An HHMM solves all of these problems (discriminative training comes for free, since if one sub-HMM becomes more likely, the other rival sub-HMMs are forced to become less likely due to the sum-to-one constraint).

7.12 HMM/SCFG hybrids for inferring high-level behavior from video

In [IB00], it is pointed out that recognizing high-level behaviors from visual evidence can be aided by the fact that such high-level behaviors can be composed of re-usable low-level behaviors, which can be recognized using HMMs. These authors suggest running a bank of HMMs in parallel, to detect all the possible low-level events, and then picking the one with the highest likelihood to convert the signal into a string of discrete symbols (representing different event types). They then use a SCFG to model distributions over strings, and compute the most likely parse (interpretation).

[ME01] have a similar hybrid SCFG system; at the top level, they have an SCFG which encodes the rules of the card game Blackjack, and at the bottom level they track objects such as hands, cards and chips.

SCFGs are more powerful than HHMMs; for instance, they support recursion. However, all of the examples in [IB00] involve at most tail recursion, which can easily be represented in an HHMM using a self-transition. We suspect that HHMMs would provide a simpler way to implement these and other systems; this conjecture is something we plan to investigate in the future.

[4] Code for this is available in BNT. See `BNT/examples/dynamic/HHMM/Motif`.

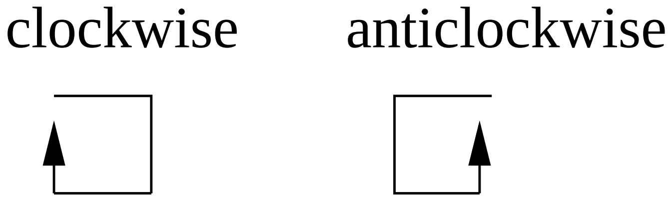


Figure 17: Two ways of drawing a square.

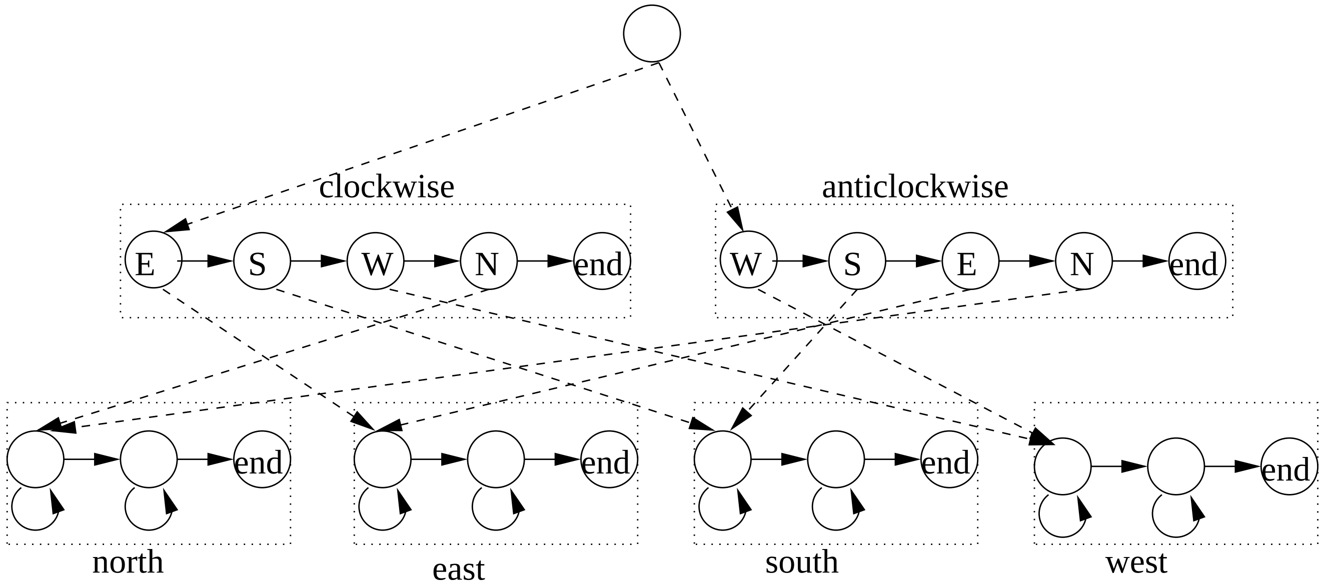


Figure 18: An HHMM corresponding to a simple SCFG for classifying squares, based on [IB00].

7.12.1 Toy gesture recognition problem

As a warmup task, we consider the problem of distinguishing two kinds of hand-drawn squares: clockwise and anticlockwise (see Figure 17). These squares could be drawn with a mouse, on a tablet, or in the air (using gesture recognition). This task was first proposed in [IB00] to illustrate how HMMs, representing up (north), down (south), left (west) and right (east) strokes, could be pieced together in different ways according to a very simple grammar. Specifically, the grammar is as follows: (Production states (terminal symbols) are in lower-case, and probabilities of each rule are omitted for simplicity.)

```
SQUARE -> CLK | ACK
CLK -> east south west north
ACK -> west south east north
```

Each of the “terminals” north, east, south and west can be modelled by e.g., a 2 state left-right HMM, which emits a vector in R^2, representing the (x, y) velocity of the pen. This is shown in Figure 18.[5],

There are several problems with this model. First, note that we always traverse the bottom level HMMs in a left-right fashion, even if we are drawing the square in “reverse” order. (One way to fix this is to simply use one-state HMMs at the bottom level.) Second, there is no constraint that the duration spent in each sub-HMM is the same; hence the sides of the “square” are not guaranteed to be the same length. This could be fixed by adding a global duration variable which is shared across sub-models.

[5]Code for this example is available in BNT. See `BNT/examples/dynamic/HHMM/Square`.

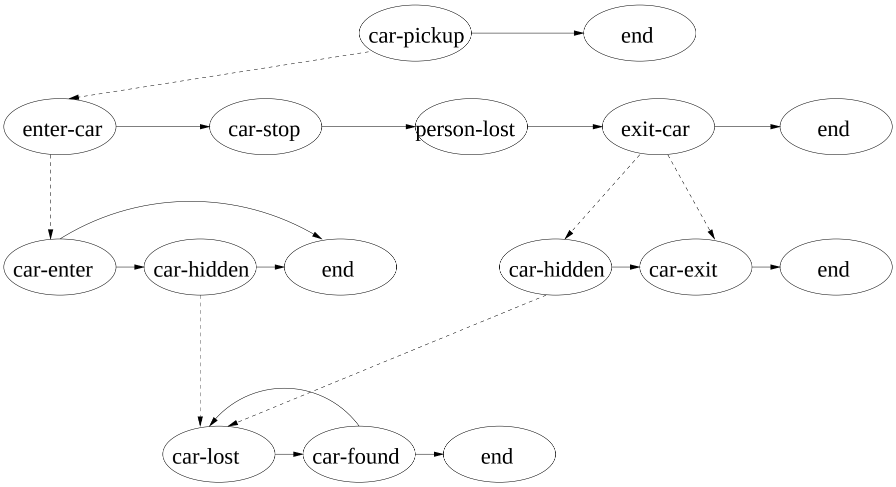


Figure 19: An HHMM corresponding to a simple SCFG for detecting events from video, based on [IB00].

7.12.2 Recognizing events in videos of parking lots

We now give an example of how a more complex grammar could be used to generate high-level descriptions such as "a car picked up a person" from a video sequence of a parking lot. (The low-level events are detected by a motion tracker, not an HMM, which may lose and then re-find an object, although not necessarily the same one.) A (slightly modified) piece of their SCFG grammar is given below.

```
CAR-PICKUP -> ENTER-CAR  car-stop  person-lost  EXIT-CAR
ENTER-CAR  -> car-enter | car-enter CAR-HIDDEN
CAR-HIDDEN -> car-lost car-found | car-lost car-found CAR-HIDDEN
EXIT-CAR   -> car-exit | CAR-HIDDEN car-exit
```

A corresponding HHMM is shown in Figure 19. (Notice how the tail-recursion in the CAR-HIDDEN rule is handled by a simple loop.) When the tracker first finds a moving object, it may be uncertain of its identity. Hence it may say `person-enter` with probability 0.7 and `car-enter` with probability 0.3. These probabilities can be used as "soft evidence" on the appropriate production states. For example, we associate the conditional likelihood $b = (0.7, 0.3)$ with the `person-enter` state, and $b = (0.3, 0.7)$ with the `car-enter` state. All other production states would receive the all-0s vector as evidence.

The [IB00] system allows the likelihood of terminal symbols to depend on the space-time location of the event thay they represent. One way to represent this in the HHMM framework is to add hidden state variables, as in the AHMM approach to plan recognition.

7.13 Hierarchical variable-length Markov models for visual motion

[GJH01] discusses learning the structure of hierarchical variable length Markov models for modelling exercise routines with repetitive behavior from video data. Even though VLMMs are fully observed (they use vector quantization to convert the B-spline representation of the person's profile to a unique discrete symbol), they need to use dynamic time warping (i.e., forwards-backwards analysis) if the input is uncertain. The HHMM offers a more unified framework for this.

Acknowledgements

I would like to thank Dr Christopher Schlick for giving me his Matlab implementation of the $O(T^3)$ algorithm, which was used to create part of Figure 7.

Appendices

A worked example

In this section, we explicitly compute various quantities for the model shown in Figure 20. The transition matrix for level 3 is

$$A^3 = \begin{pmatrix} 0.5 & 0.5 & 0 & 0 & 0 & 0.0 \\ 0.6 & 0.0 & 0 & 0 & 0 & 0.4 \\ 0 & 0 & 0.0 & 0.5 & 0.0 & 0.5 \\ 0 & 0 & 0.3 & 0.0 & 0.7 & 0.0 \\ 0 & 0 & 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

The rescaled transition matrices for the DBNs are

$$A_1^3 = \begin{pmatrix} 0.5 & 0.5 & 0 \\ 1.0 & 0.0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

and

$$A_2^3 = \begin{pmatrix} 0.0 & 1.0 & 0.0 \\ 0.3 & 0.0 & 0.7 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}$$

The fact that the HMM below q_1^2 only has 2 states is reflected in the "structural zeros" in A_1^3; if the prior topology of the HMM is known, these constraints can be hard-coded in. (Entries that are initially 0 will remain 0 after maximum likelihood estimation.)

Now suppose we observe the sequence $O = (1, 2, 3, 4, 5)$. To simplify the calculations, we assume that each production state is fully observable, i.e., state i emits symbol i with probability 1, for $i = 1, \ldots, 5$. There are two ways to generate this sequence: the first is shown in Figure 21; the second takes an indirect route between q_3^3 to q_4^3, and is shown in Figure 22. The first path is the most probable; the corresponding variable assignment is shown in Figure 23, and has probability 0.0017. The transition out of q_2^3 at time 2 is represented by $F_2^2 = 1$ (the level numbers in the DBN are 1 less than in the HHMM, since we exclude the vacuous root state q^1). This path probability can be computed as follows:

$$\begin{aligned} P_1 &= P(q^1 \to q_1^2) \times P(q_1^2 \to q_1^3) \times P(q_1^3 \rightsquigarrow O_1) \times P(q_1^3 \to q_2^3) \times \cdots \\ &= 0.5 \times 0.4 \times 1.0 \times 0.5 \times \cdots \end{aligned}$$

The second path is 10 times less likely. The corresponding assignment differs only at time $t = 3$: in this case, $F_3^2 = 1$ *and* $F_3^1 = 1$, since we exit levels 2 and 1 at time 3, only to re-enter them at time 4. The log-likelihood of the data is given by

$$L = \log \sum_{\text{paths}} P(\text{path}|O_{1:T}) = \log(0.0017 + 0.000168) = -6.2937$$

If we use max-product instead of sum-product, the log-likelihood is given by

$$L = \log \max_{\text{paths}} P(\text{path}|O_{1:T}) = \log(\max\{0.0017, 0.000168\}) = -6.3890$$

which is just the log probability of the most likely (Viterbi) path.

Now we consider some marginal probabilities. The term $\xi(t,d,i,j,k) = P(Q_{t-1}^d = i, F_{t-1}^d = 0, Q_t^{d-1} = k, Q_t^d = j|O_{1:T})$ when $t = 4, d = 2, i = 1, j = 2, k = 2$ stores the probability of the $q_3^3 \to q_4^3$ transition, since $(i = 1, k = 2)$ encodes q_3^3 and $(j = 2, k = 2)$ encodes q_4^3; this probability is 0.9091. The term $\xi_e(t,d,i,k) = P(Q_t^{d-1} = k, Q_t^d = i, F_t^d = 1|O_{1:T})$, when $t = 3, d = 2, i = 1, k = 2$ stores the probability of the $q_3^3 \to q_e^3$ transition, which is 0.0909. We see that these terms sum to 1, and are in a 10:1 ratio, as they should be. The term $\chi(t,d,i,k) = P(F_{t-1}^d = 1, Q_t^{d-1} = k, Q_t^d = i|O_{1:T})$ when $t = 4, d = 2, i = 2, k = 2$ stores the probability of the $q_2^2 \to q_4^3$ transition, which is 0.0909. The probability of the upwards and downwards

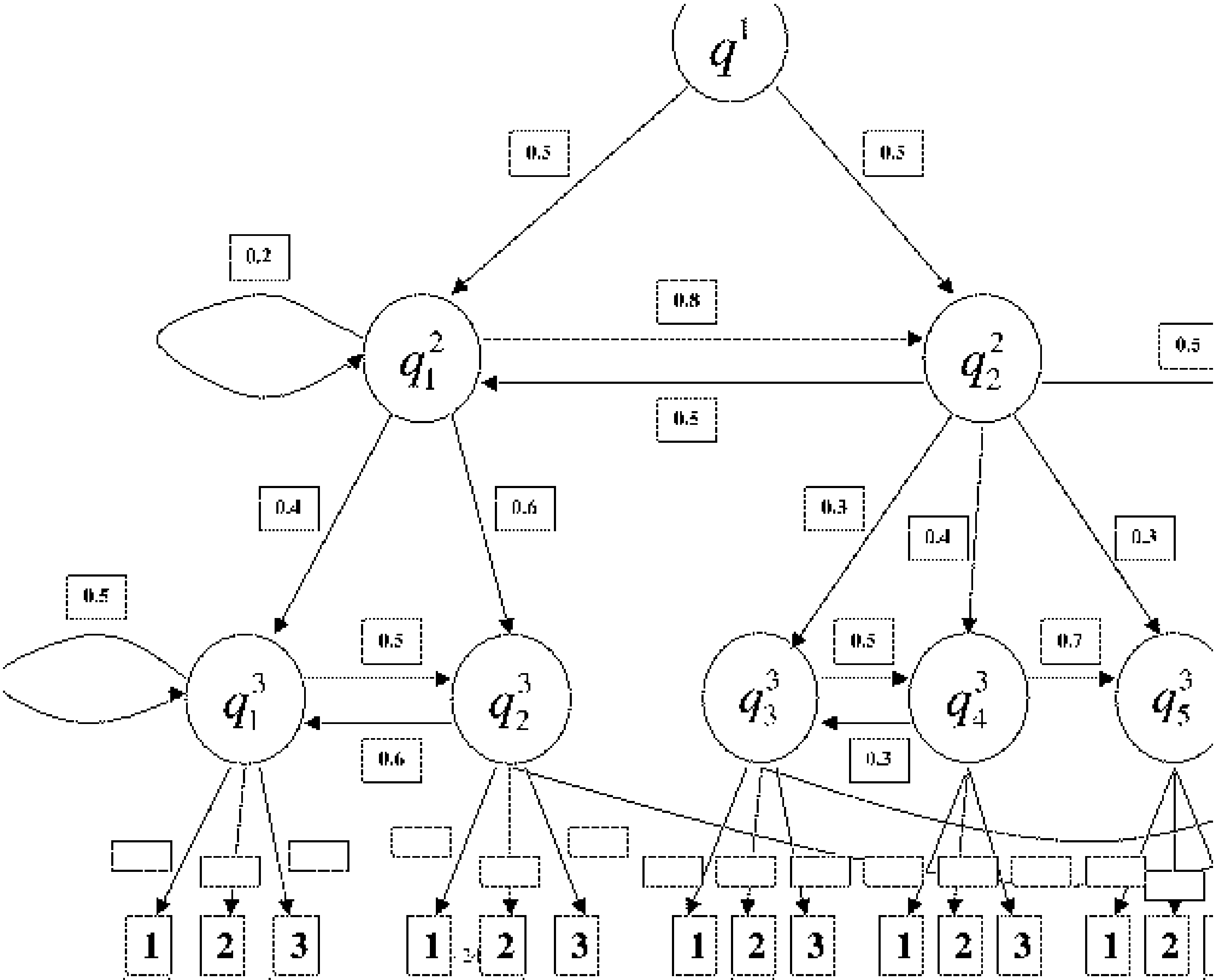


Figure 20: An example of an HHHM represented as a probabilistic finite state automaton, taken from [Sch00]. In this example, the observations (1, 2 or 3) are all emitted at the bottom level. We see that the number of states of the bottom-level HMM depends on the state from which it was called.

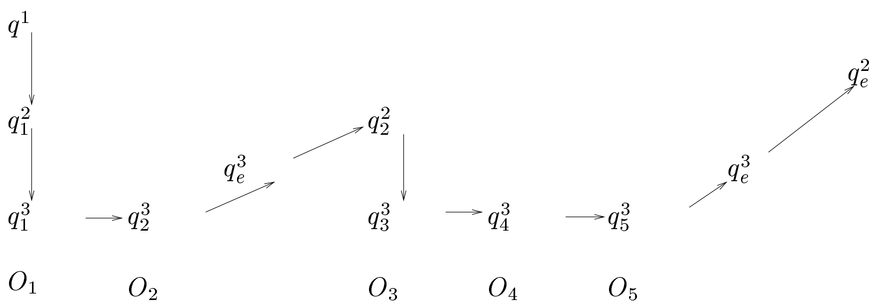


Figure 21: The most probable path through the HHMM state lattice.

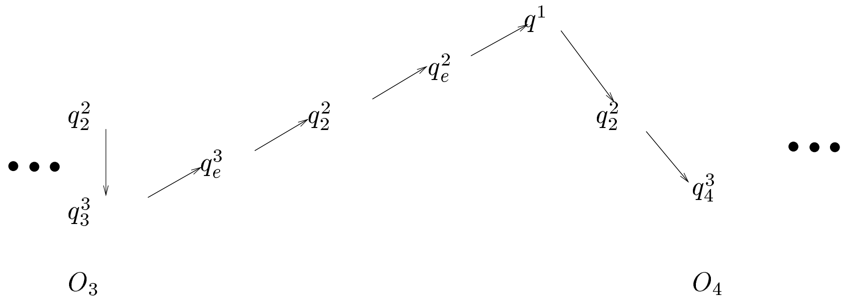


Figure 22: The second most probable path takes a long route from q_3^3 to q_4^3. Note that q_2^2 cannot make a self-transition, but it can enter q_e^2 and then re-enter itself via q^1.

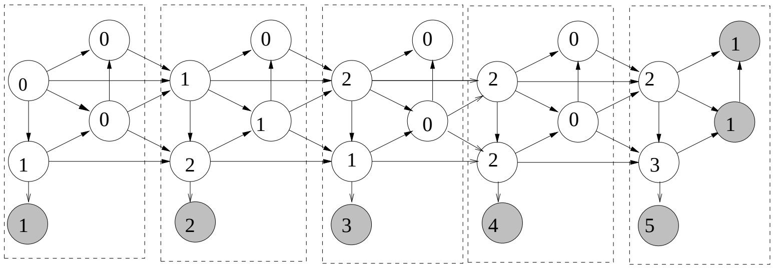


Figure 23: The most probable assignment to all the nodes.

transitions are equal, as they should be. Finally, consider the marginal probabilities of the states themselves. The matrix $M(i,t) = P(Q_t^2 = i|O_{1:T})$ is shown below

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

and $M'(i,t) = P(Q_t^1 = i|O_{1:T})$ is shown below

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

which indicates the correct state sequence. If we used max-product instead of sum-product, the probabilities at time $t = 4$ would be 0.9091 instead of 1.0 in both matrices, because we would not be summing over all paths.

References

[BB96] S. Bengio and Y. Bengio. An EM algorithm for asynchronous input/output HMMs. In *Proc. Intl. Conf. on Neural Info. Processing (ICONIP)*, 1996.

[BFGK96] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-Specific Independence in Bayesian Networks. In *UAI*, 1996.

[Bil98] J. Bilmes. Data-driven extensions to HMM statistical dependencies. In *Intl. Conf. Speech Lang. Proc.*, 1998.

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

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

[Bra99] M. Brand. Structure learning in conditional probability models via an entropic prior and parameter extinction. *Neural Computation*, 11:1155–1182, 1999.

[BVW00a] H. Bui, S. Venkatesh, and G. West. On the recognition of abstract Markov policies. In *AAAI*, 2000.

[BVW00b] H. Bui, S. Venkatesh, and G. West. Policy Recognition in the Abstract Hidden Markov Model. Technical Report 4/2000, School of Computing Science, Curtin Univ. of Technology, Perth, 2000.

[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.

[DdFMR00] A. Doucet, N. de Freitas, K. Murphy, and S. Russell. Rao-blackwellised particle filtering for dynamic Bayesian networks. In *UAI*, 2000.

[EB96] S. ElHihi and Y. Bengio. Hierarchical recurrent neural networks for long-term dependencies. In *NIPS-8*, 1996.

[FM99] D. Freitag and A. McCallum. Information extraction with hmms and shrinkage, 1999.

[FM00] Dayne Freitag and Andrew McCallum. Information extraction with HMM structures learned by stochastic optimization. In *AAAI/IAAI*, pages 584–589, 2000.

[FST98] Shai Fine, Yoram Singer, and Naftali Tishby. The hierarchical Hidden Markov Model: Analysis and applications. *Machine Learning*, 32:41, 1998.

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

[GJH01] A. Galata, N. Johnson, and D. Hogg. Learning Variable Length Markov Models of Behaviour. *Int. J. of Computer Vision and Image Understanding (CVIU)*, 81(3):398–413, 2001.

[HIM+00] M. Hu, C. Ingram, M.Sirski, C. Pal, S. Swamy, and C. Patten. A Hierarchical HMM Implementation for Vertebrate Gene Splice Site Prediction. Technical report, Dept. Computer Science, Univ. Waterloo, 2000.

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

[IB00] Y. Ivanov and A. Bobick. Recognition of visual activities and interactions by stochastic parsing. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22(8):852–872, 2000.

[JGS96] M. I. Jordan, Z. Ghahramani, and L. K. Saul. Hidden Markov decision trees. In *NIPS*, 1996.

[JJD94] F. V. Jensen, F. Jensen, and S. L. Dittmer. From influence diagrams to junction trees. In *UAI*, 1994.

[JM00] Daniel Jurafsky and James H. Martin. *Speech and language processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* Prentice-Hall, 2000.

[JR85] B. H. Juang and L. R. Rabiner. A probabilistic distance measure for hidden Markov models. *AT&T Technical Journal*, 64(2):391–408, 1985.

[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.

[Kja90] U. Kjaerulff. Triangulation of graphs – algorithms giving small total state space. Technical Report R-90-09, Dept. of Math. and Comp. Sci., Aalborg Univ., Denmark, 1990.

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

[LY90] K. Lari and S. J. Young. The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech and Language*, 4:35–56, 1990.

[ME01] D. Moore and I. Essa. Recognizing multitasked activities using stochastic context-free grammar. In *CVPR Workshop on Models vs Exemplars in Computer Vision*, 2001.

[MN01] K. Murphy and A. Nefian. Embedded graphical models. Technical report, Intel MRL, 2001.

[Mur99] K. Murphy. Pearl's algorithm and multiplexer nodes. Technical report, U.C. Berkeley, Dept. Comp. Sci., 1999.

[Mur01] K. Murphy. Applying the junction tree algorithm to variable-length DBNs. Technical report, Comp. Sci. Div., UC Berkeley, 2001.

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

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

[Pfl98] K. Pfleger. Categorical Boltzmann Machines. Technical Report KSL-98-05, Knowledge systems lab, Computer Science Department, Stanford University, 1998.

[Pfl00] K. Pfleger. Learning Predictive Compositional Hierarchies. Technical Report KSL-01-09, Computer Science Department, Stanford University, 2000.

[Pfl01] K. Pfleger. On-line Learning of Undirected Sparse n-grams. Technical report, Computer Science Department, Stanford University, 2001.

[PPL97] S. Della Pietra, V. Della Pietra, and J. Lafferty. Inducing features of random fields. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(4), 1997.

[PR97] R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. In *NIPS*, 1997.

[PW96] D. Pynadath and M. Wellman. Generalized Queries on Probabilistic Context-Free Grammars. In *UAI*, 1996.

[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.

[RTM00] K. Rohanimanesh, G. Theocharous, and S. Mahadevan. Hierarchical map learning for robot navigation. In *AIPS Workshop on Decision-Theoretic Planning*, 2000.

[Sch00] C. Schlick. Fine's Algorithms for Hierarchical Hidden Markov Models. Technical report, FGAN Forschungsgesellschaft fur Angewandte Naturwissenschaften, 2000.

[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.

[SO92] A. Stolcke and S. M. Omohundro. Hidden Markov Model Induction by Bayesian Model Merging. In *NIPS-5*, 1992.

[SPS99] R.S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

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

[Zwe97] G. Zweig. *Speech Recognition with Dynamic Bayesian Networks.* PhD thesis, U.C. Berkeley, Dept. Comp. Sci., 1997.