

Learning Markov Processes *

Kevin P. Murphy
Department of Computer Science
University of California, Berkeley
Berkeley, CA 94720-1776. USA
Tel: (510) 642 2128
Fax: (510) 642 5775
murphyk@cs.berkeley.edu

16 May 2001

Keywords: Dynamical systems, probabilistic models, hidden Markov models, EM algorithm, Kalman filter

1 Introduction

The ability to predict the future, at least to a certain degree, is fundamental to probably all intelligent systems. Since in general we cannot hope to make perfect predictions, it makes sense to adopt a probabilistic approach (see REASONING UNDER UNCERTAINTY). For example, although we may not be able to predict the exact price of a stock tomorrow, we may be able to predict its *expected* price; deviations around this mean will be modelled as “noise”, assumed to result from all the unknown factors that were omitted from the model.

The standard way to formalise the prediction problem mathematically is to model the system to be predicted as a dynamical system (see DYNAMICAL SYSTEMS: MATHEMATICS). That is, the system is assumed to be in some state, $X_t \in \Omega$, at each time step t ; the next state of the system is determined by the state transition function, $X_{t+1} = f(X_t)$. (In this article, we restrict our attention to discrete time dynamical systems.) Typically we do not know the exact dynamics of the system, so instead we consider a probabilistic state transition function: $P(X_{t+1}|X_t)$. Such a probabilistic formulation will be particularly useful when we try to *learn* the model from data.

The state space, Ω , might be discrete (finite) or continuous (infinite). For example, we might just try to predict the probability that a stock goes up or down, in which case $\Omega = \{\uparrow, \downarrow\}$; more ambitiously, we might try to predict its expected value, in which case $\Omega = \mathbb{R}$.

In general, the state is a *vector* of state variables, which we can partition into three kinds: input variables (ones which we can control), output variables (ones which we can observe), and hidden or latent variables (internal variables which we cannot directly control or observe); we shall denote these by U_t , Y_t and X_t respectively. See Figure 1.

In this article, we shall consider how to learn models of this kind. We start by considering the special case in

*To be published in *The Encyclopedia of Cognitive Science*, Macmillan, 2002

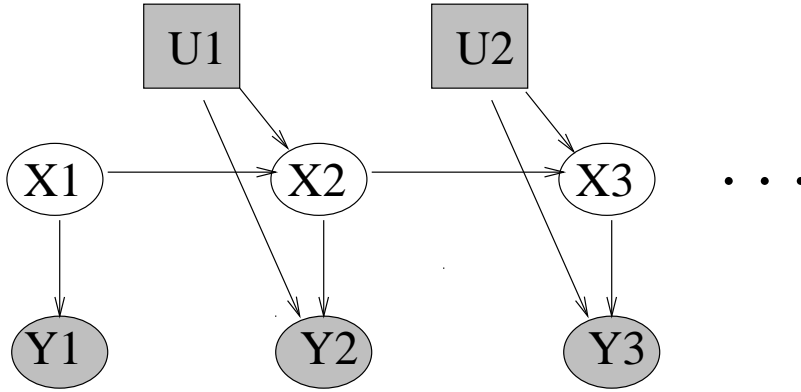


Figure 1: A generic discrete-time dynamical system represented as a dynamic Bayesian network (DBN) (see BAYESIAN BELIEF NETWORKS for a definition). U_t is the input, X_t is the hidden state, and Y_t is the output. Shaded nodes are observed, clear nodes are hidden. Square nodes are fixed inputs (controls), round nodes are random variables. Notice how what we see, Y_t , may depend on the actions that we take, U_t : this can be used to model active perception.

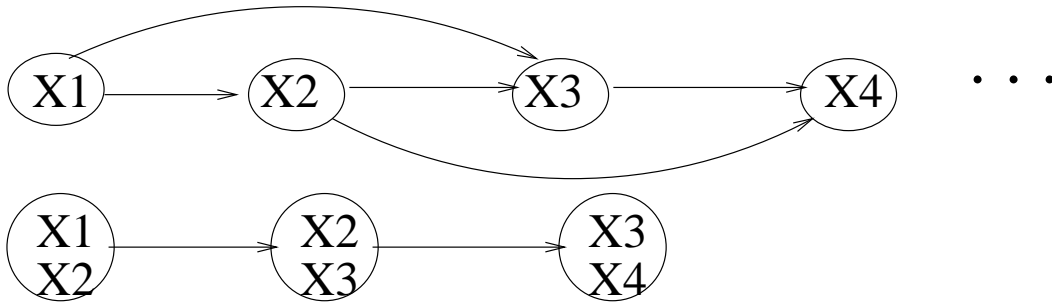


Figure 2: Converting a second order Markov model (top) into a first order Markov model (bottom).

which all variables are observed (i.e., $Y_t \equiv X_t$). We then consider the harder, but more realistic, case where we only observe part of the system.

When the goal is to predict the consequences of our actions (as in REINFORCEMENT LEARNING), we represent our intended actions as external inputs to the system, U_t . If, however, our goal is to predict the behavior of a “closed” system, we omit the input variables. In this article, we mostly focus on the latter case, but this is just for notational simplicity: the techniques generalize easily.

Finally, a remark on the (first-order) Markov property which we have implicitly assumed. This states that the current state, X_t , contains all the information needed to predict the next state, X_{t+1} , and that past states, X_{t-1} , X_{t-2} , etc. are irrelevant. (Formally, we write this as $X_{t+i} \perp\!\!\!\perp X_{t-j} | X_t$, for positive integers i, j . Informally, this says that “the future is independent of the past given the present.”) This can always be made true by augmenting the state space appropriately (see Figure 2). We will consider some examples of this below.

2 Fully observed models

2.1 Discrete state spaces: Markov chains

In a finite state Markov chain, the system can be in one of S states, $X_t \in \Omega = \{1, 2, \dots, S\}$; the probability of a transition from state i to state j is specified by a transition matrix, $A_t(i, j) \stackrel{\text{def}}{=} P(X_t = j | X_{t-1} = i)$. If A_t is independent of time, then this is called a homogeneous Markov chain; we shall assume this throughout. The initial state distribution is specified by $\pi(i) \stackrel{\text{def}}{=} P(X_1 = i)$. Since π is a probability distribution, it must satisfy the constraint that $\sum_{i=1}^S \pi(i) = 1$. Similarly, each row of A must satisfy $\sum_j A(i, j) = 1$; such a matrix is called stochastic.

A simple example of a finite state Markov chain is a bigram model, widely used in STATISTICAL APPROACHES TO NATURAL LANGUAGE PROCESSING. In this case, Ω is the set of words, and the goal is to predict the next word given the previous. An n -gram model uses the last $n - 1$ words to predict the next word, e.g., in a trigram model, the goal is to learn $P(X_t | X_{t-1}, X_{t-2})$ (see Section 2.1.3).

Another example of a finite state Markov chain is a Markov decision process (MDP), widely used in REINFORCEMENT LEARNING. In an MDP, the state transitions are “triggered” by an input U_t . If we suppose the inputs are also discrete valued, we may write $P(X_t = j | X_{t-1} = i, U_{t-1} = k) = A_k(i, j)$; that is, the input just specifies which transition matrix to use. This can be represented as in Figure 1 by omitting the observation nodes Y_t (since we are assuming that we can directly observe X_t ; if this were not the case, the model would be a *partially observed* MDP).

2.1.1 Maximum likelihood estimation

Given a training set D of sequences, the goal of learning is to find the maximum likelihood estimate of the parameters: $\hat{\theta}_{ML} \stackrel{\text{def}}{=} \text{argmax}_{\theta} P(D|\theta)$, where $\theta = (\pi, A)$. If the training data contains N independent sequences, we can express the likelihood as $P(D|\theta) = \prod_{i=1}^N P(D^{(i)}|\theta)$. To compute the likelihood of an individual sequence, consider an example: $X_{1:4} = (1, 2, 1, 2)$. This has likelihood

$$\begin{aligned} P(X|\theta) &= P(X_1 = 1)P(X_2 = 2|X_1 = 1)P(X_3 = 1|X_2 = 2)P(X_4 = 2|X_3 = 1) \\ &= \pi(1)Aa(1,2)A(2,1)A(1,2) \\ &= \pi(1)A(1,2)^2A(2,1)^1 \end{aligned}$$

In general, the likelihood is

$$P(D|\theta) = \prod_{i=1}^S \pi(i)^{\#_1(i)} \prod_{i=1}^S \prod_{j=1}^S A(i, j)^{\#(i \rightarrow j)}$$

where $\#(i \rightarrow j)$ is the number of times an i to j state transition is observed in the whole training set:

$$\#(i \rightarrow j) \stackrel{\text{def}}{=} \sum_{n=1}^N \sum_{t=2}^{T_n} I(X_{t-1}^{(n)} = i, X_t^{(n)} = j)$$

Here, $I(e)$ is the indicator function that is 1 if event e occurs, and is 0 otherwise. $\#_1(i)$ is defined analogously to be the number of times the first state is observed to be i .

To maximize the likelihood, we must introduce Lagrange multipliers to enforce the sum-to-one constraints. Some simple calculus yields the intuitive results that the parameter estimates are just the normalised counts:

$$\hat{A}_{ML}(i, j) = \frac{\#(i \rightarrow j)}{\sum_k \#(i \rightarrow k)}$$

$$\hat{\pi}_{ML}(i) = \frac{\#_1(i)}{\sum_k \#_1(k)} = \frac{\#_1(i)}{N}$$

2.1.2 Bayesian estimation

The problem with maximum likelihood (ML) estimation is that it assigns a probability of 0 to any event that was not seen in the training data; this is called the “sparse data” problem. To see why this is a problem, consider the case of bigram models of language, where Ω is the set of all words. If the training corpus contains the phrase “good dog” but not “bad dog”, the ML parameter estimates will assign a probability of 0 to any test sentence that contains “bad dog”, even though intuitively this ought to be as probable as a sentence that contains “good dog”.

A simple and well-principled solution to the sparse data problem is to compute the maximum *a posteriori* (MAP) estimate instead of the ML estimate: $\hat{\theta}_{MAP} \stackrel{\text{def}}{=} \arg \max_{\theta} P(\theta|D)$. By Bayes’ rule (see REASONING UNDER UNCERTAINTY),

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$$

or, in words,

$$\text{posterior} = \frac{\text{conditional likelihood} \times \text{prior}}{\text{likelihood}}$$

Let us assume that each row i of the transition matrix A is an independent multinomial random variable with prior $P(\theta_i)$. A suitable prior is the Dirichlet [DeGroot, 1970, Heckerman, 1998]; we write this as $P(\theta_i) = \text{Dir}(\theta_i|\alpha(i, 1), \dots, \alpha(i, S))$. The hyper parameters $\alpha(i, j)$ have an intuitive interpretation in terms of “pseudo counts”: $\alpha(i, j)$ is the number of times an $i \rightarrow j$ transition was observed in some prior “virtual” training set. Using this prior, one can show that the MAP estimate has the intuitive form

$$\hat{A}_{MAP}(i, j) = \frac{\#(i \rightarrow j) + \alpha(i, j)}{\sum_k \#(i \rightarrow k) + \alpha(i, k)}$$

Using the uninformative prior $\alpha(i, j) = 1$ (also known as Laplace’s prior) has the effect of preventing any 0’s in the estimated parameters, but otherwise letting the data “speak for itself.” It is also possible to use more complex priors, e.g., clustering words into similar categories, or using hierarchical priors [MacKay and Peto, 1995].

MAP estimation computes a point estimate of the parameter values. A fully Bayesian solution would compute a *distribution* over parameter values. This can be useful for computing the confidence in one’s estimate (which is useful for active learning), and for online learning, in which one updates the parameter estimates one step at a time.

The Dirichlet distribution is the conjugate prior to the multinomial likelihood function, which means that, after applying Bayes’ rule, the form of the posterior is also Dirichlet:

$$P(\theta_i|D) = \text{Dir}(\theta_i|\alpha(i, 1) + \#(i \rightarrow 1), \dots, \alpha(i, S) + \#(i \rightarrow S))$$

(The mode of this posterior is the MAP expression given above.) So we see that sequential Bayesian updating consists of simply updating the pseudo counts using the observed transitions. See [Strens, 2000] for an application of this to MDPs, where online learning is particularly important.

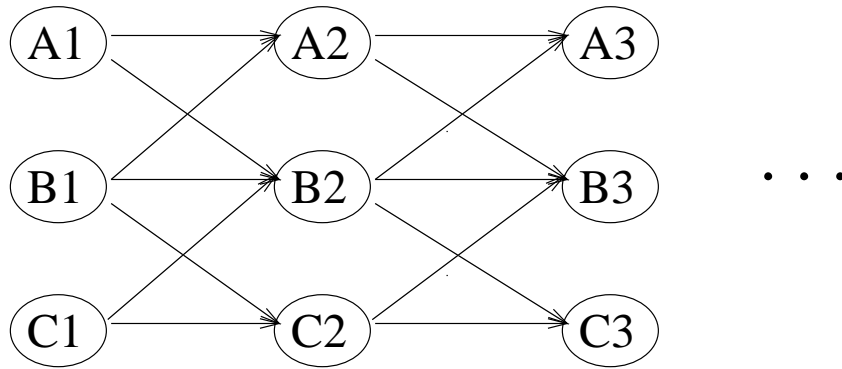


Figure 3: Three coupled Markov chains.

2.1.3 Higher order Markov chains

As was mentioned in the introduction, a k 'th order Markov model can always be converted to a first-order Markov model by creating a “mega variable” $\tilde{X}_t = (X_t, X_{t-1}, \dots, X_{t-k})$. Unfortunately, the size of the state space is now S^k , which means the number of parameters that we need to estimate has gone up exponentially. One way to ameliorate this is to use n -gram smoothing or a hierarchical prior [MacKay and Peto, 1995]. Another is to approximate the conditional distribution $P(X_t | X_{t-1}, \dots, X_{t-k})$ as a mixture of k bigram models [Saul and Jordan, 1999]. We can learn the parameters of such a mixture model using the EM algorithm (see Section 3.1).

The problem of choosing the appropriate number of steps of history to maintain, k , is called *model order determination*, and is a simple example of *model selection*. The basic problem is that we can get a better fit to our training data by using larger k (since the number of parameters increases with k), but this may result in overfitting (see MACHINE LEARNING for a discussion of this point).

One approach is to choose the best model by cross-validation. A computationally cheaper approach is to choose the model M that maximizes a penalized log-likelihood function: $M^* = \arg \max_M \log P(D | \hat{\theta}_M) - g(d_M)$, where $\hat{\theta}_M$ is the ML estimate of the parameters for model M , d_M is the number of parameters in M , and $g(d)$ is some complexity penalty that increases with d . Some popular choices for this penalty function are the Akaike information criterion (AIC), $g(d) = d$, the Bayesian information criterion (BIC), $g(d) = \frac{d}{2} \log N$, where N is the number of samples, and the minimum description length (MDL) criterion, which is the same as BIC [Heckerman, 1998].

2.1.4 Factorial Markov chains

Another way that a Cartesian state space can arise is if the state X_t is in fact composed of several discrete state variables. For example, Figure 3 illustrates the case where $X_t = (A_t, B_t, C_t)$. This situation frequently arises in AI applications, where the state of the world is represented using fluents; this kind of representation is known as a dynamic Bayesian network (DBN) [Dean and Wellman, 1991]. If there are n variables in the state vector, each of which has S possible values, then a fully interconnected model requires $O(S^n)$ parameters. However, if each variable has at most k parents, we only need $O(nS^k)$ parameters. In other words, sparse models are easier to learn (in terms of the number of samples needed, or sample complexity).

2.2 Continuous state spaces

The most common form for a Markov process with a continuous state space is $X_t = f(X_{t-1}, U_{t-1}) + W_t$ where f is some function, and W_t is a zero-mean random variable, representing noise.

2.2.1 Linear systems

A widely studied special case is when f is a *linear* function, and the noise is Gaussian, in which case we can write $X_t = AX_{t-1} + BU_{t-1} + W_t$ where $W_t \sim N(0, \Sigma)$. Equivalently, we may write

$$P(X_t = x_t | X_{t-1} = x, U_{t-1} = u) = N(x_t; Ax + Bu, \Sigma).$$

where the notation $N(x; \mu, \Sigma)$ means evaluating a Normal (Gaussian) probability density function with mean μ and covariance Σ at the point (vector) x . The distribution of the initial state is $P(X_1 = x) = N(x; \mu_1, \Sigma_1)$. It is possible to estimate the parameters of such a linear system using techniques based on linear regression [Ljung, 1987].

2.2.2 Non-linear systems

The usual approach to learning non-linear dynamical systems is to represent the f function using a feedforward NEURAL NETWORK (a.k.a. a MULTI-LAYER PERCEPTRON). We create a training set from the matrices

$$X = \begin{pmatrix} X'_1 & U'_1 \\ \vdots & \\ X'_{T-1} & U'_{T-1} \end{pmatrix}, \quad Y = \begin{pmatrix} X'_2 \\ \vdots \\ X'_T \end{pmatrix}$$

where the t 'th row of X is the t 'th input vector, and the t 'th row of Y is the corresponding output. (X'_t denotes the transpose of the vector X_t .) Now we use some non-linear learning procedure, such as backpropagation or a conjugate gradient method [Bishop, 1995], to estimate the parameters of the f function; Σ can be set to the empirical covariance matrix of the residuals.

2.2.3 Higher order models

It is easy to convert a k 'th-order linear model to a first-order model. For example, consider the following auto-regressive process of order 2:

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

This can be represented as a first-order model as follows:

$$\tilde{X}_t \stackrel{\text{def}}{=} \begin{pmatrix} X_t \\ X_{t-1} \end{pmatrix} = \begin{pmatrix} A_1 & A_2 \\ I & 0 \end{pmatrix} \begin{pmatrix} X_{t-1} \\ X_{t-2} \end{pmatrix} \stackrel{\text{def}}{=} \tilde{A} \tilde{X}_{t-1}$$

Now we should estimate each block in \tilde{A} separately. (Note that, in contrast to the case of discrete state spaces discussed in Section 2.1.3, the number of parameters increases *linearly* with k , not exponentially. The most common way to choose k is to use the AIC scoring metric discussed in Section 2.1.3; this is built-in to many software packages.

For a non-linear model, we can use a tapped delay line, i.e., we feed in a window of the last k time slices into f . (Note that this does not exploit the prior knowledge that successive input vectors (windows) are overlapping, and hence highly correlated.) In this case, k is usually chosen by cross-validation.

3 Partially observed models

Consider trying to predict a speech signal, represented, for example, as a m -dimensional vector encoding the instantaneous power of the signal in k different frequency bands (typically $m \approx 20$). One approach would be to build a non-linear model to directly predict Y_{t+1} as a function of Y_t, Y_{t-1} , etc. Such a model would need a very large number of parameters.

An alternative approach would be to try to infer the word that the person is saying, based on the acoustic evidence, then predict the next word that they are likely to say, and finally predict the corresponding sound. (More realistically, we would work with phonemes instead of words, since phonemes have less acoustic variability, and suffer less from the sparse data problems mentioned in Section 2.1.2.) This is the basis of the hidden Markov model (HMM) approach to speech recognition [Rabiner, 1989], which has proved to be quite successful in practice. We will discuss how to learn such models in Section 3.2 below.

Another reason to consider models with hidden variables is that often we are more interested in inferring the hidden causes of a signal than in predicting the signal itself. As in Bayesian networks (see BAYESIAN BELIEF NETWORKS), the approach is to build a generative model of the observed signal (a mapping from X_t to Y_t : see Figure 1), and then to apply Bayes' rule to "invert the causal arrow", i.e., to infer $P(X_t|y_{1:t})$. We will discuss how to do this below.

3.1 The EM algorithm

Parameter learning (also known as system identification) in the case of partially observed models is much harder than in the case of fully observed models because the likelihood surface has multiple maxima. One approach is to use gradient ascent (e.g., [Binder et al., 1997]), but there is a simpler and more effective algorithm: expectation maximization (EM) algorithm [Dempster et al., 1977, Neal and Hinton, 1998].

EM is guaranteed to converge to a local maximum of the likelihood (or MAP) surface, and works roughly as follows. We first perform probabilistic inference to try to estimate the hidden state given the observation sequence (this is the E step). We then use the estimated values of the hidden variables as if they were observed, in order to update the parameters using the techniques discussed in Section 2 (this is the M step). Since the inferred values depends on the parameter settings, we then need to re-estimate the hidden values, and then re-estimate the parameters, etc. Hence learning calls inference as a subroutine. (Gradient-based learning methods also use an inference algorithm [Binder et al., 1997], but have the disadvantage of requiring a step-size parameter and a way to enforce constraints on the parameters.) We will explain the EM algorithm in more detail below.

3.2 Discrete state spaces: hidden Markov models

A hidden Markov model (HMM) is a partially observed Markov process. The dynamics of the hidden variable X_t are modelled using a standard finite-state Markov chain (see Section 2.1); the observed variable Y_t is some function of X_t (as in Figure 1, but without the inputs U_t). For example, in speech recognition, in which $Y_t \in \mathbb{R}^m$, we assume that the distribution of Y_t is Gaussian (or perhaps a mixture of Gaussians), whose parameters are determined by the hidden state X_t : $P(Y_t = y|X_t = i) = N(y; \mu_i, \Sigma_i)$. If Y_t is discrete, we specify the observation model using another matrix: $P(Y_t = j|X_t = i) = B(i, j)$.

The complete log-likelihood of a sequence is given by

$$\log P(X_{1:T}, Y_{1:T}) = \log P(X_1) + \sum_{t=1}^T \log P(Y_t | X_t) + \sum_{t=2}^T \log P(X_t | X_{t-1})$$

Using the trick discussed in Section 2.1.1, we may rewrite the last term as

$$\sum_{t=2}^T \sum_{i=1}^S \sum_{j=1}^S I(X_{t-1} = i, X_t = j) \log A(i, j)$$

and similarly for the other terms. Taking expectations with respect to the hidden variables X_t , using the current parameter settings θ , we find

$$E \log P(X_{1:T}, Y_{1:T}) = \dots + \sum_{i=1}^S \sum_{j=1}^S E \left[\sum_{t=2}^T I(X_{t-1} = i, X_t = j) \right] \log A(i, j)$$

where we have omitted terms not involving A for simplicity. The terms inside the square brackets are the expected number of times we see an $i \rightarrow j$ transition:

$$E(i, j) \stackrel{\text{def}}{=} E \left[\sum_{t=2}^T I(X_{t-1} = i, X_t = j) \right] = \sum_{t=2}^T P(X_{t-1} = i, X_t = j | y_{1:T})$$

If we could compute these expected sufficient statistics (ESS), we could update the parameters to

$$\hat{A}_{ML} = \frac{E(i, j)}{\sum_k E(i, k)}$$

by analogy with the fully observed case. We could then use the new parameters to recompute the ESS, and so on, until convergence. (Note: in the case of HMMs, the EM algorithm is also known as the Baum-Welch algorithm.)

3.2.1 The forwards-backwards algorithm

We will now explain how to compute $\gamma_t(i) \stackrel{\text{def}}{=} P(X_t = i | y_{1:T})$ and $\xi_t(i, j) \stackrel{\text{def}}{=} P(X_{t-1} = i, X_t = j | y_{1:T})$ in $O(TS^2)$ time and space using dynamic programming.

In the forwards pass, we compute

$$\begin{aligned} \alpha_t(j) &\stackrel{\text{def}}{=} P(X_t = j, y_{1:t}) \\ &= P(y_t | X_t = j, y_{1:t-1}) P(X_t = j | y_{1:t-1}) \\ &= P(y_t | X_t = j) \sum_i P(X_t = j | X_{t-1} = i, y_{1:t-1}) P(X_{t-1} = i | y_{1:t-1}) \\ &= B(j, y_t) \sum_i A(i, j) \alpha_{t-1}(i) \end{aligned}$$

Notice how this algorithm contains two key steps: computing the one step-ahead prediction, $P(X_t | y_{1:t-1})$, and combining it with the likelihood $P(y_t | X_t)$ to get the updated estimate $P(X_t | y_{1:t})$.

If we make a diagonal matrix B_t in which $B_t(j, j) \stackrel{\text{def}}{=} B(j, y_t)$, we may rewrite each step of the algorithm as a simple vector-matrix multiplication: $\alpha_t = B_t A' \alpha_{t-1}$.

In the backwards pass, we compute

$$\begin{aligned}
\beta_t(i) &\stackrel{\text{def}}{=} P(y_{t+1:T}|X_t = i) \\
&= \sum_j P(X_{t+1} = j|X_t = i)P(y_{t+1:T}|X_t = i, X_{t+1} = j) \\
&= \sum_j P(X_{t+1} = j|X_t = i)P(y_{t+1}|X_{t+1} = j)P(y_{t+2:T}|X_{t+1} = j) \\
&= \sum_j A(i, j)B(j, y_{t+1})\beta_{t+1}(j)
\end{aligned}$$

Again, we may rewrite this as a simple vector-matrix multiplication: $\beta_t = AB_{t+1}\beta_{t+1}$.

Finally, we combine the results of the forwards and backwards steps to obtain

$$\gamma_t(i) \stackrel{\text{def}}{=} P(X_t = i|y_{1:T}) \propto P(y_{t+1:T}|X_t = i, y_{1:t})P(X_t = i, y_{1:t}) = \beta_t(i)\alpha_t(i)$$

and

$$\begin{aligned}
\xi_t(i, j) &\stackrel{\text{def}}{=} P(X_{t-1} = i, X_t = j|y_{1:T}) \\
&\propto P(X_{t-1} = i|y_{1:t-1})P(X_t = j|X_{t-1} = i)P(y_t|X_t = j)P(y_{t+1:T}|X_t = j) \\
&= \alpha_{t-1}(i)A(i, j)B_t(j, j)\beta_t(j)
\end{aligned}$$

The ESS can be computed using $E(i, j) = \sum_{t=2}^T \xi_t(i, j)$, and similarly for the other terms.

3.2.2 The Viterbi algorithm

A useful task in speech recognition and other applications is to compute the most probable state sequence, $\arg\max P(X_{1:T}|y_{1:T})$. This can be done using the Viterbi algorithm, which is identical to the forwards algorithm, except that one replaces the \sum operator with \max [Rabiner, 1989].

3.2.3 Classifying sequences using HMMs

The normalization constant used to compute γ_t is equal to the likelihood of the data:

$$P(y_{1:T}) = \sum_i P(y_{t+1:T}|X_t = i, y_{1:t})P(X_t = i, y_{1:t})$$

This is a useful quantity, because it can be used to classify a sequence. For example, suppose we train up 10 HMMs, one for each spoken digit 0 to 9; given a test sequence $y_{1:T}$, we compute the likelihood that HMM i generated $y_{1:T}$; we then classify $y_{1:T}$ as digit i if HMM i has higher likelihood than any of the rival models.

3.2.4 Factored state spaces

We can create partially observed versions of the kind of factored models discussed in Section 2.1.4, which yields such models as coupled HMMs and factorial HMMs (see Figure 4). Unfortunately, since the state space is exponential in the number of variables, exact inference in these models is computationally intractable (requiring $O(TS^{2n})$ time and space, which is exponential in n). We therefore have to use *approximate* inference (see e.g., [Ghahramani, 1998]).

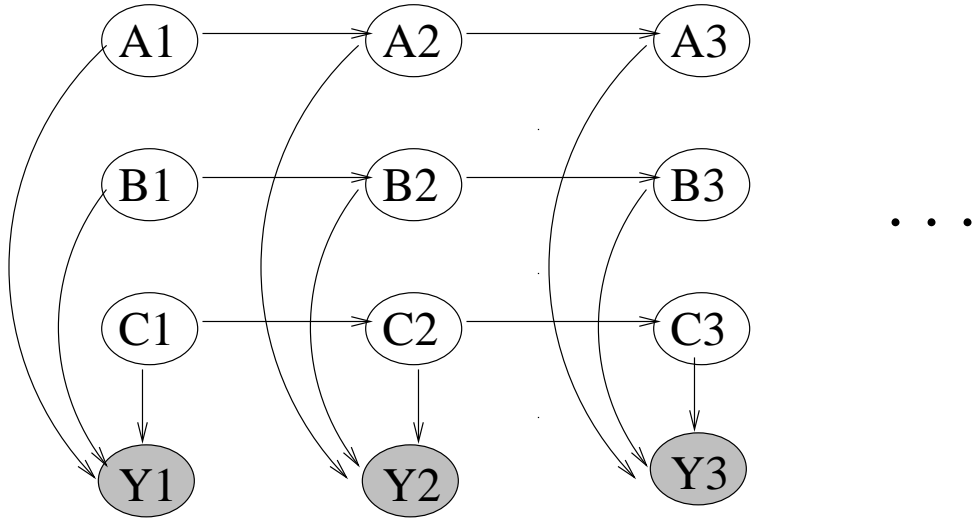


Figure 4: A factorial hidden Markov model. The hidden state is represented in a distributed fashion, in terms of three *a priori* independent discrete random variables, $X_t = (A_t, B_t, C_t)$. However, after observing the output Y_t , the hidden variables all become dependent, because of the explaining away phenomenon (see BAYESIAN BELIEF NETWORKS). Hence the cost of exact inference in this model grows exponentially with the number of hidden variables (chains).

3.3 Continuous state spaces

3.3.1 Linear systems and the Kalman filter

A linear dynamical system (LDS) has the generic form

$$X_t = AX_{t-1} + BU_{t-1} + W_t$$

$$Y_t = CX_t + DU_t + V_t$$

where $W_t \sim N(0, Q)$ and $V_t \sim N(0, R)$ are independent random variables representing Gaussian noise (see Figure 1).

The Kalman filter is an algorithm to compute $P(X_t|y_{1:t}, u_{1:t})$ in an LDS. It is entirely analogous to the forwards algorithm for HMMs [Minka, 1999]. (In particular, it contains the same kind of predict-update cycle, which, it has been claimed [Rao, 1997], has analogs in the brain in terms of top-down and bottom-up visual processing.) For offline learning, one can get better performance by estimating the hidden states based on “future” data, as well as past, i.e., by using the “smoothed” quantities $P(X_t|y_{1:T}, u_{1:T})$. The Rauch-Tung-Striebel (RTS) algorithm is a way to compute $P(X_t|y_{1:T}, u_{1:T})$ in an LDS, and is entirely analogous to the forwards-backwards algorithm for HMMs.¹ To do parameter estimation in an LDS, we can use the EM algorithm, using the RTS algorithm as a subroutine to compute the required expected sufficient statistics (see [Ghahramani, 1998] for details).

¹In the general case, the RTS algorithm takes $O(Tm^3)$ time and $O(Tn^2)$ space, where $Y_t \in \mathbb{R}^m$ and $X_t \in \mathbb{R}^n$, because at each step, the algorithm must invert an $m \times m$ matrix and must store an $n \times n$ covariance matrix.

3.3.2 Non-linear systems

Inference, and therefore learning, is much harder in systems which are nonlinear and/or have non-Gaussian noise, and one typically has to resort to approximate methods. A popular method for computing $P(X_t|y_{1:t})$ online is particle filtering [Doucet et al., 2001] or the extended Kalman filter [Bar-Shalom and Fortmann, 1988]; for offline computation of $P(X_t|y_{1:T})$, one can use Gibbs sampling [Gilks et al., 1996] or the extended Kalman smoother [Roweis and Ghahramani, 2001]. These inference routines can be used as subroutines for the E step of EM, as before.

3.4 Bayesian estimation

EM and gradient ascent both attempt to compute a point-estimate (either ML or MAP) of the parameters. In the Bayesian approach, we treat the parameters as *random variables*, and attempt to compute a distribution over them. This is particularly useful for non-stationary processes, since we can compute $P(\theta_t|y_{1:t})$ online. Unfortunately, models in which the parameters are represented as hidden variables fall into the continuous-state, non-linear category², for which inference is hard, as discussed in Section 3.3.2.

4 Conclusion

We have discussed a variety of increasingly complex models and learning algorithms, ranging from simple counting in finite-state Markov chains, to the iterative EM algorithm for partially observed, continuous-state Markov models. Nevertheless, all of the models have used propositional representations of the state (i.e., fixed-size vectors). To work at a higher level of abstraction, we need models that use first-order representations, i.e., that represent the world in terms of objects and relations. How to efficiently learn and reason with such first-order probabilistic dynamic models is one of the major open problems in AI (see [Pasula and Russell, 2001] for a first attempt).

References

- [Bar-Shalom and Fortmann, 1988] Bar-Shalom, Y. and Fortmann, T. (1988). *Tracking and data association*. Academic Press.
- [Binder et al., 1997] Binder, J., Koller, D., Russell, S. J., and Kanazawa, K. (1997). Adaptive probabilistic networks with hidden variables. *Machine Learning*, 29:213–244.
- [Bishop, 1995] Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Clarendon Press.
- [Dean and Wellman, 1991] Dean, T. L. and Wellman, M. P. (1991). *Planning and Control*. Morgan Kaufmann.
- [DeGroot, 1970] DeGroot, M. (1970). *Optimal Statistical Decisions*. McGraw-Hill.
- [Dempster et al., 1977] Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *J. of the Royal Statistical Society, Series B*, 34:1–38.

²To see this, consider the linear system $X_{t+1} = AX_t$. If we add A to the state space to get $\tilde{X}_t = (X_t \ A)'$, then the update equation becomes $\tilde{X}_{t+1} = f(\tilde{X}_t)$, where f is a non-linear function.

- [Doucet et al., 2001] Doucet, A., de Freitas, N., and Gordon, N. J. (2001). *Sequential Monte Carlo Methods in Practice*. Springer Verlag.
- [Ghahramani, 1998] Ghahramani, Z. (1998). Learning Dynamic Bayesian Networks. In Giles, C. and Gori, M., editors, *Adaptive Processing of Sequences and Data Structures. Lecture Notes in Artificial Intelligence*, pages 168–197. Springer-Verlag.
- [Gilks et al., 1996] Gilks, W., Richardson, S., and Spiegelhalter, D. (1996). *Markov Chain Monte Carlo in Practice*. Chapman and Hall.
- [Heckerman, 1998] Heckerman, D. (1998). A tutorial on learning with Bayesian networks. In Jordan, M., editor, *Learning in Graphical Models*. MIT Press.
- [Ljung, 1987] Ljung, L. (1987). *System Identification: Theory for the User*. Prentice Hall.
- [MacKay and Peto, 1995] MacKay, D. and Peto, L. (1995). A hierarchical Dirichlet language model. *Natural Language Engineering*, 1(3):1–19.
- [Minka, 1999] Minka, T. (1999). From Hidden Markov Models to Linear Dynamical Systems. Technical report, MIT.
- [Neal and Hinton, 1998] Neal, R. M. and Hinton, G. E. (1998). A new view of the EM algorithm that justifies incremental and other variants. In Jordan, M., editor, *Learning in Graphical Models*. MIT Press.
- [Pasula and Russell, 2001] Pasula, H. and Russell, S. (2001). Approximate inference for first-order probabilistic languages. In *IJCAI*.
- [Rabiner, 1989] Rabiner, L. R. (1989). A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proc. of the IEEE*, 77(2):257–286.
- [Rao, 1997] Rao, P. (1997). Kalman filter model of the visual cortex. *Neural Computation*, 9(4):721–763.
- [Roweis and Ghahramani, 2001] Roweis, S. and Ghahramani, Z. (2001). Learning Nonlinear Dynamical Systems using the EM Algorithm. In Haykin, S., editor, *Kalman Filtering and Neural Networks*. Wiley.
- [Saul and Jordan, 1999] Saul, L. and Jordan, M. (1999). Mixed memory markov models: Decomposing complex stochastic processes as mixture of simpler ones. *Machine Learning*, 37(1):75–87.
- [Strens, 2000] Strens, M. (2000). A Bayesian framework for reinforcement learning. In *Intl. Conf. on Machine Learning*.