Encyclopedia of Cognitive Science ECS-40A
Learning Markov Processes

Dr. 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

20 June 2002

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

Contents

Summary

Markov models are probabilistic models of dynamical systems, which can be used to predict the future and the consequences of one's actions. We discuss how to learn such models from time-series data.

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. Similarly, although we may not be able to exactly predict the next word that someone will speak or type at a keyboard, we can predict a set of likely or probable words based on what we have heard/seen so far.

It is standard 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, since although no model may fit the data perfectly, some models might be more probable than others. The state space, Ω, might be discrete (finite) or continuous (infinite). For example, we might try to predict the probability that a stock goes up or down, in which case $\Omega = \{\uparrow, \downarrow\}$, or we might try to predict its expected value, in which case $\Omega = \mathbb{R}$. Of course, the state X_t may also be vector-valued.

A (first-order) Markov process is one such that $P(X_{t+h}|X_{1:t}) = P(X_{t+h}|X_t)$, where $X_{1:t} = X_1, \ldots, X_t$ is the past observation sequence and $h > 0$ is the amount of lookahead (the prediction horizon). This is often paraphrased as: the future X_{t+h} is independent of the past $X_{1:t-1}$ give the present X_t. A k'th order Markov process is such that $P(X_{t+h}|X_{1:t}) = P(X_{t+h}|X_{t-k+1:t})$. $X_{t-k+1:t}$ is called a sufficient statistic, since it can be used to predict the future with the same accuracy as would be obtained if we conditioned on the whole sequence of past observations. A k'th order Markov model can always be converted to a first-order Markov model by creating a "mega variable" containing the last k observations (a sliding window). For example, if the system is second-order Markov, we can convert it to first-order by defining a new state-space, $\tilde{X}_t = (X_t, X_{t-1})$, and set $P(\tilde{X}_t = (x_t, x_{t-1})|\tilde{X}_{t-1} = (x'_{t-1}, x_{t-2})) = \delta(x_{t-1}, x'_{t-1})P(x_t|x_{t-1}, x_{t-2})$. The δ function ensures that $\tilde{X}_t$ and $\tilde{X}_{t-1}$ assign the same value to the variables that they share (in this case, X_{t-1}).

Often a system has no finite (or reasonably small) sufficient statistic, which means we need to base our predictions on all the past data; this is clearly impractical, especially for high-dimensional data such as audio or video signals. However, we may posit the existence of a hidden or latent variable, which generates the observations at each step, and whose transition dynamics do satisfy the (first-order) Markov property. This is called a hidden Markov model (HMM) if the latent variable is discrete, or a state-space model if the latent variable is continuous. We will denote the hidden variable by X_t and the observed variable by Y_t. (Since the value of X_t is hidden, the model is called "partially observed".) In addition to the transition function $P(X_t|X_{t-1})$, we must now specify the observation function, $P(Y_t|X_t)$. We will explain this in more detail below.

In addition to hidden and observed variables, we may have control or input variables U_t. In this case, the transition function becomes $P(X_t|X_{t-1}, U_t)$ and the observation model becomes $P(Y_t|X_t, U_t)$. (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.) The resulting model can be represented as a BAYESIAN NETWORK, as shown in Figure 1.

[Figure 1 about here]

In the following sections, we will explain how to learn Markov models from data. We start with the case where the system is fully observed, and then consider the harder case in which X_t is hidden, and we only

get to observe Y_t, which is a probabilistic function of X_t.

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, $X_t \in \Omega = \{1, 2, \ldots, S\}$; 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 or stationary 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 satisy $\sum_j A(i,j) = 1$; 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 2.1.2).

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 Parameter estimation

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

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

In general, the likelihood is

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

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

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

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.

Maximizing the log-likelihood is equivalent to maximizing the likelihood (since log is a monotonic transformation), but is mathematically more convenient. The log-likelihood of a sequence is given by

$$\log P(D|\theta) = \sum_{i=1}^{S} \#_1(i) \log \pi(i) + \sum_{i=1}^{S} \sum_{j=1}^{S} \#_{i \to j} \log A(i,j)$$

To maximize the log-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:

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

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

One 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,

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

If we use a suitable prior $P(\theta)$, it can be shown that the MAP estimate of the transition matrix of a Markov chain is given by

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

where the $\alpha(i,j)$ have an intuitive interpretation in terms of "pseudo counts", i.e., $\alpha(i,j)$ is the number of times an $i \to j$ transition was observed in some prior "virtual" training set. 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."

We can estimate the parameters of a k'th order Markov model in a similar way. Unfortunately, the number of parameters that we need to specify $P(X_t|X_{t-k:t-1})$ is now $O(S^k)$, making the sparse data problem particularly severe. One solution is to approximate the conditional distribution $P(X_t|X_{t-k:t-1})$ 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). Another solution is to use more complex parameter priors [MacKay and Peto, 1995], or to to cluster words into similar categories; in this case, the category is a hidden variable, making the model partially observable.

2.1.2 Model selection

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*. Maximum likelihood is not an appropriate metric, since the model with the highest likelihood on the training set will always be the one with the largest possible value of k (since this has the largest number of parameters, and hence can fit the data the best: this is called overfitting). One approach to model selection is to use cross-validation, i.e., to choose k by measuring performance on a hold-out set (as opposed to the training set). Alternatively, we can can 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].

2.2 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. The ML estimate of Σ is the empiricial covariance matrix of the residuals (i.e., the difference between the predicted output and the actual output).

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 is called auto-regressive since X_t is computed by using linear regression applied to "older" values of itself.) 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 2.1.2, 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 2.1.2; 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 m 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. (In practice, it is more common to work with phonemes instead of words, since phonemes have less acoustic variability, and suffer less from the sparse data problems mentioned in Section 2.1.1.) This is the basis of the hidden Markov model (HMM) approach to speech recognition [Rabiner, 1989, Jelinek, 1997], 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), a good approach is to build a generative model of the observed signal (a mapping from X_t to Y_t) 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 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 often more effective algorithm called expectation maximization (EM) [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, repeating this until convergence. We will explain the EM algorithm in more detail below. (Note: when applied to HMMs, EM is also known as the Baum-Welch algorithm.)

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. 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.

We cannot maximize this directly, since X_t is hidden. Instead, we will try to maximize the expected complete-data log-likelihood. Taking expectations with respect to the hidden variables X_t, using the current parameter settings θ, we find

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

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 $i \to j$ 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.

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 $O(TS)$ space using dynamic programming.

In the forwards pass, we compute

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

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

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

In the backwards pass, we compute

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

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

3.2.2 The Viterbi algorithm

A useful task in a variety of applications is to compute the most probable state sequence given the observations: $\arg\max P(X_{1:T}|y_{1:T})$. For example, in speech recognition, this might be the most likely sequence of

words given the acoustic signal, and in biology, it might be the most likely alignment of a protein sequence to some model. This quantity can be computed using the Viterbi algorithm, which is like forwards-backwards except that we replace the $\sum$ operator with max in the forwards pass, and do pointer-following in the backwards pass: see [Rabiner, 1989] for details.

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

[Figure 2 about here.]

An HMM represents the state of the system using a single discrete random variable X_t. To represent k bits of information, we need to use 2^k values; this results in high computational complexity (i.e., inference using forwards-backwards is slow) and high sample complexity (i.e., the amount of data needed by EM is large). An alternative is to use a distributed representation of state, i.e., to use k binary variables. Such a model is called a factorial HMM [Ghahramani and Jordan, 1997], and is shown in Figure 2. This model has exponentially fewer parameters, and hence is easier to learn. Unfortunately, inference is still slow, because the k hidden random variables become correlated, acting like a single large random variable. However, it is possible to exploit the structure of the model to enable efficient approximations.

A factorial HMM is an example of a more general class of models called dynamic Bayesian networks; see [Dean and Wellman, 1991, Ghahramani, 1998, Murphy, 2002] for details.

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.

The Kalman filter is an algorithm to compute $P(X_t|y_{1:t}, u_{1:t})$ in an LDS. It is 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 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 [Roweis and Ghahramani, 1999] for details).

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 $P(X_t|y_{1:t})$ 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.

[1] In 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.

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. and Wellman, M. (1991). *Planning and Control.* Morgan Kaufmann.

[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.

[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.

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

[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.

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

[Ljung, 1987] Ljung, L. (1987). *System Identificiation: 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.

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

[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.

[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, 1999] Roweis, S. and Ghahramani, Z. (1999). A Unifying Review of Linear Gaussian Models. *Neural Computation*, 11(2).

[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.

Figures

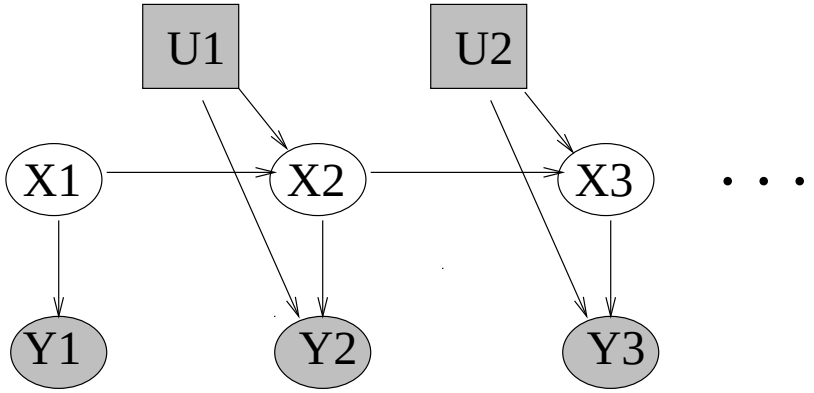

Figure 1: A generic discrete-time dynamical system represented as a dynamic Bayesian network (DBN) (see BAYESIAN BELIEF NETWORKS for a definition). U_t is the input, X_t is the hidden state, and Y_t is the output. Shaded nodes are observed, clear nodes are hidden. Square nodes are fixed inputs (controls), round nodes are random variables.

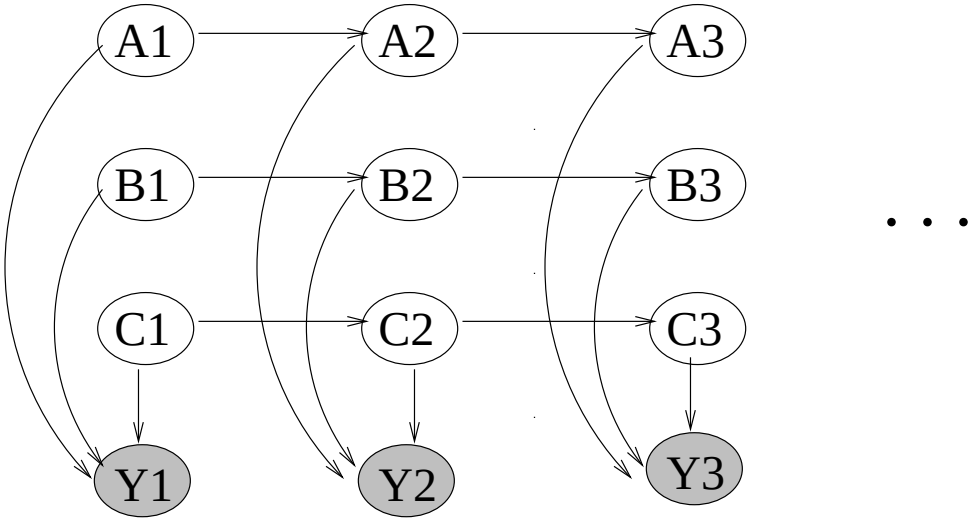

Figure 2: A factorial HMM. The hidden state is represented in a distributed fashion, in terms of three discrete random variables, $X_t = (A_t, B_t, C_t)$.

Cross references

Automatic speech recognition, Bayesian belief networks, Bayesian learning in games, Computational morphology, Dynamical systems: mathematics, Finite state processing, Language learning (compuational models), Machine learning, Reasoning under uncertainty, Reinforcement learning, Speech recognition, Statistical approaches to natural language processing Statistical pattern recognition

Document software

This document was prepared with LaTeX.