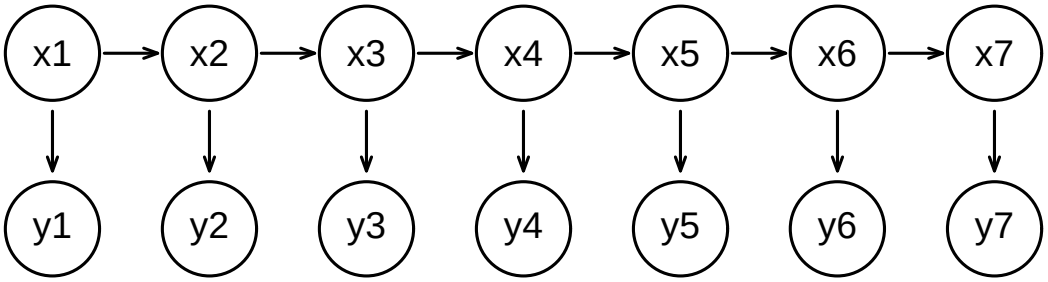
BCCS 2008/09: Graphical models and complex stochastic systems: Lecture 7: Hidden Markov models and State space models

These classes of models, which are often treated as distinct, are really two flavours of a similar idea, and they collectively provide a flexible way of modelling dependent random systems evolving in 'time', where 'time' may be time, or linear position, or location in a genome, or ... The linear structure means that short cuts can often be taken in computing inferences.

The key idea is that there are two sequences – a 'hidden' one $\{x_t, t = 0, 1, 2, \ldots\}$ and an observed one $\{y_t, t = 0, 1, 2, \ldots\}$. The structure in the system is provided by assuming that $\{x_t\}$ is a Markov chain, and the simplicity from assuming that only x_t has a *direct* influence of y_t; loosely, y_t is a 'noisy version' of x_t.

What this means essentially is that the dependence is in the system, rather than the observation process, and this is realistic in very many applications.

In the language of graphical modelling, these models are represented by this generic DAG:



7.1 Hidden Markov models

In HMM's as usually defined, the distinctive feature is that x_t has a finite state space. Sometimes the states are known, but not always. Usually, the transition probabilities between the states are unknown. In a *finite* HMM, the values of y_t are also in a finite set, so that everything is discrete. In a *normal* HMM, the distribution of y_t given x_t is normal.

Examples .

- communication channels
- DNA and protein sequencing
- ion channels
- speech recognition

7.2 State space models

As usually defined, the term state space models covers cases where the process x_t is continuous-valued. The classic version is the gaussian linear state space model

$$\begin{aligned} x_{t+1} &= ax_t + ru_t \\ y_t &= bx_t + sv_t \end{aligned}$$

where a, r, b, s are constants (known or unknown) and u_t and v_t are independent sequences of i.i.d. normal random variables. In many applications, these quantities are all vectors and matrices.

Examples .

- automatic control, signal processing
- time series, econometrics
- tracking

7.3 Conditional independences

These can be read off the DAG above, straightforwardly. The key properties are that for any times s, t,

- $y_s \perp\!\!\!\perp y_t \mid (x_s, x_t)$
- $y_t \perp\!\!\!\perp x_s \mid x_t$

(both of these extend to more than two time points).

7.4 Filtering, smoothing and prediction

In many applications, where t represents actual time, we will wish to make *online* inference, that is to report what we know about the x process immediately after each y_t is observed. *Filtering* refers to estimating x_t, given $y_{\leq t} \equiv y_t, y_{t-1}, y_{t-2}, \ldots$ *Smoothing* refers to estimating x_s for some $s < t$, given $y_{\leq t}$. *Prediction* refers to estimating x_s for some $s > t$, given $y_{\leq t}$.

Even when there is no requirement to do inference online, it may still be an attractive option since it may be much cheaper to compute (say) $p(x_t|y_{\leq t})$ (filtering) than $p(x_t|\text{all } y)$, although this means throwing information away (since it is not true that $x_t \perp\!\!\!\perp y_{>t} \mid y_{\leq t}$).

7.5 Kalman filtering

For the gaussian linear state space model of section 7.2, and vector generalisations of it, there is a well-known and long-standing algorithm called the Kalman filter for computing $p(x_t|y_{\leq t})$; because in a multivariate normal (gaussian) distribution, all conditional distributions are also normal, all that the algorithm needs to do is compute the mean and variance of the filtering distribution, that is $m_t = E(x_t|y_{\leq t})$ and $w_t = \text{var}(x_t|y_{\leq t})$. These can be calculated by the following recursion:

$$\begin{aligned} m_t &= \frac{s^2 a m_{t-1} + (a^2 w_{t-1} + r^2) b y_t}{s^2 + (a^2 w_{t-1} + r^2) b^2} \\ w_t &= \frac{s^2 (a^2 w_{t-1} + r^2)}{s^2 + (a^2 w_{t-1} + r^2) b^2} \end{aligned}$$

There are many different equivalent ways of writing this, and of course in the vector case the expressions involve matrices and look more complicated.

The Kalman filter is probably one of the most-often used algorithms in the whole of electronic engineering.

If the state-space model is not gaussian, and/or not linear, there is no general recursive formula for $p(x_t|y_{\leq t})$. Various adaptations of the idea have been devised to solve the filtering problem approximately. In recent years, the idea of *particle filtering*, where the distributions are represented by large random samples, and the calculations are all done by simulation, has become very popular.

7.6 Forwards/backwards recursions

We can read off the joint distribution of all variables from the DAG: letting $x = (x_0, x_1, \ldots, x_T)$ and $y = (y_1, y_2, \ldots, y_T)$ (note that we begin x at $t = 0$), we have

$$p(x, y) = p(x_0) \prod_{t=1}^{T} [p(x_t|x_{t-1})p(y_t|x_t)],$$

assuming there are no unknown parameters. Once the data are observed, they are fixed, so let us remove them from the notation, and abbreviate: $g_1(x_0, x_1) = p(x_0)p(x_1|x_0)p(y_1|x_1)$ and $g_t(x_{t-1}, x_t) = p(x_t|x_{t-1})p(y_t|x_t)$, for $t = 2, \ldots, T$, then

$$p(x, y) = \prod_{t=1}^{T} g_t(x_{t-1}, x_t)$$

To calculate $p(x_t|y) = p(x_t, y)/p(y)$ we need to sum this over all values of $x_0, x_1, \ldots, x_{t-1}, x_{t+1}, \ldots, x_T$, i.e.

$$p(x_t, y) = \sum_{x_0} \cdots \sum_{x_{t-1}} \sum_{x_{t+1}} \cdots \sum_{x_T} \prod_{t=1}^{T} g_t(x_{t-1}, x_t)$$

We can permute the order of the sums and products to find that the right hand side is the same as $r_t(x_t)s_t(x_t)$ where

$$r_t(x_t) = \sum_{x_{t-1}} g_t(x_{t-1}, x_t) \sum_{x_{t-2}} g_{t-1}(x_{t-2}, x_{t-1}) \ldots$$

and

$$s_t(x_t) = \sum_{x_{t+1}} g_{t+1}(x_t, x_{t+1}) \sum_{x_{t+2}} g_{t+2}(x_{t+1}, x_{t+2}) \ldots$$

But note the recursive structure:

$$r_t(x_t) = \sum_{x_{t-1}} g_t(x_{t-1}, x_t) r_{t-1}(x_{t-1}) \quad \text{and} \quad s_t(x_t) = \sum_{x_{t+1}} g_{t+1}(x_t, x_{t+1}) s_{t+1}(x_{t+1})$$

So we can make an enormous saving of computing effort by performing these two recursions, starting from $r_0(x_0) \equiv 1$ and $s_T(x_T) \equiv 1$. Having found all the r_t and s_t functions, you then just set

$$p(x_t|y) = \frac{r_t(x_t)s_t(x_t)}{\sum_{x_t} r_t(x_t)s_t(x_t)}$$

This argument can be easily modified for specific filtering, smoothing or predicting tasks, for example,

$$p(x_t|y_{\leq t}) = \frac{r_t(x_t)}{\sum_{x_t} r_t(x_t)}.$$

You can view the junction tree probability propagation algorithms we saw in lecture 6 as a generalisation of forwards/backwards recursion to graphs more general than linear chains.

7.7 Related algorithms: Viterbi and dynamic programming

There are also modifications in the same spirit to deal with other (static) parameters, and with maximising or sampling rather than marginalising. For example the Viterbi algorithm find the sequence $(x_0, x_1, \ldots, x_T)$ that maximises $p(x_0, x_1, \ldots, x_T | y_1, y_2, \ldots, y_T)$. All of these exploit the same kind of trick – using the linear graph and conditional independence to organise computation to avoid the workload growing exponentially with the length of the sequence.

7.8 Reading

Cappé, Moulines and Rydén discusses everything you could possibly want to know about these models, although it is written in a rather formal and rigorous style. Chapters 1 and 3 certainly cover everything in this lecture.