Probabilistic & Unsupervised Learning

Latent Variable Models for Time Series

Yee Whye Teh
ywteh@gatsby.ucl.ac.uk

Gatsby Computational Neuroscience Unit, and
MSc in Intelligent Systems, Dept Computer Science
University College London

Term 1, Autumn 2007

Modeling time series

Consider a sequence of observations:

$$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_t$$

which are *not* iid.
For example:

- Sequence of images
- Speech signals, English sentences
- Stock prices
- Kinematic variables in a robot
- Sensor readings from an industrial process
- Amino acids, DNA, etc. . .

Goal: To build a probabilistic model of the data $p(\mathbf{x}_1, \ldots, \mathbf{x}_t)$. This can be used to:

- Predict $p(\mathbf{x}_t | \mathbf{x}_1, \ldots, \mathbf{x}_{t-1})$
- Detect abnormal/changed behaviour (if $p(\mathbf{x}_t, \mathbf{x}_{t+1}, \ldots | \mathbf{x}_1, \ldots, \mathbf{x}_{t-1})$ small)
- Recover underlying/latent/hidden causes

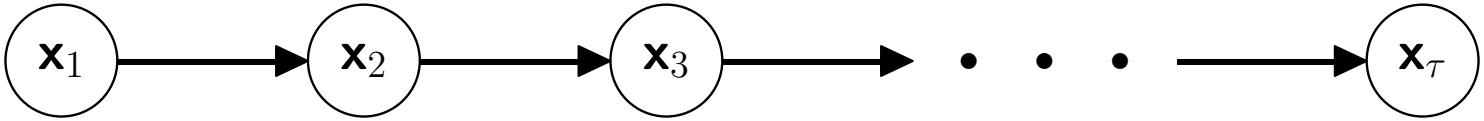
Markov models

In general:

$$P(\mathbf{x}_1, \ldots, \mathbf{x}_t) = P(\mathbf{x}_1)P(\mathbf{x}_2|\mathbf{x}_1)P(\mathbf{x}_3|\mathbf{x}_1, \mathbf{x}_2) \cdots P(\mathbf{x}_t|\mathbf{x}_1, \mathbf{x}_2 \ldots \mathbf{x}_{t-1})$$

First-order Markov model:

$$P(\mathbf{x}_1, \ldots, \mathbf{x}_t) = P(\mathbf{x}_1)P(\mathbf{x}_2|\mathbf{x}_1) \cdots P(\mathbf{x}_t|\mathbf{x}_{t-1})$$

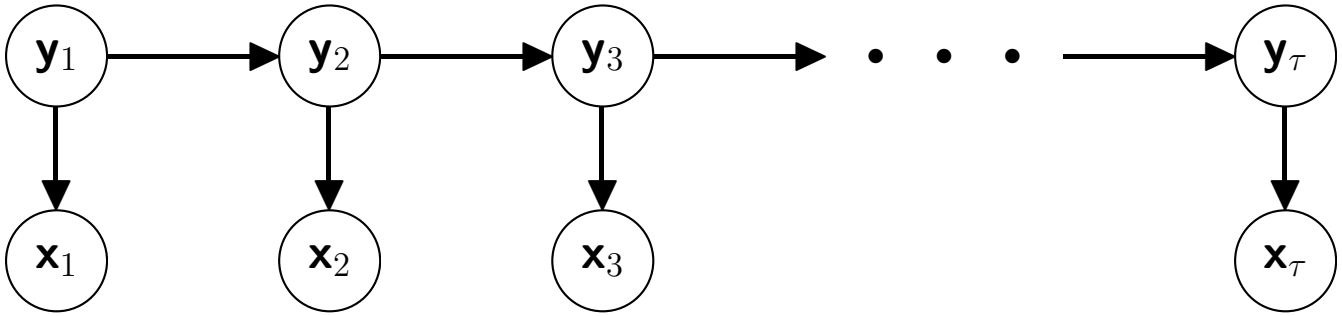


The term *Markov* refers to a conditional independence relationship. In this case, the Markov property is that, given the present observation ($\mathbf{x}_t$), the future ($\mathbf{x}_{t+1}, \ldots$) is independent of the past ($\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}$).

Second-order Markov model:

$$P(\mathbf{x}_1, \ldots, \mathbf{x}_t) = P(\mathbf{x}_1)P(\mathbf{x}_2|\mathbf{x}_1) \cdots P(\mathbf{x}_{t-1}|\mathbf{x}_{t-3}, \mathbf{x}_{t-2})P(\mathbf{x}_t|\mathbf{x}_{t-2}, \mathbf{x}_{t-1})$$

Causal structure and latent variables



Speech recognition:

- **y** - underlying phonemes or words
- **x** - acoustic waveform

Vision:

- **y** - object identities, poses, illumination
- **x** - image pixel values

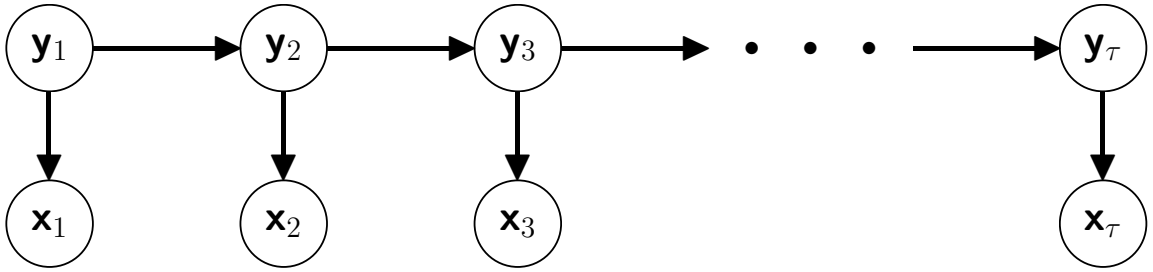
Industrial Monitoring:

- **y** - current state of molten steel in caster
- **x** - temperature and pressure sensor readings

Two frequently-used tractable models:

- Linear-Gaussian state-space models
- Hidden Markov models

Linear-Gaussian state-space models (SSMs)



Joint probability factorizes:

$$P(\mathbf{y}_{1:\tau}, \mathbf{x}_{1:\tau}) = P(\mathbf{y}_1)P(\mathbf{x}_1|\mathbf{y}_1)\prod_{t=2}^{\tau} P(\mathbf{y}_t|\mathbf{y}_{t-1})P(\mathbf{x}_t|\mathbf{y}_t)$$

where $\mathbf{y}_t$ and $\mathbf{x}_t$ are both real-valued vectors, and $\mathbf{z}_{1:\tau} \equiv \mathbf{z}_1, \ldots, \mathbf{z}_\tau$.

In a linear Gaussian SSM all conditional distributions are linear and Gaussian:

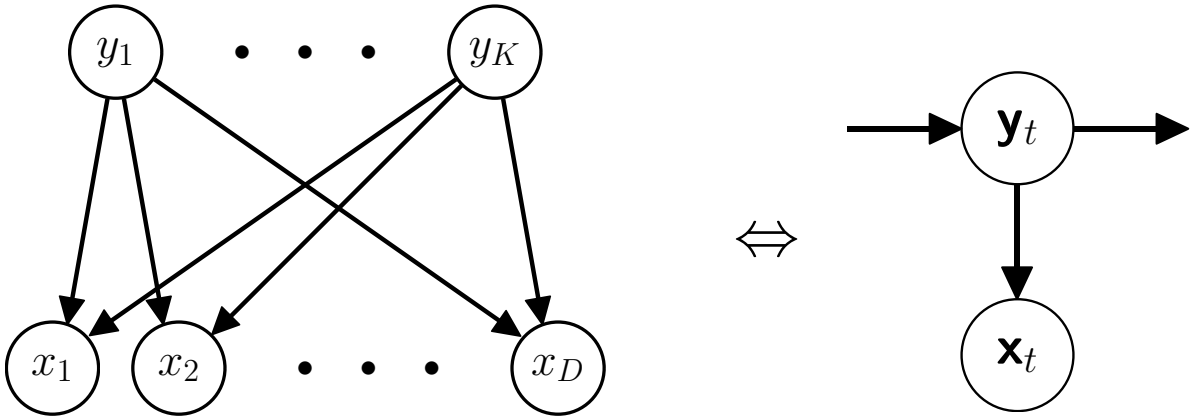
Output equation:	$\mathbf{x}_t = C\mathbf{y}_t + \mathbf{v}_t$
State dynamics equation:	$\mathbf{y}_t = A\mathbf{y}_{t-1} + \mathbf{w}_t$

where $\mathbf{v}_t$ and $\mathbf{w}_t$ are uncorrelated zero-mean multivariate Gaussian noise vectors.

Also assume $\mathbf{y}_1$ is multivariate Gaussian. The joint distribution over all variables $\mathbf{x}_{1:\tau}, \mathbf{y}_{1:\tau}$ is (one big) multivariate Gaussian. Why?

These models are also known as stochastic linear dynamical systems, Kalman filter models.

From factor analysis to state space models



Factor analysis: $x_i = \sum_{j=1}^{K} \Lambda_{ij}\, y_j + \epsilon_i$ vs SSM output equation: $x_{t,i} = \sum_{j=1}^{K} C_{ij}\, y_{t,j} + v_i$.

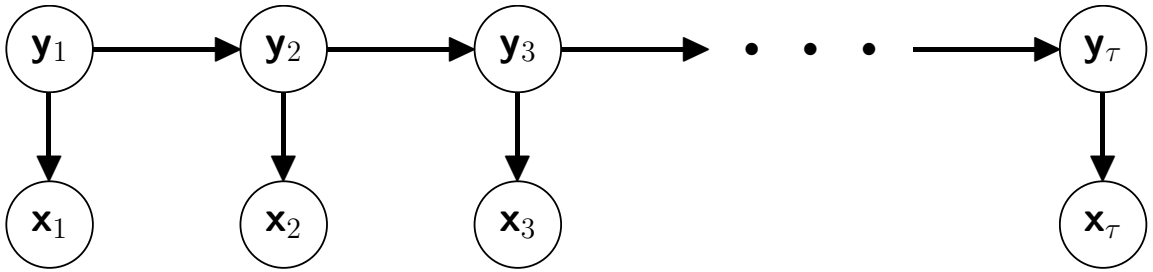
Interpretation 1:

In both models the observations are linearly related to the hidden factors (state-variables) and all variables are Gaussian.

Linear Gaussian state-space models can therefore be seen as a dynamical generalization of factor analysis where $y_{t,j}$ can depend linearly on $y_{t-1,k}$.

Note: while factor analysis only makes sense for $K < D$ and with Ψ diagonal, in SSM it is possible to have $K \geq D$ and Ψ not diagonal. Why?

Linear dynamical systems



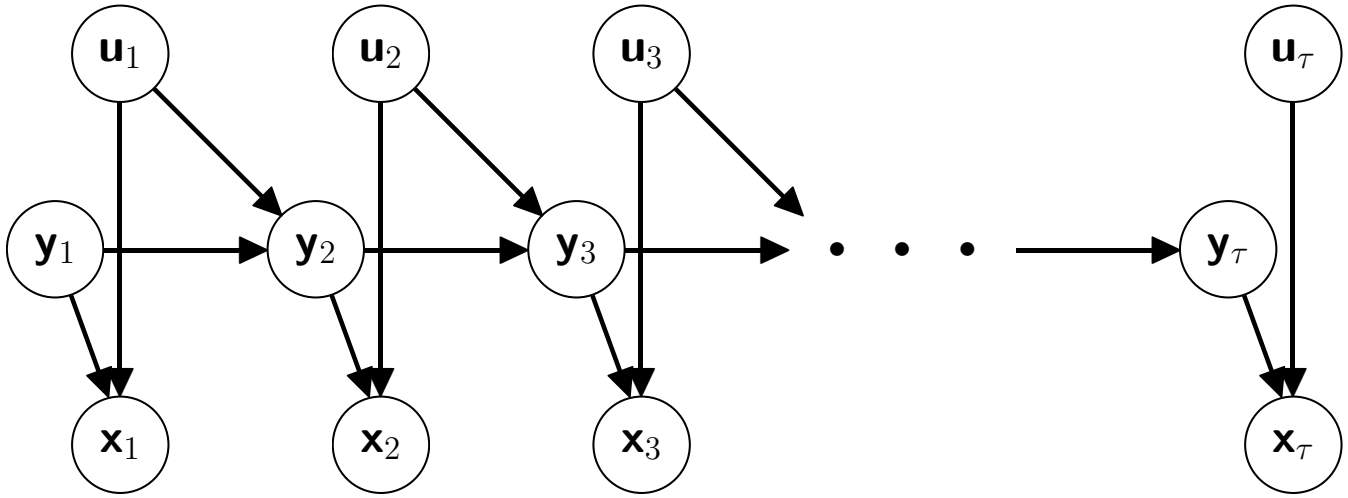
Interpretation 2:

Markov chain with linear Gaussian dynamics $\mathbf{y}_{t-1} \longrightarrow \mathbf{y}_t$.

Observation variables $\mathbf{x}_t$ are a linear projection of latent variables $\mathbf{y}_t$, with Gaussian observation noise.

Note: Latent space of dynamics $\mathbf{y}_t$ can be in a higher dimensional space than the observation space $\mathbf{x}_t$.

State Space Models with Control Inputs



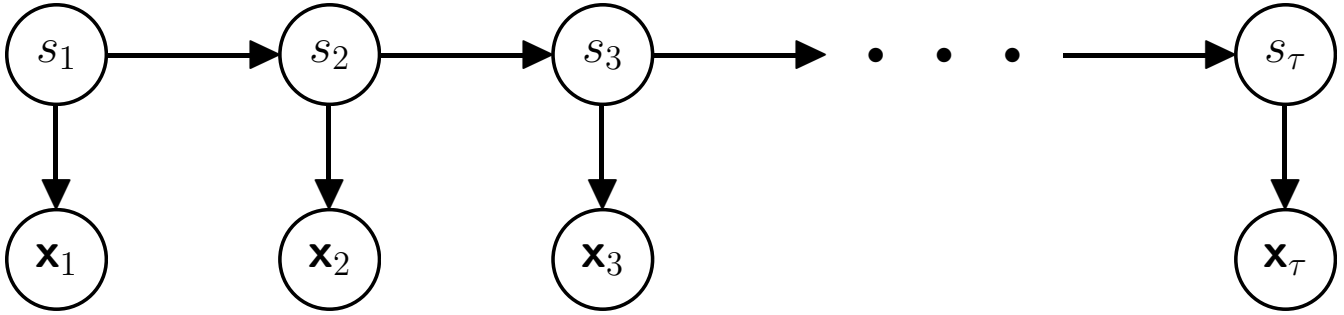
State space models can be used to model the input–output behaviour of controlled systems. The observed variables are divided into inputs ($\mathbf{u}_t$) and outputs ($\mathbf{x}_t$).

State dynamics equation: $\mathbf{y}_t = A\mathbf{y}_{t-1} + B\mathbf{u}_{t-1} + \mathbf{w}_t.$

Output equation: $\mathbf{x}_t = C\mathbf{y}_t + D\mathbf{u}_t + \mathbf{v}_t.$

Note that we can have many variants, e.g. $\mathbf{y}_t = A\mathbf{y}_{t-1} + B\mathbf{u}_t + \mathbf{w}_t$ or even $\mathbf{y}_t = A\mathbf{y}_{t-1} + B\mathbf{x}_{t-1} + \mathbf{w}_t$.

Hidden Markov models



Joint probability factorizes:

$$\mathsf{P}(s_{1:\tau}, \mathbf{x}_{1:\tau}) = \mathsf{P}(s_1)\mathsf{P}(\mathbf{x}_1|s_1) \prod_{t=2}^{\tau} \mathsf{P}(s_t|s_{t-1})\mathsf{P}(\mathbf{x}_t|s_t)$$

Discrete hidden states $s_t \in \{1 \ldots, K\}$, while outputs $\mathbf{x}_t$ can be discrete or continuous.

Generative process:

1. A first-order Markov chain generates the hidden state sequence (path):

   initial state probs: $\pi_j = \mathsf{P}(s_1 = j)$ transition matrix: $T_{ij} = \mathsf{P}(s_{t+1} = j|s_t = i)$

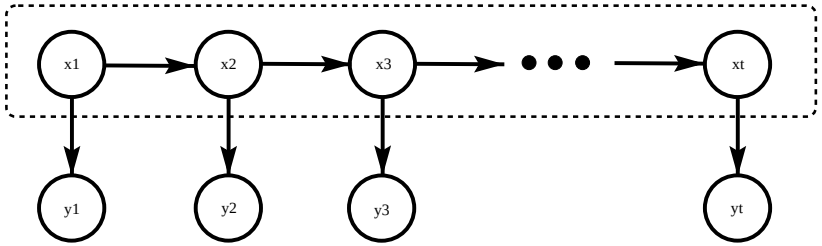
2. A set of emission (output) distributions $A_j(\cdot)$ (one per state) converts this state path into a sequence of observations $\mathbf{x}_t$.

$$\begin{aligned} A_j(\mathbf{x}) &= \mathsf{P}(\mathbf{x}_t = \mathbf{x}|s_t = j) \quad \text{(for continuous } \mathbf{x}_t\text{)} \\ A_{jk} &= \mathsf{P}(\mathbf{x}_t = k|s_t = j) \quad \text{(for discrete } \mathbf{x}_t\text{)} \end{aligned}$$

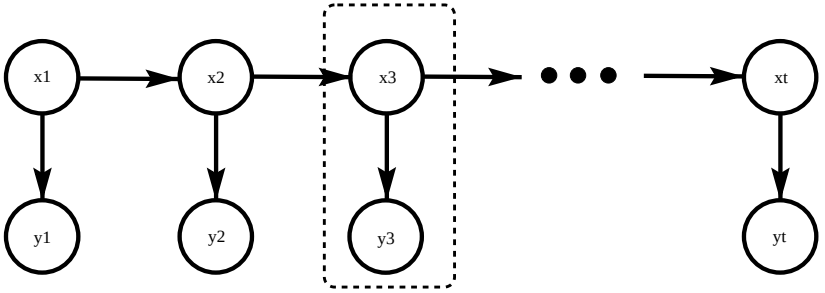
Hidden Markov models

Two interpretations:

- a Markov chain with stochastic measurements:



- or a mixture model with states coupled across time:

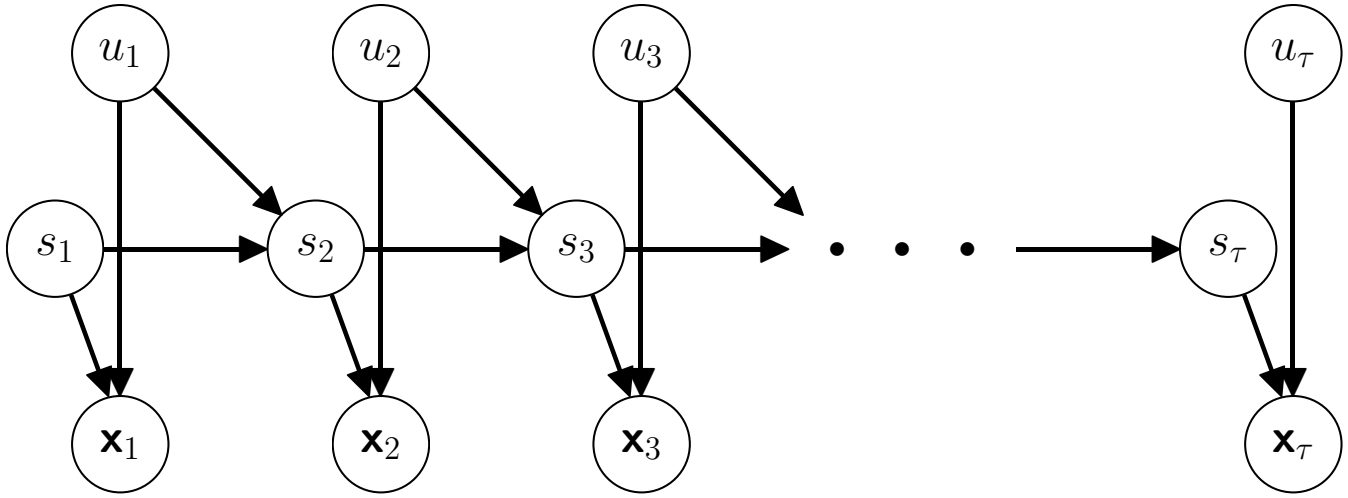


Even though hidden state sequence is first-order Markov, the output process may not be Markov of any order (for example: $1111121111311121111131\ldots$).

Discrete state, discrete output models can approximate any continuous dynamics and observation mapping even if nonlinear; however this is usually not practical.

HMMs are related to stochastic finite state machines/automatas.

Input-output hidden Markov models



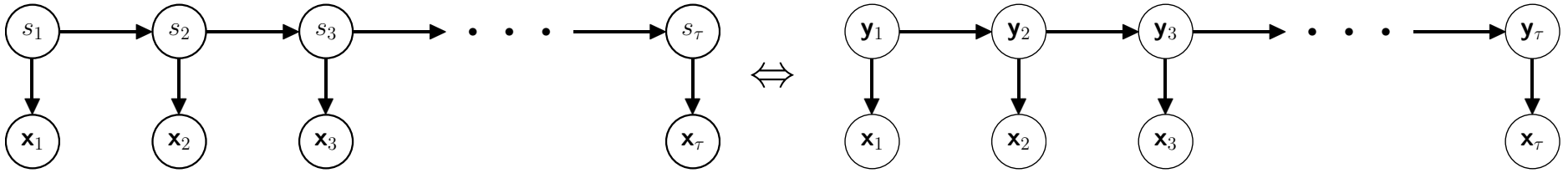
Hidden Markov models can also be used to model sequential input-output behaviour:

$$\mathsf{P}(s_{1:\tau}, \mathbf{x}_{1:\tau} | u_{1:\tau}) = \mathsf{P}(s_1|u_1)\mathsf{P}(\mathbf{x}_1|s_1, u_1) \prod_{t=2}^{\tau} \mathsf{P}(s_t|s_{t-1}, u_{t-1})\mathsf{P}(\mathbf{x}_t|s_t, u_t)$$

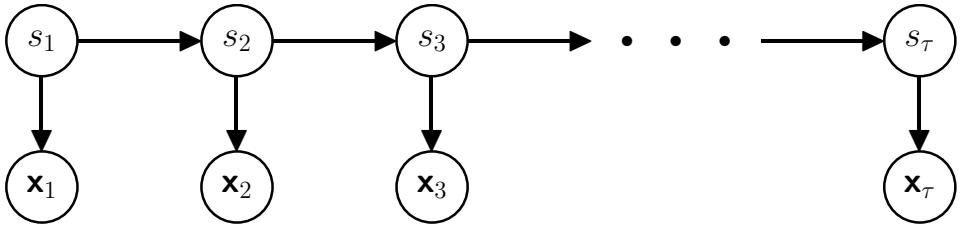
IOHMMs can capture arbitrarily complex input-output relationship, however the number of states required is often impractical.

HMMs and SSMs

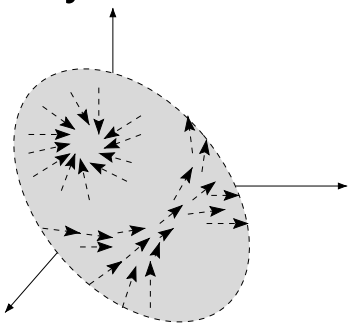
State space models (linear dynamical systems with Gaussian noise) are exactly the continuous state analogue of hidden Markov models.



- Continuous vector of states is a very powerful representation.
  For an HMM to communicate N bits of information about the past, it needs 2^N states!
  But a real-valued state vector can store an arbitrary number of bits in principle.

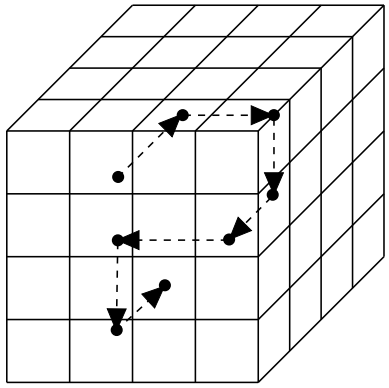


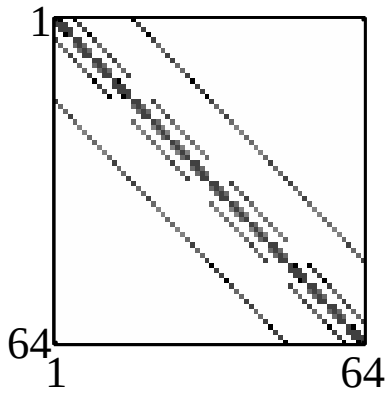
- Linear-Gaussian output/dynamics are very weak.
  The types of dynamics linear SSMs can capture is very limited. However, HMMs can in principle represent arbitrary stochastic dynamics and output mappings.



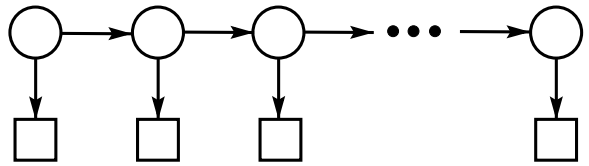
Some Extensions

- Constrained HMMs

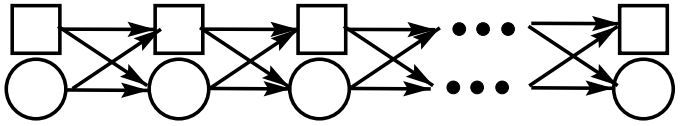




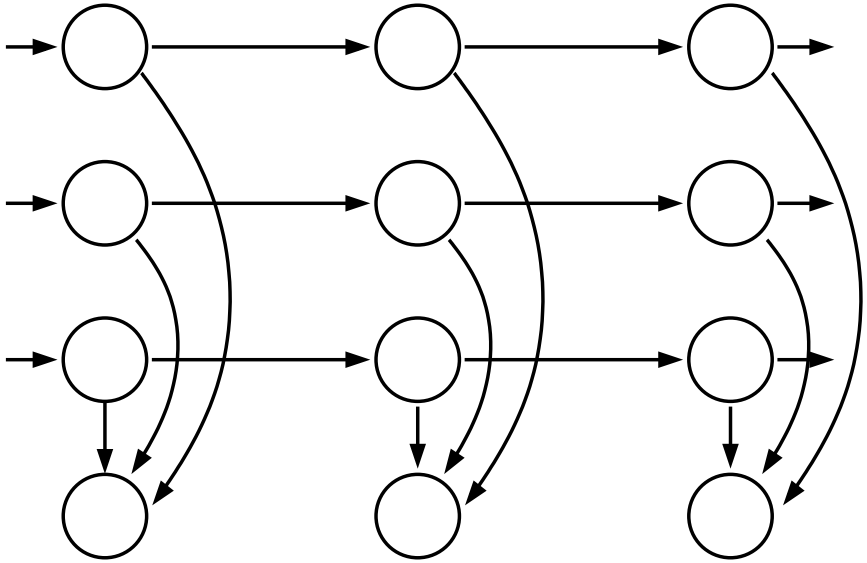
- Continuous state models with discrete outputs for time series and static data

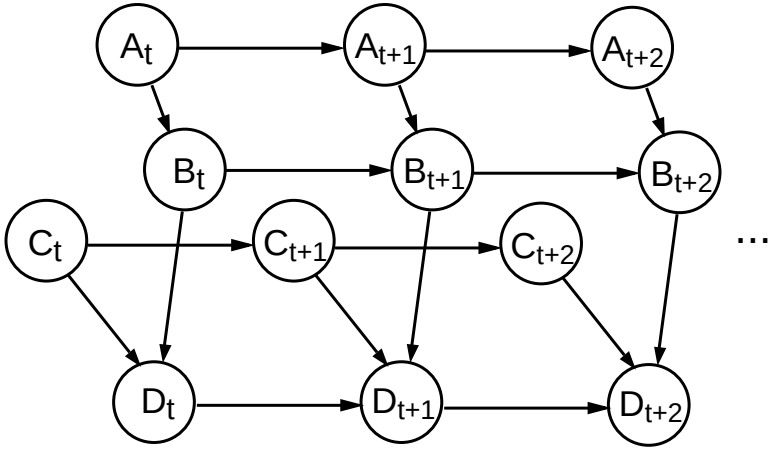


- Hierarchical HMMs
- Hybrid systems $\Leftrightarrow$ Mixed continuous & discrete states, switching state-space models



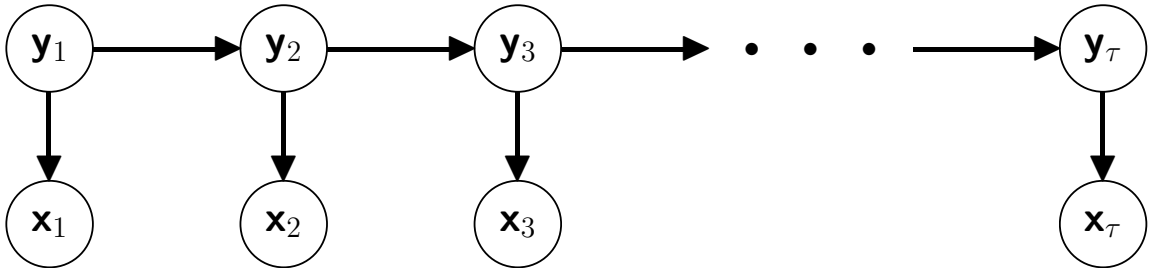
Factorial hidden Markov models and dynamic Bayesian networks





- These are hidden Markov models with many state variables (i.e. a distributed representation of the state).
- The state can capture many more bits of information about the sequence (linear in the number of state variables).

State space models: inference and learning



$\mathbf{y}_t \sim \mathcal{N}(A\mathbf{y}_{t-1}, Q)$ $\mathbf{x}_t \sim \mathcal{N}(C\mathbf{y}_t, R)$

Three inference problems:

- Filtering:

$$P(\mathbf{y}_t|\mathbf{x}_1, \ldots, \mathbf{x}_t)$$

- Smoothing:

$$P(\mathbf{y}_t|\mathbf{x}_1, \ldots, \mathbf{x}_{t+\Delta t})$$

- Prediction:

$$P(\mathbf{y}_t|\mathbf{x}_1, \ldots, \mathbf{x}_{t-\Delta t})$$

Learning:

- EM algorithm

A very simple idea: running averages

$$\hat{\mathbf{y}}_t = \frac{1}{t}\sum_{\tau=1}^{t}\mathbf{x}_\tau$$

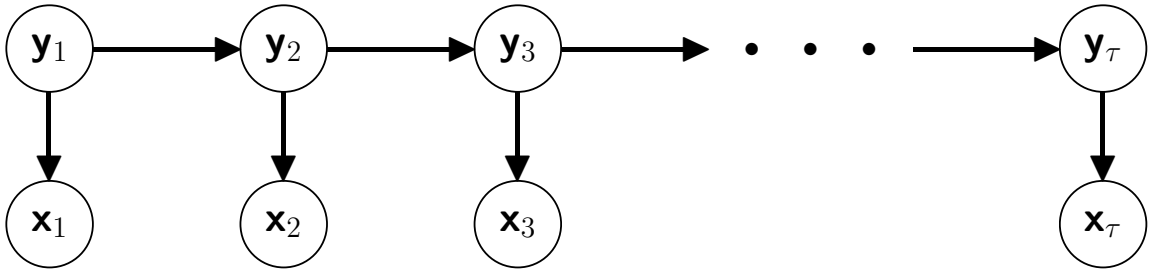
$$\hat{\mathbf{y}}_{t-1} = \frac{1}{t-1}\sum_{\tau=1}^{t-1}\mathbf{x}_\tau$$

$$\hat{\mathbf{y}}_t = \left(\frac{t-1}{t}\right)\hat{\mathbf{y}}_{t-1} + \frac{1}{t}\mathbf{x}_t$$

$$\hat{\mathbf{y}}_t = \hat{\mathbf{y}}_{t-1} + \frac{1}{t}(\mathbf{x}_t - \hat{\mathbf{y}}_{t-1})$$

we can call $K_t = \frac{1}{t}$ the “Kalman gain”

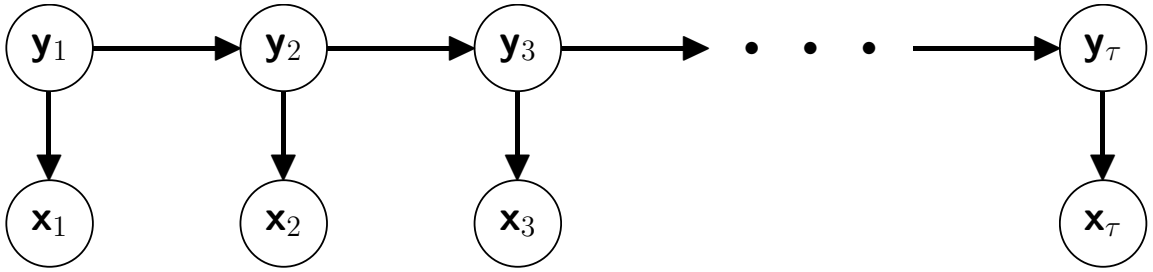
The Kalman Filter



$$
\begin{aligned}
P(\mathbf{y}_t|\mathbf{x}_{1:t}) &= \int P(\mathbf{y}_t, \mathbf{y}_{t-1}|\mathbf{x}_t, \mathbf{x}_{1:t-1})\, d\mathbf{y}_{t-1} \\
&= \int \frac{P(\mathbf{y}_t, \mathbf{y}_{t-1}, \mathbf{x}_t|\mathbf{x}_{1:t-1})}{P(\mathbf{x}_t|\mathbf{x}_{1:t-1})}\, d\mathbf{y}_{t-1} \\
&\propto \int P(\mathbf{y}_{t-1}|\mathbf{x}_{1:t-1}) P(\mathbf{y}_t|\mathbf{y}_{t-1}, \mathbf{x}_{1:t-1}) P(\mathbf{x}_t|\mathbf{y}_t, \mathbf{y}_{t-1}, \mathbf{x}_{1:t-1})\, d\mathbf{y}_{t-1} \\
&\underset{\text{Markov property}}{=} \int P(\mathbf{y}_{t-1}|\mathbf{x}_{1:t-1}) P(\mathbf{y}_t|\mathbf{y}_{t-1}) P(\mathbf{x}_t|\mathbf{y}_t)\, d\mathbf{y}_{t-1}
\end{aligned}
$$

This is a **forward recursion** based on Bayes rule.

The Kalman Filter



Notation: $\hat{\mathbf{y}}_t^\tau \equiv E[\mathbf{y}_t | \mathbf{x}_1, \ldots, \mathbf{x}_\tau]$

Prediction: $\hat{\mathbf{y}}_t^{t-1} = A\hat{\mathbf{y}}_{t-1}^{t-1}$

Correction: $\hat{\mathbf{y}}_t^t = \hat{\mathbf{y}}_t^{t-1} + K_t(\mathbf{x}_t - C\hat{\mathbf{y}}_t^{t-1})$

Kalman gain: $K_t = \hat{V}_t^{t-1} C^\top (C\hat{V}_t^{t-1} C^\top + R)^{-1}$

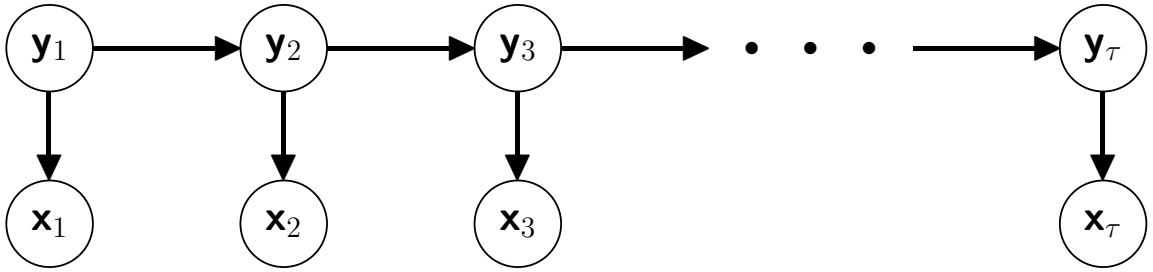
Prediction variance: $\hat{V}_t^{t-1} = A\hat{V}_{t-1}^{t-1} A^\top + Q$

Corrected variance: $\hat{V}_t^t = \hat{V}_t^{t-1} - K_t C \hat{V}_t^{t-1}$

R and Q are the covariance matrices of the output noise $\mathbf{v}_t$, and state dynamics noise $\mathbf{w}_t$, respectively.
To get these equations we need the Gaussian integral: $\int \exp\left\{-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right\} dx = |2\pi\Sigma|^{1/2}$
and the Matrix Inversion Lemma: $(\Phi + \Lambda\Psi\Lambda^\top)^{-1} = \Phi^{-1} - \Phi^{-1}\Lambda(\Psi^{-1} + \Lambda^\top\Phi^{-1}\Lambda)^{-1}\Lambda^\top\Phi^{-1}$
assuming Φ and Ψ are symmetric and invertible.

There are simpler forms for these equations. Why are these complex ones necessary?

The Kalman Smoother

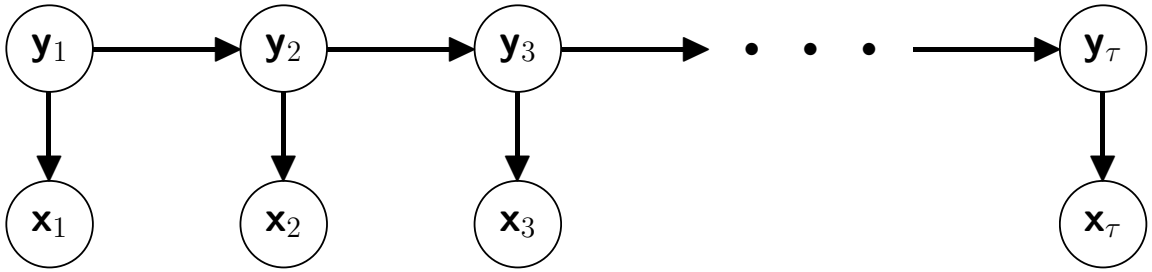


$$
\begin{aligned}
P(\mathbf{y}_t|\mathbf{x}_{1:\tau}) &= \int P(\mathbf{y}_t, \mathbf{y}_{t+1}|\mathbf{x}_{1:\tau})\, d\mathbf{y}_{t+1} \\
&= \int P(\mathbf{y}_t|\mathbf{y}_{t+1}, \mathbf{x}_{1:\tau})(\mathbf{y}_{t+1}|\mathbf{x}_{1:\tau})\, d\mathbf{y}_{t+1} \\
&\underset{\text{Markov property}}{=} \int P(\mathbf{y}_t|\mathbf{y}_{t+1}, \mathbf{x}_{1:t})(\mathbf{y}_{t+1}|\mathbf{x}_{1:\tau})\, d\mathbf{y}_{t+1}
\end{aligned}
$$

Additional **backward recursion**:

$$
\begin{aligned}
J_t &= \hat{V}_t^t A^\top (\hat{V}_{t+1}^t)^{-1} \\
\hat{\mathbf{y}}_t^\tau &= \hat{\mathbf{y}}_t^t + J_t(\hat{\mathbf{y}}_{t+1}^\tau - A\hat{\mathbf{y}}_t^t) \\
\hat{V}_t^\tau &= \hat{V}_t^t + J_t(\hat{V}_{t+1}^\tau - \hat{V}_{t+1}^t)J_t^\top
\end{aligned}
$$

Learning SSM using batch EM



Any distribution $q(\mathbf{y})$ over the hidden states defines a lower bound on $\ell(\theta) = \ln p(\mathbf{x}|\theta)$:

$$\ell(\theta) = \ln p(\mathbf{x}|\theta) = \ln \int d\mathbf{y}\; q(\mathbf{y}) \frac{p(\mathbf{y}, \mathbf{x}|\theta)}{q(\mathbf{y})} \geq \int d\mathbf{y}\; q(\mathbf{y}) \ln \frac{p(\mathbf{y}, \mathbf{x}|\theta)}{q(\mathbf{y})} = \mathcal{F}(q, \theta)$$

E-step: Maximise $\mathcal{F}$ w.r.t. q with θ fixed: $\quad q^*(\mathbf{y}) = p(\mathbf{y}|\mathbf{x}, \theta)$
This can be achieved with a two-state extension of the Kalman smoother.

M-step: Maximize $\mathcal{F}$ w.r.t. θ with q fixed.
This boils down to solving a few weighted least squares problems, since all the variables in:

$$p(\mathbf{y}, \mathbf{x}|\theta) = p(\mathbf{y}_1) p(\mathbf{x}_1|\mathbf{y}_1) \prod_{t=2}^{\tau} p(\mathbf{y}_t|\mathbf{y}_{t-1}) p(\mathbf{x}_t|\mathbf{y}_t)$$

form a multivariate Gaussian.

Solving the M step for SSM

Example: M-step for C using $p(\mathbf{x}_t|\mathbf{y}_t) \propto \exp\left\{-\frac{1}{2}(\mathbf{x}_t - C\mathbf{y}_t)^\top R^{-1}(\mathbf{x}_t - C\mathbf{y}_t)\right\}$:

$$
\begin{aligned}
C_{\text{new}} &= \underset{C}{\text{argmax}} \left\langle \sum_t \ln p(\mathbf{x}_t|\mathbf{y}_t) \right\rangle_q \\
&= \underset{C}{\text{argmax}} \left\langle -\frac{1}{2}\sum_t (\mathbf{x}_t - C\mathbf{y}_t)^\top R^{-1}(\mathbf{x}_t - C\mathbf{y}_t) \right\rangle_q + \text{const} \\
&= \underset{C}{\text{argmax}} \left\{ -\frac{1}{2}\sum_t \mathbf{x}_t{}^\top R^{-1}\mathbf{x}_t - 2\mathbf{x}_t{}^\top R^{-1} C\langle \mathbf{y}_t\rangle + \langle \mathbf{y}_t{}^\top C^\top R^{-1} C\mathbf{y}_t\rangle \right\} \\
&= \underset{C}{\text{argmax}} \left\{ \text{Tr}\left[C\sum_t \langle \mathbf{y}_t\rangle \mathbf{x}_t{}^\top R^{-1}\right] - \frac{1}{2}\text{Tr}\left[C^\top R^{-1} C \left\langle \sum_t \mathbf{y}_t\mathbf{y}_t{}^\top \right\rangle \right] \right\}
\end{aligned}
$$

using $\frac{\partial \text{Tr}[AB]}{\partial A} = B^\top$, we get: $\frac{\partial\{\cdot\}}{\partial C} = R^{-1}\sum_t \mathbf{x}_t\langle \mathbf{y}_t\rangle^\top - R^{-1}C\left\langle \sum_t \mathbf{y}_t\mathbf{y}_t{}^\top \right\rangle$

Solving, we get: $C_{\text{new}} = \left(\sum_t \mathbf{x}_t\langle \mathbf{y}_t\rangle^\top\right)\left(\sum_t \langle \mathbf{y}_t\mathbf{y}_t{}^\top\rangle\right)^{-1}$

Notice that this is exactly the *same equation* as in factor analysis and linear regression!

Learning (online gradient)

This alternative to EM learns online using the output of a Kalman filter.
We can recursively compute the log likelihood of each new data point as it arrives:

$$\ell = \sum_{t=1}^{\tau} \ln p(\mathbf{x}_t | \mathbf{x}_1, \ldots, \mathbf{x}_{t-1}) = \sum_{t=1}^{\tau} \ell_t$$

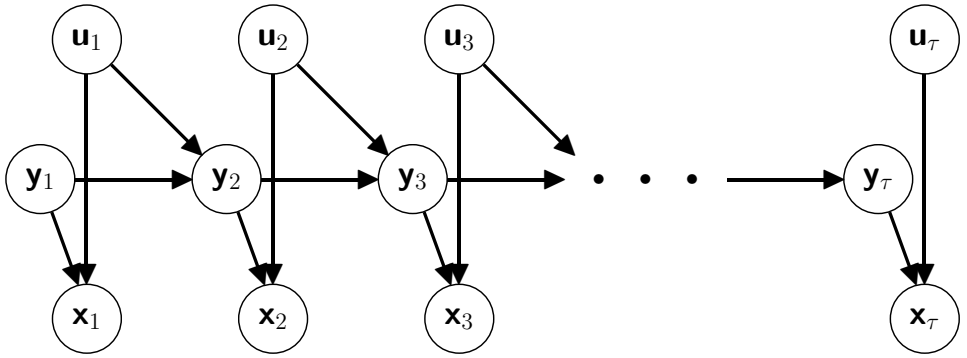
$$\ell_t = -\frac{D}{2} \ln 2\pi - \frac{1}{2} \ln |\Sigma| - \frac{1}{2} (\mathbf{x}_t - C\hat{\mathbf{y}}_t^{t-1})^\top \Sigma^{-1} (\mathbf{x}_t - C\hat{\mathbf{y}}_t^{t-1})$$

where D is dimension of $\mathbf{x}$, and:

$$\begin{aligned} \hat{\mathbf{y}}_t^{t-1} &= A\hat{\mathbf{y}}_{t-1}^{t-1} \\ \Sigma &= C\hat{V}_t^{t-1} C^\top + R \\ \hat{V}_t^{t-1} &= A\hat{V}_{t-1}^{t-1} A^\top + Q \end{aligned}$$

Differentiate ℓ_t to obtain gradient rules for A, C, Q, R.
Learning rate allows for modelling nonstationarity.

Nonlinear dynamical systems

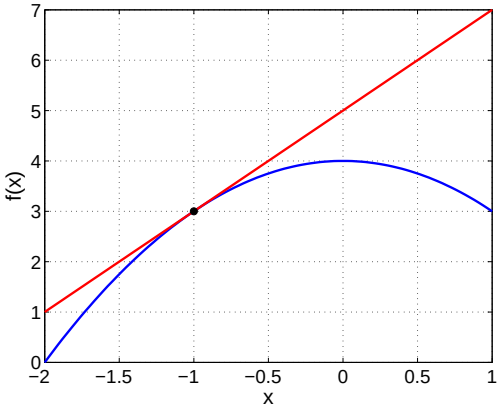


$$\begin{aligned}\mathbf{y}_{t+1} &= f(\mathbf{y}_t, \mathbf{u}_t) + \mathbf{w}_t \\ \mathbf{x}_t &= g(\mathbf{y}_t, \mathbf{u}_t) + \mathbf{v}_t\end{aligned}$$

Extended Kalman Filter (EKF): linearise nonlinear functions about current estimate, $\hat{\mathbf{y}}_t^t$:

$$\mathbf{y}_{t+1} \approx f(\hat{\mathbf{y}}_t^t, \mathbf{u}_t) + \left.\frac{\partial f}{\partial \mathbf{y}_t}\right|_{\hat{\mathbf{y}}_t^t} (\mathbf{y}_t - \hat{\mathbf{y}}_t^t) + \mathbf{w}_t$$

$$\mathbf{x}_t \approx g(\hat{\mathbf{y}}_t^t, \mathbf{u}_t) + \left.\frac{\partial g}{\partial \mathbf{y}_t}\right|_{\hat{\mathbf{y}}_t^t} (\mathbf{y}_t - \hat{\mathbf{y}}_t^t) + \mathbf{v}_t$$



Run the Kalman filter (smoother) on linearised system:

- No guarantees: approximates non-Gaussian by a Gaussian
- Works OK in practice, for approximately linear systems
- Other approaches: sigma-point; quadrature; sequential Monte Carlo.

Can base EM-like algorithm on EKF/EKS or alternatives.

Learning (online EKF)

Augment state vector to include the model parameters

$$\tilde{\mathbf{y}}_t = [\mathbf{y}_t, A, C]$$

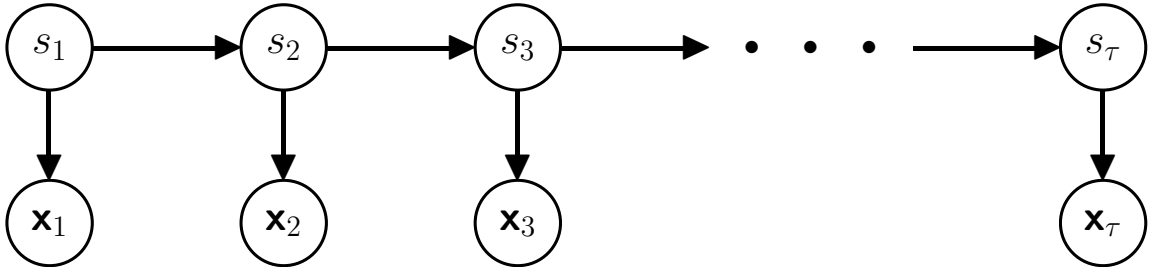
$$\tilde{\mathbf{y}}_{t+1} = f(\tilde{\mathbf{y}}_t) + \text{noise}$$

Use EKF to compute online $E[\tilde{\mathbf{y}}_t|\mathbf{x}_1, \ldots, \mathbf{x}_t]$ and Cov$[\tilde{\mathbf{y}}_t|\mathbf{x}_1, \ldots, \mathbf{x}_t]$.

- Pseudo-Bayesian approach: gives distributions over parameters.
- One can deal with nonstationarity by assuming noise added to A, C at each time step..
- Not clear that it works for Q and R (e.g. how does it deal with covariance constraints?).
- May be faster than gradient approaches.

Also known as “joint-EKF” approach. Also available is the “dual-EKF” approach.

Hidden Markov models: inference and learning



$$P(s_1 = i) = \pi_i \qquad P(s_{t+1} = j | s_t = i) = T_{ij} \qquad P(\mathbf{x}_t | s_t = i) = A_i(\mathbf{x}_t)$$

Inference:

- Filtering:

$$P(s_t | \mathbf{x}_1, \ldots, \mathbf{x}_t)$$

- Smoothing:

$$P(s_t | \mathbf{x}_1, \ldots, \mathbf{x}_\tau)$$

- Prediction:

$$P(s_{t+\Delta_t} | \mathbf{x}_1, \ldots, \mathbf{x}_t)$$

Learning:

- EM algorithm:

$$\operatorname*{argmax}_{A,T,\pi} \log P(\mathbf{x}_1, \ldots, \mathbf{x}_\tau)$$

$$\log P(\mathbf{x}_1, \ldots, \mathbf{x}_\tau), \quad P(s_{t-1}, s_t | \mathbf{x}_1, \ldots, \mathbf{x}_\tau)$$

Likelihood of an observed sequence

The likelihood $P(\mathbf{x}_1, \ldots, \mathbf{x}_\tau|\theta)$ is:

$$\sum_{s_1,\ldots,s_\tau} P(\mathbf{x}_1, \ldots, \mathbf{x}_\tau, s_1, \ldots, s_\tau, \theta)$$

which looks like an extremely hard computation because the number of possible paths grows exponentially with number of time steps τ (number of paths = K^τ).
Fortunately, there exists a **forward recursion** to compute the sum efficiently. Define:

$$\alpha_t(i) = P(\mathbf{x}_1, \ldots, \mathbf{x}_t,\ s_t = i\ |\theta)$$

Now, the Markov property and dynamic programming comes to our rescue:

$$\alpha_1(i) = \pi_i A_i(\mathbf{x}_1) \qquad\qquad \alpha_{t+1}(i) = \left(\sum_{j=1}^{K} \alpha_t(j) T_{ji}\right) A_i(\mathbf{x}_{t+1})$$

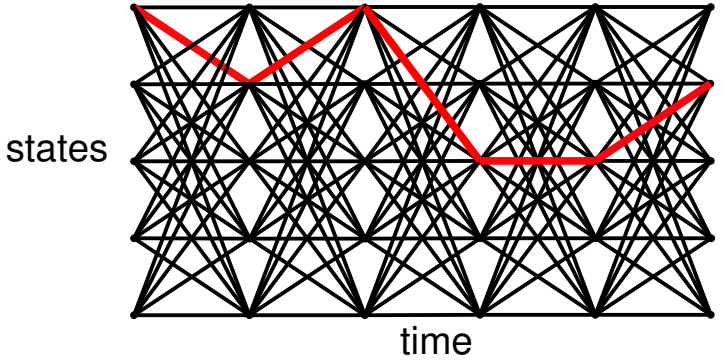
This enables us to compute the likelihood for $\theta = \{A, T, \pi\}$ efficiently in $\mathcal{O}(\tau K^2)$ time.

$$P(\mathbf{x}_1 \ldots \mathbf{x}_\tau|\theta) = \sum_{k=1}^{K} \alpha_k(\tau)$$

Bugs on a Lattice

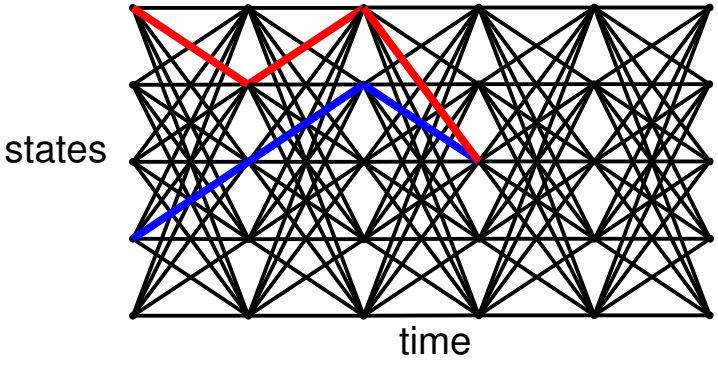
Naïve algorithm:

1. start bug in each state at t=1 holding value 1
2. move each bug forward in time: make copies of each bug to each subsequent state and multiply the value of each copy by transition prob. $\times$ output emission prob.
3. go to 2 until all bugs have reached time τ
4. sum up values on all bugs (there will be one bug per state path)



Clever recursion:

adds a step between 2 and 3 above which says: at each node, replace all the bugs with a single bug carrying the sum of their values



Forward–Backward Algorithm

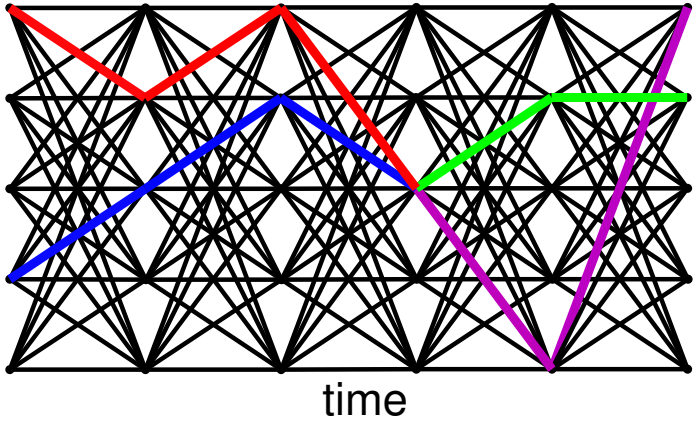
State estimation: compute posterior distribution over states at time t:

$$\gamma_t(i) \equiv P(s_t = i|\mathbf{x}_{1:\tau}) = \frac{P(s_t = i, \mathbf{x}_{1:t})P(\mathbf{x}_{t+1:\tau}|s_t = i)}{P(\mathbf{x}_{1:\tau})} = \frac{\alpha_t(i)\beta_t(i)}{\sum_j \alpha_t(j)\beta_t(j)}$$

where there is a simple **backward recursion** for

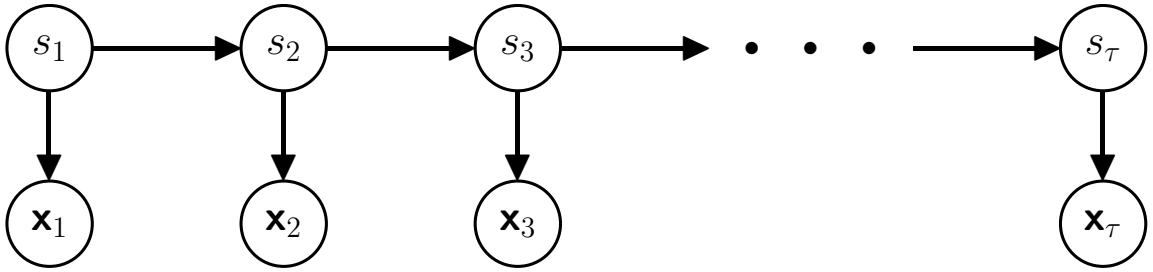
$$\beta_t(i) \equiv \mathsf{P}(\mathbf{x}_{t+1:\tau}|s_t = i) = \sum_{j=1}^{K} T_{ij}\beta_{t+1}(j)A_j(\mathbf{x}_{t+1})$$

$\alpha_t(i)$ gives total *inflow* of probabilities to node (t, i);
$\beta_t(i)$ gives total *outflow* of probabiilties.



Bugs again: the bugs run forward from time 0 to t and backward from time τ to t.

Learning HMMs using EM



Parameters: $\theta = \{\pi, T, A\}$

Free energy:

$$\mathcal{F}(q, \theta) = \sum_{s_{1:\tau}} q(s_{1:\tau})(\log P(x_{1:\tau}, s_{1:\tau}|\theta) - \log q(s_{1:\tau}))$$

E-step: Maximise $\mathcal{F}$ w.r.t. q with θ fixed: $q^*(s_{1:\tau}) = P(s_{1:\tau}|\mathbf{x}_{1:\tau}, \theta)$

We will only need the marginal probabilities $q(s_t, s_{t+1})$, which can also be obtained from the forward–backward algorithm.

M-step: Maximize $\mathcal{F}$ w.r.t. θ with q fixed.

We can re-estimate the parameters by computing the expected number of times the HMM was in state i, emitted symbol k and transitioned to state j.

This is the **Baum-Welch algorithm** and it predates the (more general) EM algorithm.

M step: Parameter updates are given by just ratios of expected counts

We can derive the following updates by taking derivatives of $\mathcal{F}$ w.r.t. θ.

- The initial state distribution is the expected number of times in state i at $t = 1$:

$$\hat{\pi}_i = \gamma_1(i)$$

- The expected number of transitions from state i to j which begin at time t is:

$$\xi_t(ij) \equiv P(s_t = i, s_{t+1} = j|\mathbf{x}_{1:\tau}) = \alpha_t(i)T_{ij}A_j(\mathbf{x}_{t+1})\beta_{t+1}(j)/P(x_{1:\tau})$$

  so the estimated transition probabilities are:

$$\hat{T}_{ij} = \sum_{t=1}^{\tau-1} \xi_t(ij) \Big/ \sum_{t=1}^{\tau-1} \gamma_t(i)$$

- The output distributions are the expected number of times we observe a particular symbol in a particular state:

$$\hat{A}_{ik} = \sum_{t:\mathbf{x}_t=k} \gamma_t(i) \Big/ \sum_{t=1}^{\tau} \gamma_t(i)$$

  (or the state-probability-weighted mean and variance for a Gaussian model).

Viterbi decoding

- The numbers $\gamma_t(i)$ computed by forward-backward gave the posterior distribution over states at each time.
- By choosing the state i_t^* with the largest $\gamma_t(i)$ at each time, we can make a "best" state path. This is the path with the maximum expected number of correct states.
- But it is not the single path with the highest probability of generating the data.
  In fact it may be a path of probability zero!
- To find the single best path, we use the *Viterbi decoding algorithm* which is just Bellman's dynamic programming algorithm applied to this problem. This is an inference algorithm which computes the most probable state sequences: $\operatorname{argmax}_{s_{1:\tau}} P(s_{1:\tau}|\mathbf{x}_{1:\tau}, \theta)$
- The recursions look the same as forward-backward, except with $\max$ instead of $\sum$.
- Bugs once more: same trick except at each step kill all bugs but the one with the highest value at the node.
- There is also a modified Baum-Welch training based on the Viterbi decoder (assignment).

HMM practicalities

- Numerical scaling: the probability values that the bugs carry get tiny for big times and so can easily underflow. Good rescaling trick:

$$\rho_t = \sum_{i=1}^{K} \tilde{\alpha}_t(i) \qquad \tilde{\alpha}_t(i) = \alpha_t(i)/\rho_t$$

  Exercise: show that:

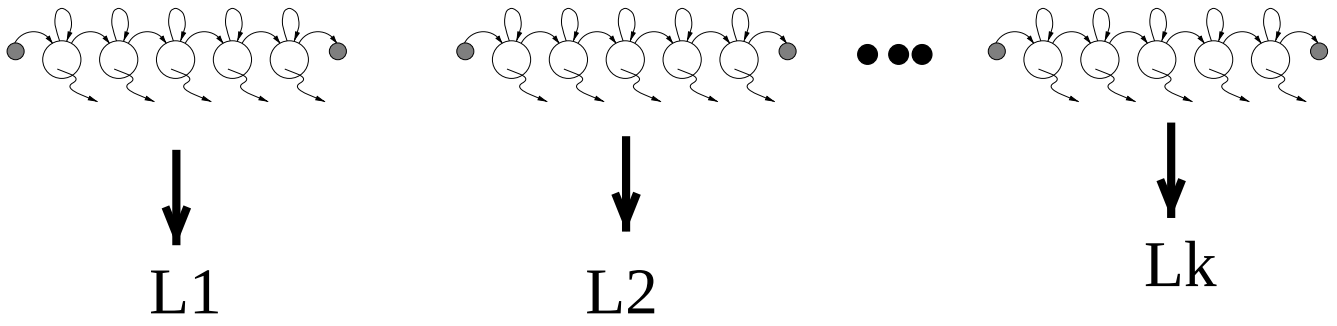
$$\rho_t = P(\mathbf{x}_t|\mathbf{x}_{1:t-1}, \theta) \qquad \prod_{t=1}^{\tau} \rho_t = P(\mathbf{x}_{1:\tau}|\theta)$$

- Multiple observation sequences: can be dealt with by averaging numerators and averaging denominators in the ratios given above.
- Training data requirements: full covariance matrices in high dimensions or discrete symbol models with many symbols have *lots* of parameters.
- How do we pick the topology of the HMM? How many states?

Using HMMs for recognition 1

Use many HMMs for recognition by:

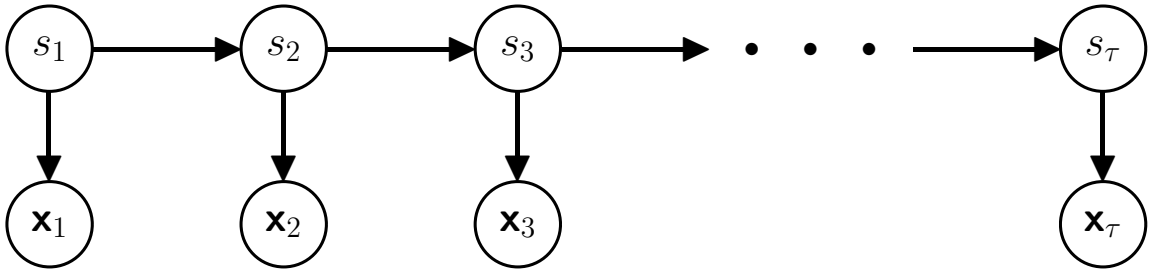
1. training one HMM for each class (this requires each sequence to be labelled by the class)
2. evaluating the probability of an unknown sequence under each HMM
3. classifying the unknown sequence by the HMM which gave it the highest likelihood



Using HMMs for recognition 2

Use a single HMM to label sequences:

1. train a single HMM on sequences of data $\mathbf{x}_1, \ldots, \mathbf{x}_\tau$ and corresponding labels $s_1, \ldots, s_\tau$.
2. On an unlabelled test sequence, compute the posterior distribution over label sequences $P(s_1, \ldots, s_\tau | \mathbf{x}_1, \ldots, \mathbf{x}_\tau)$.
3. Return the label sequence either with highest expected number of correct states, or highest probability under the posterior (Viterbi).



HMM pseudocode: inference (E step)

Forward-backward including scaling tricks

$$
\begin{array}{lrlll}
\text{for } t = 1:\tau, i = 1:K & p_t(i) = A_i(\mathbf{x}_t) & & & \\
& \alpha_1 = \pi \cdot\!* \, p_1 & & \rho_1 = \sum_{i=1}^{K} \alpha_1(i) & \alpha_1 = \alpha_1 / \rho_1 \\
\text{for } t = 2:\tau & \alpha_t = (T^\top * \alpha_{t-1}) \cdot\!* \, p_t & & \rho_t = \sum_{i=1}^{K} \alpha_t(i) & \alpha_t = \alpha_t / \rho_t \\
& \beta_\tau = 1 & & & \\
\text{for } t = \tau - 1:1 & \beta_t = T * (\beta_{t+1} \cdot\!* \, p_{t+1}) / \rho_{t+1} & & & \\
& \log \mathsf{P}(\mathbf{x}_{1:\tau}) = \sum_{t=1}^{\tau} \log(\rho_t) & & & \\
\text{for } t = 1:\tau & \gamma_t = \alpha_t \cdot\!* \, \beta_t & & & \\
\text{for } t = 1:\tau - 1 & \xi_t = T \cdot\!* (\alpha_t * (\beta_{t+1} \cdot\!* \, p_{t+1})^\top) / \rho_{t+1} & & &
\end{array}
$$

HMM pseudocode: parameter re-estimation (M step)

Baum-Welch parameter updates:

For each sequence $l = 1 : L$, run forward–backward to get $\gamma^{(l)}$ and $\xi^{(l)}$, then

$$\pi_i = \frac{1}{L} \sum_{l=1}^{L} \gamma_1^{(l)}(i)$$

$$T_{ij} = \frac{\sum_{l=1}^{L} \sum_{t=1}^{\tau^{(l)}-1} \xi_t^{(l)}(ij)}{\sum_{l=1}^{L} \sum_{t=1}^{\tau^{(l)}-1} \gamma_t^{(l)}(i)}$$

$$A_{ik} = \frac{\sum_{l=1}^{L} \sum_{t=1}^{\tau^{(l)}} \delta(\mathbf{x}_t = k) \gamma_t^{(l)}(i)}{\sum_{l=1}^{L} \sum_{t=1}^{\tau^{(l)}} \gamma_t^{(l)}(i)}$$

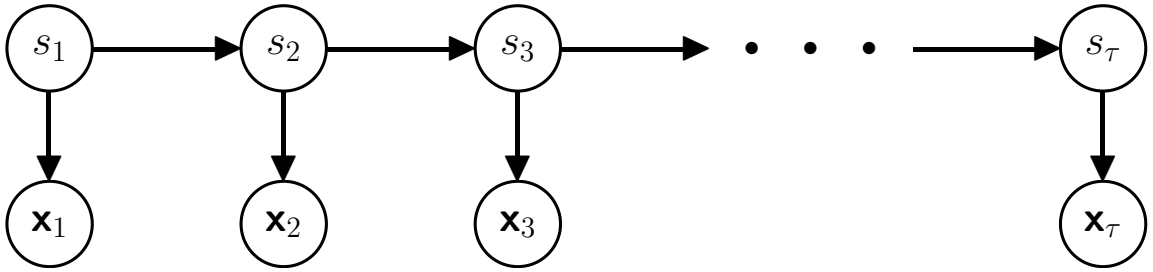
Some HMM history

- Markov (’13) and later Shannon (’48,’51) studied *Markov chains*.
- Baum and colleagues developed much of the theory of “probabilistic functions of Markov chains”.
- Viterbi (’67) came up with an efficient optimal decoder for state inference.
- Applications to speech were pioneered independently by:
  - Baker (’75) at CMU
  - Jelinek’s group (’75) at IBM
  - communications research division of IDA (Ferguson ’74 unpublished)
- Dempster, Laird & Rubin (’77) recognized a general form of the Baum-Welch algorithm and called it the *EM* algorithm.

Conditional random fields

Use a single HMM to label sequences:

1. train a single HMM on sequences of data $\mathbf{x}_1, \ldots, \mathbf{x}_\tau$ and corresponding labels $s_1, \ldots, s_\tau$.
2. On an unlabelled test sequence, compute the posterior distribution over label sequences $P(s_1, \ldots, s_\tau | \mathbf{x}_1, \ldots, \mathbf{x}_\tau)$.
3. Return the label sequence either with highest expected number of correct states, or highest probability under the posterior (Viterbi).



We modelled the whole joint distribution $P(\mathbf{x}_{1:\tau}, s_{1:\tau})$, but during test time we only used $P(s_{1:\tau} | \mathbf{x}_{1:\tau})$.
May be more accurate and more efficient use of data to model $P(s_{1:\tau} | x_{1:\tau})$ directly.
Conditional Random Fields are a way to do this.

Discriminative vs generative modelling

Labelled training data comes from a true underlying distribution $\tilde{P}(s_{1:\tau}, \mathbf{x}_{1:\tau})$.

Generative modelling: train a HMM using by maximizing likelihood:

$$\theta_{\text{Joint}} = \underset{\theta}{\text{argmax}}\, E_{\tilde{P}}[\log P(s_{1:\tau}, \mathbf{x}_{1:\tau}|\theta)]$$

(note do not need EM here, since no latent variables)

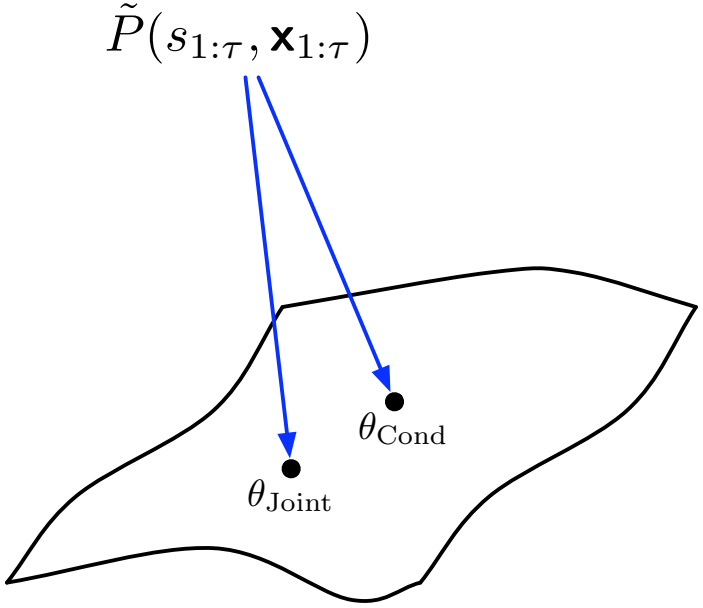
Discriminative modelling: train another HMM by maximizing conditional likelihood:

$$\theta_{\text{Cond}} = \underset{\theta}{\text{argmax}}\, E_{\tilde{P}}[\log P(s_{1:\tau}|\mathbf{x}_{1:\tau}, \theta)]$$

By construction:

$$E_{\tilde{P}}[\log P(s_{1:\tau}|\mathbf{x}_{1:\tau}, \theta_{\text{Cond}})] \geq E_{\tilde{P}}[\log P(s_{1:\tau}|\mathbf{x}_{1:\tau}, \theta_{\text{Joint}})]$$

If $\tilde{P}$ belongs to model class, $P(\cdot|\theta_{\text{Joint}}) = \tilde{P}$ and equality holds.



Caveats:

- Underlying distribution $\tilde{P}$ not usually in model class.
- training set differs from $\tilde{P}$.
- Overfitting easier in discriminative setting.
- Generative modelling often much simpler (fits each conditional probability separately, not iterative).

Major point of debate in machine learning.

Conditional distribution in a HMM

Conditional distribution over label sequences of a HMM:

$$
\begin{aligned}
P(s_{1:\tau}|\mathbf{x}_{1:\tau},\theta) &= \frac{P(s_{1:\tau},\mathbf{x}_{1:\tau}|\theta)}{\sum_{s_{1:\tau}} P(s_{1:\tau},\mathbf{x}_{1:\tau}|\theta)} \\
&\propto P(s_1|\pi)\prod_{t=1}^{\tau-1} P(s_{t+1}|s_t,T)\prod_{t=1}^{\tau} P(\mathbf{x}_t|s_t,A) \\
&= \exp\left(\sum_i \delta(s_1=i)\log\pi_i + \sum_{t=1}^{\tau-1}\sum_{ij}\delta(s_t=i,s_{t+1}=j)\log T_{ij}\right. \\
&\qquad \left. + \sum_{t=1}^{\tau}\sum_{ik}\delta(s_t=i,\mathbf{x}_t=k)\log A_{ik}\right)
\end{aligned}
$$

can be computed using the forward-backward algorithm.
The functional form above gives a well-defined conditional distribution, even if we do not enforce the constraints

$$T_{ij} \geq 0 \qquad\qquad \sum_j T_{ij} = 1$$

similarly for π and A. The forward-backward algorithm can still be applied to compute the conditional distribution.
This is an example of a conditional random field.

Conditional random fields

Define two sets of functions: single label and label-pair functions.
Single label functions:

$$f_i(s_t, \mathbf{x}_t) \qquad \text{for } i = 1, \ldots, I$$

Label-pair functions:

$$g_j(s_t, s_{t+1}, \mathbf{x}_t, \mathbf{x}_{t+1}) \qquad \text{for } j = 1, \ldots, J$$

Each function is associated with a real-valued parameter: λ_i, κ_j.
A conditional random field defines a conditional distribution over $s_{1:\tau}$ given $\mathbf{x}_{1:\tau}$ as follows:

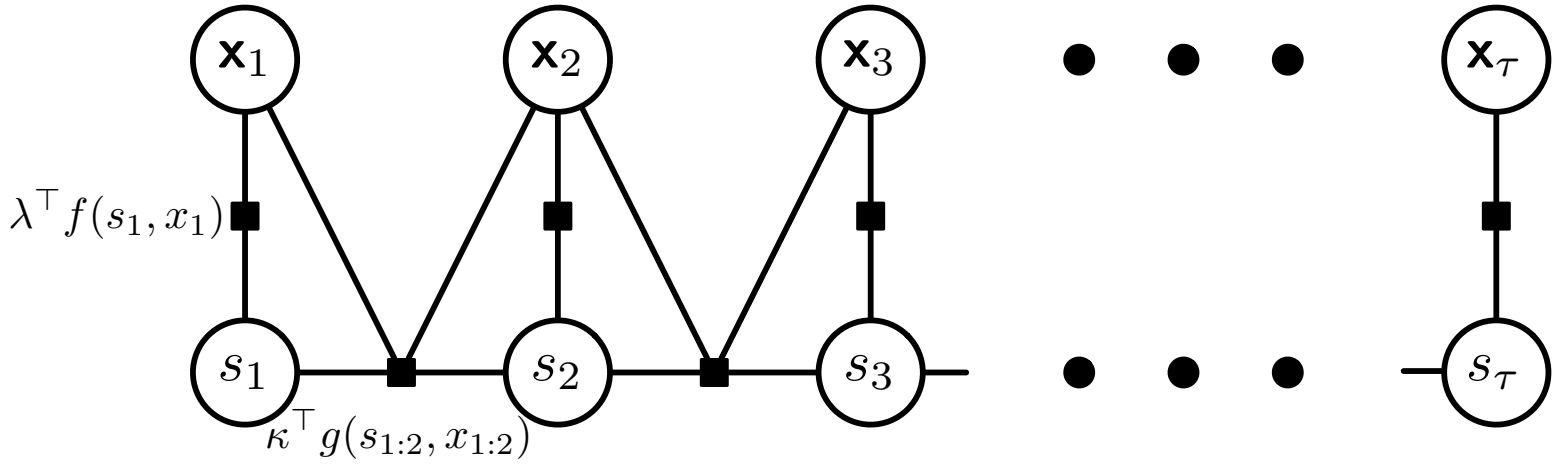
$$P(s_{1:\tau}|\mathbf{x}_{1:\tau}, \boldsymbol{\lambda}, \boldsymbol{\kappa}) \propto \exp\left(\sum_{t=1}^{\tau}\sum_{i}\lambda_i f_i(s_t, \mathbf{x}_t) + \sum_{t=1}^{\tau-1}\sum_{j}\kappa_j g_j(s_t, s_{t+1}, \mathbf{x}_t, \mathbf{x}_{t+1})\right)$$

The forward-backward algorithm can be used to compute:

$$P(s_t|\mathbf{x}_{1:\tau}, \boldsymbol{\lambda}, \boldsymbol{\kappa}) \qquad P(s_t, s_{t+1}|\mathbf{x}_{1:\tau}, \boldsymbol{\lambda}, \boldsymbol{\kappa}) \qquad \operatorname*{argmax}_{s_{1:\tau}} P(s_{1:\tau}|\mathbf{x}_{1:\tau}, \boldsymbol{\lambda}, \boldsymbol{\kappa})$$

Factor graph notation for CRFs

$$P(s_{1:\tau}|\mathbf{x}_{1:\tau}, \boldsymbol{\lambda}, \boldsymbol{\kappa}) \propto \exp\left(\sum_{t=1}^{\tau} \sum_{i} \lambda_i f_i(s_t, \mathbf{x}_t) + \sum_{t=1}^{\tau-1} \sum_{j} \kappa_j g_j(s_t, s_{t+1}, \mathbf{x}_t, \mathbf{x}_{t+1}) \right)$$



Structured generalized linear models

$$P(s_{1:\tau}|\mathbf{x}_{1:\tau}, \boldsymbol{\lambda}, \boldsymbol{\kappa}) \propto \exp\left(\sum_{t=1}^{\tau}\sum_{i}\lambda_i f_i(s_t, \mathbf{x}_t) + \sum_{t=1}^{\tau-1}\sum_{j}\kappa_j g_j(s_t, s_{t+1}, \mathbf{x}_t, \mathbf{x}_{t+1})\right)$$

The conditional distribution over $s_{1:\tau}$ forms an exponential family parameterized by $\boldsymbol{\lambda}, \boldsymbol{\kappa}$ and dependent on $\mathbf{x}_{1:\tau}$.

CRFs are a multivariate generalization of generalized linear models (GLMs).

The labels s_t in a CRF are not independently predicted, but they have a Markov property: $s_{1:t-1}$ is independent of $s_{t+1:\tau}$ given s_t and $\mathbf{x}_{1:\tau}$.
This allows efficient inference using the forward-backward algorithm.

CRFs are models for structured prediction (another major machine learning frontier).

CRFs are very flexible.

CRFs have found wide spread applications across a number of fields: natural language processing (part-of-speech tagging, named-entity recognition, coreference resolution), information retrieval (information extraction), computer vision (image segmentation, object recognition, depth perception), bioinformatics (protein structure prediction, gene finding)...

Learning CRFs

$$P(s_{1:\tau}|\mathbf{x}_{1:\tau}, \boldsymbol{\lambda}, \boldsymbol{\kappa}) \propto \exp\left(\sum_{t=1}^{\tau}\sum_{i}\lambda_i f_i(s_t, \mathbf{x}_t) + \sum_{t=1}^{\tau-1}\sum_{j}\kappa_j g_j(s_t, s_{t+1}, \mathbf{x}_t, \mathbf{x}_{t+1})\right)$$

Given labelled data $\{s_{1:\tau}^{(c)}, \mathbf{x}_{1:\tau}^{(c)}\}_{c=1}^{N}$, we train CRFs by maximum likelihood:

$$\frac{\partial \sum_c \log P(s_{1:\tau}^{(c)}|\mathbf{x}_{1:\tau}^{(c)}, \boldsymbol{\lambda}, \boldsymbol{\kappa})}{\partial \lambda_i} = \sum_{c=1}^{N}\sum_{t=1}^{\tau} f_i(s_t^{(c)}, \mathbf{x}_t^{(c)}) - E_{P(s_{1:\tau}|\mathbf{x}_{1:\tau}^{(c)})}[f_i(s_t^{(c)}, \mathbf{x}_t^{(c)})]$$

$$\frac{\partial \sum_c \log P(s_{1:\tau}^{(c)}|\mathbf{x}_{1:\tau}^{(c)}, \boldsymbol{\lambda}, \boldsymbol{\kappa})}{\partial \kappa_j} = \sum_{c=1}^{N}\sum_{t=1}^{\tau-1} g_j(s_{t:t+1}^{(c)}, \mathbf{x}_{t:t+1}^{(c)}) - E_{P(s_{1:\tau}|\mathbf{x}_{1:\tau}^{(c)})}[g_j(s_{t:t+1}, \mathbf{x}_{t:t+1}^{(c)})]$$

There is no closed-form solution for the parameters, so we use gradient ascent instead.

Note: expectations are computed using the forward-backward algorithm.

The log likelihood is concave, so unlike EM we will get to global optimum (another major frontier in machine learning).

References

- Ghahramani, Z. and Hinton, G.E. (1996) Parameter estimation for linear dynamical systems. University of Toronto Technical Report CRG-TR-96-2, 6 pages (short note). *This derives the EM algorithm for linear-Gaussian state-space models*
- Roweis, S. and Ghahramani, Z. (1999) A Unifying Review of Linear Gaussian Models. *Neural Computation* 11(2):305–345. *This paper relates factor analysis, PCA, mixtures of Gaussians, k-means, ICA, state-space models and hidden Markov models*
- Lafferty, J. and McCallum, A. and Pereira, F. (2001) Conditional random fields: Probabilistic models for segmenting and labeling sequence data. Proceedings of the 18th International Conference on Machine Learning.

References