Unsupervised Learning

Latent Variable Time Series Models

Zoubin Ghahramani
zoubin@gatsby.ucl.ac.uk

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

Autumn 2003

Modeling time series

Sequence of observations:

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

For example:

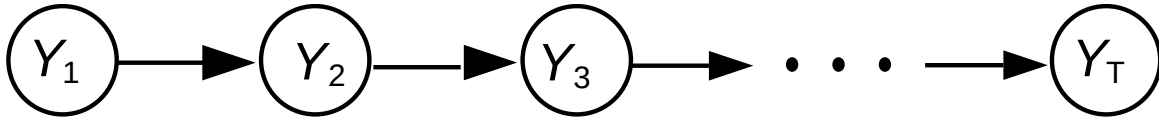
- Sequence of images
- Kinematic variables in a robot
- Speech signals
- Stock prices
- Sensor readings from an industrial process
- Amino acids, etc...

Goal: To build a probilistic model of the data:
something that can predict $p(\mathbf{y}_t|\mathbf{y}_{t-1}, \mathbf{y}_{t-2}, \mathbf{y}_{t-3} \ldots)$

Markov models

First-order Markov model:

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



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

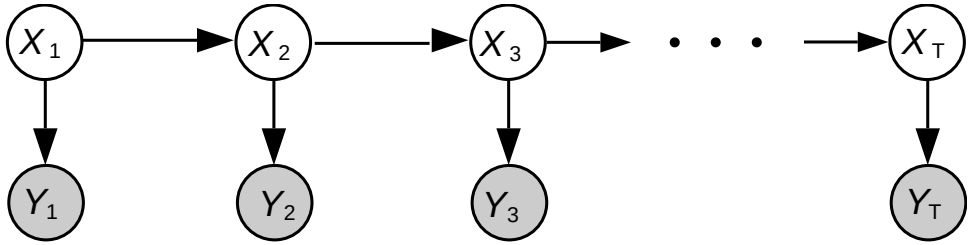
Second-order Markov model:

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

Causal structure and "hidden variables"

Speech recognition:

- $\mathbf{x}$ - underlying phonemes or words
- $\mathbf{y}$ - acoustic waveform



Vision:

- $\mathbf{x}$ - object identities, poses, illumination
- $\mathbf{y}$ - image pixel values

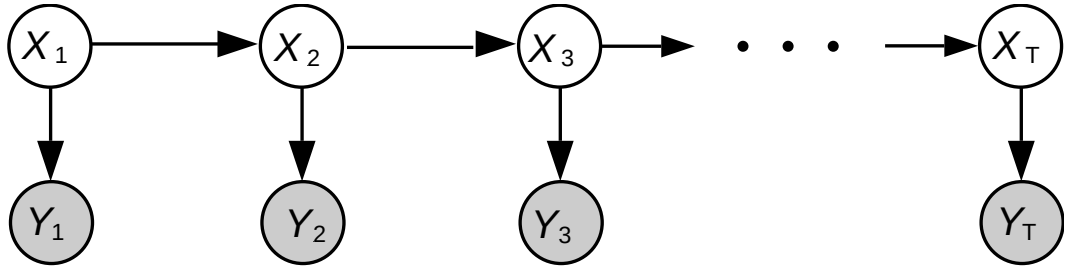
Industrial Monitoring:

- $\mathbf{x}$ - current state of molten steel in caster
- $\mathbf{y}$ - temperature and pressure sensor readings

Two frequently-used tractable models:

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

Linear-Gaussian State-space models (SSMs)



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

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

Output equation: $y_{t,i} = \sum_j C_{ij}\, x_{t,j} + v_{t,i}$

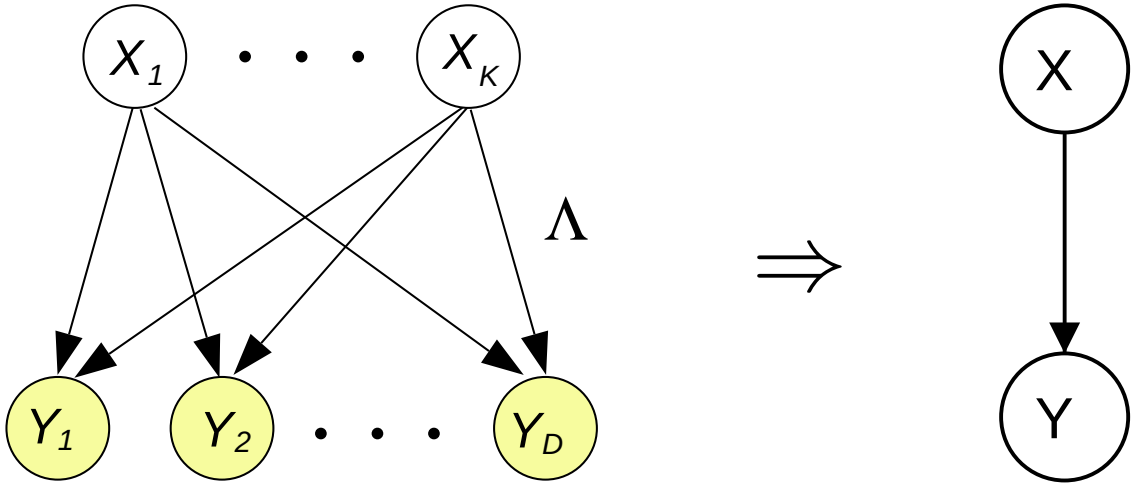
which is, in matrix form: $\mathbf{y}_t = C\mathbf{x}_t + \mathbf{v}_t$

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

where $\mathbf{v}$ and $\mathbf{w}$ are uncorrelated zero-mean multivariate Gaussian noise vectors. We assume that $\mathbf{x}_1$ is also multivariate Gaussian. Since Gaussian variables remain Gaussian after linear transformations and the addition of Gaussian noise, all variables in this model are Gaussian.

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

From Factor Analysis to State Space Models



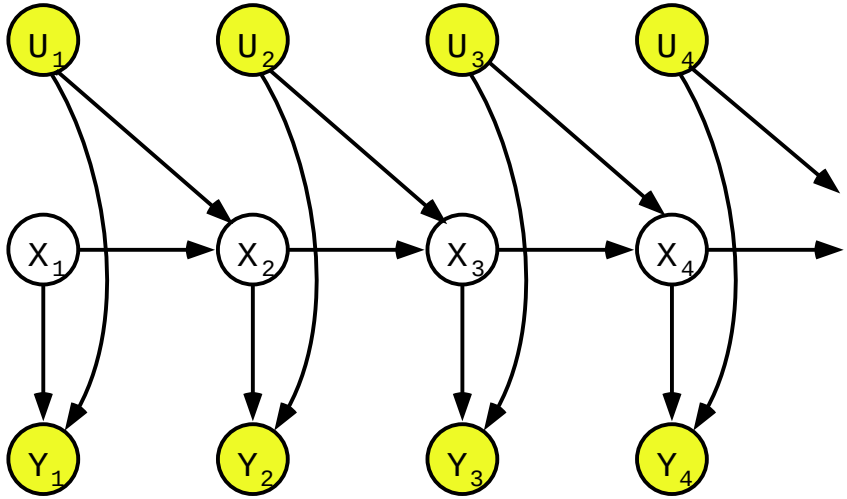
Factor analysis: $y_i = \sum_{j=1}^{K} \Lambda_{ij}\, x_j + \epsilon_i$ vs SSM output equation: $y_{t,i} = \sum_{j=1}^{K} C_{ij}\, x_{t,j} + v_i$.

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 $x_{t,j}$ can depend linearly on $x_{t-1,\ell}$.

However, while FA 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?

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 ($\mathbf{y}_t$).

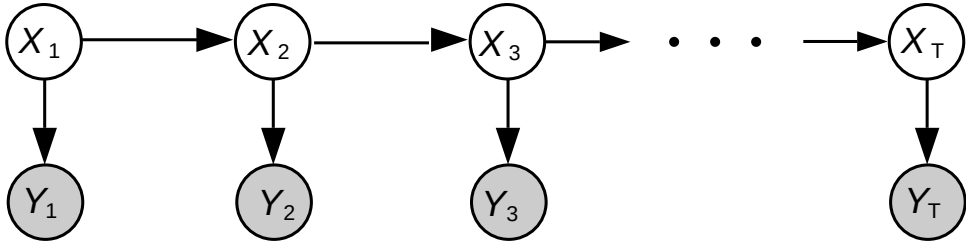
State dynamics equation: $\mathbf{x}_t = A\mathbf{x}_{t-1} + B\mathbf{u}_{t-1} + \mathbf{w}_t.$

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

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

`demo` here...

Three Inference Problems



Filtering:

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

Smoothing:

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

Prediction:

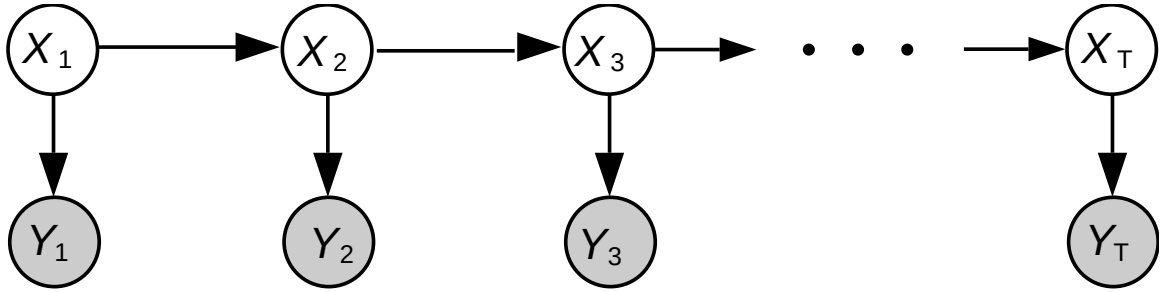
$$P(\mathbf{x}_t|\mathbf{y}_1, \ldots, \mathbf{y}_{t-\Delta t})$$

A very simple idea: running averages

$$
\begin{aligned}
\hat{\mathbf{x}}_t &= \frac{1}{t}\sum_{\tau=1}^{t}\mathbf{y}_\tau \\
\hat{\mathbf{x}}_{t-1} &= \frac{1}{t-1}\sum_{\tau=1}^{t-1}\mathbf{y}_\tau \\
\hat{\mathbf{x}}_t &= \left(\frac{t-1}{t}\right)\hat{\mathbf{x}}_{t-1} + \frac{1}{t}\mathbf{y}_t \\
\hat{\mathbf{x}}_t &= \hat{\mathbf{x}}_{t-1} + \frac{1}{t}(\mathbf{y}_t - \hat{\mathbf{x}}_{t-1})
\end{aligned}
$$

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

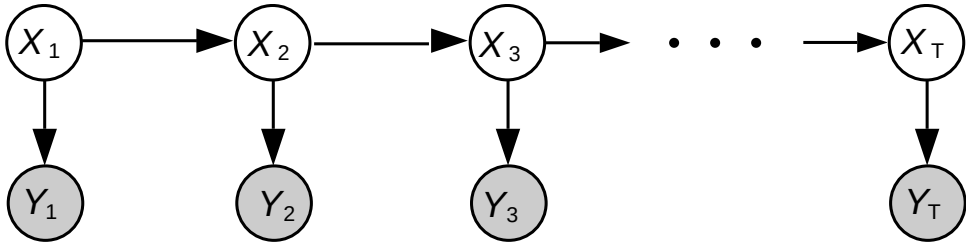
The Kalman Filter



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

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

The Kalman Filter



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

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

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

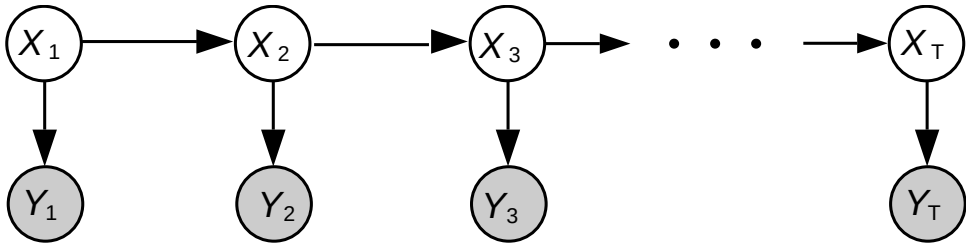
Kalman gain: $K_t = V_t^{t-1}C^\top(CV_t^{t-1}C^\top + R)^{-1}$

Prediction variance: $V_t^{t-1} = AV_{t-1}^{t-1}A^\top + Q$

Corrected variance: $V_t^t = V_t^{t-1} - K_tCV_t^{t-1}$

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: $(X + \mathbf{y}Z\mathbf{y}^\top)^{-1} = X^{-1} - X^{-1}\mathbf{y}(Z^{-1} + \mathbf{y}^\top X^{-1}\mathbf{y})^{-1}\mathbf{y}^\top X^{-1}$
assuming X and Z are symmetric and invertible.

The Kalman Smoother



Compute $P(\mathbf{x}_t|\mathbf{y}_1, \ldots, \mathbf{y}_T)$, $T > t$.

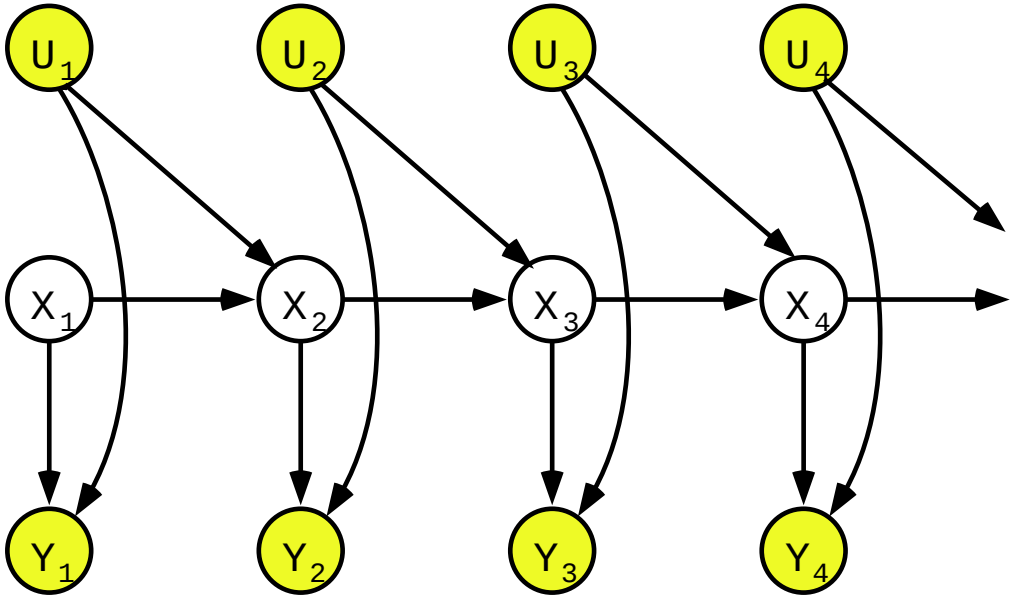
Additional backward recursion:

$$J_t = V_t^t A^\top (V_{t+1}^t)^{-1}$$

$$\mathbf{x}_t^T = \mathbf{x}_t^t + J_t(\mathbf{x}_{t+1}^T - A\mathbf{x}_t^t)$$

$$V_t^T = V_t^t + J_t(V_{t+1}^T - V_{t+1}^t){J_t}^\top$$

Control Inputs and Forward Models



State equation:

$$\mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}_t + \mathbf{w}_t$$

Output equation:

$$\mathbf{y}_t = C\mathbf{x}_t + D\mathbf{u}_t + \mathbf{v}_t$$

Internal (Forward/Generative) Model:

$$\begin{aligned} \mathbf{x}_{t+1} &= \hat{A}\mathbf{x}_t + \hat{B}\mathbf{u}_t + \mathbf{w}_t \\ \mathbf{y}_t &= \hat{C}\mathbf{x}_t + \hat{D}\mathbf{u}_t + \mathbf{v}_t \end{aligned}$$

Kalman Filters: An Example from Biology/Robotics

For example:

- $\mathbf{u}_t$ motor commands sent to the arm ("efference copy" sent back to brain)
- $\mathbf{x}_t$ true state of the hand (including position and velocity of hand)
- $\mathbf{y}_t$ proprioceptive signals (sensor readings)

A *forward model* of the dynamics of the system can be used in a Kalman filter to integrate efference copy ($\mathbf{u}$) and sensory signals ($\mathbf{y}$) to obtain an estimate of $\mathbf{x}$.

A forward model can also be used for

- prediction
- mental simulations
- learning a controller

Learning SSMs using EM

Assume a model parameterised by $\theta = \{A, C, Q, R\}$ with observable variables $\mathbf{y}$ and hidden variables $\mathbf{x}$. [1]

Goal: maximise log likelihood of parameters given observed data:

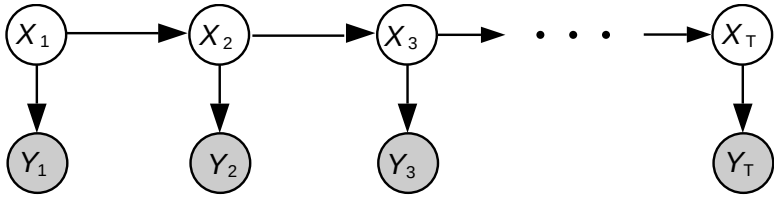
$$\mathcal{L}(\theta) = \ln p(\mathbf{y}|\theta) = \ln \int d\mathbf{x}\, p(\mathbf{x}, \mathbf{y}|\theta)$$

- **E-step**: infer $p(\mathbf{x}|\mathbf{y}, \theta_{\text{old}})$
- **M-step**: find θ_{new} using using "filled-in" values for the sufficient statistics of $\mathbf{x}$

The E-step requires solving the inference a.k.a. state estimation problem: finding a distribution over explanations, $\mathbf{x}$, for the data, $\mathbf{y}$, given the current model parameters, θ.

[1] This can be easily extended to include inputs $\mathbf{u}$ and the corresponding parameter matrices B and D.

Learning SSM using batch EM



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

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

E-step: Maximise $\mathcal{F}$ w.r.t. q with θ fixed: $\qquad q^*(\mathbf{x}) = p(\mathbf{x}|\mathbf{y},\theta)$

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:

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

are linearly related and have Gaussian distributions.

Solving the M step for SSM using Weighted Least Squares

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

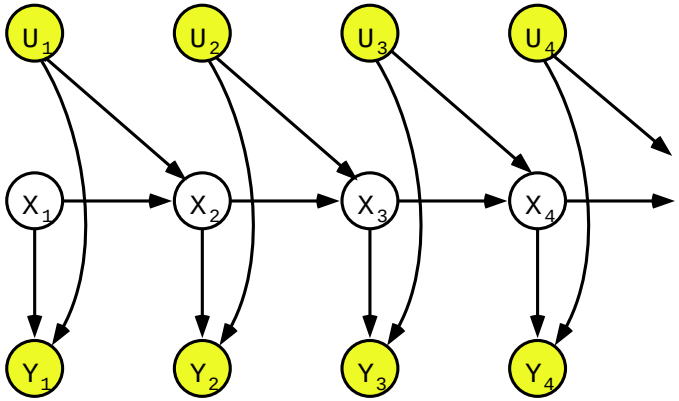
$$
\begin{aligned}
C_{\text{new}} &= \operatorname*{argmax}_C \left\langle \sum_t \ln p(\mathbf{y}_t|\mathbf{x}_t) \right\rangle_q \\
&= \operatorname*{argmax}_C \left\langle -\frac{1}{2}\sum_t (\mathbf{y}_t - C\mathbf{x}_t)^\top R^{-1}(\mathbf{y}_t - C\mathbf{x}_t) \right\rangle_q + \text{const} \\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

using $\frac{\partial \mathrm{tr}[AB]}{\partial A} = B^\top$, we get: $\frac{\partial\{\cdot\}}{\partial C} = -2R^{-1}\sum_t \mathbf{y}_t\langle \mathbf{x}_t\rangle + 2R^{-1}C\left\langle \sum_t \mathbf{x}_t\mathbf{x}_t{}^\top \right\rangle$

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

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

Nonlinear Dynamical Systems



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

The Extended Kalman Filter: linearise about the current estimate, i.e. $\mathbf{x}_t^t, \mathbf{u}_t$:

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

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

Run the Kalman filter (smoother) on linearised system:

- No guarantees
- Approximates non-Gaussian by a Gaussian
- Works OK in practice, for approximately linear systems

EM for nonlinear dynamical systems in (Ghahramani & Roweis, 1999)

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:

$$\mathcal{L} = \sum_{t=1}^{T} \ln p(\mathbf{y}_t | \mathbf{y}_1, \ldots, \mathbf{y}_{t-1}) = \sum_{t=1}^{T} \ell_t$$

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

where d is dimension of $\mathbf{y}$, and:

$$\begin{aligned} \mathbf{x}_t^{t-1} &= A\mathbf{x}_{t-1}^{t-1} \\ \Sigma &= CV_t^{t-1}C^\top + R \\ V_t^{t-1} &= AV_{t-1}^{t-1}A^\top + Q \end{aligned}$$

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

Learning (Online EKF)

Augment state vector to include the model parameters

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

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

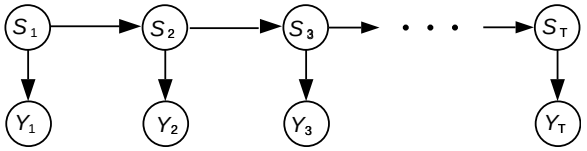
Use EKF to compute online $E[\tilde{\mathbf{x}}_t|\mathbf{y}_1, \ldots, \mathbf{y}_t]$ and Cov$[\tilde{\mathbf{x}}_t|\mathbf{y}_1, \ldots, \mathbf{y}_t]$.

- 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?).
- Faster than gradient approaches.

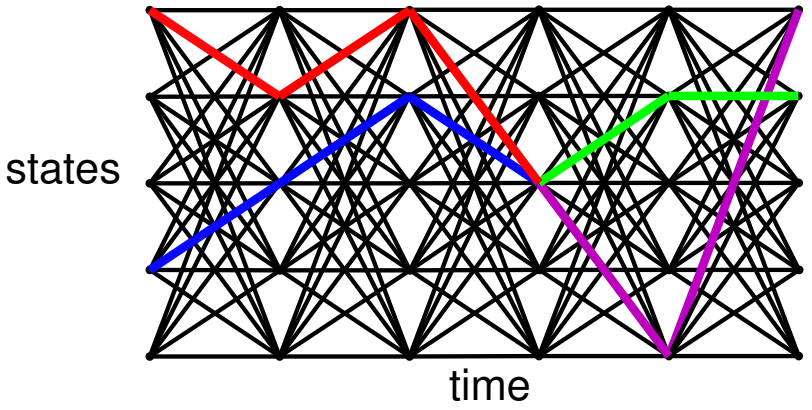
Also known as "joint-EKF" approach. Also available is the "dual-EKF" approach.

Hidden Markov Model: Outline

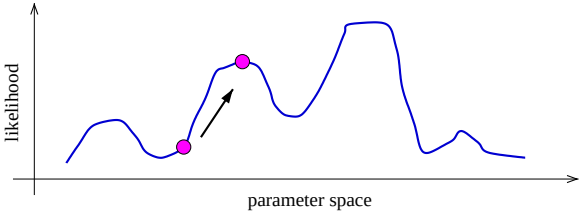
- Generative Model



- Likelihood Evaluation

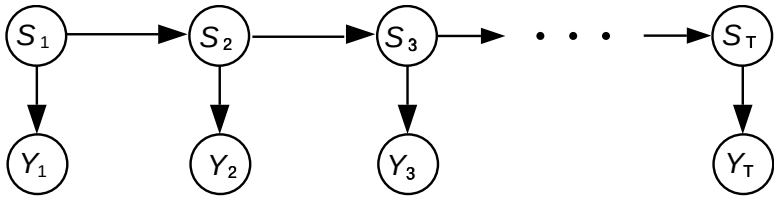


- State inference



- Parameter Estimation

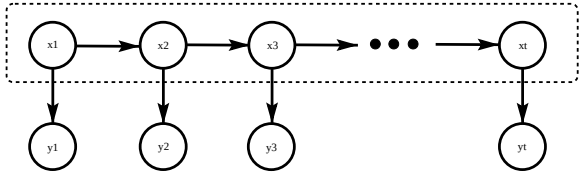
Graphical Model for HMM



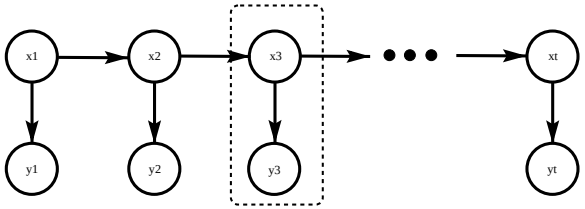
- Discrete hidden states $s_t \in \{1 \ldots, K\}$, and outputs $\mathbf{y}_t$ (discrete or continuous). Joint probability factorizes:

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

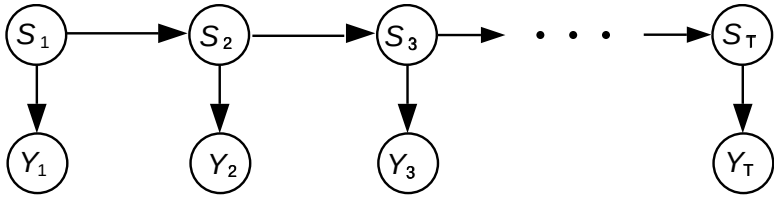
- a Markov chain with stochastic measurements:



- or a mixture model with states coupled across time:



HMM Generative Model



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 $\mathbf{y}_t$.

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

- Even though hidden state sequence is first-order Markov, the output process may not be Markov of any order

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

Probability of an Observed Sequence

- The likelihood $\mathsf{P}(\mathbf{y}_1, \ldots, \mathbf{y}_\tau|\theta)$ is: $\sum_{\text{all paths}} \mathsf{P}(\text{ observations } | \text{ state path }) \mathsf{P}(\text{state path})$

  which looks like an extremely hard computation because the number of possible paths grows exponentially with number of time steps τ (number of paths $= K^\tau$)

- Fortunately, there exists a forward recursion to compute the sum efficiently. Define:

$$\alpha_j(t) = \mathsf{P}(\, \mathbf{y}_1 \ldots \mathbf{y}_t \,,\, s_t = j \,|\theta)$$

  Now, induction (Dynamic Programming) comes to our rescue:

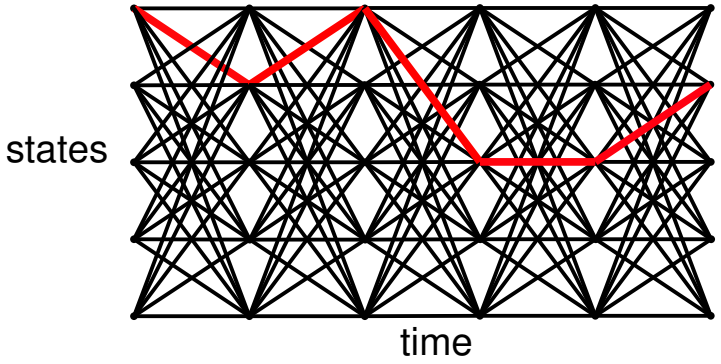
$$\alpha_j(1) = \pi_j A_j(\mathbf{y}_1) \qquad\qquad \alpha_k(t+1) = \left(\sum_j \alpha_j(t) T_{jk} \right) A_k(\mathbf{y}_{t+1})$$

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

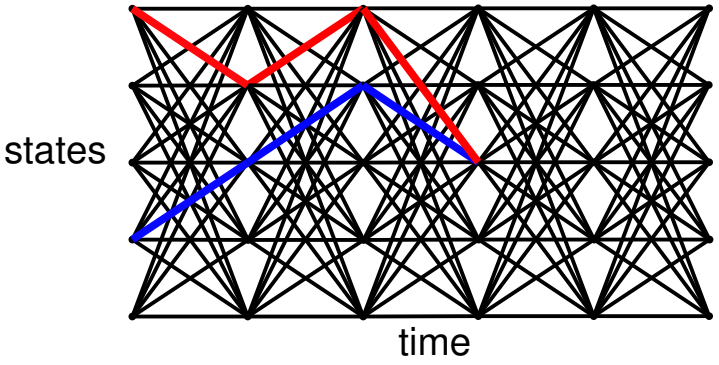
$$\mathsf{P}(\, \mathbf{y}_1 \ldots \mathbf{y}_\tau|\theta) = \sum_{k=1}^{K} \alpha_k(\tau)$$

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



Thanks to Sam Roweis for bugs analogy

Forward–Backward Algorithm

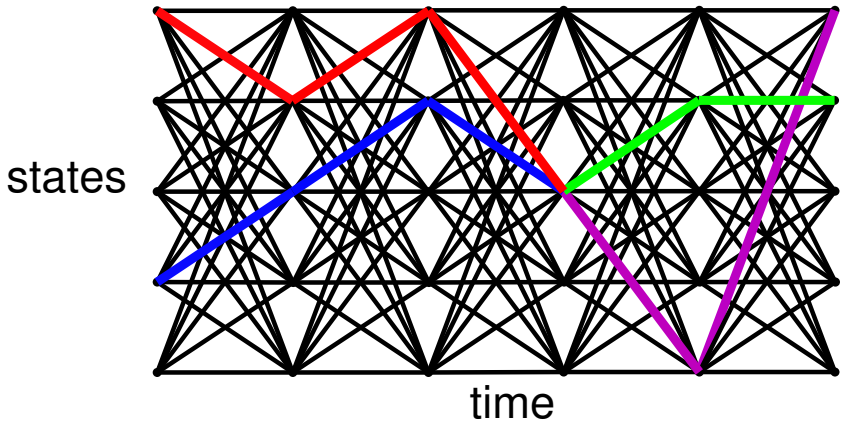
- If we knew the total prob. of all paths going through state i at time t we could compute the *conditional* prob. of being in state i at time t given the data:

$$\gamma_i(t) \equiv \mathsf{P}(s_t = i \mid \mathbf{y}_{1\ldots\tau}) = \frac{\mathsf{P}(s_t = i, \mathbf{y}_{1\ldots t})\mathsf{P}(\mathbf{y}_{t+1\ldots\tau} \mid s_t = i)}{\mathsf{P}(\mathbf{y}_{1\ldots\tau})} = \frac{\alpha_i(t)\ \beta_i(t)}{L}$$

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

$$\beta_j(t) \equiv \mathsf{P}(\mathbf{y}_{t+1\ldots\tau} \mid s_t = j\ ) = \sum_i T_{ij}\beta_i(t+1)A_i(\mathbf{y}_{t+1})$$

- $\alpha_i(t)$ gives total *inflow* of prob. to node (t, i);
  $\beta_i(t)$ gives total *outflow* of prob.



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

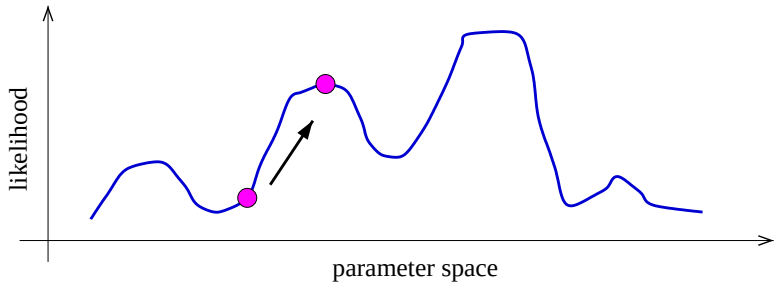
Learning HMMs Using EM: Baum-Welch Training

1. **Intuition**: if we knew the true state path, then ML parameter estimation would be trivial (count co-occurences to get π vector and T and A matrices)

2. But we can estimate the state path using a trick similar to the one above.

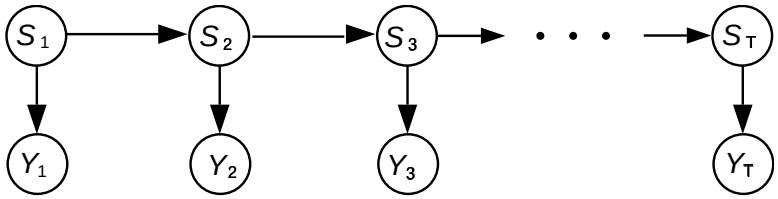
3. What if we estimate the states, then compute params, then re-estimate states, etc ?

4. This works and we can prove that it always improves likelihood. This is the *Baum-Welch algorithm* and it is a special case of the *EM algorithm*.



However, it has been proven that finding the globally-optimal ML parameters is NP-complete, so initial conditions matter a lot.

Learning HMMs using EM



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

E-step: Maximise $\mathcal{F}$ w.r.t. q with θ fixed: $q^*(\mathbf{s}) = q(\mathbf{s}|\mathbf{y}, \theta)$
We need the probability of being in each state at each point in time, which is given by the forward–backward algorithm, which we will see is a special case of belief propagation

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 $t = 1$:

$$\hat{\pi}_i = \gamma_i(1)$$

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

$$\xi_{ij}(t) \equiv \mathsf{P}(s_t = i, s_{t+1} = j | \mathbf{y}_{1\ldots\tau}) = \alpha_i(t) T_{ij} A_j(\mathbf{y}_{t+1}) \beta_j(t+1)/L$$

  so the estimated transition probabilities are: $$\hat{T}_{ij} = \sum_{t=1}^{\tau-1} \xi_{ij}(t) \Big/ \sum_{t=1}^{\tau-1} \gamma_i(t)$$

- The output distributions are the expected number of times we observe a particular symbol in a particular state: $$\hat{A}_j(y) = \sum_{t:\mathbf{y}_t = y} \gamma_j(t) \Big/ \sum_{t=1}^{\tau} \gamma_j(t)$$

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

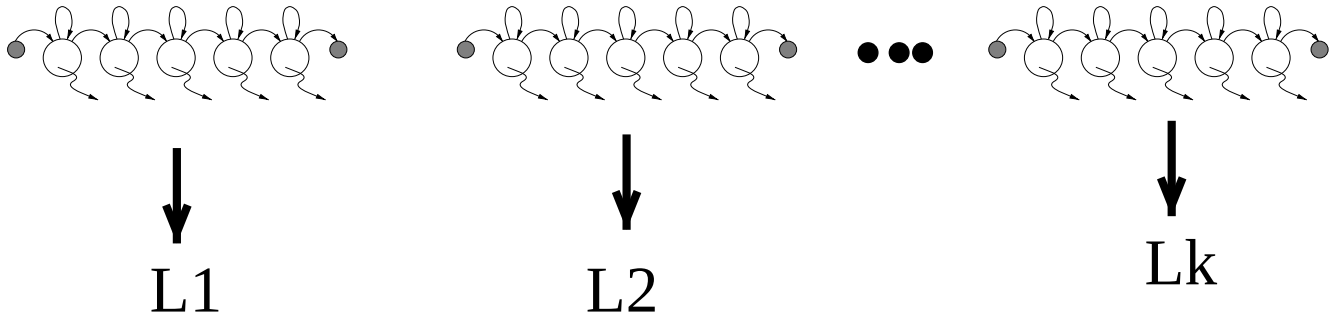
Viterbi Decoding

- The numbers $\gamma_j(t)$ computed by forward-backward gave the posterior distribution over states at each time.
- By choosing the state $j^*(t)$ with the largest $\gamma_j(t)$ 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: $\underset{\mathbf{s}}{\mathrm{argmax}}\, \mathrm{P}(\mathbf{s}|\mathbf{y}, \theta)$
- 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.

Using HMMs for Recognition

- Use many HMMs for recognition by:

  1. training one HMM for each class (this requires *labelled* training data)
  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



- This requires the solution of two problems:

  1. Given model, evaluate prob. of a sequence. (We can do this exactly and efficiently.)
  2. Give some training sequences, estimate model parameters. (We can find a local maximum of parameter space using EM.)

HMM Practicalities

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

$$\rho_t = \mathsf{P}(\mathbf{y}_t | \mathbf{y}_{1\ldots t-1}) \qquad \alpha(t) = \tilde{\alpha}(t) \prod_{t'=1}^{t} \rho_{t'}$$

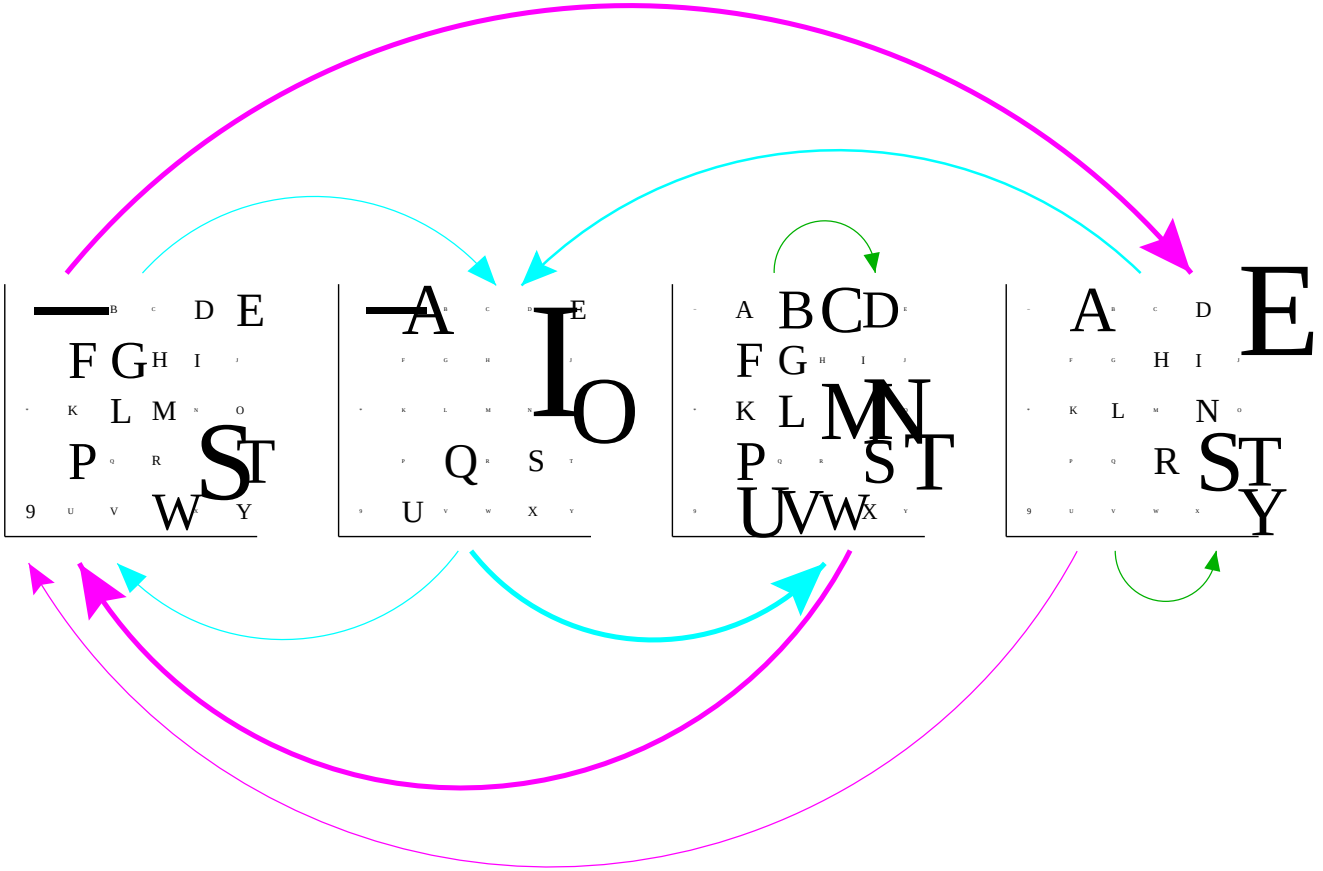
- Multiple observation sequences: can be dealt with by averaging numerators and averaging denominators in the ratios given above.

- 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?

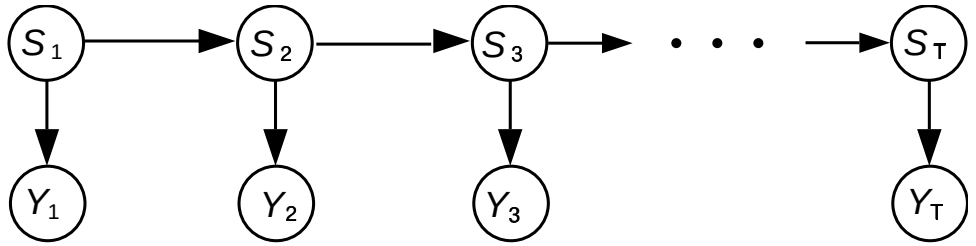
HMM Example

- Character sequences (discrete outputs)



Relationship to State Space Models

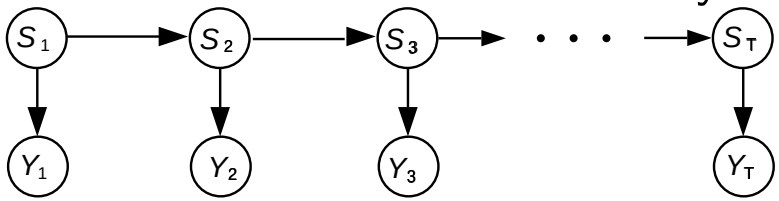
- State space models (linear dynamical systems with Gaussian noise) are exactly the continuous state analogue of Hidden Markov Models.



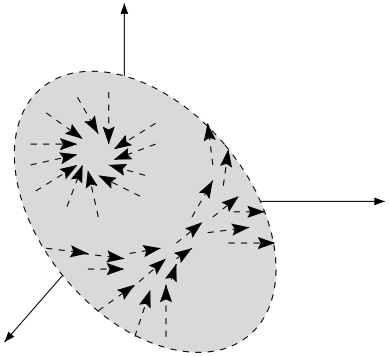
- forward algorithm $\Leftrightarrow$ Kalman Filter
  forward-backward $\Leftrightarrow$ Kalman Smoother
  Viterbi decoding $\Leftrightarrow$ Kalman Smoother

Strengths and Weaknesses of SSMs and HMMs

- 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.

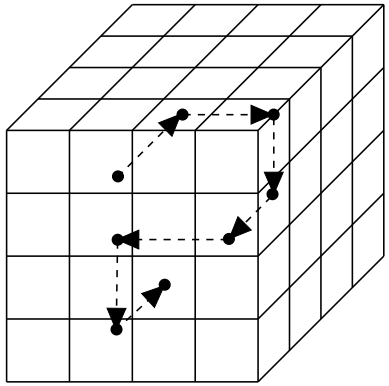


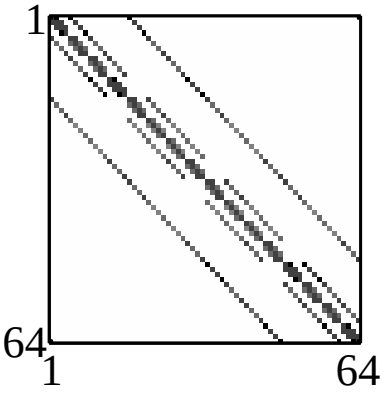
- Linear-Gaussian output/dynamics are very weak.
  The types of dynamics linear SSMs can capture is very limited. However, HMMs can in principle represent arbitrary stochastic dynamics and output mappings.



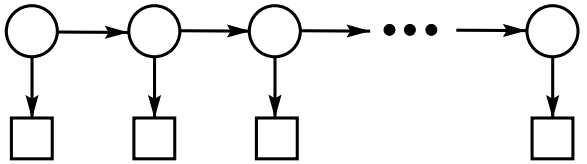
Some Extensions

- Constrained HMMs



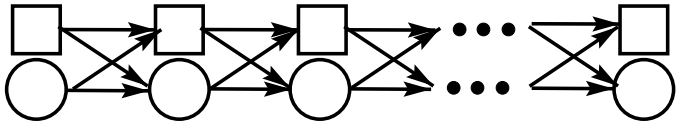


- Continuous state models with discrete outputs for time series and static data

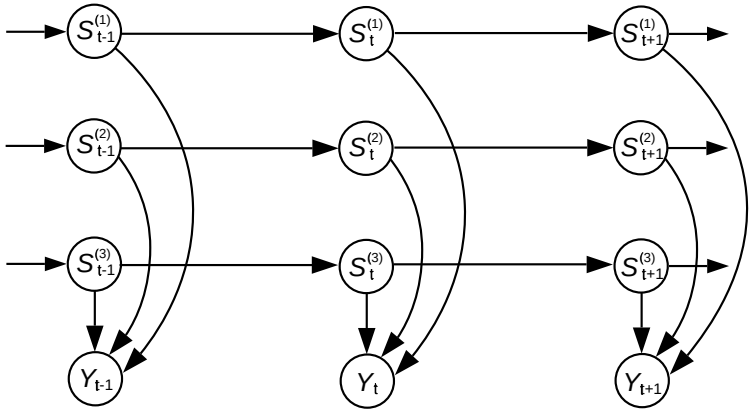


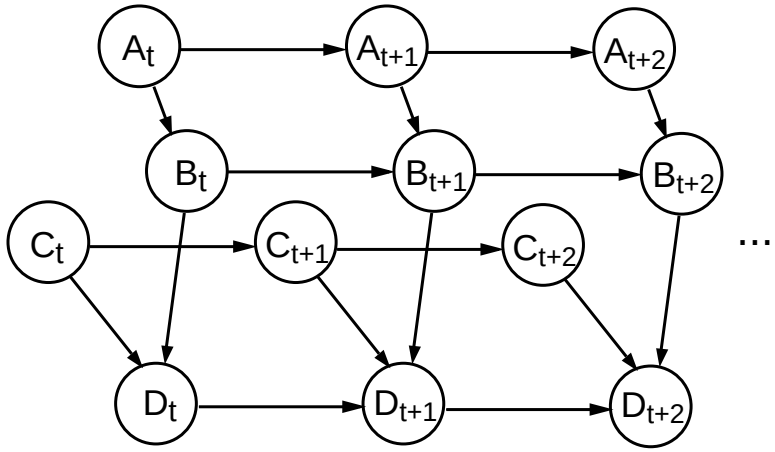
- Hierarchical HMMs

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



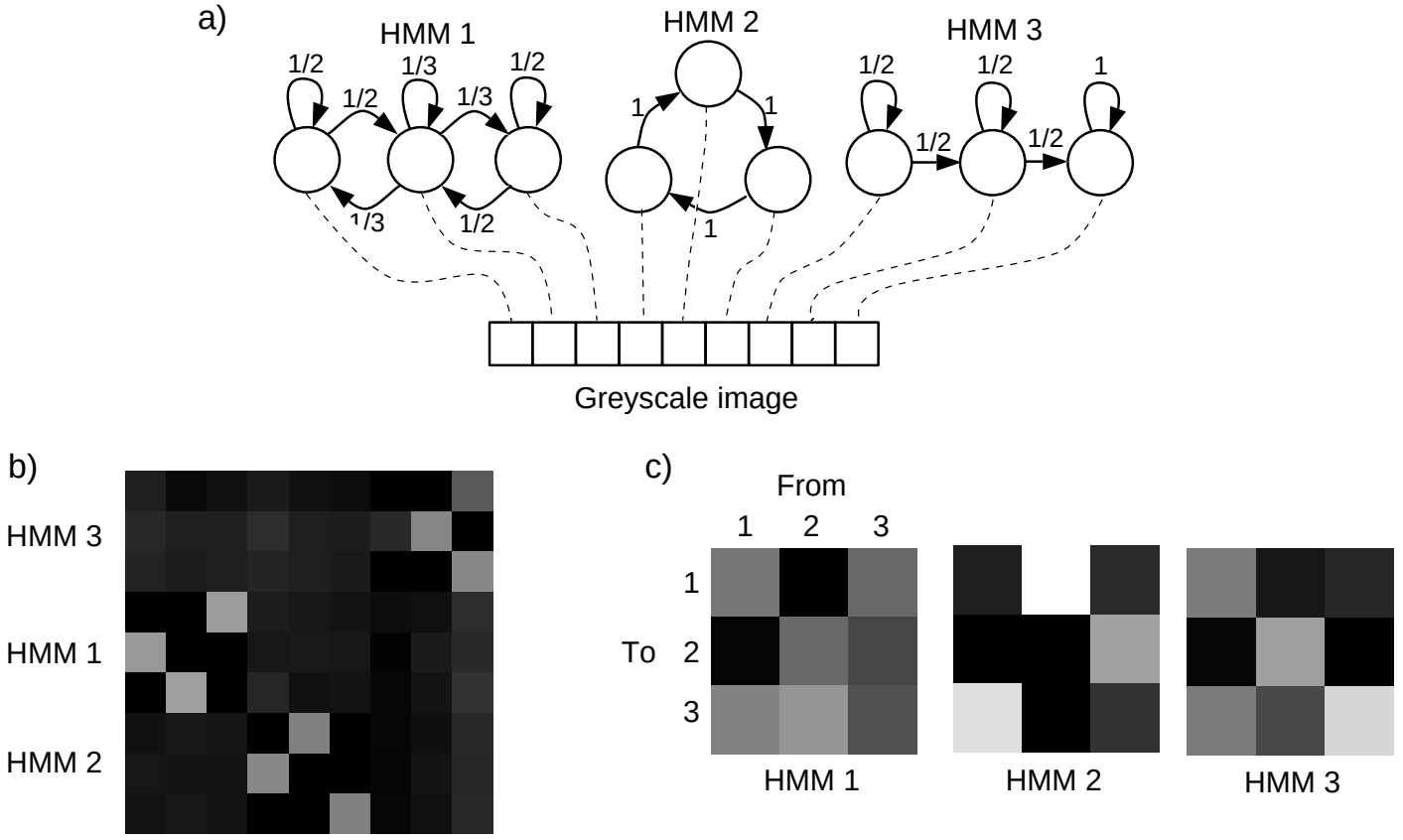
Factorial Hidden Markov Models and Dynamic Bayesian Networks





- 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).
- E step is intractable, but we can use sampling or variational methods.

Factorial Hidden Markov Models: An Toy Example



- Three parallel HMMs: (1) random walk, (2) cyclical, (3) absorbing
- Each HMM had 3 hidden states and 9 outputs
- Data set of 100 sequences of 8 observables each
- EM run for 20 iterations.

Factorial HMMs: Modeling J.S. Bach's Chorales

Discrete event sequences:

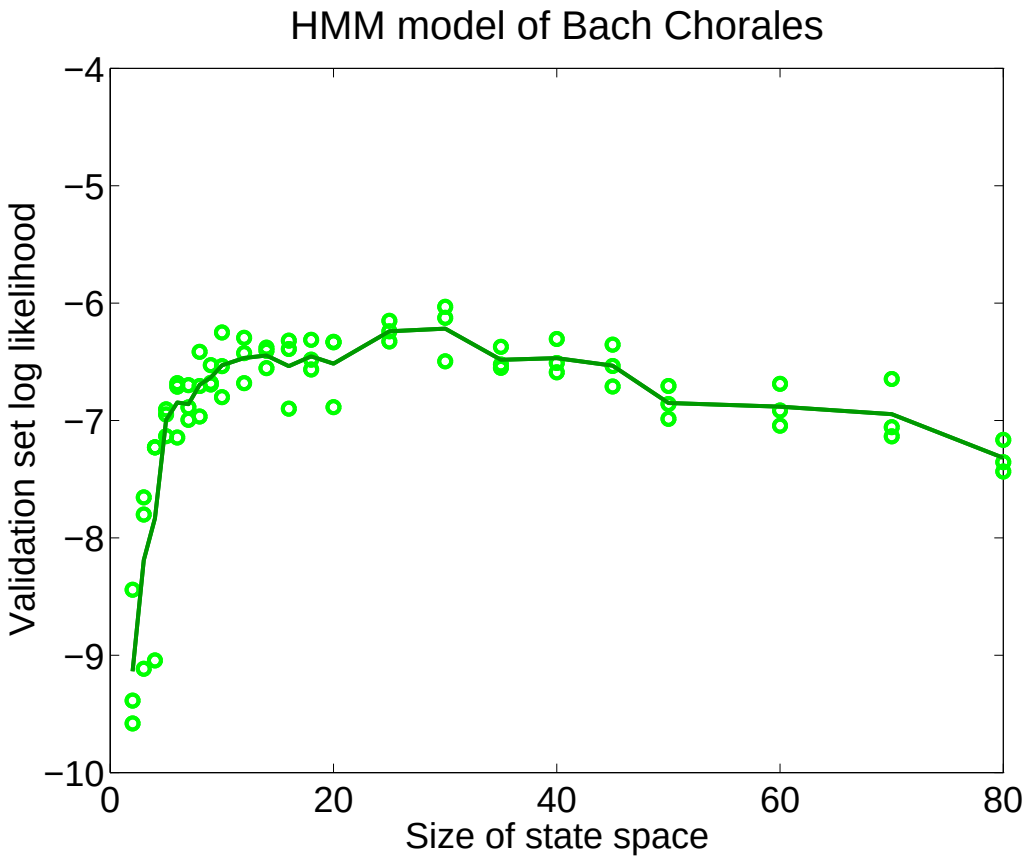
Attribute	Description	Representation
pitch	pitch of the event	int $[0, 127]$
keysig	key signature of the chorale (num of sharps and flats)	int $[-7, 7]$
timesig	time signature of the chorale	int (1/16 notes)
fermata	event under fermata?	binary
st	start time of event	int (1/16 notes)
dur	duration of event	int (1/16 notes)

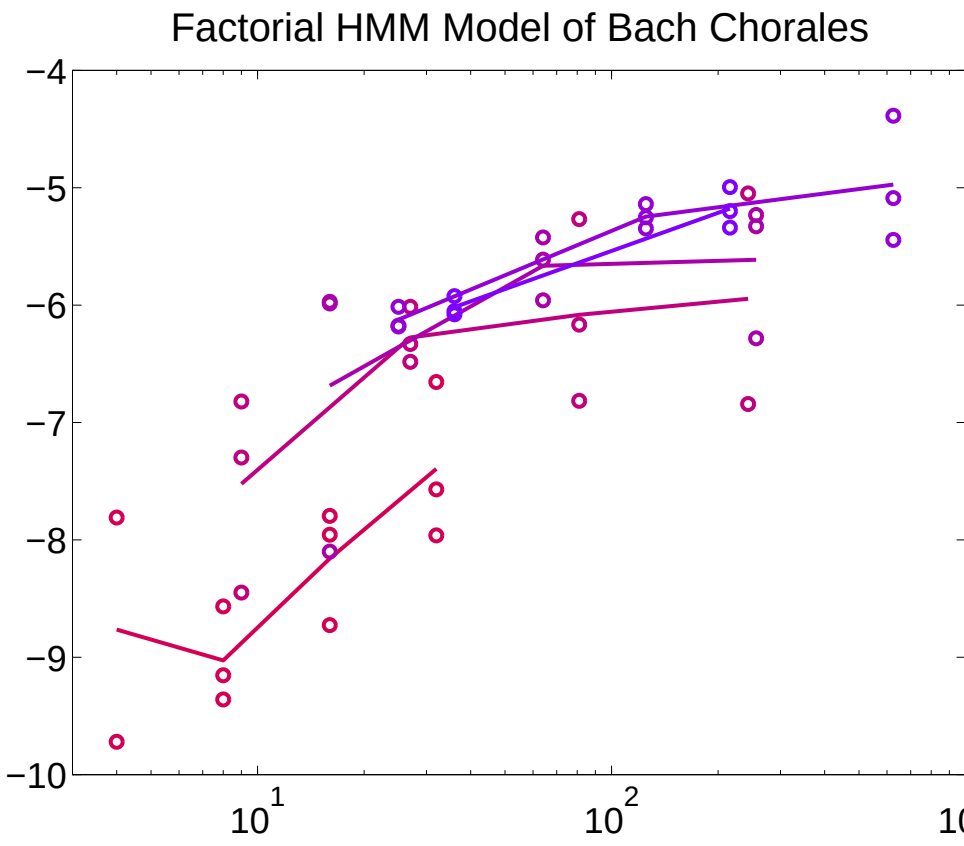
66 chorale melodies of 40 events each:

- training: 30 melodies
- validation: 18 melodies
- test: 18 melodies

See Conklin and Witten (1995) for data and a more musically informed model.

Factorial HMMs: Results on Bach Chorales





HMM Pseudocode: Inference (E step)

Forward-backward including scaling tricks

$$q_j(t) = A_j(\mathbf{y}_t)$$

$$\alpha(1) = \pi .* q(1) \qquad \rho(1) = \sum \alpha(1) \qquad \alpha(1) = \alpha(1)/\rho(1)$$

$$\alpha(t) = (T' * \alpha(t-1)) .* q(t) \qquad \rho(t) = \sum \alpha(t) \qquad \alpha(t) = \alpha(t)/\rho(t) \qquad [t = 2 : \tau]$$

$$\beta(\tau) = 1$$

$$\beta(t) = T * (\beta(t+1) .* q(t+1)/\rho(t+1) \qquad [t = (\tau - 1) : 1]$$

$$\xi = 0$$

$$\xi = \xi + T .* (\alpha(t) * (\beta(t+1) .* q(t+1))')/\rho(t+1) \qquad [t = 1 : (\tau - 1)]$$

$$\gamma = (\alpha .* \beta)$$

$$\log \mathsf{P}(\mathbf{y}_1^\tau) = \sum_t \log(\rho(t))$$

HMM Pseudocode: Parameter Re-estimation (M step)

Baum-Welch parameter updates:

For each sequence, run forward–backward to get γ and ξ, then

$$\delta_j = 0 \qquad \hat{T}_{ij} = 0 \qquad \hat{\pi} = 0 \qquad \hat{A} = 0$$

$$\hat{T} = \hat{T} + \xi \qquad \hat{\pi} = \hat{\pi} + \gamma(1) \qquad \delta = \delta + \sum_t \gamma(t)$$

$$\hat{A}_j(\mathbf{y}) = \sum_{t:\mathbf{y}_t = y} \gamma_j(t) \qquad \text{or} \qquad \hat{A} = \hat{A} + \sum_t \mathbf{y}_t \gamma(t)$$

$$\hat{T}_{ij} = \hat{T}_{ij} / \sum_k \hat{T}_{kj} \qquad \hat{\pi} = \hat{\pi} / \sum \hat{\pi} \qquad \hat{A}_j = \hat{A}_j / \delta_j$$

Some HMM History

- Markov ('13) and later Shannon ('48,'51) studied *Markov chains*.
- Baum and colleagues (BP'66, BE'67, BS'68, BPSW'70, B'72) 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.

Some References for SSMs and HMMs

ZG papers available at: www.gatsby.ucl.ac.uk/∼zoubin/papers.html

- 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*

- Roweis, S. and Ghahramani, Z. (2000) An EM Algorithm for Identification of Nonlinear Dynamical Systems. Preprint. *We show how EM can be exteded to learning nonlinear dynamics, using an Extended Kalman smoother in the approximate E step*

- Minka, T. (1999) From Hidden Markov Models to Linear Dynamical Systems http://www.stat.cmu.edu/∼minka/papers/learning.html

- Welling (2002) The Kalman Filter (class notes). Available at: http://www.cs.toronto.edu/∼welling/classnotes/classnotes.html