Time Series Models:

Hidden Markov Models & Linear Dynamical Systems

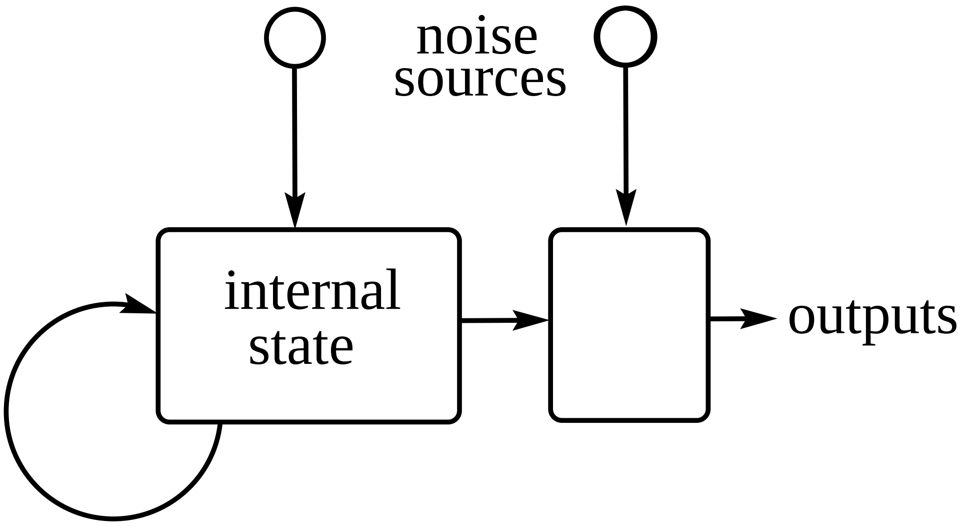
Sam Roweis

Gatsby Computational Neuroscience Unit

August 1999

Noise Driven Feedback Systems

- Generative models for time-series:
  Need *noise* and system *state*.



- Time: discrete
  States: discrete or continuous
  Outputs: discrete or continuous
- Familiar territory: you've probably seen some of these models before.
  Discrete state:
  - Moore and Mealy machines (engineering)
  - stochastic finite state automata (CS)

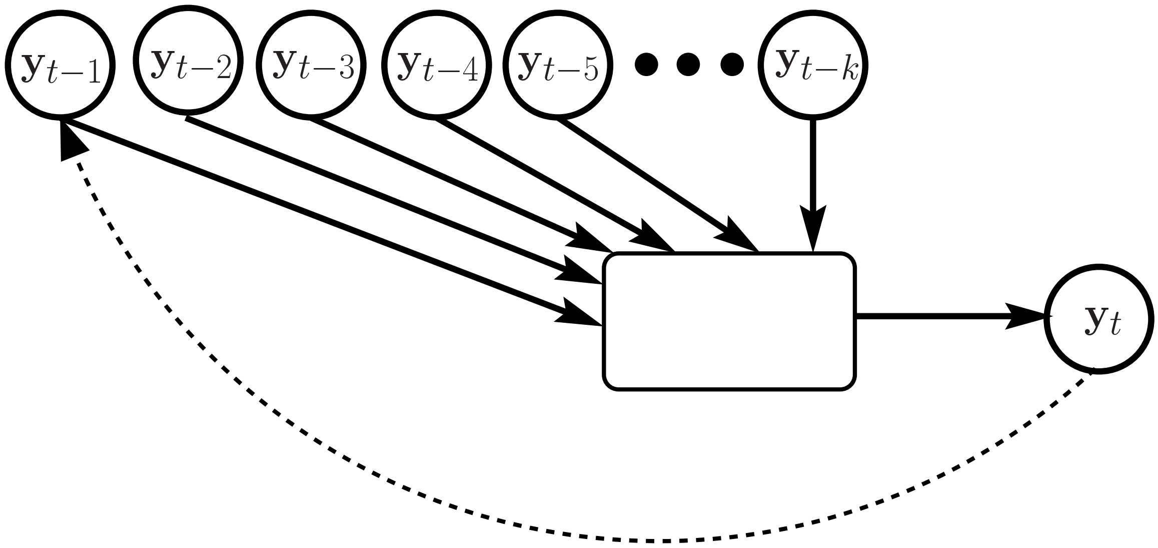
  Continuous state:
  - ARMA models (engineering)
  - Jordan and Elman recurrent networks

Markov Models

- Next output depends on previous output(s):

$$\mathbf{y}_t = f(\mathbf{y}_{t-1}, \mathbf{y}_{t-2}, \dots)$$

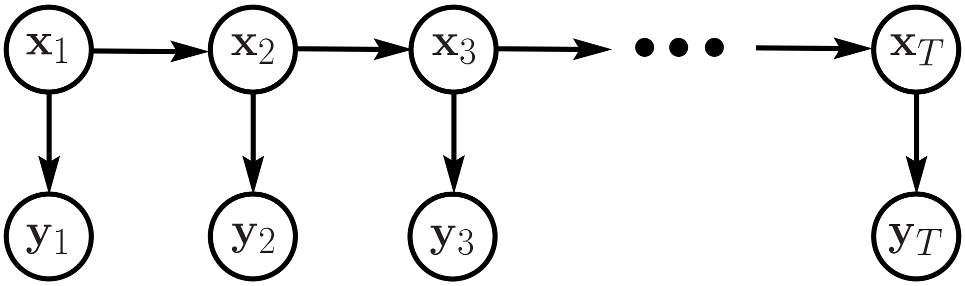
order is number of previous outputs



- Discrete state: finite state automaton
  Continuous state: AR model (IIR filter)
  if $f(\cdot)$ is linear

- Problems:
  - need big order to remember past "events"
  - output noise is confounded with state noise

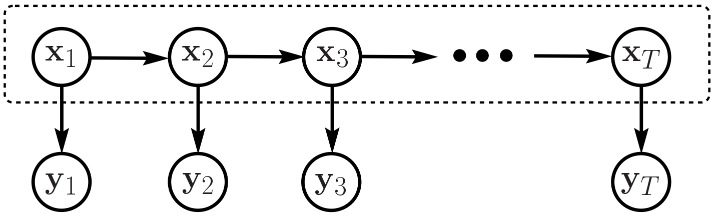
Graphical Model for HMM/LDS



- Hidden states $\{\mathbf{x}_t\}$, outputs $\{\mathbf{y}_t\}$
  Joint probability factorizes:

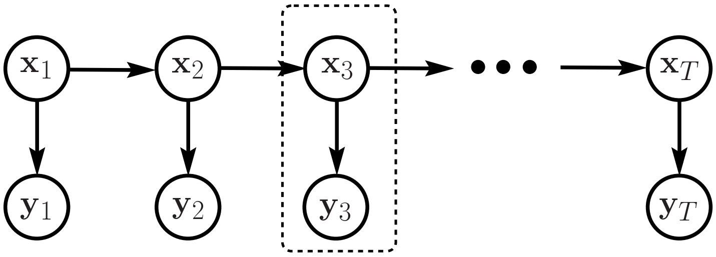
$$\mathsf{P}(\{\mathbf{x}\}, \{\mathbf{y}\}) = \prod_{t=1}^{T} \mathsf{P}(\mathbf{x}_t|\mathbf{x}_{t-1})\mathsf{P}(\mathbf{y}_t|\mathbf{x}_t)$$

- you can think of this as:
  Markov chain with stochastic measurements.
  Gauss-Markov process in a pancake.



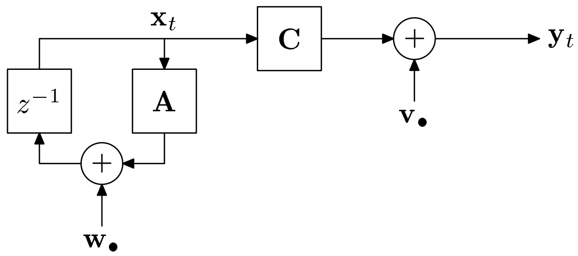
or

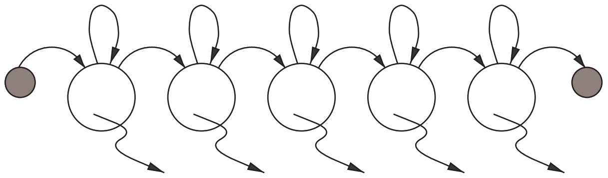
Mixture model with states coupled across time.
Factor analysis through time.



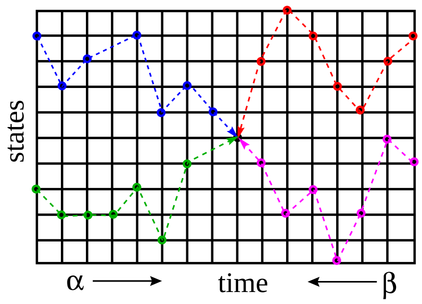
Outline x2

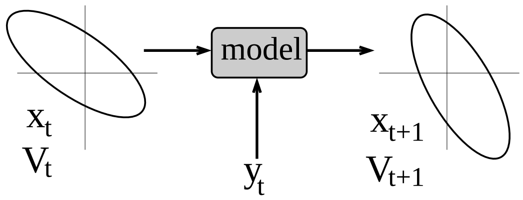
- Generative Models



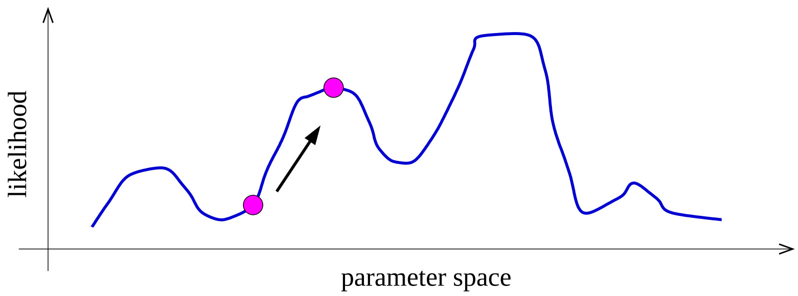


- Likelihood Evaluation
- State inference: why and how?





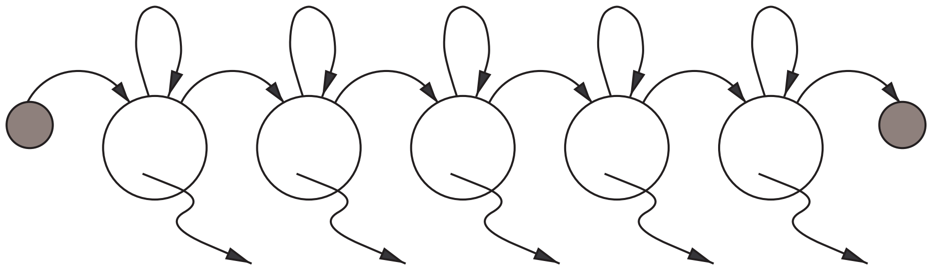
- Parameter Estimation



- Practical Pitfalls

HMM Generative Model

- plain-vanilla HMM $\equiv$
  "probabilistic function of a Markov chain":
  1. Use a 1st-order Markov chain to generate a hidden state sequence (path):
  $$\mathsf{P}(x_1 = j) = \pi_j$$
  $$\mathsf{P}(x_{t+1} = j | x_t = i) = T_{ij}$$
  2. Use a set of output prob. distributions $A_j(\cdot)$ (one per state) to convert this state path into a sequence of observable symbols or vectors
  $$\mathsf{P}(\mathbf{y}_t = y | x_t = j) = A_j(y)$$



- Notes:
  - Even though hidden state seq. is 1st-order Markov, the output process is not Markov of *any* order
    [ex. 1111121111311121111131 . . .]
  - Discrete state, discrete output models can approximate any continuous dynamics and observation mapping even if nonlinear; however lose ability to interpolate

Probability of an Observed Sequence

- To evaluate the probability, we want:

$$\sum_{\text{all paths}} \mathsf{P}(\text{ observed outputs} \,|\, \text{state path })$$

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

- But: there exists a clever "forward recursion" to compute the sum efficiently. Define $\alpha_j(t)$:

$$\alpha_j(t) = \mathsf{P}(\, \mathbf{y}_1^t \,,\, x_t = j \,)$$

  Now induction comes to our rescue:

$$\alpha_j(1) = \pi_j A_j(\mathbf{y}_1)$$

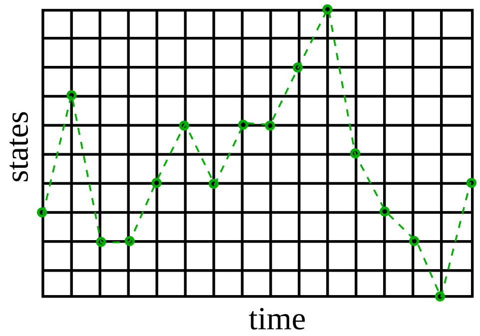
$$\alpha_k(t+1) = \left(\sum_j \alpha_j(t) T_{jk} \right) A_k(\mathbf{y}_{t+1})$$

  Which enables us to easily (cheaply) compute the desired likelihood L since we know we must end in some possible state:

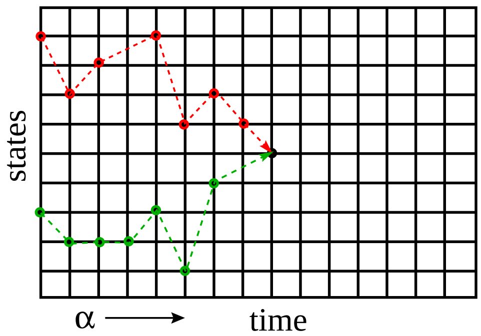
$$L = \sum_k \alpha_k(\tau)$$

Bugs on a Grid

- Naive algorithm:
  1. start bug in each state at $t=1$ holding value 0
  2. move each bug forward in time: make copies & increment the value of each copy by transition prob. $+$ output emission prob.
  3. go to 2 until all bugs have reached time τ
  4. sum up values on all bugs



- 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

- 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 observations:

$$\gamma_i(t) = \mathsf{P}(x_t = i \mid \mathbf{y}_1^T) = \alpha_i(t)\beta_i(t)/L$$

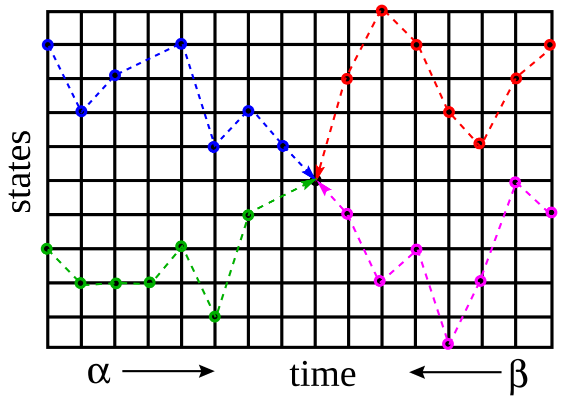
where we defined:

$$\beta_j(t) = \mathsf{P}(\mathbf{y}_{t+1}^T \mid x_t = j\ )$$

- there is also a simple recursion for $\beta_j(t)$:

$$\beta_j(t) = \sum_i T_{ij}\beta_i(t+1)A_i(\mathbf{y}_{t+1})$$

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

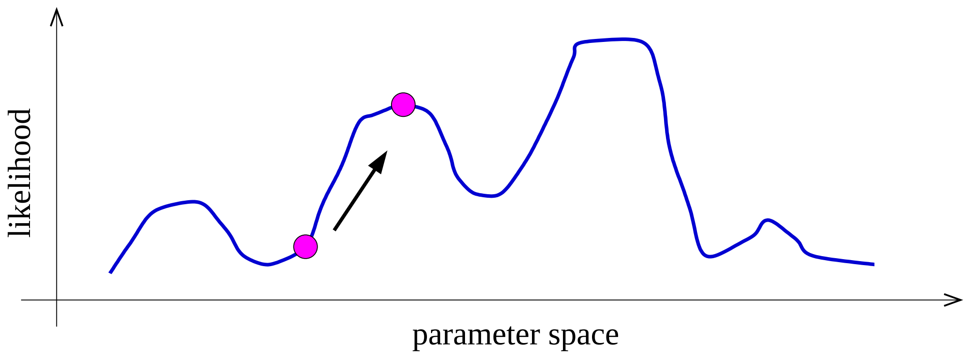


- Bugs again: we just let the bugs run forward from time 0 to t and backward from time τ to t.

Baum-Welch Training

1. Intuition: if only we *knew* the true state path then ML parameter estimation would be trivial.
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, and so on ...
4. Turns out: 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: we can also *prove* that finding the ML parameters is NP complete, so initial conditions matter a lot and convergence is hard to tell.

Viterbi Decoding

- The numbers $\gamma_j(t)$ above gave the probability distribution over all states at any time.
- By choosing the state $\gamma_*(t)$ with the largest probability 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 likelihood of generating the data. In fact it may be a path of prob. zero!
- To find the single best path, we do *Viterbi decoding* which is just Bellman's dynamic programming algorithm applied to this problem.
- The recursions look the same, 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 decode.

New Parameters are just Ratios of Frequency Counts

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

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

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

$$\xi_{ij}(t) = \alpha_i(t) T_{ij} A_j(\mathbf{y}_{t+1}) \beta_j(t+1)/L$$

so the estimated transition probabilities are:

$$\hat{T}_{ij} = \sum_{t=1}^{\tau-1} \xi_{ij}(t) \Bigg/ \sum_{t=1}^{\tau-1} \gamma_j(t)$$

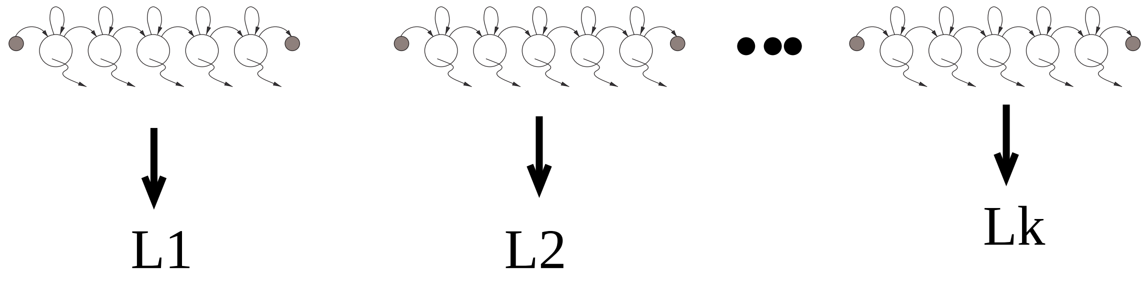
- The output distributions are the expected number of times we observe a particular symbol in a particular state:

$$\hat{A}_j(y) = \sum_{t|\mathbf{y}_t=y} \gamma_j(t) \Bigg/ \sum_{t=1}^{\tau} \gamma_j(t)$$

or merely the state-probability weighted mean and variance for a continuous Gaussian model.

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



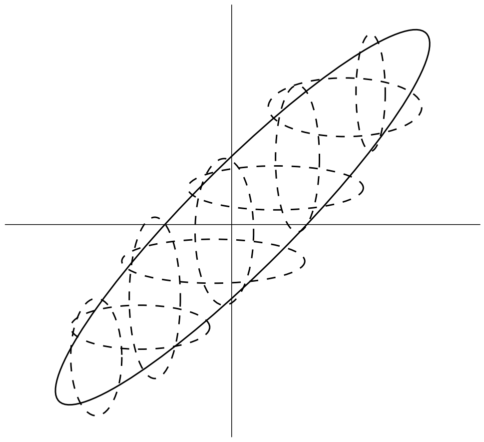
- This requires the solution of two problems:
  1. Given model, evaluate prob. of a sequence.
     (We can do this exactly and efficiently.)
  2. Give some training sequences, estimate model
     parameters. (We can find the local maximum
     of parameter space nearest our starting point.)

HMM Practicalities

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

$$\rho_t = \mathsf{P}(\mathbf{y}_t|\mathbf{y}_1^{t-1}) \qquad \alpha(t) = \tilde{\alpha}(t)\prod_{t'=1}^{t}\rho_{t'}$$

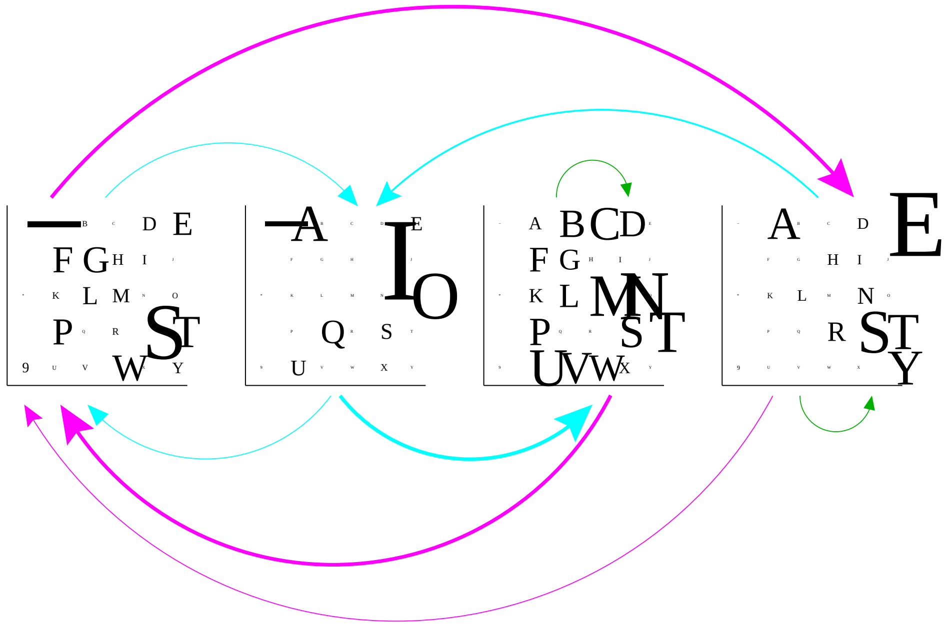
- Training data requirements: full covariance matrices in high dimensions or discrete symbol models with many symbols have *lots* of parameters. Instead, we often use mixtures of diagonal covariance Gaussians or tie states together in other ways.



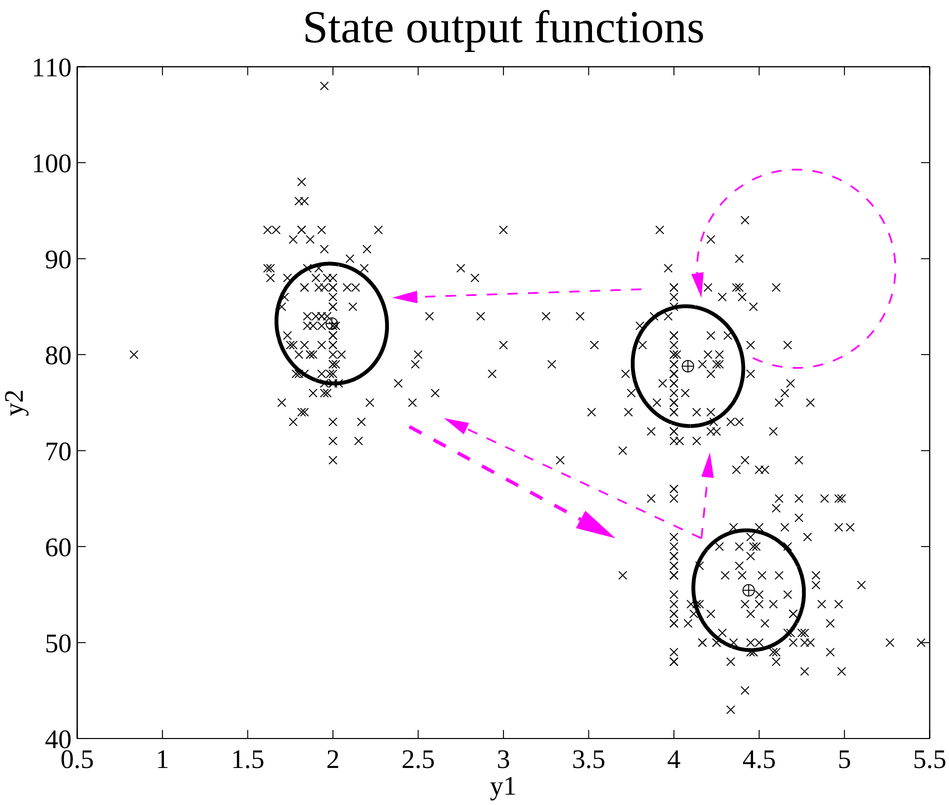
- Multiple observation sequences: can be dealt with by averaging numerators and averaging denominators in the ratios given above.

HMM Example

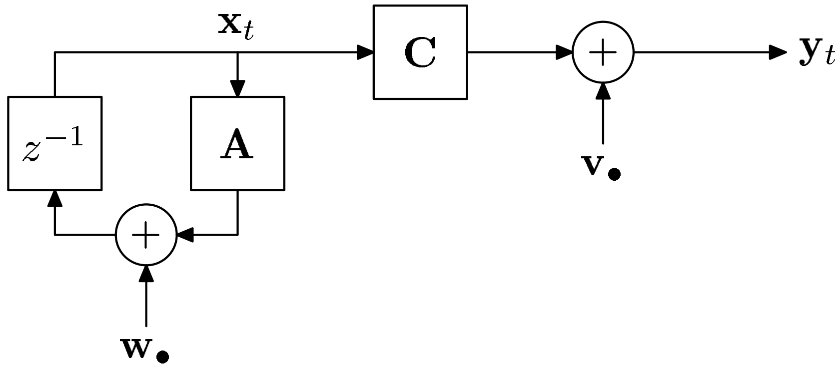
- Character sequences (discrete outputs)



- Geyser data (continuous outputs)



Linear Dynamical Systems



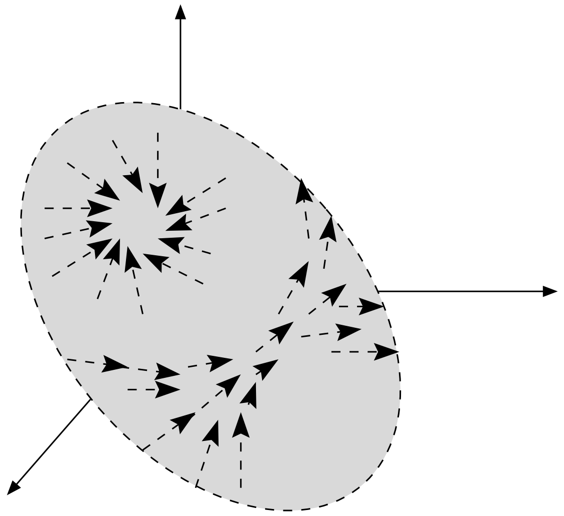
- Also called state-space models.
- Gauss-Markov continuous state process:

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

  observed through the "lens" of a noisy linear embedding:

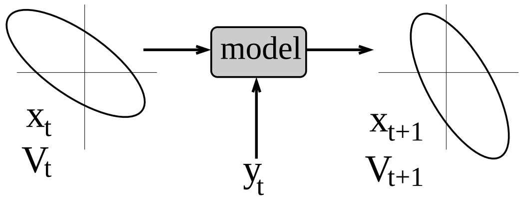
$$\mathbf{y}_t = \mathbf{C}\mathbf{x}_t + \mathbf{v}_t$$

- Noises $\mathbf{w}_\bullet$ and $\mathbf{v}_\bullet$ are temporally white and uncorrelated with everything else
- Think of this as "matrix flow in a pancake"



Discrete Kalman Filter

- DKF computes likelihood and optimal (causal) state estimates given model and observations.
- Elegant recursive form just like forward algorithm



- But now we have a distribution over state at each time rather than a histogram.
  $\Rightarrow$ Approximation! (& associated headaches)
- Constraint for linear-time algorithms:
  Form of distribution needs to be
  closed under multiplication.
  $\Rightarrow$ Exponential family (Pitman-Koopman thm.)
- If Gaussians are used, then *any* combination of hidden variables $\mathbf{x}_t$ and observed variables $\mathbf{y}_t$ is jointly Gaussian.

Innovations and Bayes Rule

- Say you have a Gaussian prior on a variable and you get some Gaussian evidence.
  Q: What is your posterior?

$$
\begin{aligned}
\mathsf{P}(\mathbf{x}) &= \mathcal{N}\left(\mathbf{x}_0, \mathbf{V}_0\right) \\
\mathsf{P}(\mathbf{y}|\mathbf{x}) &= \mathcal{N}\left(\mathbf{x}, \mathbf{R}\right) \\
\mathsf{P}(\mathbf{x}|\mathbf{y}) &= \mathcal{N}\left(\mathbf{V}\mathbf{V}_0^{-1}\mathbf{x}_0 + \mathbf{V}\mathbf{R}^{-1}\mathbf{y}, \mathbf{V}\right) \\
\mathbf{V} &= (\mathbf{V}_0^{-1} + \mathbf{R}^{-1})^{-1}
\end{aligned}
$$

  A: Linear combination of your prior and your measurement, weighted by relative covariances.

- Simple example:
  online mean/covariance estimators.

$$
\boldsymbol{\mu}_n = \left(1 - \frac{1}{n}\right)\boldsymbol{\mu}_{n-1} + \left(\frac{1}{n}\right)\mathbf{x}_n
$$

$$
\boldsymbol{\Sigma}_n = \left(1 - \frac{1}{n}\right)\boldsymbol{\Sigma}_{n-1} + \left(\frac{1}{n} - \frac{1}{n^2}\right)\left(\mathbf{x}_n - \boldsymbol{\mu}_{n-1}\right)^2
$$

  You are already doing Kalman filtering!

Kalman Filter Equations

- Want to compute $\mathsf{P}(\mathbf{x}_t|\mathbf{y}_1^t) = \mathcal{N}(\mathbf{x}_t, \mathbf{V}_t)$

1. Run the model forward:

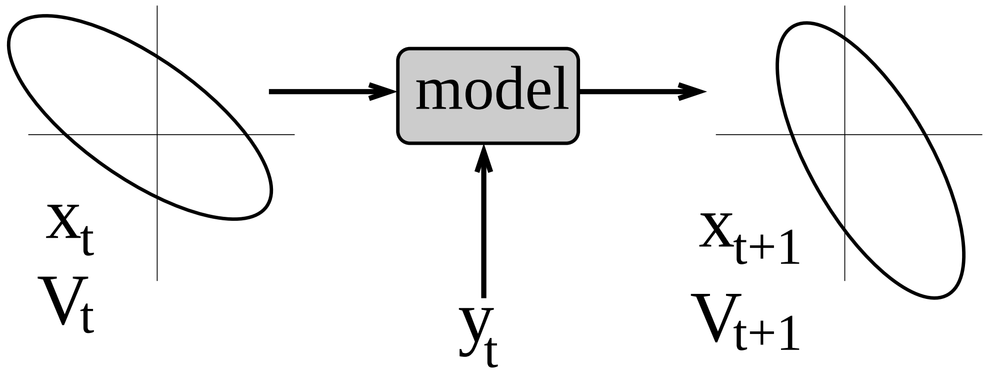
$$\mathbf{x}^+ = \mathbf{A}\mathbf{x}_t$$
$$\mathbf{V}^+ = \mathbf{A}\mathbf{V}_t\mathbf{A}^T + \mathbf{Q}$$

2. Compute how "reliable" your prediction is:

$$\mathbf{K} = \mathbf{V}^+\mathbf{C}^T\left(\mathbf{C}\mathbf{V}^+\mathbf{C}^T + \mathbf{R}\right)^{-1}$$

3. Update your estimate as a mix of prediction and data correction:

$$\mathbf{x}_{t+1} = \mathbf{x}^+ + \mathbf{K}\left(\mathbf{y}_{t+1} - \mathbf{C}\mathbf{x}^+\right)$$
$$\mathbf{V}_{t+1} = (\mathbf{I} - \mathbf{K}\mathbf{C})\,\mathbf{V}^+$$

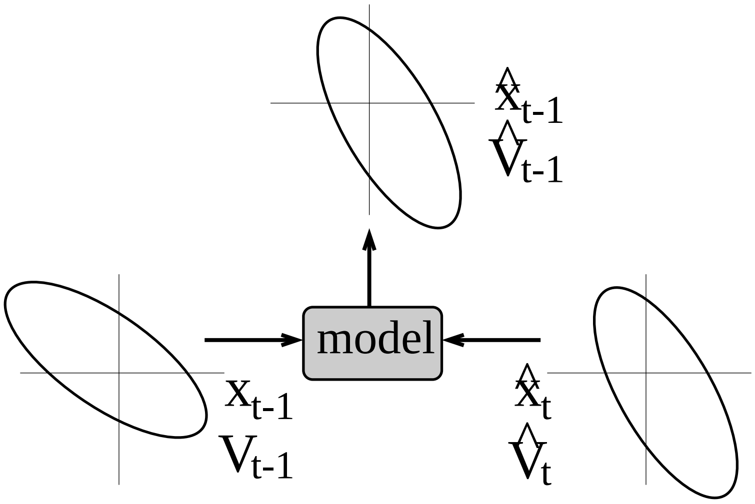


- initialize using $\mathbf{x}^+ = \mathbf{x}_0$ and $\mathbf{V}^+ = \mathbf{V}_0$
- NB: Kalman gain $\mathbf{K}$ evolves independent of data

Kalman smoothing

- Want to compute $\mathsf{P}(\mathbf{x}_t|\mathbf{y}_1^{\mathcal{T}}) = \mathcal{N}\left(\hat{\mathbf{x}}_t, \hat{\mathbf{V}}_t\right)$

$$\mathbf{V}^+ = \mathbf{A}\mathbf{V}_{t-1}\mathbf{A}^T + \mathbf{Q}$$
$$\mathbf{J} = \mathbf{V}_{t-1}\mathbf{A}^T\left(\mathbf{V}^+\right)^{-1}$$
$$\hat{\mathbf{x}}_{t-1} = \mathbf{x}_{t-1} + \mathbf{J}\left(\hat{\mathbf{x}}_t - \mathbf{A}\mathbf{x}_{t-1}\right)$$
$$\hat{\mathbf{V}}_{t-1} = \mathbf{V}_{t-1} + \mathbf{J}\left(\hat{\mathbf{V}}_t - \mathbf{V}^+\right)\mathbf{J}^T$$



- initialize using $\hat{\mathbf{x}}_{\mathcal{T}} = \mathbf{x}_{\mathcal{T}}$ and $\hat{\mathbf{V}}_{\mathcal{T}} = \mathbf{V}_{\mathcal{T}}$
- For the afficionado:
  This is a “triangular” backwards-recursive filter unlike the “two-filter” version for HMMs.

  It is possible to derive triangular HMM filters and two-filter Kalman smoothers.

  This has to do with different versions of the *junction tree algorithm* for belief propagation.

Parameter Estimation with EM

- For EM we need the cross-time covariances:

$$\mathsf{P}(\mathbf{x}_{t-1}, \mathbf{x}_t | \mathbf{y}_1^{\mathcal{T}}) = \mathcal{N}\left(\begin{bmatrix} \hat{\mathbf{x}}_{t-1} \\ \hat{\mathbf{x}}_t \end{bmatrix}, \begin{bmatrix} \hat{\mathbf{V}}_{t-1} & \hat{\mathbf{V}}_{t-1}^{t} \\ \hat{\mathbf{V}}_{t}^{t-1} & \hat{\mathbf{V}}_t \end{bmatrix}\right)$$

- These can be computed directly from filtered and smoothed estimates (although a recursion is also possible)

$$\hat{\mathbf{V}}_{t-1}^{t} = \mathbf{V}_{t-1}\mathbf{A}^{T}(\mathbf{A}\mathbf{V}_{t-1}\mathbf{A}^{T} + \mathbf{Q})^{-1}\hat{\mathbf{V}}_t$$

- Given these moments, maximize prob. of joint data with respect to parameters

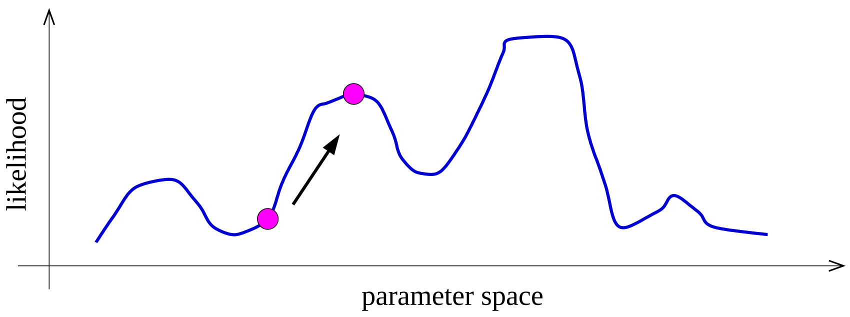
$$\mathbf{C} = \left(\sum \mathbf{y}_t \hat{\mathbf{x}}_t\right)\left(\sum \hat{\mathbf{x}}_t \hat{\mathbf{x}}_t + \hat{\mathbf{V}}_t\right)^{-1}$$

$$\mathbf{R} = \sum \left(\mathbf{y}_t \mathbf{y}_t - \mathbf{C}\hat{\mathbf{x}}_t \mathbf{y}_t\right)/\tau$$

$$\mathbf{A} = \left(\sum \hat{\mathbf{x}}_{t+1}\hat{\mathbf{x}}_t + \hat{\mathbf{V}}_{t+1}^{t}\right)\left(\sum \hat{\mathbf{x}}_t \hat{\mathbf{x}}_t + \hat{\mathbf{V}}_t\right)^{-1}$$

$$\mathbf{Q} = \sum \left(\hat{\mathbf{x}}_t \hat{\mathbf{x}}_t + \hat{\mathbf{V}}_t - \mathbf{A}\left(\hat{\mathbf{x}}_{t-1}\hat{\mathbf{x}}_t + \hat{\mathbf{V}}_{t-1}^{t}\right)\right)/(\tau - 1)$$

$$\mathbf{x}_0 = \hat{\mathbf{x}}_1 \qquad \mathbf{V}_0 = \hat{\mathbf{V}}_1$$

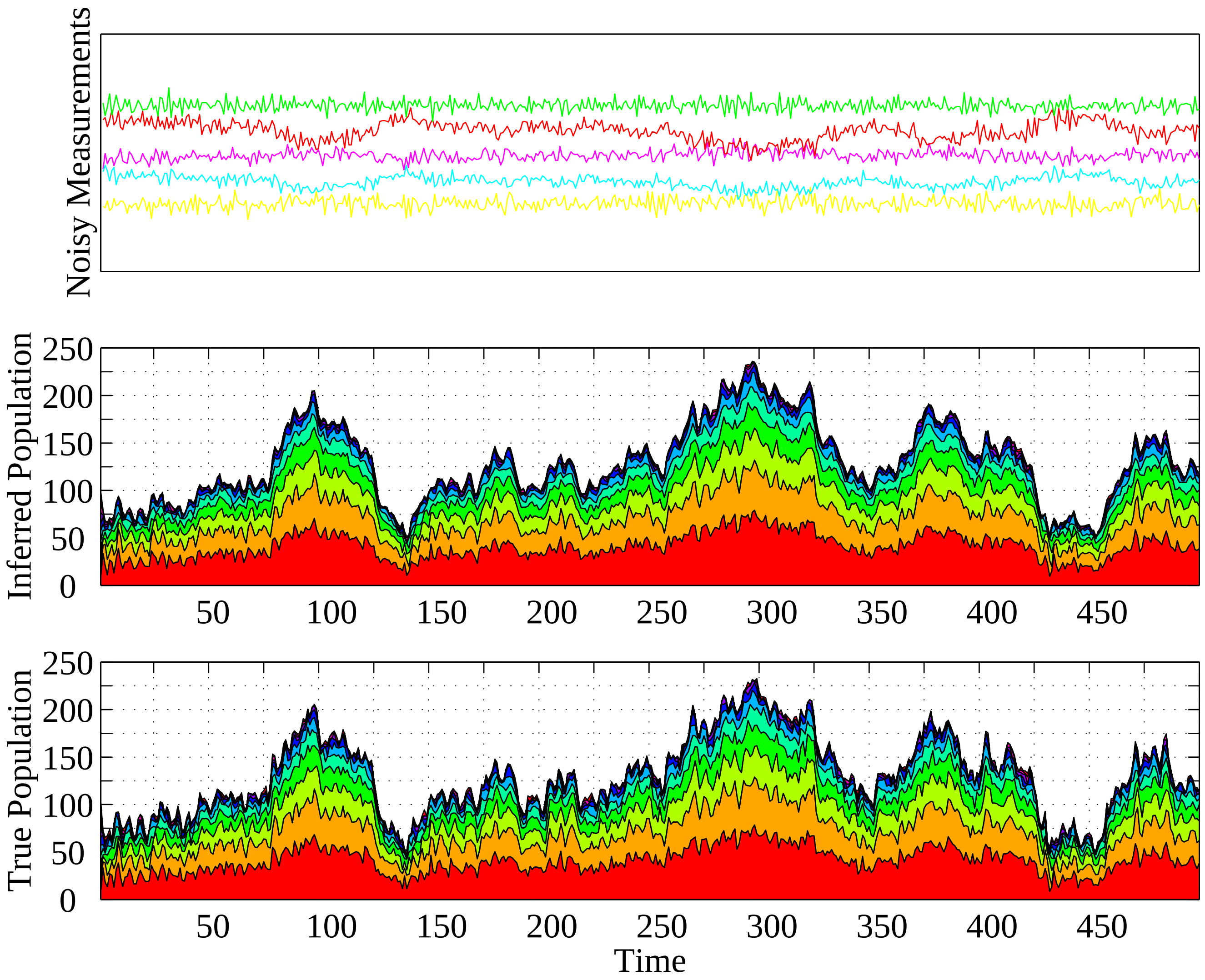


LDS Practicalities

- Initialization is very important.
  Factor analysis on the outputs is a good choice.
- Scaling again.
  But this time just to compute likelihood.
  $$\rho_t = \mathsf{P}(\mathbf{y}_t|\mathbf{y}_1^{t-1}) = \mathcal{N}\left(\mathbf{C}\mathbf{x}^+, \mathbf{C}\mathbf{V}^+\mathbf{C}^T + \mathbf{R}\right)|_{\mathbf{y}_t}$$
- Computational load: you spend most of your cycles working out tiny contributions.
  "Sparse" variants of EM skip computation of most estimators on most iterations; computing fully only every once in a while.
- Multiple observation sequences. Only tricky part is the initial covariance estimate $\mathbf{V}_0$.
- Inputs are easy to incorporate into the state evolution and output functions.
  (Also possible for HMMs.)

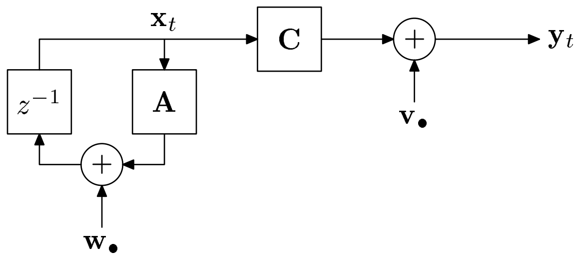
LDS Example

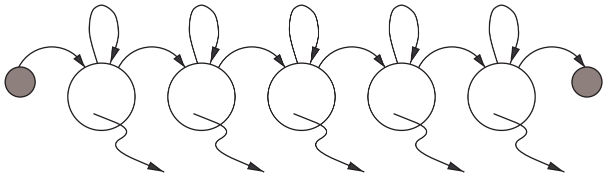
- Population model:
  state ⇔ population histogram
  first row of A ⇔ birthrates
  subdiagonal of A ⇔ 1-deathrates
  Q ⇔ immigration/emmigration
  C ⇔ noisy indicators



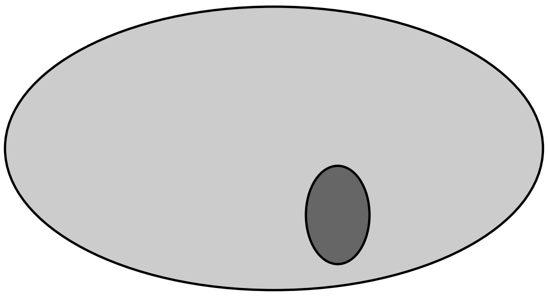
Relationship to HMMs

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



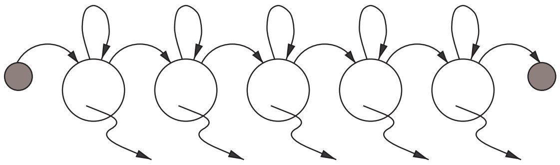


- forward algorithm $\Leftrightarrow$ Discrete Kalman Filter
  forward-backward $\Leftrightarrow$ Kalman Smoothing
  Viterbi decoding $\Leftrightarrow$ no equivalent
- Equivalent number of output symbols?
  Compare volume of data with det(R).
- Equivalent number of states?
  Compare volume of data with det(Q)det(C).
- What sets (arbitrary) scale of hidden state?
  Scale of Q (usually fixed at I).

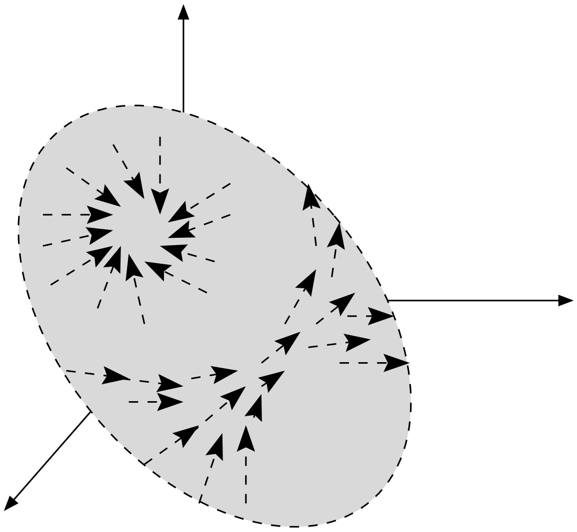


Strengths and Weaknesses

- Continuous vector state is very powerful.
  (For an HMM to communicate N bits of information into future, it needs 2^N states!)

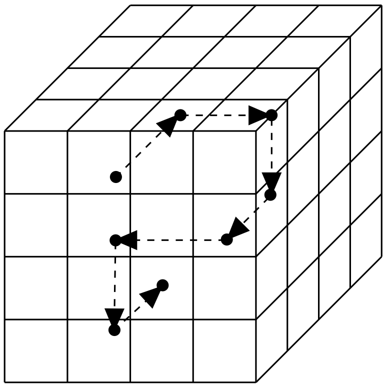


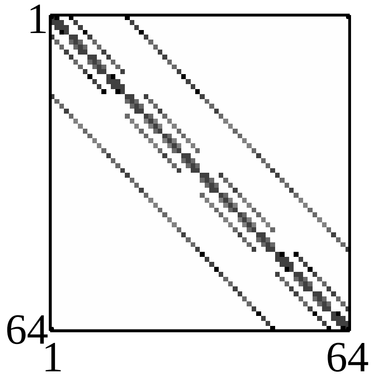
- Linear-Gaussian output/dynamics are very weak.
  (e.g. even using a mixture of Gaussians in place of single Gaussians is tricky)
  HMMs can represent arbitrary stochastic dynamics and output mappings.



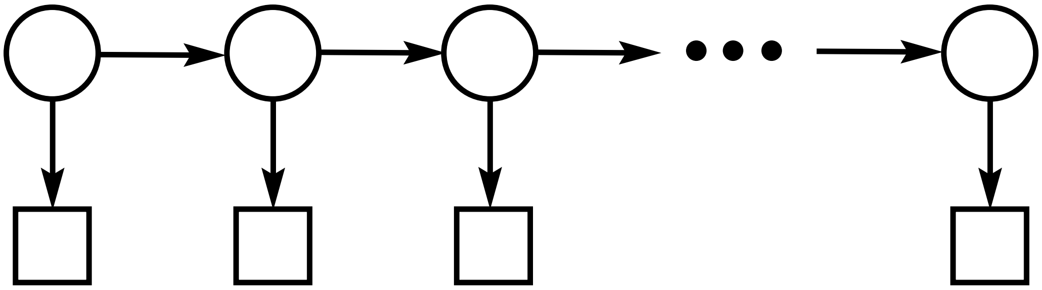
A Taste of Research

- Constrained HMMs

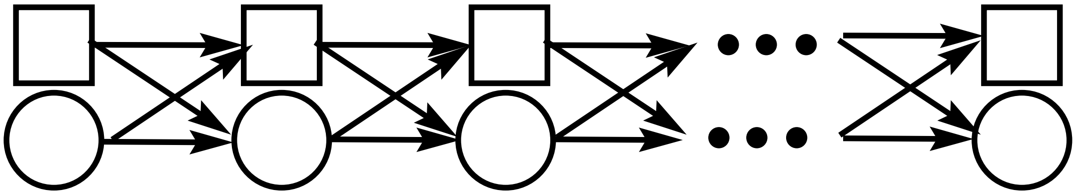




- Continuous state models with discrete outputs
  for time series and static data



- Hybrid systems ⇔
  Mixed continuous & discrete states



HMM Pseudocode

- Forward-backward including scaling tricks

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

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

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

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

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

$$\xi = 0$$

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

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

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

- Baum-Welch parameter updates

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

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

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

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

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

LDS Pseudocode

- Kalman filter/smoother including scaling tricks

$$\mathbf{x}^+ = \mathbf{x}_0 \qquad \mathbf{V}^+ = \mathbf{V}_0$$

$$\rho_t = \mathcal{N}\left(\mathbf{C}\mathbf{x}^+, \mathbf{C}\mathbf{V}^+\mathbf{C}' + \mathbf{R}\right)|_{\mathbf{y}_t} \qquad [t = 1 : \tau]$$

$$\mathbf{K} = \mathbf{V}^+\mathbf{C}'(\mathbf{C}\mathbf{V}^+\mathbf{C}' + \mathbf{R})^{-1}$$

$$\mathbf{x}_t = \mathbf{x}^+ + \mathbf{K}(\mathbf{y}_t - \mathbf{C}\mathbf{x}^+) \qquad \mathbf{V}_t = (\mathbf{I} - \mathbf{K}\mathbf{C})\mathbf{V}^+$$

$$\mathbf{x}^+ = \mathbf{A}\mathbf{x}_t \qquad \mathbf{V}^+ = \mathbf{A}\mathbf{V}_t\mathbf{A}' + \mathbf{Q}$$

$$\hat{\mathbf{x}}_\tau = \mathbf{x}_\tau \qquad \hat{\mathbf{V}}_\tau = \mathbf{V}_\tau$$

$$\mathbf{V}^+ = \mathbf{A}\mathbf{V}_t\mathbf{A}' + \mathbf{Q} \qquad [t = (\tau - 1) : 1]$$

$$\mathbf{J} = \mathbf{V}_t\mathbf{A}'(\mathbf{V}^+)^{-1}$$

$$\hat{\mathbf{x}}_t = \mathbf{x}_t + \mathbf{J}(\hat{\mathbf{x}}_{t+1} - \mathbf{A}\mathbf{x}_t) \qquad \hat{\mathbf{V}}_t = \mathbf{V}_t + \mathbf{J}(\hat{\mathbf{V}}_{t+1} - \mathbf{V}^+)\mathbf{J}'$$

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

- EM parameter updates

$$\delta = 0 \qquad \beta = 0 \qquad \gamma = 0 \qquad \alpha = 0 \qquad \mathbf{x}_0 = 0$$

for each sequence, run Kalman smoother to get $\mathbf{x}_t, \hat{\mathbf{x}}_t, \mathbf{V}_t$ and $\hat{\mathbf{V}}_t$

$$\mathbf{x}_0 = \mathbf{x}_0 + \hat{\mathbf{x}}_1/N \qquad \hat{\mathbf{V}}_{t-1}^t = \mathbf{V}_{t-1}\mathbf{A}'(\mathbf{A}\mathbf{V}_{t-1}\mathbf{A}' + \mathbf{Q})^{-1}\hat{\mathbf{V}}_t$$

$$\delta = \delta + \sum_t \mathbf{y}_t\hat{\mathbf{x}}_t' \qquad \beta = \beta + \sum_{t=2}^{\tau} \hat{\mathbf{x}}_t\hat{\mathbf{x}}_{t-1}' + \hat{\mathbf{V}}_t^{t-1} \qquad \alpha = \alpha + \sum_t \mathbf{y}_t\mathbf{y}_t'$$

$$\gamma = \gamma + \sum_t \hat{\mathbf{x}}_t\hat{\mathbf{x}}_t' + \hat{\mathbf{V}}_t \qquad \gamma_\tau = \gamma_\tau + \hat{\mathbf{x}}_\tau\hat{\mathbf{x}}_\tau' + \hat{\mathbf{V}}_\tau \qquad \gamma_1 = \gamma_1 + \hat{\mathbf{x}}_1\hat{\mathbf{x}}_1' + \hat{\mathbf{V}}_1$$

$$\mathbf{C} = \delta\gamma^{-1} \qquad \mathbf{A} = \beta(\gamma - \gamma_\tau)^{-1}$$

$$\mathbf{R} = (\alpha - \mathbf{C}\delta')/\tau_N \qquad \mathbf{Q} = (\gamma - \gamma_1 - \mathbf{A}\beta')/(\tau_N - N)$$

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) (now Qualcomm) came up with an efficient optimal decoder for state inference.
- Applications to speech were pioneered independently by:
  - Baker ('75) at CMU (now Dragon)
  - Jelinek's group ('75) at IBM (now Hopkins)
  - 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.
- A landmark open symposium in Princeton ('80) hosted by IDA reviewed work till then.

References

- *A Unifying Review of Linear Gaussian Models.*
  Sam Roweis & Zoubin Ghahramani.
  Neural Computation 11(2) (1999) pp.305-345
  http://www.gatsby.ucl.ac.uk/~roweis/publications.html
- *Parameter estimation for linear dynamical systems.*
  Zoubin Ghahramani & Geoff Hinton.
  University of Toronto Technical Report CRG-TR-96-2
  ftp://ftp.cs.toronto.edu/pub/zoubin/tr-96-2.ps.gz
- Online software
  http://www.gatsby.ucl.ac.uk/~roweis/code.html