Lecture 10:

Markov and Hidden Markov Models

Sam Roweis

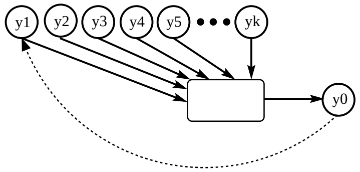
November 13, 2002

Markov Models

- Use past as state. Next output depends on previous output(s):

$$\mathbf{y}_t = f[\mathbf{y}_{t-1}, \mathbf{y}_{t-2}, \ldots]$$

order is number of previous outputs



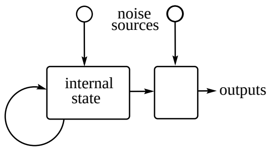
- Add noise to make the system probabilistic:

$$p(\mathbf{y}_t | \mathbf{y}_{t-1}, \mathbf{y}_{t-2}, \ldots, \mathbf{y}_{t-k})$$

- Markov models have two problems:
  - need big order to remember past "events"
  - output noise is confounded with state noise

Probabilistic Models for Time Series

- Generative models for time-series:
  To get interesting variability need *noise*.
  To get correlations across time, need some system *state*.



- Time: discrete
  States: discrete or continuous
  Outputs: discrete or continuous
- Today: discrete state
  similar to finite state automata; Moore/Mealy machines

Learning Markov Models

- The ML parameter estimates for a simple Markov model are easy:

$$p(\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_T) = \prod_{t=k+1}^{T} p(\mathbf{y}_t | \mathbf{y}_{t-1}, \mathbf{y}_{t-2}, \ldots, \mathbf{y}_{t-k})$$

$$\log p(\{\mathbf{y}\}) = \sum_{t=k+1}^{T} \log p(\mathbf{y}_t | \mathbf{y}_{t-1}, \mathbf{y}_{t-2}, \ldots, \mathbf{y}_{t-k})$$

- Each window of $k+1$ outputs is a training case for the model $p(\mathbf{y}_t | \mathbf{y}_{t-1}, \mathbf{y}_{t-2}, \ldots, \mathbf{y}_{t-k})$.
- Example: for discrete outputs (symbols) and a 2nd-order markov model we can use the multinomial model:

$$p(y_t = m | y_{t-1} = a, y_{t-2} = b) = \alpha_{mab}$$

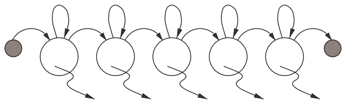
The maximum likelihood values for α are:

$$\alpha^*_{mab} = \frac{\mathrm{num}[t \ s.t. \ y_t = m, y_{t-1} = a, y_{t-2} = b]}{\mathrm{num}[t \ s.t. \ y_{t-1} = a, y_{t-2} = b]}$$

Hidden Markov Models (HMMs)

Add a latent (hidden) variable x_t to improve the model.

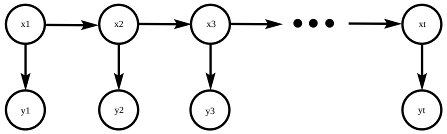
- HMM $\equiv$ " probabilistic function of a Markov chain":
  1. 1st-order Markov chain generates hidden state sequence (path):
  $$\mathsf{P}(x_{t+1} = j | x_t = i) = T_{ij} \qquad \mathsf{P}(x_1 = j) = \pi_j$$
  2. A set of output probability distributions $\mathbf{A}_j(\cdot)$ (one per state) converts state path into sequence of observable symbols/vectors
  $$\mathsf{P}(\mathbf{y}_t = y | x_t = j) = \mathbf{A}_j(\mathbf{y})$$



(state transition diagram)

- Even though hidden state seq. is 1st-order Markov, the output process is not Markov of *any* order
  [ex. 1111121111311121111131. . .]

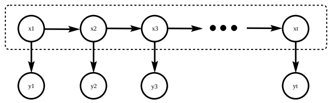
HMM Model Equations



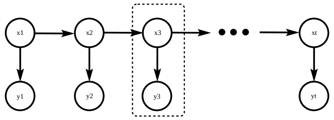
- Hidden states $\{x_t\}$, outputs $\{\mathbf{y}_t\}$
  Joint probability factorizes:
$$\begin{aligned}\mathsf{P}(\{x\},\{\mathbf{y}\}) &= \prod_{t=1}^{T} \mathsf{P}(x_t|x_{t-1})\mathsf{P}(\mathbf{y}_t|x_t) \\ &= \pi_{x_1} \prod_{t=1}^{\tau-1} T_{x_t,x_{t+1}} \prod_{t=1}^{\tau} A_{x_t}(\mathbf{y}_t)\end{aligned}$$
- NB: Data are *not* i.i.d. Everything is coupled across time.
- Three problems: computing probabilities of observed sequences, inference of hidden state sequences, learning of parameters.

Links to Other Models

- You can think of an HMM as:
  A Markov chain with stochastic measurements.



  or
  A mixture model with states coupled across time.



- The future is independent of the past given the present.
  However, conditioning on all the observations couples hidden states.

Probability of an Observed Sequence

- To evaluate the probability $\mathsf{P}(\{\mathbf{y}\})$, we want:
$$\mathsf{P}(\{\mathbf{y}\}) = \sum_{\{x\}} \mathsf{P}(\{x\},\{\mathbf{y}\})$$
$$\mathsf{P}(\text{observed sequence}) = \sum_{\text{all paths}} \mathsf{P}(\text{ observed outputs , state path })$$
- Looks hard! (#paths $= N^\tau$). But joint probability factorizes:
$$\begin{aligned}\mathsf{P}(\{\mathbf{y}\}) &= \sum_{x_1}\sum_{x_2}\cdots\sum_{x_\tau}\prod_{t=1}^{T} \mathsf{P}(x_t|x_{t-1})\mathsf{P}(\mathbf{y}_t|x_t) \\ &= \sum_{x_1} \mathsf{P}(x_1)\mathsf{P}(\mathbf{y}_1|x_1) \sum_{x_2} \mathsf{P}(x_2|x_1)\mathsf{P}(\mathbf{y}_2|x_2)\cdots \\ &\qquad \sum_{x_\tau} \mathsf{P}(x_\tau|x_{\tau-1})\mathsf{P}(\mathbf{y}_\tau|x_\tau)\end{aligned}$$
- By moving the summations inside, we can save a lot of work.

The forward (α) recursion

- We want to compute:

$$L = \mathsf{P}(\{\mathbf{y}\}) = \sum_{\{x\}} \mathsf{P}(\{x\}, \{\mathbf{y}\})$$

- There exists a clever "forward recursion" to compute this huge sum very efficiently. Define $\alpha_j(t)$:

$$\alpha_j(t) = \mathsf{P}(\ \mathbf{y}_1^t\ ,\ x_t = j\ )$$
$$\alpha_j(1) = \pi_j \mathbf{A}_j(\mathbf{y}_1) \qquad \text{induction to the rescue...}$$
$$\alpha_k(t+1) = \{\sum_j \alpha_j(t) T_{jk}\} A_k(\mathbf{y}_{t+1})$$

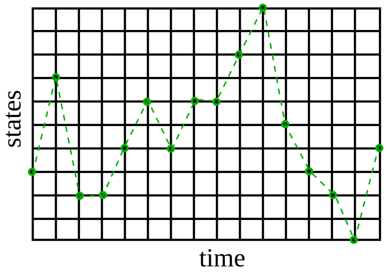
- Notation: $x_a^b \equiv \{x_a, \ldots, x_b\}$; $\mathbf{y}_a^b \equiv \{\mathbf{y}_a, \ldots, \mathbf{y}_b\}$
- This enables us to easily (cheaply) compute the desired likelihood L since we know we must end in some possible state:

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

Bugs on a Grid

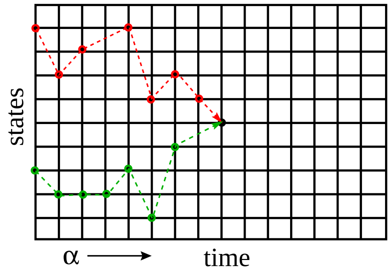
- Naive algorithm:

1. start bug in each state at t=1 holding value 0
2. move each bug forward in time by making copies of it and incrementing the value of each copy by the probability of the transition and output emission
3. go to 2 until all bugs have reached time τ
4. sum up values on all bugs



Bugs on a Grid - Trick

- 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



- This is exactly dynamic programming.

Inference of Hidden States

- What if we we want to estimate the hidden states given observations? To start with, let us estimate a single hidden state:

$$p(x_t|\{\mathbf{y}\}) = \gamma(x_t) = \frac{p(\{\mathbf{y}\}|x_t)p(x_t)}{p(\{\mathbf{y}\})}$$
$$= \frac{p(\mathbf{y}_1^t|x_t)p(\mathbf{y}_{t+1}^\tau|x_t)p(x_t)}{p(\mathbf{y}_1^\tau)}$$
$$= \frac{p(\mathbf{y}_1^t, x_t)p(\mathbf{y}_{t+1}^\tau|x_t)}{p(\mathbf{y}_1^\tau)}$$
$$p(x_t|\{\mathbf{y}\}) = \gamma(x_t) = \frac{\alpha(x_t)\beta(x_t)}{p(\mathbf{y}_1^\tau)}$$

where

$$\alpha_j(t) = \mathsf{P}(\ \mathbf{y}_1^t\ ,\ x_t = j\ )$$
$$\beta_j(t) = p(\mathbf{y}_{t+1}^\tau \mid x_t = j\ )$$
$$\gamma_i(t) = p(x_t = i \mid \mathbf{y}_1^\tau)$$

Forward-Backward Algorithm

- We compute these quantities efficiently using another recursion. Use total prob. of all paths going through state i at time t to compute the *conditional* prob. of being in state i at time t:

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

  where we defined:

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

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

$$\beta_j(t) = \sum_k T_{jk} A_k(\mathbf{y}_{t+1}) \beta_k(t+1)$$
$$\beta_j(\tau) = 1$$

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

Likelihood from Forward-Backward Algorithm

- Since $\sum_{x_t} \gamma(x_t) = 1$, we can compute the likelihood at *any* time using the results of the $\alpha - \beta$ recursions:

$$L = p(\{\mathbf{y}\}) = \sum_{x_t} \alpha(x_t)\beta(x_t)$$

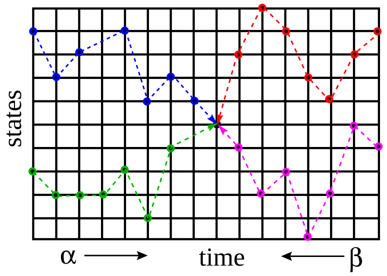
- In the forward calculation we proposed originally, we did this at the final timestep $t = \tau$:

$$L = \sum_{x_\tau} \alpha(x_\tau)$$

  because $\beta_\tau = 1$.
- This is a good way to check your code!

Forward-Backward Algorithm

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



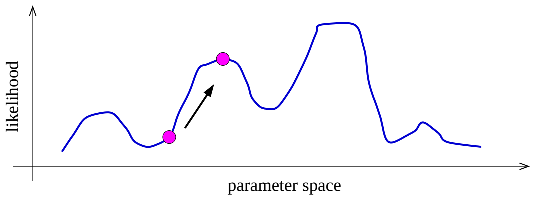
- Bugs again: we just let the bugs run forward from time 0 to t and backward from time τ to t.
- In fact, we can just do one forward pass to compute all the $\alpha_i(t)$ and one backward pass to compute all the $\beta_i(t)$ and then compute any $\gamma_i(t)$ we want. Total cost is $O(M^2T)$.

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 an "average" 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 probability 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. Like K-means instead of mixtures of Gaussians.

Baum-Welch Training

1. Intuition: if only we *knew* the true state path then ML parameter estimation would be trivial.
2. But: can *estimate* state path using the DP trick.
3. *Baum-Welch algorithm* (special case of EM): estimate the states, then compute params, then re-estimate states, and so on ...
4. This works and we can *prove* that it always improves likelihood.
5. However: finding the ML parameters is NP hard, so initial conditions matter a lot and convergence is hard to tell.



Two-frame inference

- Need the cross-time statistics for adjacent time steps:

$$\xi_{ij} = p(x_t = i, x_{t+1} = j | \{\mathbf{y}\})$$

- This can be done by rewriting:

$$\begin{aligned} p(x_t, x_{t+1}|\{\mathbf{y}\}) &= p(x_t, x_{t+1}, \{\mathbf{y}\})/p(\{\mathbf{y}\}) \\ &= p(x_t, \mathbf{y}_1^t) p(x_{t+1}, \mathbf{y}_{t+1}^\tau | x_t, \mathbf{y}_1^t)/L \\ &= p(x_t, \mathbf{y}_1^t) p(x_{t+1}|x_t) p(\mathbf{y}_{t+1}|x_{t+1}) p(y_{t+2}^\tau|x_{t+1})/L \\ &= \alpha_i(t) T_{ij} \mathbf{A}_j(\mathbf{y}_{t+1}) \beta_j(t+1)/L \\ &= \xi_{ij} \end{aligned}$$

- This is the expected number of transitions from state i to state j that begin at time t, given the observations.
- It can be computed with the same α and β recursions.

Parameter Estimation

- Complete log likelihood:

$$\begin{aligned} \log p(x, y) &= \log\{\pi_{x_1} \prod_{t=1}^{\tau-1} T_{x_t, x_{t+1}} \prod_{t=1}^{\tau} A_{x_t}(\mathbf{y}_t)\} \\ &= \log\{\prod_i \pi_i^{[x_1^i]} \prod_{t=1}^{\tau-1} \prod_j T_{ij}^{[x_t^i, x_{t+1}^j]} \prod_{t=1}^{\tau} \prod_k A_k(\mathbf{y}_t)^{[x_t^k]}\} \\ &= \sum_i [x_1^i] \log \pi_i + \sum_{t=1}^{\tau-1} \sum_j [x_t^i, x_{t+1}^j] \log T_{ij} + \sum_{t=1}^{\tau} \sum_k [x_t^k] \log A_k(\mathbf{y}_t) \end{aligned}$$

  where the indicator $[x_t^i] = 1$ if $x_t = i$ and 0 otherwise
- Statistics we need from the E-step are: $p(x_t|\{\mathbf{y}\})$ and $p(x_t, x_{t+1}|\{\mathbf{y}\})$.
- We saw how to get single time marginals $p(x_t|\{\mathbf{y}\})$, but what about two-frame estimates $p(x_t, x_{t+1}|\{\mathbf{y}\})$?

New Parameters are just Ratios of Frequency Counts

- Initial state distribution: expected #times in state i at time 1:

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

- Expected #transitions from state i to j which begin at time t:

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

  so the estimated transition probabilities are:

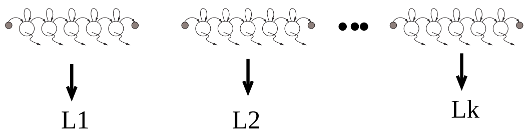
$$\hat{T}_{ij} = \sum_{t=1}^{\tau-1} \xi_{ij}(t) \Bigg/ \sum_{t=1}^{\tau-1} \gamma_i(t)$$

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

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

Using HMMs for Recognition

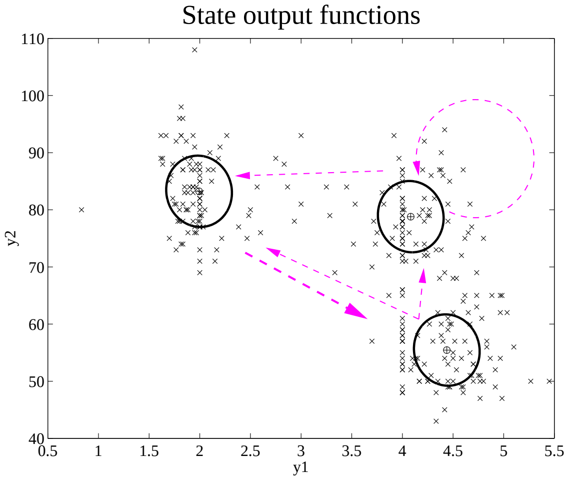
- Use many HMMs for recognition by:
  1. training one HMM for each class (requires *labelled* training data)
  2. evaluating probability of an unknown sequence under each HMM
  3. classifying unknown sequence: HMM with highest likelihood



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

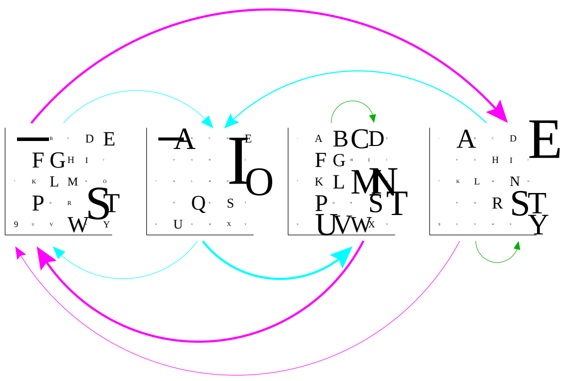
Mixture HMM Example

- Geyser data (continuous outputs)



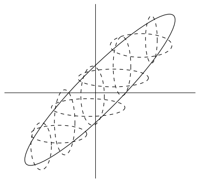
Symbol HMM Example

- Character sequences (discrete outputs)



Regularizing HMMs

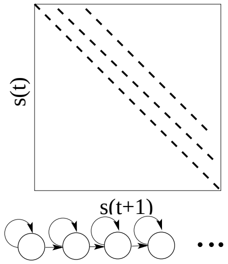
- Two problems:
  - for high dimensional outputs, lots of parameters in each $\mathbf{A}_j(\mathbf{y})$
  - with many states, transition matrix has many^2 elements
- First problem: full covariance matrices in high dimensions or discrete symbol models with many symbols have *lots* of parameters. To estimate these accurately requires a lot of training data. Instead, we often use mixtures of diagonal covariance Gaussians.



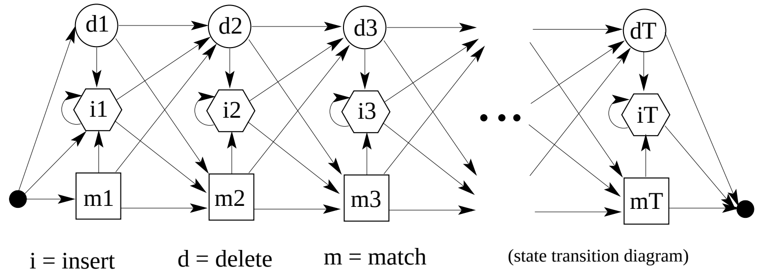
- For discrete data, we can use mixtures of base rates.
- We can also tie parameters across states.

Regularizing Transition Matrices

- One way to regularize large transition matrices is to *constrain* them to be relatively *sparse*: instead of being allowed to transition to *any* other state, each state has only a few possible successor states.
- For example if each state has at most p possible next states then the cost of inference is $O(pKT)$ and the number of parameters is $O(pK + KM)$ which are both *linear* in the number of states.
- An extremely effective way to constrain the transitions is to *order* the states in the HMM and allow transitions only to *states that come later in the ordering*. Such models are known as "linear HMMs", "chain HMMs" or "left-to-right HMMs". Transition matrix is upper-diagonal (usually only has a few bands).

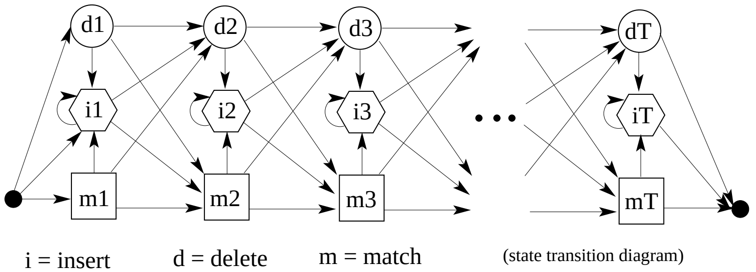


String-Edit HMMs



- number of states = 3(length_template)
- Only insert and match states can generate output symbols.
- Why linear? Because once you visit or skip a match state you can never return to it.
- At most 3 destination states from any state.
- State variables and observations no longer in sync.
  (e.g. y1:s1=m1 ; y2:s2=d2 ; y3:s2=i2 ; y4:s2=m2 ; . . .)

String-Edit HMMs

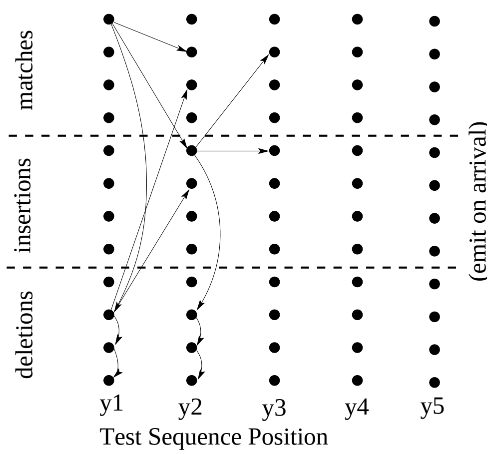


- Another linear architecture is the string-edit HMM.
- Used for probabilistically matching an observed input string to a stored template pattern with possible insertions and deletions.
- Three kinds of states:
  m_j: use position j in the template to match an observed symbol
  i_j: insert extra symbol(s) observations after template position j
  d_j: delete (skip) template position j

String-Edit HMMs

- How do we fill in the costs for a DP grid using a string-edit HMM?
- Almost the same as normal except:
  - Now the grid is 3 times its normal height.
  - It is possible to move down without moving right if you move into a deletion state.

eg: template length=4, test sequence length=5

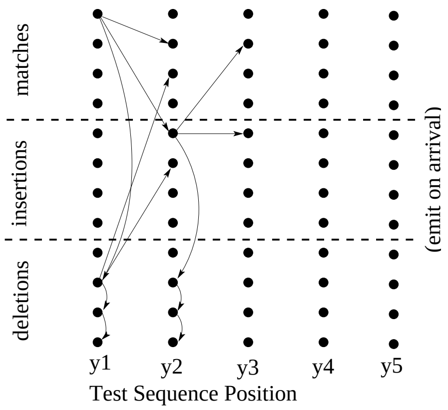


String-Edit HMM Grid Costs

$C_{x_t \to x_{t+1}} = -\log T_{x_t,x_{t+1}} - \log A_{x_{t+1}}(\mathbf{y}_t)$ if x_{t+1} is match or insert

$C_{x_t \to x_{t+1}} = -\log T_{x_t,x_{t+1}}$ if x_{t+1} is a delete state

State $x_t \in \{m_j, i_j, d_j\}$ has nonzero transition probabilities only to states $x_{t+1} \in \{m_{j+1}, i_j, d_{j+1}\}$.



HMM Practicalities

- If you just implement things as I have described them, *they will not work at all*. Why? Remember `logsum`...
- Numerical scaling: the probability values that the bugs carry get tiny for big times and so can easily underflow. Good rescaling trick:

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

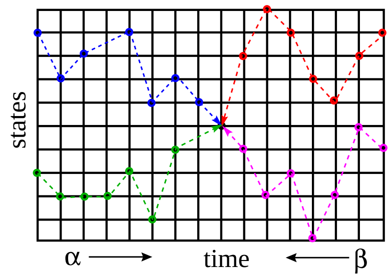
- Multiple observation sequences: can be dealt with by averaging numerators and averaging denominators in the ratios given above.
- Initialization: mixtures of base rates or mixtures of Gaussians
- Generation of new sequences. Just roll the dice!
- Sampling a single state sequence from the posterior $p(\{x\}|\{\mathbf{y}\})$. Harder...but possible. (can you think of how?)

Applications of HMMs

- Speech recognition.
- Language modeling.
- Information retrieval.
- Motion video analysis/tracking.
- Protein sequence and genetic sequence alignment and analysis.
- Financial time series prediction.
- ...

Computational Costs in HMMs

- The number of parameters in the model was $O(K^2 + KM)$ for M output symbols or dimensions.
- Recall the forward-backward algorithm for inference of state probabilities $p(x_t|\{\mathbf{y}\})$.
- The storage cost of this procedure was $O(KT + K^2)$ for K states and a sequence of length T.
- The time complexity was $O(K^2T)$.



Some HMM History

- Markov ('13) and later Shannon ('48,'51) studied *Markov chains*.
- Baum et. al (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.

HMM Pseudocode

- Forward-backward including scaling tricks

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

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

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

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

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

$$\xi = 0$$

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

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

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

Example Problem on HMM Inference

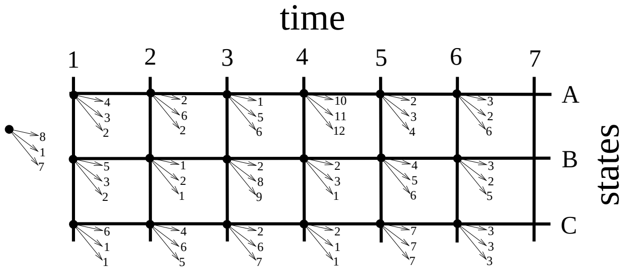
Consider an HMM with 3 states and a sequence of length 7. As you know, HMM inference can be converted into path problems on discrete grids and thus made isomorphic to dynamic programming. The rows of the grid correspond to states and the columns to time steps. Any state sequence is a path through the grid that starts at the leftmost column and ends in the rightmost column, going through exactly one node per column. The cost of moving from one node to another corresponds to *negative log probability*. Such a grid is shown below, with all the costs filled in.

(a) Write an expression for the cost of moving from node $(i, t-1)$ to node (j, t) in terms of the probabilities T_{ij} of transitioning from state i to state j and the probabilities $A_j(y_t)$ of emitting observation y_t from state j $(t > 1)$.

(b) Write an expression for the cost of starting at any node $(k, 1)$ in the leftmost column in terms of the probability of the initial distribution π_k amd the probability of emitting the first symbol y_1 from state k.

(c) Find the log probability (negative of the total cost) of the sequence in the grid below.

(d) Find the most likely hidden state sequence (smallest cost) and its cost given the grid below. Use {A,B,C} to denote the states.



HMM Pseudocode

- 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

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

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

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