Lecture 17:

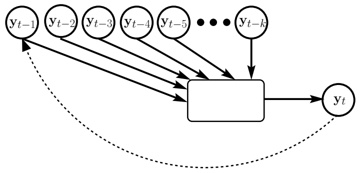
Markov and Hidden Markov Models

Sam Roweis

March 8, 2004

Markov Models

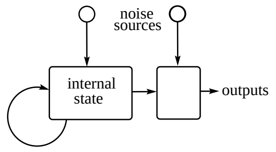
- Use past as state. Next output depends on previous output(s):
$$\mathbf{y}_t = f[\mathbf{y}_{t-1}, \mathbf{y}_{t-2}, \ldots]$$
order is number of previous outputs



- 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

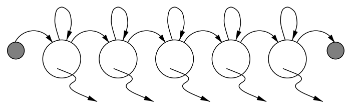
Learning Markov Models

- The ML parameter estimates for a simple Markov model are easy:
$$p(\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_T) = p(\mathbf{y}_1 \ldots \mathbf{y}_k) \prod_{t=k+1}^{T} p(\mathbf{y}_t | \mathbf{y}_{t-1}, \mathbf{y}_{t-2}, \ldots, \mathbf{y}_{t-k})$$
$$\log p(\{\mathbf{y}\}) = \log p(\mathbf{y}_1 \ldots \mathbf{y}_k) + \sum_{t=k+1}^{T} \log p(\mathbf{y}_t | \mathbf{y}_{t-1}, \mathbf{y}_{t-2}, \ldots, \mathbf{y}_{t-k})$$
- 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.

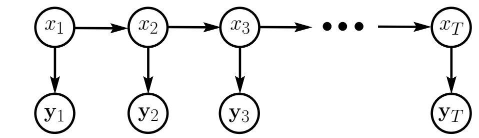
- HMM $\equiv$ " probabilistic function of a Markov chain":
  1. 1st-order Markov chain generates hidden state sequence (path):
  $$\mathsf{P}(x_{t+1} = j | x_t = i) = S_{ij} \qquad \mathsf{P}(x_1 = j) = \pi_j$$
  2. A set of output probability distributions $A_j(\cdot)$ (one per state) converts state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



(state transition diagram)

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

HMM Graphical Model



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

$$\mathsf{P}(\{x\},\{\mathbf{y}\}) = \prod_{t=1}^{T} \mathsf{P}(x_t|x_{t-1})\mathsf{P}(\mathbf{y}_t|x_t)$$
$$= \pi_{x_1} \prod_{t=1}^{T-1} S_{x_t,x_{t+1}} \prod_{t=1}^{T} A_{x_t}(\mathbf{y}_t)$$

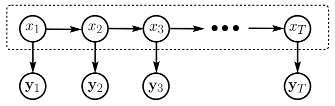
- NB: Data are *not* i.i.d.
  There is no easy way to use plates to show this model. (Why?)

Applications of HMMs

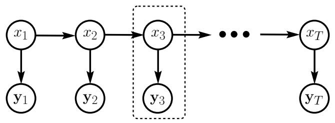
- Speech recognition.
- Language modeling.
- Information retrieval.
- Motion video analysis/tracking.
- Protein sequence and genetic sequence alignment and analysis.
- Financial time series prediction.
- . . .

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 = #states^T). But joint probability factorizes:

$$\mathsf{P}(\{\mathbf{y}\}) = \sum_{x_1}\sum_{x_2}\cdots\sum_{x_T}\prod_{t=1}^{T} \mathsf{P}(x_t|x_{t-1})\mathsf{P}(\mathbf{y}_t|x_t)$$

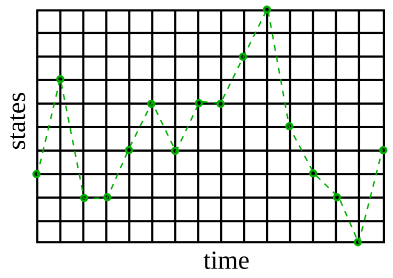
$$= \sum_{x_1}\mathsf{P}(x_1)\mathsf{P}(\mathbf{y}_1|x_1)\sum_{x_2}\mathsf{P}(x_2|x_1)\mathsf{P}(\mathbf{y}_2|x_2)\cdots\sum_{x_T}\mathsf{P}(x_T|x_{T-1})\mathsf{P}(\mathbf{y}_T|x_T)$$

- By moving the summations inside, we can save a lot of work.

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 T
4. sum up values on all bugs



The forward (α) recursion

- We want to compute:

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

- There is a clever "forward recursion" to compute the sum efficiently.

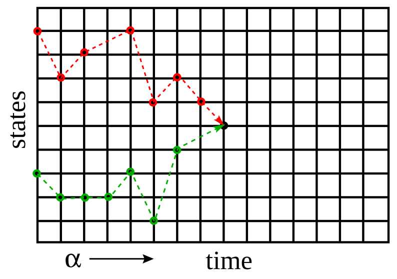
$$\alpha_j(t) = \mathsf{P}(\ \mathbf{y}_1^t\ ,\ x_t = j\ )$$
$$\alpha_j(1) = \pi_j A_j(\mathbf{y}_1)$$
$$\alpha_k(t+1) = \{\sum_j \alpha_j(t) S_{jk}\} A_k(\mathbf{y}_{t+1})$$

- 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:

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

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 trick is called *dynamic programming*, and can be used whenever we have a summation, search, or maximization problem that can be set up as a grid in this way. The axes of the grid don't have to be "time" and "states".

Inference of Hidden States

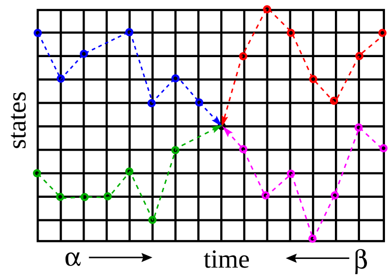
- What if we we want to estimate the hidden states given observations? To start with, let us estimate a single hidden state:

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

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

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 T 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 $O(K^2T)$.

Forward-Backward Algorithm

- We compute these quantites 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:

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

  where we defined:

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

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

$$\beta_j(t) = \sum_i S_{ji}\beta_i(t+1)A_i(\mathbf{y}_{t+1})$$
$$\beta_j(T) = 1$$

- $\alpha_i(t)$ 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:

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

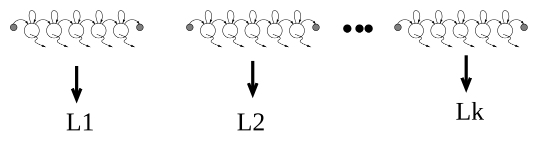
- In the forward calculation we proposed originally, we did this at the final timestep $t = T$:

$$L = \sum_{x_T} \alpha(x_T)$$

  because $\beta_T = 1$.

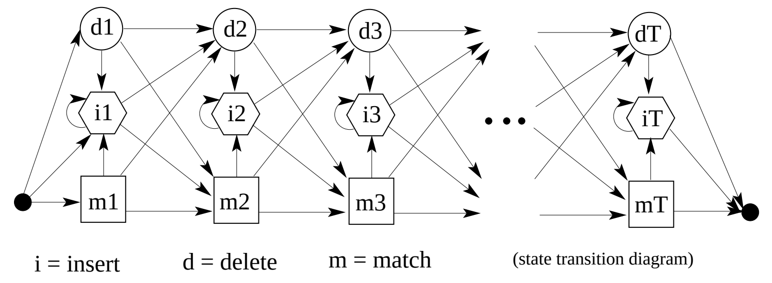
Using HMMs for Recognition

- 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 using Baum-Welch (EM).)

Profile (String-Edit) HMMs



- A "profile HMM" or "string-edit" HMM is used for probabilistically matching an observed input string to a stored template pattern with possible insertions and deletions.
- Three kinds of states: match, insert, delete.
  m_n – use position n in the template to match an observed symbol
  i_n – insert extra symbol(s) observations after template position n
  d_n – delete (skip) template position n

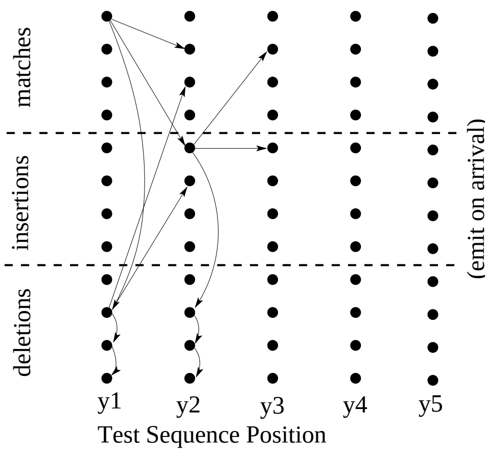
Viterbi Decoding

- 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.

DP for Profile 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

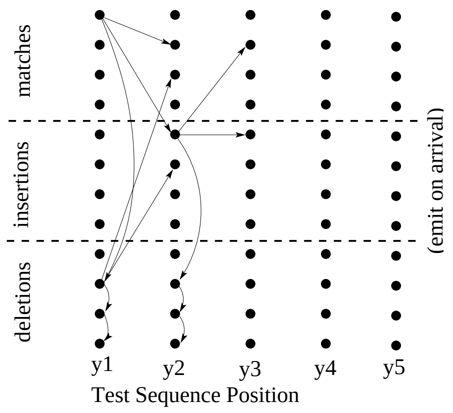


String-Edit HMM Grid Costs

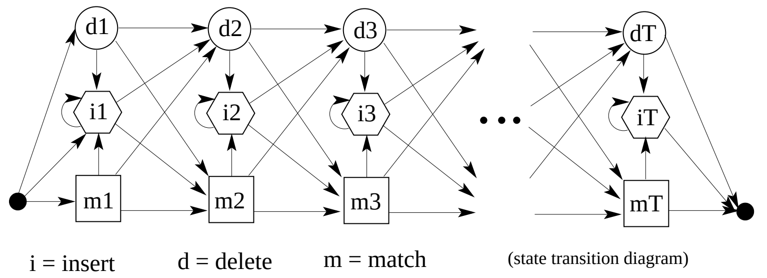
$C_{x \to x'} = -\log T_{x,x'} - \log A_{x'}(\mathbf{y}_t)$ if x' is match or insert

$C_{x \to x'} = -\log T_{x,x'}$ if x' is a delete state

State $x \in \{m_n, i_n, d_n\}$ has nonzero transition probabilities only to states $x' \in \{m_{n+1}, i_n, d_{n+1}\}$.



Profile HMMs have Linear Costs



- number of states = 3(length_template)
- Only insert and match states can generate output symbols.
- Once you visit or skip a match state you can never return to it.
- At most 3 destination states from any state, so S_{ij} very sparse.
- Storage/Time cost *linear* in #states, not quadratic.
- State variables and observations no longer in sync. (e.g. y1:m1 ; d2 ; y2:i2 ; y3:i2 ; y4:m3 ; ...)

Forward-Backward for Profile HMMs

- The equations for the delete states in profile HMMs need to be modified slightly, since they don't emit any symbols.
- For delete states k, the forward equations become:
$$\alpha_k(t) = \sum_j \alpha_j(t) S_{jk}$$
which should be evaluated after the insert and match state updates.
- For all states, the backward equations become:
$$\beta_k(t) = \sum_{i \in \text{match,ins}} S_{ki}\beta_i(t+1)A_i(\mathbf{y}_{t+1}) + \sum_{j \in \text{del}} S_{kj}\beta_j(t)$$
which should be evaluated first for delete states k; then for the rest.
- The gamma equations remain the same:
$$\gamma_i(t) = p(x_t = i \mid \mathbf{y}_1^T) = \alpha_i(t)\beta_i(t)/L$$
- Notice that each summation above contains only three terms, regardless of the number of states!

Initializing Forward-Backward for Profile HMMs

- The initialization equations for Profile HMMs also need to be fixed up, to reflect the fact that the model can only begin in states m_1, i_1, d_1 and can only finish in states m_N, i_N, d_N.
- In particular, $\pi_j = 0$ if j is not one of m_1, i_1, d_1.
- When initializing $\alpha_k(1)$, delete states k have zeros, and all other states have the product of the transition probabilities through only delete states up to them, plus the final emission probability.
- When initializing $\beta_k(T)$, the same kind of adjustment must be made.

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.

Example Problem on HMM Inference

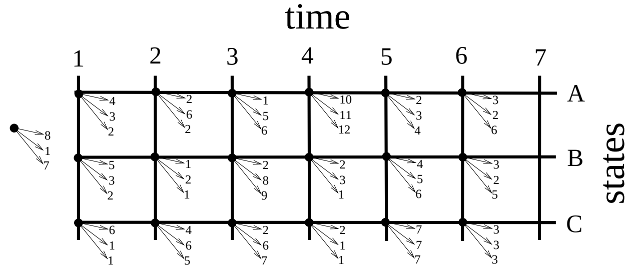
Consider an HMM with 3 states and a sequence of length 7. As you know, HMM inference can be converted into path problems on discrete grids and thus made isomorphic to dynamic programming. The rows of the grid correspond to states and the columns to time steps. Any state sequence is a path through the 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 S_{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.


time
1
2
3
4
5
6
7
A
B
C
states
8
1
7
4
3
2
2
6
2
1
5
6
10
11
12
2
3
4
3
2
6
5
3
2
1
2
1
2
8
9
2
3
1
4
5
6
3
2
5
6
1
1
4
6
5
2
6
7
2
1
1
7
7
7
3
3
3