

# Lecture 9

## Intro to Hidden Markov Models

### Big questions from last time

- Evaluation
  - How likely is a sequence given a model?
  - More formally, given a model  $M$  and a sequence  $s$ , find  $\Pr(x | M)$ .
- Decoding (or inference)
  - Given a sequence and a model, try and figure out which states were visited.
  - More formally, given a model  $M$  and an observation sequence  $s$ , find a state sequence  $t$  such that  $\Pr(s, t | M)$  is maximal.

### Observed sequence, hidden path and Viterbi path

```

Rolls  315116246446644245311321631164152133625144543631656626566666
Die    FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL
Viterbi FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL

Rolls  651166453132651245636664631636663162326455236266666625151631
Die    LLLLLLFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLLLFFL
Viterbi LLLLLLFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL

Rolls  222555441666566563564324364131513465146353411126414626253356
Die    FFFFFFFFFLLLLLLLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL
Viterbi FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL

Rolls  36616366646623253441366166116325256246225526525226643535336
Die    LLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL
Viterbi LLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL

Rolls  233121625364414432335163243633665562466662632666612355245242
Die    FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL
Viterbi FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL
    
```

**Figure 3.5** The numbers show 300 rolls of a die as described in the example. Below is shown which die was actually used for that roll (F for fair and L for loaded). Under that the prediction by the Viterbi algorithm is shown.

From Durbin

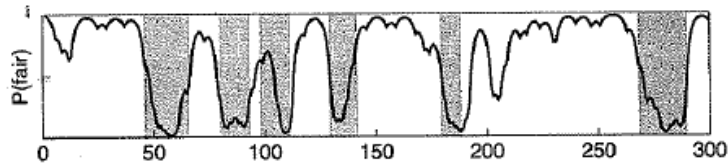
### Review of last lecture

**Algorithm: Forward algorithm**  
 Initialisation ( $i = 0$ ):  $f_0(0) = 1, f_k(0) = 0$  for  $k > 0$ .  
 Recursion ( $i = 1 \dots L$ ):  $f_i(i) = e_i(x_i) \sum_k f_k(i-1) a_{ki}$ .  
 Termination:  $P(x) = \sum_k f_k(L) a_{k0}$ .

**Algorithm: Viterbi**  
 Initialisation ( $i = 0$ ):  $v_0(0) = 1, v_k(0) = 0$  for  $k > 0$ .  
 Recursion ( $i = 1 \dots L$ ):  $v_i(i) = e_i(x_i) \max_k (v_k(i-1) a_{ki})$ ;  
 $\text{ptr}_i(i) = \text{argmax}_k (v_k(i-1) a_{ki})$ .  
 Termination:  $P(x, \pi^*) = \max_k (v_k(L) a_{k0})$ ;  
 $\pi_L^* = \text{argmax}_k (v_k(L) a_{k0})$ .  
 Traceback ( $i = L \dots 1$ ):  $\pi_{i-1}^* = \text{ptr}_i(\pi_i^*)$ .

The structure of the Forward algorithm is essentially the same as that of the Viterbi algorithm, except that a maximization operation is replaced by summation.

## The occasionally dishonest casino: Posterior decoding



**Figure 3.6** The posterior probability of being in the state corresponding to the fair die in the casino example. The  $x$  axis shows the number of the roll. The shaded areas show when the roll was generated by the loaded die.

## Backward algorithm to compute posterior state probabilities

$$\begin{aligned} P(x, \pi_i = k) &= P(x_1 \dots x_i, \pi_i = k) P(x_{i+1} \dots x_L | x_1 \dots x_i, \pi_i = k) \\ &= P(x_1 \dots x_i, \pi_i = k) P(x_{i+1} \dots x_L | \pi_i = k), \end{aligned}$$

### Algorithm: Backward algorithm

Initialisation ( $i = L$ ):  $b_k(L) = a_{k0}$  for all  $k$ .

Recursion ( $i = L - 1, \dots, 1$ ):  $b_k(i) = \sum_l a_{kl} e_l(x_{i+1}) b_l(i + 1)$ .

Termination:  $P(x) = \sum_l a_{0l} e_l(x_1) b_l(1)$ .

$$b_k(i) = P(x_{i+1} \dots x_L | \pi_i = k).$$

$$P(\pi_i = k | x) = \frac{f_k(i) b_k(i)}{P(x)},$$

## Class today

- Although these three algorithms have separate tasks, they also have use when we would like to train a model.
- Training a model is where HMMs are most powerful; they can be used to find genes, recognize speech, and detect other patterns.

## Parameter estimation for HMMs

- We generally need to estimate transition and emission probabilities  $a_{ij}$  and  $e_k(b)$ .
- We have in hand a set of training examples, that correspond to output from the HMM.
- Two potential strategies:
  - Estimation when state sequence is known
  - Estimation when paths are unknown

# Learning

- If state path is known and there are no hidden states, this is easy and involves:
  - Counting how often each parameter is used
  - Normalizing to get probabilities
  - Then treating it just like Markov chain models
- Harder without knowing state paths
  - Idea: estimate counts by considering every path weighted by its probability

## Estimation when state sequence is known

- Easier than estimation when paths unknown
- Maximum likelihood estimators are:

$$a_{kl} = \frac{A_{kl}}{\sum_{l'} A_{kl'}} \quad e_k(b) = \frac{E_k(b)}{\sum_{b'} E_k(b')}$$

- $A_{kl}$  = number of transitions  $k$  to  $l$  in training data +  $r_{kl}$
- $E_k(b)$  = number of emissions of  $b$  from  $k$  in training data +  $r_k(b)$

## Potential problems

- Maximum likelihood estimators are prone to overfitting
  - For example, states never encountered
- For this reason, we introduce  $r_{kl}$  and  $r_{k(b)}$ , which reflect prior biases
- Can be interpreted as parameters of a Bayesian Dirichlet prior.

## Estimation when paths are unknown

- More complex than when paths are known
- Because we can't use maximum likelihood estimators, we will use an iterative algorithm
  - Baum-Welch

## Baum-Welch Algorithm

- *Aka* the Forward-Backward algorithm
- Also an example of an expectation maximization (EM) algorithm
- Idea: hidden state path is the best that explains a training sequence

## Overview

- We will estimate  $A_k$  and  $E_k(b)$  by considering probable paths for training sequences using current values of  $A_k$  and  $E_k(b)$
- This will converge to a local maximum
- Unfortunately, there are many such local maximum
  - Dealing with these is outside the scope of the course

## Overview

- More formally, Baum-Welch calculates  $A_{kl}$  and  $E_k(b)$  as the expected number of times each transition or emission is used.
- This will use the same Forward and Backward probabilities as posterior decoding.

Baum-Welch algorithm: formulae and computational recipe

$$A_{kl} = \sum_j \frac{1}{P(x^j)} \sum_i f_k^j(i) a_{kl} e_l(x_{i+1}^j) b_l^j(i+1), \quad E_k(b) = \sum_j \frac{1}{P(x^j)} \sum_{\{i|x_i^j=b\}} f_k^j(i) b_k^j(i),$$

**Algorithm: Baum-Welch**

Initialisation: Pick arbitrary model parameters.

Recurrence:

Set all the  $A$  and  $E$  variables to their pseudocount values  $r$  (or to zero).

For each sequence  $j = 1 \dots n$ :

Calculate  $f_k(i)$  for sequence  $j$  using the forward algorithm (p. 58).

Calculate  $b_k(i)$  for sequence  $j$  using the backward algorithm (p. 59).

Add the contribution of sequence  $j$  to  $A$  (3.20) and  $E$  (3.21).

Calculate the new model parameters using (3.18).

Calculate the new log likelihood of the model.

Termination:

Stop if the change in log likelihood is less than some predefined threshold or the maximum number of iterations is exceeded.

## Drawbacks

- ML estimators
  - Vulnerable to overfitting if not enough data
  - Estimations can be undefined if never used in training set (so use of pseudocounts)
- Baum-Welch
  - Many local maximums instead of global maximum can be found, depending on starting values of parameters
  - This problem will be worse for large HMMs

## Example from Durbin

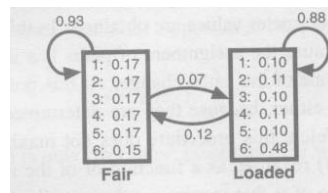
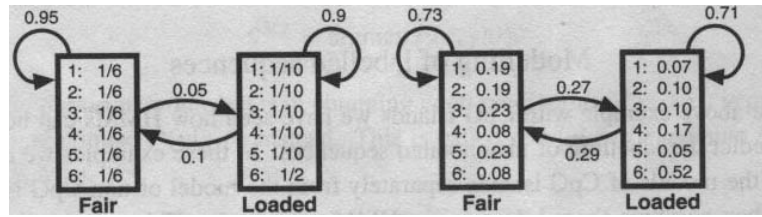
1: 1/6  
2: 1/6  
3: 1/6  
4: 1/6  
5: 1/6  
6: 1/6

1: 1/10  
2: 1/10  
3: 1/10  
4: 1/10  
5: 1/10  
6: 1/2

1: 0.19  
2: 0.19  
3: 0.23  
4: 0.08  
5: 0.23  
6: 0.06

1: 0.07  
2: 0.10  
3: 0.10  
4: 0.17  
5: 0.05  
6: 0.52

Note transition probabilities are different from real ones  
Partly a result of local minima, but its never possible to  
Estimate parameters exactly



## Other methods

- Durbin also discusses an alternative method called Viterbi training based on the Viterbi algorithm.
- Does not maximize the true likelihood as a function of model parameters, but rather finds the model from the most probable paths.
- For this reason it generally does worse than Baum-Welch, but it is widely used.

## Topology

- Some characteristics: number of nodes, alphabet, subset of edges
- We want to exploit domain knowledge
  - Limits number of states / edges
  - Still expressive enough to model relationships
- Length distributions (Durbin 3.4)

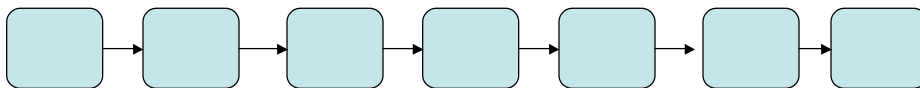
## Silent states

- States that do not emit symbols

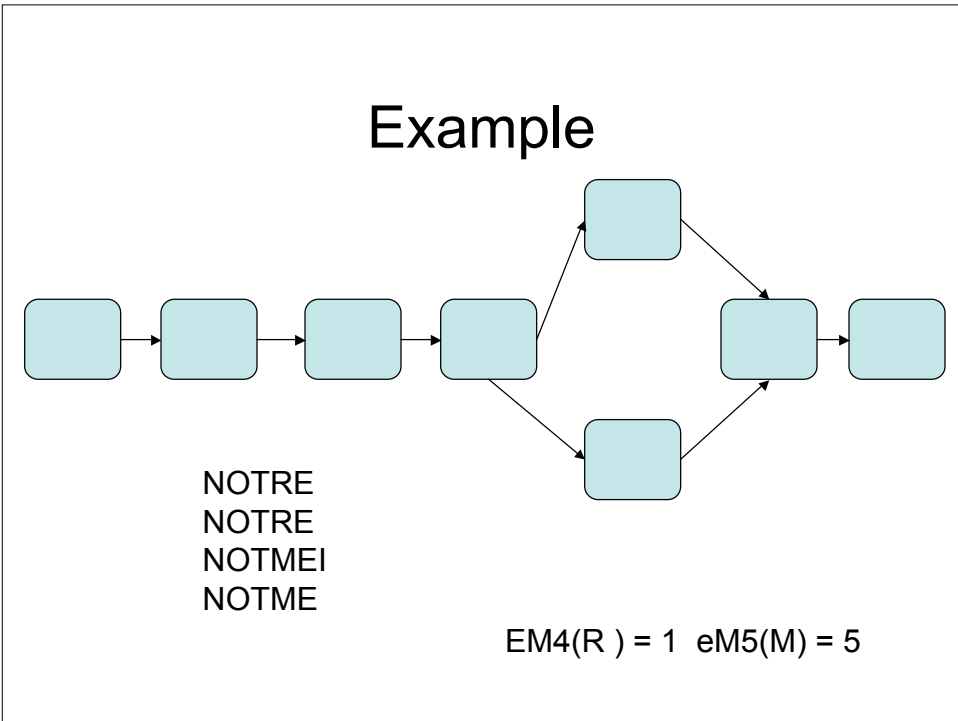
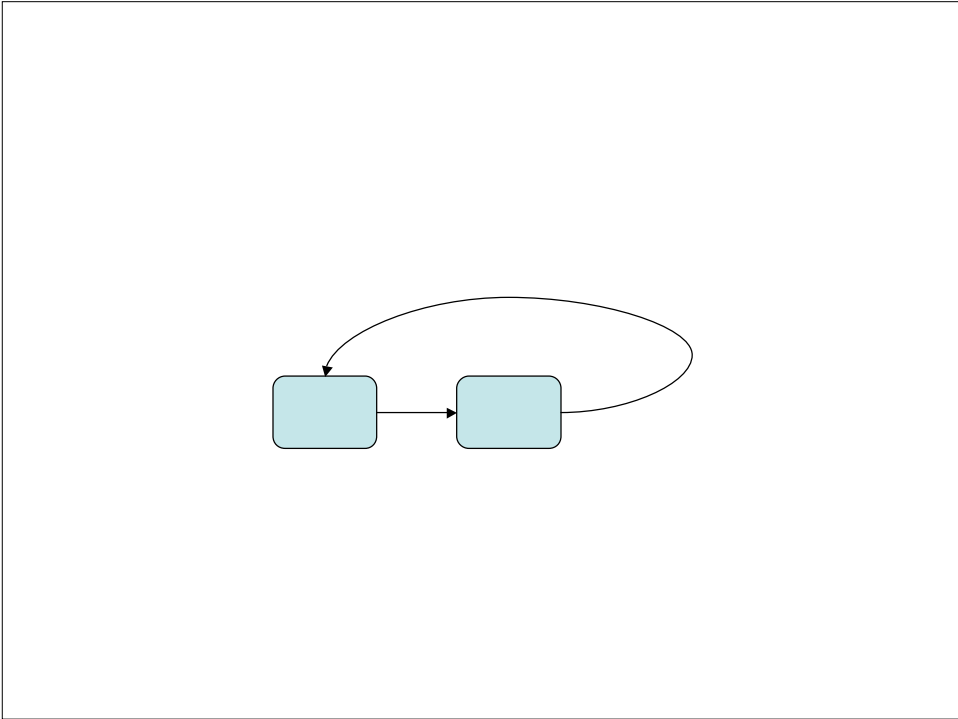


- Also in other places in HMM

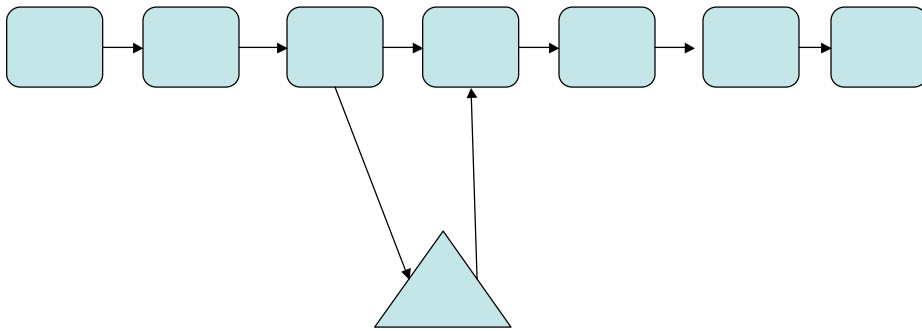
## Example



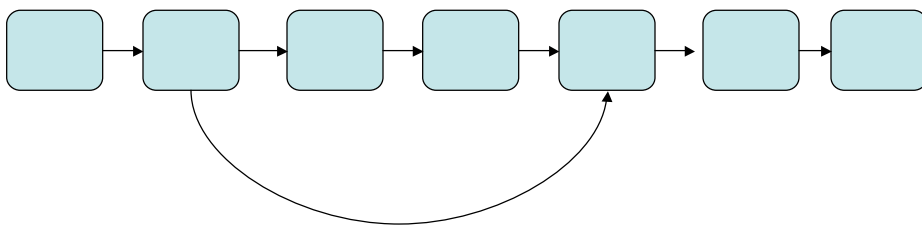
NOTRE  
NOTRE  
NOTRI  
NOTAM



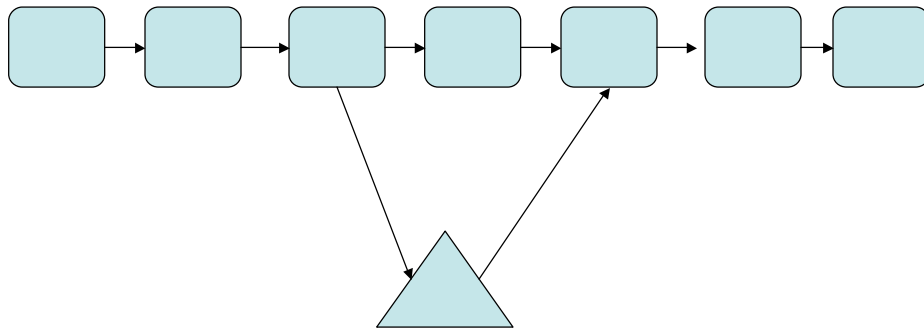
## Insertions



## Deletions



## Deletions



## Affine Gap Penalty

- Gotoh's single affine gap penalty scheme can be represented by pair HMMs
- Pair HMMs simultaneously output two sequences

## Example

- Suppose we have 2 sequences:
  - TCTGGATTC
  - TCTTTCG
- A pair HMM will have these states:
  - MMMIIIMDM
- And will emit:
  - TCTGGATTC-
  - TCT---TTCG

## Model

