# CS711008Z Algorithm Design and Analysis

## Lecture 6. Hidden Markov model and Viterbi's decoding algorithm

Dongbo Bu

Institute of Computing Technology
Chinese Academy of Sciences, Beijing, China

# Outline

- The occasionally dishonest casino: an example of HMM
- Formal definition of HMM
- Finding the most probable state path: Viterbi algorithm

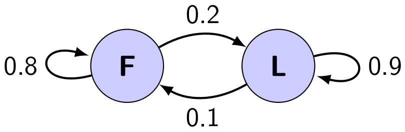
## The occasionally dishonest casino: an example of HMM

# The occasionally dishonest casino

- A casino have a **fair dice** and a **loaded dice**. The fair dice has identical probability $\frac{1}{6}$ for all numbers one to six while the loaded dice has probability $0.3$ of a five, $0.3$ of a six, and $0.1$ for the numbers one to four.
- For the first roll, the casino uses the fair dice with probability $\frac{3}{5}$ and uses the loaded one with probability $\frac{2}{5}$. In the subsequent rolls, the casino switches from a fair to a loaded dice with probability $0.2$ and switches back with probability $0.1$. Thus the switch between dice forms a **Markov process**.

**Fair dice**

| | |
|---|---|
| 1 | 1/6 |
| 2 | 1/6 |
| 3 | 1/6 |
| 4 | 1/6 |
| 5 | 1/6 |
| 6 | 1/6 |



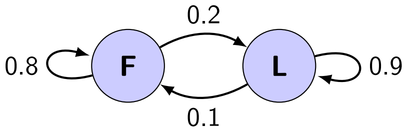

**Loaded dice**

| | |
|---|---|
| 1 | 1/10 |
| 2 | 1/10 |
| 3 | 1/10 |
| 4 | 1/10 |
| 5 | 3/10 |
| 6 | 3/10 |

# The occasionally dishonest casino cont'd

**Fair dice**

| | |
|---|---|
| 1 | 1/6 |
| 2 | 1/6 |
| 3 | 1/6 |
| 4 | 1/6 |
| 5 | 1/6 |
| 6 | 1/6 |




**Loaded dice**

| | |
|---|---|
| 1 | 1/10 |
| 2 | 1/10 |
| 3 | 1/10 |
| 4 | 1/10 |
| 5 | 3/10 |
| 6 | 3/10 |

- Question: Suppose we observed a total of $10$ rolls with the following outcomes:

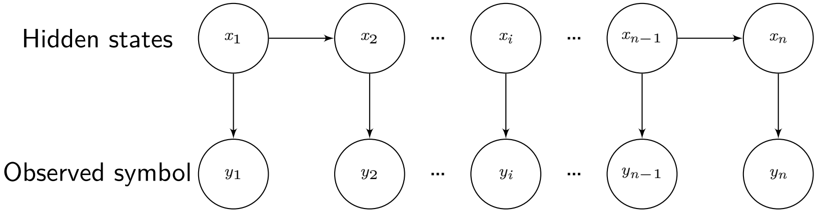
$$Y = (1, 3, 4, 5, 5, 6, 6, 3, 2, 6)$$

Could we find out the most probable state sequence, i.e. the most probable dice used for each roll?

# Trial 1: Calculating log-odd score based on Markov model

- For each observed symbol, we could calculate log-odd score for a window of $w$ rolls around it, and expect the rolls using fair dice to stand out with positive values.
- However, this is unsatisfactory since:
  - This solution depends heavily on the selection of the window size $w$.
  - The rolls generated using fair dice might have sharp boundaries and variable length.
- A better idea is to build a model to describe the switch between these two dice.

# Trial 2: Calculating the most probable state path using HMM




- In each state of the Markov process, the outcome of a roll has different probability. Thus, the whole process forms a **hidden Markov model**. Here the **state sequence**, i.e. the dice used for each roll, is hidden.
- The essential difference between a Markov chain and a hidden Markov model is that for a HMM, there is not a one-to-one correspondence between observed symbols and states.

# Formal definition of HMM

- **Transition probability**: We now distinguish the sequence of states (denoted as $X$) and the sequence of observed symbols (denoted as $Y$). The state sequence follows a simple Markov chain, so the probability of a state $x_i$ depends only on the previous one $x_{i-1}$, which is characterised using transition probability:

$$a_{kl} = P(x_i = l | x_{i-1} = k)$$

- **Begin state**: To model the beginning of the process we introduce a **begin state** (denoted as state $0$). The transition probability $a_{0k}$ represents the probability of starting in state $k$.
- **Emission probability**: A state can generate a symbol from a distribution over all possible symbols; thus, we define **emission probability**:

$$e_k(b) = P(y_i = b | x_i = k)$$

# Using HMM as a generative model

- A symbol sequence can be generated from HMM as follows:
  - Initially a state $x_1$ is chose according the probability $a_{0k}$. In this state $x_i$, a symbol is emitted according to the emission probability $e_{x_i}$.
  - Then a new state $x_2$ is generated according to the transition probability $a_{x_1 k}$ and so on. This way a symbol sequence $Y = (y_1, y_2, ..., y_n)$ is generated. Here we assume $n$ is a fixed number and thus avoid defining an "end state" for simplicity.
- The joint probability of an observed symbol sequence $Y$ and state sequence $X$ is:

$$P(X, Y) = P(x_1 x_2 \ldots x_n, y_1 y_2 \cdots y_n) = \prod_{i=1}^{n} \left( a_{x_{i-1} x_i} e_{x_i}(y_i) \right)$$

# An example

- For example, given an observed outcome of 10 rolls $Y = (1, 3, 4, 5, 5, 6, 6, 3, 2, 6)$, if $X = (\mathtt{F}, \mathtt{F}, \mathtt{F}, \mathtt{F}, \mathtt{F}, \mathtt{L}, \mathtt{L}, \mathtt{L}, \mathtt{L}, \mathtt{L})$, we have:

$$P(X, Y) = \frac{3}{5} \times (\frac{1}{6})^5 \times (0.8)^4 \times 0.2 \times (\frac{3}{10})^3 \times (\frac{1}{10})^2 \times 0.9^4$$

- There are a total of $2^n$ possible state sequence. If we are to choose just one sequence, perhaps the one with the highest joint probability should be chosen,

$$X^* = argmax_X P(X, Y)$$

# Viterbi's decoding algorithm [1967]

- In 1967, Andrew Viterbi proposed a dynamic programming algorithm for decoding over noisy communication links.
- The idea can be extended for decoding in general graphical models, including Bayesian networks, Markov random fields and CRF. The extension is usually termed as **max-sum algorithm**, which aims to finding the most probable latent variables in graphical models. In these models, the **forward-backward algorithm** was generalized to **message passing** or **belief propagation**.
- A faster implementation of Viterbi's algorithm is LazyViterbi (J. Feldman, et al, 2002). The algorithm algorithm was built upon $A^*$ algorithm, and it does not expand any nodes until it really needs to do so.

# Viterbi's decoding algorithm: recursion

- First we rewrite $\max_X P(X, Y)$ as:

$$\max_{x_n} \max_{x_{n-1}} \ldots \max_{x_1} e_{x_n}(y_n) a_{x_{n-1}x_n} e_{x_{n-1}}(y_{n-1}) \ldots a_{x_1x_2} e_{x_1}(y_1) a_{0x_1}$$

- Note that we cannot build a direct recursion between $P(x_1x_2 \ldots x_n, y_1y_2 \cdots y_n)$ and $P(x_2x_3 \ldots x_n, y_2y_3 \cdots y_n)$.
- Let's consider a smaller subproblem: define $v_i(k)$ as

$$\max_{x_{i-1}} \ldots \max_{x_1} e_k(y_i) a_{x_{i-1}k} e_{x_{i-1}}(y_{i-1}) \ldots a_{x_1x_2} e_{x_1}(y_1) a_{0x_1}$$

  We can observe the following recursion:

$$v_i(k) = e_k(y_i) \max_l (a_{lk} v_{i-1}(l))$$

- We also have

$$\max_X P(X, Y) = \max_k v_n(k)$$

# Viterbi's decoding algorithm

ViterbiDecoding($Y, a, e$)

Initialize $v_1(k) = a_{0k}e_k(y_1)$ for all state $k$;<br>
**for** $i = 2$ to $n$ **do**<br>
    **for** each state $k$ **do**<br>
        $v_i(k) = e_k(y_i) \max_l(a_{lk}v_{i-1}(l))$;<br>
        $ptr_i(k) = argmax_l(a_{lk}v_{i-1}(l))$;<br>
    **end for**<br>
**end for**<br>
$P(X^*, Y) = max_k(v_n(k))$;<br>
$x_n^* = argmax_k(v_n(k))$;<br>
**for** $i = n - 1$ to $1$ **do**<br>
    $x_i^* = ptr_{i-1}(x_{i+1}^*)$;<br>
**end for**<br>
**return** $X$;

Time complexity: $O(nK^2)$, where $K$ denotes the number of possible states

# An example

| | $y_i$ | $v_i(\texttt{F})$ | $ptr_i(\texttt{F})$ | $v_i(\texttt{L})$ | $ptr_i(\texttt{L})$ |
|---|---|---|---|---|---|
| $i=1$ | 1 | $1.000*10^{-1}$ | - | $4.000*10^{-2}$ | - |
| $i=2$ | 3 | $1.333*10^{-2}$ | F | $3.600*10^{-3}$ | L |
| $i=3$ | 4 | $1.778*10^{-3}$ | F | $3.240*10^{-4}$ | L |
| $i=4$ | 5 | $2.370*10^{-4}$ | F | $1.067*10^{-4}$ | F |
| $i=5$ | 5 | $3.161*10^{-4}$ | F | $2.880*10^{-5}$ | L |
| $i=6$ | 6 | $4.214*10^{-6}$ | F | $7.776*10^{-6}$ | L |
| $i=7$ | 6 | $5.619*10^{-7}$ | F | $2.100*10^{-6}$ | L |
| $i=8$ | 3 | $7.492*10^{-8}$ | F | $1.890*10^{-7}$ | L |
| $i=9$ | 2 | $9.989*10^{-9}$ | F | $1.701*10^{-8}$ | L |
| $i=10$ | 6 | $1.332*10^{-9}$ | F | $4.592*10^{-9}$ | L |