Chapter 15 (AIAMA)

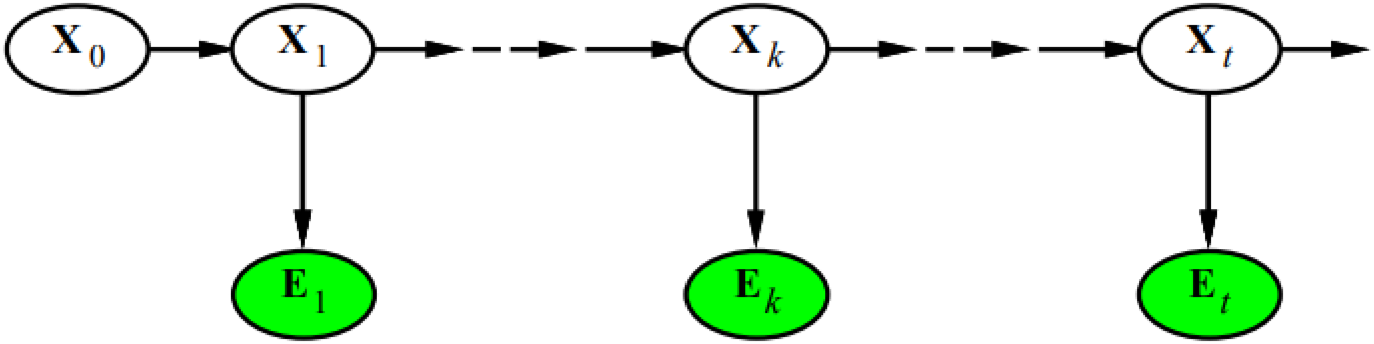
Probabilistic Reasoning Over Time-02

Sukarna Barua

Assistant Professor, CSE, BUET

Smoothing

- Compute the probability distribution over past states.
- Compute where



Smoothing

- Compute where

$$
\begin{aligned}
\mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:t}) &= \mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:k}, \mathbf{e}_{k+1:t}) \\
&= \alpha\, \mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:k}) \mathbf{P}(\mathbf{e}_{k+1:t} \mid \mathbf{X}_k, \mathbf{e}_{1:k}) \quad \text{(using Bayes' rule)} \\
&= \alpha\, \mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:k}) \mathbf{P}(\mathbf{e}_{k+1:t} \mid \mathbf{X}_k) \quad \text{(using conditional independence)}
\end{aligned}
$$

- Remember bayes expansion:
- Here,
- is a fixed term [can be obtained later as probabilities sum to 1]

Smoothing

- Compute where

$$
\begin{aligned}
\mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:t}) &= \mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:k}, \mathbf{e}_{k+1:t}) \\
&= \alpha\, \mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:k}) \mathbf{P}(\mathbf{e}_{k+1:t} \mid \mathbf{X}_k, \mathbf{e}_{1:k}) \quad \text{(using Bayes' rule)} \\
&= \alpha\, \mathbf{P}(\mathbf{X}_k \mid \mathbf{e}_{1:k}) \mathbf{P}(\mathbf{e}_{k+1:t} \mid \mathbf{X}_k) \quad \text{(using conditional independence)}
\end{aligned}
$$

- Hence,
  - Where denotes forward probabilities [*filtering problem*]
  - Note that is a vector [ ]

Smoothing

- Now, how to compute

$$
\begin{aligned}
\mathbf{P}(\mathbf{e}_{k+1:t} \mid \mathbf{X}_k) &= \sum_{\mathbf{x}_{k+1}} \mathbf{P}(\mathbf{e}_{k+1:t} \mid \mathbf{X}_k, \mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1} \mid \mathbf{X}_k) \quad \text{(conditioning on } \mathbf{X}_{k+1}\text{)} \\
&= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1:t} \mid \mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1} \mid \mathbf{X}_k) \quad \text{(by conditional independence)} \\
&= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1}, \mathbf{e}_{k+2:t} \mid \mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1} \mid \mathbf{X}_k) \\
&= \sum_{\mathbf{x}_{k+1}} P(\mathbf{e}_{k+1} \mid \mathbf{x}_{k+1})P(\mathbf{e}_{k+2:t} \mid \mathbf{x}_{k+1})\mathbf{P}(\mathbf{x}_{k+1} \mid \mathbf{X}_k)\,, \qquad (15.9)
\end{aligned}
$$

- and comes from the models [transition and sensor]
- can be obtained recursively or iteratively [*using dynamic programming*]

Smoothing

- **Recurrence relation for**
- Let [*is a vector/array of probabilities*]
- From our previous derivation:

)

[assume an specific output value at time step]

Smoothing

- **Recurrence relation for**

  -)

- What is the base condition?

  - To find the base condition, put [for the last/current state]

  - for all [Why?]

    - Probability of occurring an empty sequence is 1

Smoothing

- **Recurrence relation for**

)

- is known as backward probabilities

  - The algorithm for computing starts from the -th state [and go backward up to [*Fill DP tables for*
  - The algorithm is known as backward algorithm [*in contrast to forward algorithm*]

Smoothing

- **Finally**

- In vector form:

  [*point-wise multiplication*]

- Again is normalizing constant.

Smoothing: Example

- Compute [Probability of rain at time given umbrella observations at time and
- As per our previous formula:
- First term: [*We already know fro* $\mathbf{P}(R_1 \mid u_1, u_2) = \alpha\, \mathbf{P}(R_1 \mid u_1)\, \mathbf{P}(u_2 \mid R_1)$
- Second term: needs recursive expansion [*previous slide*]

$$\begin{aligned}\mathbf{P}(u_2 \mid R_1) &= \sum_{r_2} P(u_2 \mid r_2) P(\ \mid r_2) \mathbf{P}(r_2 \mid R_1) \\ &= (0.9 \times 1 \times \langle 0.7, 0.3 \rangle) + (0.2 \times 1 \times \langle 0.3, 0.7 \rangle) = \langle 0.69, 0.41 \rangle \ .\end{aligned}$$

Smoothing: Example

- Second term needs recursive expansion [previous slides]

$$\begin{aligned}\mathbf{P}(u_2 \mid R_1) &= \sum_{r_2} P(u_2 \mid r_2) P(\ \mid r_2) \mathbf{P}(r_2 \mid R_1) \\ &= (0.9 \times 1 \times \langle 0.7, 0.3 \rangle) + (0.2 \times 1 \times \langle 0.3, 0.7 \rangle) = \langle 0.69, 0.41 \rangle\ .\end{aligned}$$

- Finally, compute:

$$\mathbf{P}(R_1 \mid u_1, u_2) = \alpha\, \mathbf{P}(R_1 \mid u_1)\, \mathbf{P}(u_2 \mid R_1)$$

$$\mathbf{P}(R_1 \mid u_1, u_2) = \alpha\, \langle 0.818, 0.182 \rangle \times \langle 0.69, 0.41 \rangle \approx\ \langle 0.883, 0.117 \rangle\ .$$

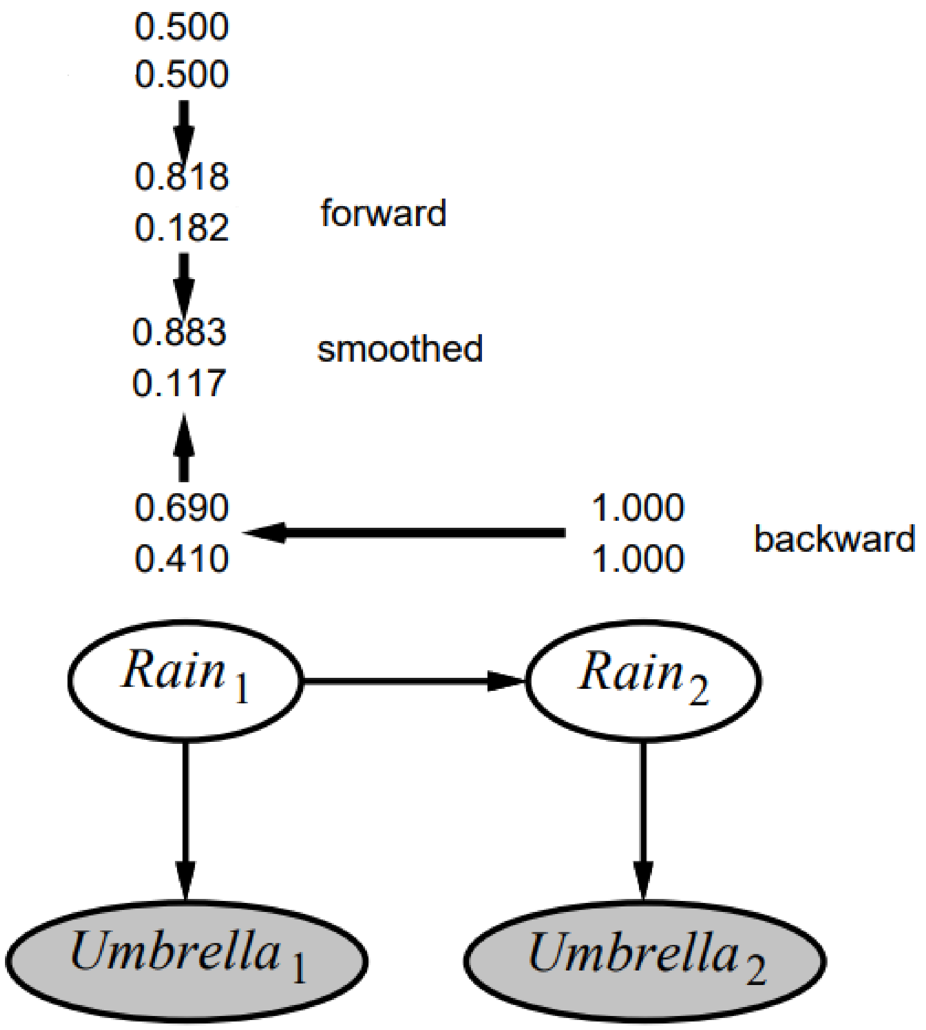
Smoothing: Example

- Smoothed estimate:

$$\mathbf{P}(R_1 \mid u_1, u_2) = \langle 0.883, 0.117 \rangle$$

- Filtered estimate:

$$\mathbf{P}(R_1 \mid u_1) = \langle 0.818, 0.182 \rangle$$



Smoothing: Example

- Smoothed estimate: $\mathbf{P}(R_1 \mid u_1, u_2) = \langle 0.883, 0.117 \rangle$

- Filtered estimate: $\mathbf{P}(R_1 \mid u_1) = \langle 0.818, 0.182 \rangle$

- Observation: Smoothed estimate for rain on day 1 is higher than the filtered estimate (0.818) [Why?]

  - This is because the umbrella on day 2 makes it more likely to have rained on day 2 which in turn, because rain tends to persist, that makes it more likely to have rained on day 1.

Smoothing: Algorithm Pseudocode

- Forward and backward algorithm can be combined to compute posterior probabilities in linear time

function FORWARD-BACKWARD(**ev**, *prior*) **returns** a vector of probability distributions
  inputs: **ev**, a vector of evidence values for steps $1, \ldots, t$
    prior, the prior distribution on the initial state, $\mathbf{P}(\mathbf{X}_0)$
  local variables: **fv**, a vector of forward messages for steps $0, \ldots, t$
    b, a representation of the backward message, initially all 1s
    sv, a vector of smoothed estimates for steps $1, \ldots, t$

  $\mathbf{fv}[0] \leftarrow prior$
  for $i = 1$ **to** t **do**
    $\mathbf{fv}[i] \leftarrow$ FORWARD($\mathbf{fv}[i-1], \mathbf{ev}[i]$)
  for $i = t$ **downto** 1 **do**
    $\mathbf{sv}[i] \leftarrow$ NORMALIZE($\mathbf{fv}[i] \times \mathbf{b}$)
    $\mathbf{b} \leftarrow$ BACKWARD($\mathbf{b}, \mathbf{ev}[i]$)
  return sv

Most Likely Explanation

- Suppose that [true, true, false, true, true] is the umbrella sequence for the security guard's first five days on the job.
- What is the weather (state) sequence most likely to explain this?
- There are possible weather sequences. Is there a way to find the most likely one?

Most Likely Explanation: Naïve Approach

- Compute the probability distribution .

  - [Bayes theorem]

  - Assume for notational convenience

  -) is the probability distribution of states at time step

- After computing the probability distribution)
  - We know the probability of each state sequence
  - We select the most probably state sequence as

Most Likely Explanation: Naïve Approach

- We select the most probably state sequence as:

- What is the problem with the naïve approach?

  - Number of state sequence is exponential
  - If number of states is , then number of state sequences is
  - Computing) for states requires exponential running time
  - Not feasible in real time
  - What can we do?
    - A better approach can be derived using dynamic programming

Most Likely Explanation

- Note that) =) [Why?]
- Consider the probability:
  - is the probability of the most likely state sequence that produce observation sequences and ends at a specific state at time step
  - We will show that instead of maximizing over all sequence of previous states, it is enough to maximize over only previous state [*we will derive a recurrence relation over the previous state*]

Most Likely Explanation

- **Recurrence relation for computing** : we can compute as follows:
  - For all possible states at time step
    - Compute the probability of most likely state sequence that produce observation sequence and ends at (this probability is
    - Move from state to [with transition probability]
    - Produce observation at state [with emission probability assuming]
  - As we don’t know the previous state leading to the most likely sequence, we just take the maximum over all possible previous states :

=

Most Likely Explanation

- Derivation of recurrence relation for :

  =

)

Most Likely Explanation

- Finally obtain the best possible state sequence:
  - is the most likely state sequence ending at at time step
  - We don’t know what is the end state for the most likely state sequence, hence we just take the maximum over all

- This algorithm is known as Viterbi algorithm
- Can be implemented using recursion or dynamic programming approach