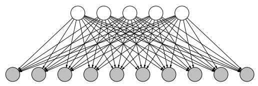
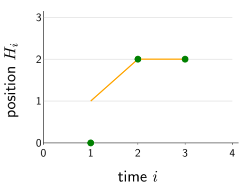
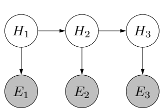
# Bayesian networks: forward-backward



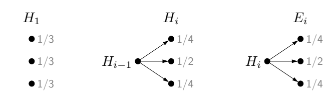
- In this module, I will introduce the forward-backward algorithm for performing efficient and exact inference in Hidden Markov models, an important special case of Bayesian networks.

# Hidden Markov models for object tracking




start / transition / emission




| $h_1$ | $p(h_1)$ |
|---|---|
| 0 | 1/3 |
| 1 | 1/3 |
| 2 | 1/3 |

| $h_i$ | $p(h_i \mid h_{i-1})$ |
|---|---|
| $h_{i-1} - 1$ | 1/4 |
| $h_{i-1}$ | 1/2 |
| $h_{i-1} + 1$ | 1/4 |

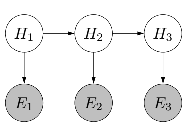
| $e_i$ | $p(e_i \mid h_i)$ |
|---|---|
| $h_i - 1$ | 1/4 |
| $h_i$ | 1/2 |
| $h_i + 1$ | 1/4 |

$$\mathbb{P}(H = h, E = e) = \underbrace{p(h_1)}_{\text{start}} \prod_{i=2}^{n} \underbrace{p(h_i \mid h_{i-1})}_{\text{transition}} \prod_{i=1}^{n} \underbrace{p(e_i \mid h_i)}_{\text{emission}}$$



- Let us revisit our object tracking example, now through the lens of HMMs. Recall that each time $i$, an object is at a location $H_i$, and what we observe is a noisy observation $E_i$. The goal is to infer where the object is / was.
- We define a probabilistic story as follows: An object starts at $H_1$ uniformly drawn over all possible locations.
- Then at each subsequent time step, the object **transitions** from the previous time step, keeping the same location with $1/2$ probability, and moves to an adjacent location each with $1/4$ probability. For example, if $p(h_3 = 3 \mid h_2 = 3) = 1/2$ and $p(h_3 = 2 \mid h_2 = 3) = 1/4$.
- At each time step, we also **emit** a sensor reading $E_i$ given the actual location $H_i$, following the same process as transitions ($1/2$ probability of the same location, $1/4$ probability of an adjacent location).
- Recall that finally, we define a joint distribution over all the actual locations $H_1, \dots, H_n$ and sensor readings $E_1, \dots, E_n$ by taking the product of all the local conditional probabilities.

# Inference questions




Question (**filtering**):

$$\mathbb{P}(H_2 \mid E_1 = 0, E_2 = 2)$$

Question (**smoothing**):

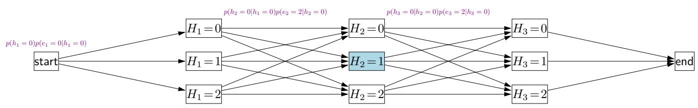
$$\mathbb{P}(H_2 \mid E_1 = 0, E_2 = 2, E_3 = 2)$$

Note: filtering is a special case of smoothing if marginalize unobserved leaves



- In principle, you could ask any type of questions on an HMM, but there are two common ones: filtering and smoothing.
- **Filtering** asks for the distribution of some hidden variable $H_i$ conditioned on only the evidence up until that point. This is useful when you're doing real-time object tracking, and you can't see the future.
- **Smoothing** asks for the distribution of some hidden variable $H_i$ conditioned on all the evidence, including the future. This is useful when you have collected all the data and want to retrospectively go and figure out what $H_i$ was.
- Note that filtering is a special case of smoothing: if we're asking for $H_i$ given $E_1, \dots, E_i$, then we can marginalize everything in the future (since they are just unobserved leaf nodes), reducing the problem to a smaller HMM, where we are smoothing.

# Lattice representation



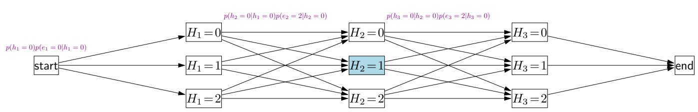

- Edge start $\Rightarrow$ $H_1 = h_1$ has weight $p(h_1)p(e_1 \mid h_1)$
- Edge $H_{i-1} = h_{i-1}$ $\Rightarrow$ $H_i = h_i$ has weight $p(h_i \mid h_{i-1})p(e_i \mid h_i)$
- Each path from start to end is an assignment with weight equal to the product of edge weights

Key: $\mathbb{P}(H_i = h_i \mid E = e)$ is the weighted fraction of paths through $H_i = h_i$



- The forward-backward algorithm is based on a form of dynamic programming.
- To develop this, we consider a **lattice representation** of HMMs. Consider a directed graph (not to be confused with the HMM) with a start node, an end node, and a node for each assignment of a value to a variable $H_i = v$. The nodes are arranged in a lattice, where each column corresponds to one variable $H_i$ and each row corresponds to a particular value $v$. Each path from the start to the end corresponds exactly to a complete assignment to the nodes.
- Each edge has a weight (a single number) determined by the local conditional probabilities (more generally, the factors in a factor graph). For each edge into $H_i = h_i$, we multiply by the transition probability into $h_i$ and emission probability $p(e_i \mid h_i)$.
- This defines a weight for each path (assignment) in the graph equal to the joint probability $P(H = h, E = e)$.
- Note that the lattice contains $O(n|\text{Domain}|)$ nodes and $O(n|\text{Domain}|^2)$ edges, where $n$ is the number of variables and $|\text{Domain}|$ is the number of values in the domain of each variable (3 in our example).
- Now comes the key point. Recall we want to compute a smoothing question $\mathbb{P}(H_i = h_i \mid E = e)$. This quantity is simply the weighted fraction of paths that pass through $H_i = h_i$. This is just a way of visualizing the definition of the smoothing question.
- There are an exponential number of paths, so it's intractable to enumerate all of them. But we can use dynamic programming...

# Forward and backward messages




Forward: $F_i(h_i) = \sum_{h_{i-1}} F_{i-1}(h_{i-1})$Weight($H_{i-1} = h_{i-1}$, $H_i = h_i$)

sum of weights of paths from start to $H_i = h_i$

Backward: $B_i(h_i) = \sum_{h_{i+1}} B_{i+1}(h_{i+1})$Weight($H_i = h_i$, $H_{i+1} = h_{i+1}$)

sum of weights of paths from $H_i = h_i$ to end

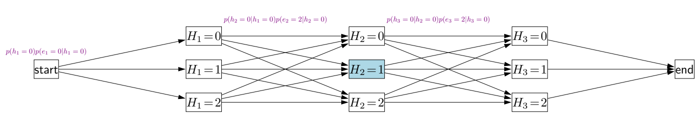
Define $S_i(h_i) = F_i(h_i)B_i(h_i)$:

sum of weights of paths from start to end through $H_i = h_i$



- First, define the forward message $F_i(v)$ to be the sum of the weights over all paths from the start node to $H_i = v$. This can be defined recursively: any path that goes $H_i = h_i$ will have to go through some $H_{i-1} = h_{i-1}$, so we can sum over all possible values of $h_{i-1}$.
- Analogously, let the backward message $B_i(v)$ be the sum of the weights over all paths from $H_i = v$ to the end node.
- Finally, define $S_i(v)$ to be the sum of the weights over all paths from the start node to the end node that pass through the intermediate node $X_i = v$. This quantity is just the product of the weights of paths going into $H_i = h_i$ ($F_i(h_i)$) and those leaving it ($B_i(h_i)$).
- This is analogous to factoring: $(a + b)(c + d) = ab + ad + bc + bd$.
- Note: $F_1(h_1) = p(h_1)p(e_1 = 0 \mid h_1)$ and $B_n(h_n) = 1$ are base cases, which don't require the recurrence.

# Putting everything together




$$\mathbb{P}(H_i = h_i \mid E = e) = \frac{S_i(h_i)}{\sum_v S_i(v)}$$

**Algorithm: forward-backward algorithm**

Compute $F_1, F_2, \dots, F_n$
Compute $B_n, B_{n-1}, \dots, B_1$
Compute $S_i$ for each $i$ and normalize

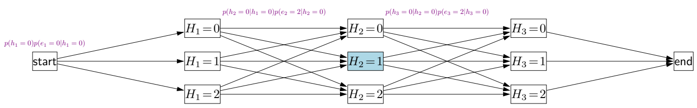
Running time: $O(n|\text{Domain}|^2)$

[demo]



- Now the smoothing question $\mathbb{P}(H_i = h_i \mid E = e)$ is just equal to the normalized version of $S_i$.
- The algorithm is thus as follows: for each node $H_i = h_i$, we compute three numbers: $F_i(h_i), B_i(h_i), S_i(h_i)$. First, we sweep forward to compute all the $F_i$'s recursively. At the same time, we sweep backward to compute all the $B_i$'s recursively. Then we compute $S_i$ by pointwise multiplication.
- The running time of the algorithm is $O(n|\text{Domain}|^2)$, which is the number of edges in the lattice.
- In the demo, we are running the variable elimination algorithm, which is a generalization of the forward-backward algorithm for arbitrary Markov networks. As you step through the algorithm, you can see that the algorithm first computes a forward message $F_2$ and then a backward message $B_2$, and then it multiplies everything together and normalizes to produce $\mathbb{P}(H_2 \mid E_1 = 0, E_2 = 2, E_3 = 2)$. The names and details don't match up exactly, so you don't need to look too closely.
- Implementation note: we technically can normalize $S_i$ to get $\mathbb{P}(H_i \mid E = e)$ at the very end but it's useful to normalize $F_i$ and $B_i$ at each step to avoid underflow. In addition, normalization of the forward messages yields $\mathbb{P}(H_i = v \mid E_1 = e_1, \dots, E_i = e_i)$ which are exactly the filtering queries!

# Summary




- Lattice representation: paths are assignments
- Dynamic programming: compute sums efficiently
- Forward-backward algorithm: compute all smoothing questions, share intermediate computations



- In summary, we have presented the forward-backward algorithm for probabilistic inference in HMMs, in particular smoothing queries.
- The algorithm is based on the lattice representation in which each path is an assignment, and the weight of path is the joint probability.
- Smoothing is just then asking for the weighted fraction of paths that pass through a given node.
- Dynamic programming can be used to compute this quantity efficiently.
- This is formalized using the forward-backward algorithm, which consists of two sets of recurrences.
- Note that the forward-backward algorithm gives you the answer to all the smoothing questions ($\mathbb{P}(H_i = h_i \mid E = e)$ for all $i$), because the intermediate computations are all shared.