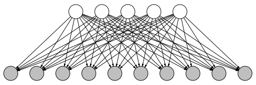
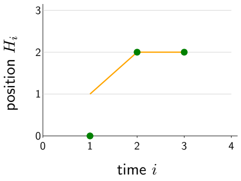
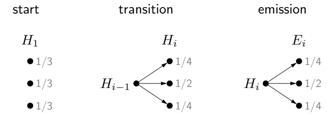
Bayesian networks: particle filtering



- In this module, I will present the particle filtering algorithm for performing approximate inference in Hidden Markov models which is useful when the size of the domain of the variables is large.

Review: Hidden Markov models for object tracking

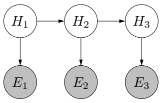




h_1	$p(h_1)$
0	1/3
1	1/3
2	1/3

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

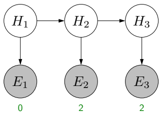
e_i	$p(e_i \mid h_i)$
h_i-1	1/4
h_i	1/2
h_i+1	1/4



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

- Recall that HMM for object tracking.
- Each each point in time, an object has an position H_i, which gives rise to a sensor reading E_i. We start with H_1 uniform over positions, transition from H_{i-1} to H_i with $1/2$ probability on the same location and $1/4$ probability on an adjacent location. We emit the sensor reading analogously. Multiply everything together to form the joint distribution over locations $H_1, \dots, H_n$ and sensor readings $E_1, \dots, E_n$.

Review: inference in Hidden Markov models



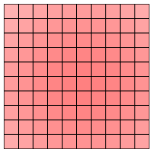
Filtering questions:

$\mathbb{P}(H_1 \mid E_1 = 0)$

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

$\mathbb{P}(H_3 \mid E_1 = 0, E_2 = 2, E_3 = 2)$

Problem: many possible location values for H_i

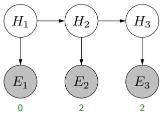


Forward-backward is too slow ($O(n|\text{Domain}|^2)$)...

- Recall that the two common types of inference questions we ask on HMMs are filtering and smoothing.
- Particle filtering, as the name might suggest, performs filtering, so let us focus on that. Filtering asks for the probability distribution over object location H_i at a current time step i given the past observations $E_1 = e_1, \dots, E_i = e_i$.
- Last time, we saw that the forward-backward algorithm could already solve this. But it runs in $O(n|\text{Domain}|^2)$, where |Domain| is the number of possible values (e.g., locations) that H_i can take on. On this example, $H_i \in \{0, 1, 2\}$ but for real applications, there could easily be hundreds of thousands of values, not to mention what happens if H_i is continuous. This could be a very large number, which makes the forward-backward algorithm very slow (even if it's not exponentially so).
- The motivation of particle filtering is to perform **approximate probabilistic inference**, and leverages the fact that most of the locations are very improbable given evidence.
- Particle filtering actually applies to general factor graphs, but we will present them for hidden Markov models for concreteness.

Beam search for HMMs

Idea: keep $\leq K$ partial assignments (**particles**)



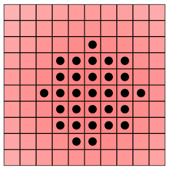
Algorithm: beam search

```
Initialize $C \leftarrow [\{\}]$
For each $i = 1, \dots, n$:
    Extend:
        $C' \leftarrow \{h \cup \{H_i : v\} : h \in C, v \in \text{Domain}_i\}$
    Prune:
        $C \leftarrow K$ particles of $C'$ with highest weights
Normalize weights to get approximate $\hat{\mathbb{P}}(H_1, \dots, H_n \mid E = e)$
Sum probabilities to get any approximate $\hat{\mathbb{P}}(H_i \mid E = e)$
```

[demo: `beamSearch({K:3})`]

- Our starting point for motivating particle filtering is beam search, an algorithm for finding an approximate maximum weight assignment in arbitrary constraint satisfaction problems (CSPs).
- Since HMMs are Bayesian networks, which are Markov networks, which have an underlying factor graph, we can simply apply beam search to HMMs (for now putting aside the goal of finding the maximum weight assignment).
- Recall that beam search maintains a list of candidate partial assignments to the first i variables. There are two phases. In the first phase, we **extend** all the existing candidates C to all possible assignments to H_i; this results in $K = |\text{Domain}|$ candidates C'. We then take the subset of K candidates with the highest weight, where the weight of a partial assignment is simply the product of all the factors (transitions, emissions) that can be computed on the partial assignment.
- In the demo, we start with partial assignments to H_1, whose weights are given by $p(h_1)p(e_1 = 0 \mid h_1)$. In the next step, we can multiply in factors $p(h_2 \mid h_1)p(e_2 = 2 \mid h_2)$, and so on.
- At the very end, we obtain $K = 3$ complete assignments, each with a weight (equal to the joint probability of the assignment and observations). We can normalize these weights to form an approximate distribution over all assignments (conditioned on the observations). From here, we can manually compute any marginal probabilities (e.g., $\mathbb{P}(H_3 = 2 \mid E = e)$) by summing the probabilities of assignments satisfying the given condition (e.g., $H_3 = 2$).

Beam search problems



Algorithm: beam search

```
Initialize $C \leftarrow [\{\}]$
For each $i = 1, \dots, n$:
    Extend:
        $C' \leftarrow \{h \cup \{H_i : v\} : h \in C, v \in \text{Domain}_i\}$
    Prune:
        $C \leftarrow K$ particles of $C'$ with highest weights
```

- Extend: slow because requires considering every possible value for H_i
- Prune: greedily taking best K doesn't provide diversity

Particle filtering solution (3 steps): **propose, weight, resample**

- There are two problems with beam search.
- First, beam search can be slow if Domain is large, since we might have to try every single candidate value h_i to assign H_i. In some cases, we can efficiently generate only the values h_i that have nonzero transition probability ($p(h_i \mid h_{i-1} > 0)$, for example, if we know that h_i must be within a certain distance of h_{i-1} (can't teleport). But if we wanted to track the object to high resolution, there might still be too many values to consider.
- Second, beam search greedily takes the K highest weight candidates at each time step. This could be dangerous, since we might end up with many assignments that are only slightly different, and not truly representative of the actual distribution. You can think of this as a form of overfitting.
- Particle filtering addresses both of these problems. It has three steps: propose, which extends the current partial assignment, and reweight + resample, which redistributes resources on the particles based on evidence.

Step 1: propose

Old particles: $\approx \mathbb{P}(H_1, H_2 \mid E_1 = 0, E_2 = 2)$

$\{H_1 : 0, H_2 : 1\}$
$\{H_1 : 1, H_2 : 2\}$

Key idea: proposal distribution

For each old particle (h_1, h_2), sample $H_3 \sim p(h_3 \mid h_2)$.

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

New particles: $\approx \mathbb{P}(H_1, H_2, H_3 \mid E_1 = 0, E_2 = 2)$

$\{H_1 : 0, H_2 : 1, H_3 : 1\}$
$\{H_1 : 1, H_2 : 2, H_3 : 2\}$

- At each stage of the particle filtering, we can think of our set of particles C as approximating a certain distribution.
- Suppose we have a set of particles that approximates the filtering distribution over H_1, H_2. The first step is to extend each current partial assignment (particle) from $(h_1, \dots, h_{i-1})$ to $(h_1, \dots, h_i)$.
- To do this, we simply go through each particle and sample a new value h_i using the transition probability $p(h_i \mid h_{i-1})$.
- We can think of advancing each particle according to the dynamics of the HMM. These extended particles approximate the probability of H_1, H_2, H_3, but still conditioned on the same evidence.
- In some cases (e.g., the transitions are Gaussian), sampling h_3 is very easy compared to enumerating all possible of h_3. (Indeed, the advantages of particle filtering are clearer in continuous state spaces.).

Step 2: weight

Old particles: $\approx \mathbb{P}(H_1, H_2, H_3 \mid E_1 = 0, E_2 = 1)$

$\{H_1 : 0, H_2 : 1 : H_3 : 1\}$
$\{H_1 : 1, H_2 : 2 : H_3 : 2\}$

Key idea: weighting based on evidence

For each old particle (h_1, h_2, h_3), weight it by $p(e_3 = 2 \mid h_3)$.

h_3	$p(e_3 = 2 \mid h_3)$
0	0
1	1/4
2	1/2

New particles: $\approx \mathbb{P}(H_1, H_2, H_3 \mid E_1 = 0, E_2 = 1, E_3 = 2)$

$\{H_1 : 0, H_2 : 1 : H_3 : 1\}$ (1/4)
$\{H_1 : 1, H_2 : 2 : H_3 : 2\}$ (1/2)

- Having generated a set of K candidates, we need to now take into account the new evidence $E_i = e_i$. This is a deterministic step that simply weights each particle by the probability of generating $E_i = e_i$, which is the emission probability $p(e_i \mid h_i)$.
- Intuitively, the proposal was just a guess about where the object will be H_3, but we need to fact check this guess.
- In this example, we observed $E_3 = 2$, so we need to weight the two particles by $p(e_3 = 2 \mid h_3 = 1) = 1/4$ and $p(e_3 = 2 \mid h_3 = 2) = 1/2$, respectively.

Step 3: resample

Old particles: $\approx \mathbb{P}(H_1, H_2, H_3 \mid E_1 = 0, E_2 = 2, E_3 = 2)$

$\{H_1 : 0, H_2 : 1 : H_3 : 1\}$ (1/4) $\Rightarrow$ 1/3
$\{H_1 : 1, H_2 : 2 : H_3 : 2\}$ (1/2) $\Rightarrow$ 2/3

Key idea: resampling

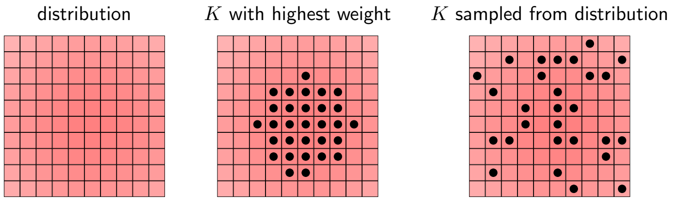
Normalize weights and draw K samples to redistribute particles to more promising areas.

New particles: $\approx \mathbb{P}(H_1, H_2, H_3 \mid E_1 = 0, E_2 = 2, E_3 = 2)$

$\{H_1 : 1, H_2 : 2 : H_3 : 2\}$
$\{H_1 : 1, H_2 : 2 : H_3 : 2\}$

- At this point, we have a set of weighted particles representing the desired filtering distribution.
- However, if some of the weights are small, this could be wasteful. In the extreme case, any particle with zero weight should just be thrown out.
- The K particles can be viewed as our limited resources for representing the distribution, the resampling step attempts to redistribute these precious resources to places in the distribution that are more promising.
- To this end, we will normalize the weights to form a distribution over the particles (similar to what we did at the end of beam search). Then we sample K times from this distribution.
- In this example, we happened to get two occurrences of the second particle, but we might have easily gotten one of each or even two of the first.

Why sampling?


distribution

K with highest weight

K sampled from distribution

not representative

more representative

Sampling is especially important when there is high uncertainty!

- You might wonder why we are resampling, leaving the result of the algorithm up to chance.
- To see why resampling can be more favorable than beam search, consider the setting where we start with a set of particles on the left where the weights are given by the shade of red (darker is more weight). Notice that the weights are all quite similar (i.e., the distribution is close to the uniform distribution).
- Beam search chooses the K locations with the highest weight, which would clump all the particles near the mode. This is risky, because we have no support out farther from the center, where there is actually substantial probability.
- However, if we sample from the distribution which is proportional to the weights, then we can hedge our bets and get a more representative set of particles which cover the space more evenly.
- In cases where the original weights much more skewed towards a few particles, then taking the highest weight particles is fine and perhaps even slightly better than resampling.

Particle filtering

Algorithm: particle filtering

```
Initialize $C \leftarrow [\{\}]$
For each $i = 1, \dots, n$:
    Propose:
        $C' \leftarrow \{h \cup \{H_i : h_i\} : h \in C, h_i \sim p(h_i \mid h_{i-1})\}$
    Weight:
        Compute weights $w(h) = p(e_i \mid h_i)$ for $h \in C'$
    Resample:
        $C \leftarrow K$ particles drawn independently from $\frac{w(h)}{\sum_{h' \in C} w(h')}$
```

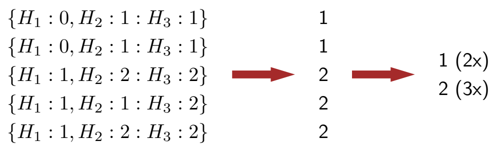
[demo: `particleFiltering({K:100})`]

- We now present the final particle filtering algorithm, which is structurally similar to beam search. We go through all the variables $H_1, \dots, H_n$.
- For each candidate $h \in C$, we propose h_i according to the transition distribution $p(h_i \mid h_{i-1})$.
- We then weight this particle using the emission probability $w(h) = p(e_i \mid h_i)$.
- Finally, we normalize the weights $\{w(h) : h \in C\}$ and sample K particles independently from this distribution.
- In the demo, we can go through the extend (propose) and prune (weight + resample) steps, ending with a final set of full assignments, which can be used to approximate the filtering distribution $\mathbb{P}(H_3 \mid E = e)$.

Particle filtering: implementation

For filtering questions, can optimize:

- Keep only value of last H_i for each particle
- Store count for each unique particle



$\{H_1 : 0, H_2 : 1 : H_3 : 1\}$ 1
$\{H_1 : 0, H_2 : 1 : H_3 : 1\}$ 1
$\{H_1 : 1, H_2 : 2 : H_3 : 2\}$ 2
$\{H_1 : 1, H_2 : 1 : H_3 : 2\}$ 2
$\{H_1 : 1, H_2 : 2 : H_3 : 2\}$ 2

1 (2x)
2 (3x)

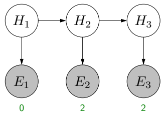
- So far, we have presented a version of particle filtering where each particle at the end is a full assignment to all the variables. This allows us to approximately answer a variety of different questions based on the induced distribution.
- However, if we're only interested in filtering questions, then we can perform two optimizations.
- First, in tracking applications, we only care about the last location H_i, and future steps only depend on the value of H_i. Therefore, we often just store the value of H_i rather than the entire trajectory.
- Second, since we have discrete variables, many particles might have the same value of H_i, so we can just store the counts of each value rather than storing duplicate values.

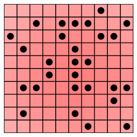
Particle filtering demo

[see web version]

- Now let us visualize particle filtering in a more realistic, interactive object tracking setting.
- Consider an object is moving around in a grid and we are trying to figure out its location $H_i \in \{1, \dots, \text{grid-width}\} \times \{1, \dots, \text{grid-height}\}$.
- The transition distribution places a uniform distribution over moving north, moving south, moving east, moving west, or staying put.
- The emission distribution places a uniform distribution over locations E_i that are within 3 steps (both vertically and horizontally) of the actual position H_i. In the textbox, you can change the emission distribution dynamically (`observeFactor`).
- When you hit ctrl-enter, you can see the noisy sensor readings (visualized as a yellow dot bouncing around).
- If you increase the number of particles, you can see a red cloud representing where the particles are, where the intensity of a square is proportional to the number of particles in that square.
- You can now set `showTruePosition = true` to see the actual H_i that generated E_i. You can see that the cloud is able to track the true location reasonably well, although there are occasional errors.

Summary





$$\mathbb{P}(H_3 \mid E_1 = 0, E_2 = 2, E_3 = 2)$$

- Use particles to represent an approximate distribution

Propose (transitions) **Weight** (emissions) **Resample**

- Can scale to large number of locations (unlike forward-backward)
- Maintains better particle diversity (compared to beam search)

- In summary, we have presented particle filtering, an inference algorithm for HMMs that approximately computes filtering questions of the form: where is the object currently given all the past noisy sensor readings?
- Particle filtering represents distributions over hidden variables with a set of particles. To advance the particles to the next time step, it proposes new positions based on transition probabilities. It then weights these guesses based on evidence from the emission probabilities. Finally, it resamples from the normalized weights to redistribute the precious particle resources.
- Compared to the forward-backward algorithm, both beam search and particle filtering can scale up to a large number of locations (assuming most of them are unlikely). Unlike beam search, however, particle filtering uses randomness to ensure better diversity of the particles.
- Particle filtering is also called sequential Monte Carlo and there are many more sophisticated extensions that I'd encourage to learn about.