Particle Filter
(Ch. 15)

WELL, WHEN WE OBSERVE THEM, THEY BECOME AMBER PARTICLES OF GRAIN.
Hidden Markov Model

To deal with information over time, we used a hidden Markov model:

\[
\begin{align*}
X_0 & \quad \xrightarrow{0.6} X_1 \\
X_1 & \quad \xrightarrow{0.9} X_2 \\
X_2 & \quad \xrightarrow{} X_3 \\
X_3 & \quad \xrightarrow{} X_4 \\
X_4 & \quad \xrightarrow{\ldots} \\
\end{align*}
\]

Often, we want more than a single variable and/or evidence for our problem (also works well for continuous variables)
This more general representation is called a dynamic Bayesian network.
Hidden Markov Model

We could always just cluster all evidence and all non-evidence to fit the HMM.

However, this would lead to an exponential amount of calculations/table size (as the cluster would have to track every combination of variables).

Rather, it is often better to relax our HMM assumptions and expand the network.
Unfortunately, it is harder to compute a “filtered” message in this new network.

We could still follow the same process:

1. Use $t_0$ to compute $t_1$, add evidence at $t_1$
2. Use $t_1$ to compute $t_2$, add evidence at $t_2$
3. (continue)

(Similar to our “forward message” in HMMs)
Dynamic Bayesian Network

The process is actually very similar to variable elimination (with factors)

You have a factor for each variable and combine them to get the next step, then you sum out the previous step

Even with this “efficient” approach, it is still $O(d^{n+k})$, where $d=$ domain size (2 if T/F), $k=$ num parents, $n=$ num var
If our network is large, finding the exact probabilities is infeasible.

Instead, we will use something similar to likelihood weighting called particle filtering.

This will estimate the filtered probability (i.e. $P(x_t|e_{1:t})$) using the previous estimate (i.e. $P(x_{t-1}|e_{1:t-1})$) ... and then repeating.
Particle filtering algorithm:
- Sample to initialize $t=0$ based on $P(x_0)$ with $N$ sample “particles”
- Loop until you reach $t$ you want:
  (1) Sample to apply transition from $t-1$: each particle samples to decide where go
  (2) Weight samples based on evidence: Weight of particle in state $x$ is $P(e|x)$
  (3) Resample $N$ particles based on weights: $P(\text{particle in } x) = \frac{\text{sum } w \text{ in } x}{\text{total sum } w}$
Particle Filtering

Particle filtering algorithm:
- Sample to initialize $t=0$ based on $P(x_0)$ with $N$ sample "particles"
- Loop until you reach $t$ you want:

(a) Propagate
(b) Weight
(c) Resample
Although the algorithm is *supposed* to be run in a more complex network... let's start small

Let's do N=100 particles

First we sample randomly to assign all 100 particles T/F in $X_0$
Particle Filtering

For each particle that is T is $X_0$:
60% chance to be T in $X_1$, 40% F in $X_1$

For each particle that is F is $X_0$:
90% chance to be T in $X_1$, 10% F in $X_1$
Particle Filtering

Then apply evidence weight, since $\neg e_1$:
- $T$ in $X_1$ weighted as 0.7
- $F$ in $X_1$ weighted as 0.2
Resample $X_1$ based on weight & counts:

Total weight = $78 \times 0.7 + 22 \times 0.2 = 59$

Weight in T samples = $78 \times 0.7 = 54.6$

Resample as $T = \frac{54.6}{59} = 92.54\%$

(100 samples still)
Particle Filtering

Start process again... first transition

For each particle:

\( X_1 \) T: 60% chance to be T in \( X_2 \), 40% F in \( X_2 \)

\( X_1 \) F: 90% chance to be T in \( X_2 \), 10% F in \( X_2 \)
Weight evidence (same evidence as last time, so same weight):
T has \( w = P(\neg e_2 | x_2) = 0.7 \)
F has \( w = P(\neg e_2 | \neg x_2) = 0.2 \)
Particle Filtering

Resample:
Total weight = 60*0.7 + 40*0.2 = 50
T weight = 60*0.7 = 42
P(sample T in X₂) = 42/50 = 0.84
Particle Filtering

You do $X_3$!

(Rather than "sampling" just round to nearest if you want to check your work with here)
Particle Filtering

You should get:

(1) 65/35
(2) w for T = 0.3, W for F = 0.8
(3) 41/59
Particle Filtering

Why does it work?

Each step computes the next “forward” message in filtering, so we can use induction

If one forward message is done right, they should all be approximately correct

(Base case is trivial as $P(x_0)$ is directly sampled, so should be approximate correct)
We compute the probabilities as:

\[ P(x_t|e_{1:t}) = \frac{\text{Number of true resamples}}{\text{Total number of samples}} = \frac{N(x_t|e_{1:t})}{N} \]

(above is our inductive hypothesis)

\[ P(x_{t+1}|e_{1:t+1}) = \sum_{x_t} P(x_{t+1}|x_t) \cdot N(x_t|e_{1:t}) \]

\[ \approx P(e_{t+1}|x_{t+1}) \sum_{x_t} P(x_{t+1}|x_t) \cdot (N \cdot P(x_t|e_{1:t})) \]

\[ = N \cdot P(e_{t+1}|x_{t+1}) \sum_{x_t} P(x_{t+1}|x_t) \cdot (P(x_t|e_{1:t})) \]

\[ = \alpha \cdot P(e_{t+1}|x_{t+1}) \sum_{x_t} P(x_{t+1}|x_t) \cdot (P(x_t|e_{1:t})) \]

Step (3) should look a lot like normalize
If we had a dynamic Bayes net (below), what do we need to change about particle filtering?

Real World Complications
If we had a dynamic Bayes net (below), what do we need to change about particle filtering?

For multiple variables, a particle should represent a value in each variable.

So if A, B, C are T/F variables, each color in the DBN represents a single particle (e.g. blue = \{T,T,F\})
Thus, when finding where “blue particle” should go, you probabilistically determine a position for $A_1$, $B_1$ and $C_1$ (again, the particle spans all variables).

The weighting is similar to likelihood, where you just multiply all weights together (e.g. for blue particle this is:

$$P(e_1|\neg c_1) \cdot P(d_1|a_1)$$)
Real World Complications

Biggest real world simplifications?
Real World Complications

Biggest real world simplifications?

The sensors are only considered to be “uncertain”, but quite often they fail

Temporarily failures (i.e. incorrect sensor readings for a few steps) can be handled by ensuring the transition is high enough

Assume 0 is a sensor failure

(i.e. \( P(\text{reading} = 0 \mid \text{reading} = \text{valid}) = 0.01 \))
Real World Complications

This can handle cases where there is a brief moment of failure:
Real World Complications

To handle cases where the sensor completely fails, you can add another variable.

This new variable should have a small chance of going “false” and when false, it will always stay there and give bad readings.

You can then ask the network which variable is more likely to be true, and judge off of that.
Real World Complications

SensOK$_0$ → SensOK$_1$ → SensOK$_2$ → SensOK$_3$

GPS$_1$ → SensOK$_1$

Position$_0$ → Position$_1$ → Position$_2$ → Position$_3$

Speed$_0$ → Speed$_1$ → Speed$_2$ → Speed$_3$

Spin$_1$ → Spin$_2$ → Spin$_3$

SensOK$_0$ → SensOK$_1$ → SensOK$_2$ → SensOK$_3$