CS 343H: Honors Artificial Intelligence

Particle Filters and Applications of HMMs

Prof. Peter Stone — The University of Texas at Austin

[These slides based on those of Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley. All CS188 materials are available at http://ai.berkeley.edu.]
Recap: Reasoning Over Time

- Markov models

\[ P(X_1) \quad P(X|X_{-1}) \]

- Hidden Markov models

\[ P(E|X) \]

<table>
<thead>
<tr>
<th>X</th>
<th>E</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>rain</td>
<td>umbrella</td>
<td>0.9</td>
</tr>
<tr>
<td>rain</td>
<td>no umbrella</td>
<td>0.1</td>
</tr>
<tr>
<td>sun</td>
<td>umbrella</td>
<td>0.2</td>
</tr>
<tr>
<td>sun</td>
<td>no umbrella</td>
<td>0.8</td>
</tr>
</tbody>
</table>
Recap: Filtering

**Elapse time:** compute \( P( X_t | e_{1:t-1} ) \)

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

**Observe:** compute \( P( X_t | e_{1:t} ) \)

\[
P(x_t|e_{1:t}) \propto P(x_t|e_{1:t-1}) \cdot P(e_t|x_t)
\]

**Belief:** \( <P(rain), P(sun)> \)

- \( P(X_1) \) \(<0.5, 0.5> \) **Prior on** \( X_1 \)
- \( P(X_1 | E_1 = umbrella) \) \(<0.82, 0.18> \) **Observe**
- \( P(X_2 | E_1 = umbrella) \) \(<0.63, 0.37> \) **Elapse time**
- \( P(X_2 | E_1 = umbrella, E_2 = umbrella) \) \(<0.88, 0.12> \) **Observe**
Ghostbusters: Time elapse and observation
Particle Filtering
Particle Filtering

- Filtering: approximate solution

- Sometimes $|X|$ is too big to use exact inference
  - $|X|$ may be too big to even store $B(X)$
  - E.g. $X$ is continuous

- Solution: approximate inference
  - Track samples of $X$, not all values
  - Samples are called particles
  - Time per step is linear in the number of samples
  - But: number needed may be large
  - In memory: list of particles, not states

- This is how robot localization works in practice

- Particle is just new name for sample
Our representation of $P(X)$ is now a list of $N$ particles (samples)
- Generally, $N \ll |X|$ (...but not in project 4)
- Storing map from $X$ to counts would defeat the point

$P(x)$ approximated by number of particles with value $x$
- So, many $x$ may have $P(x) = 0!$
- More particles, more accuracy

For now, all particles have a weight of 1

Particle filtering uses three repeated steps:
- Elapse time and observe (similar to exact filtering) and resample
Particle Filtering: Elapse Time

- Each particle is moved by sampling its next position from the transition model

\[ x' = \text{sample}(P(X'|x)) \]

- Sample frequencies reflect the transition probabilities
- Here, most samples move clockwise, but some move in another direction or stay in place

- This captures the passage of time
  - If enough samples, close to exact values before and after (consistent)
Slightly trickier:

- Don’t sample observation, fix it
- Similar to likelihood weighting, downweight samples based on the evidence

\[ w(x) = P(e|x) \]

\[ B(X) \propto P(e|X)B'(X) \]

As before, the probabilities don’t sum to one, since all have been downweighted (in fact they now sum to (N times) an approximation of P(e))
Particle Filtering: Resample

- Rather than tracking weighted samples, we resample

- N times, we choose from our weighted sample distribution (i.e. draw with replacement)

- This is equivalent to renormalizing the distribution

- Now the update is complete for this time step, continue with the next one
Recap: Particle Filtering

- **Particles:** track samples of states rather than an explicit distribution
Moderate Number of Particles
One Particle
Huge Number of Particles
In robot localization:
- We know the map, but not the robot’s position
- Observations may be vectors of range finder readings
- State space and readings are typically continuous (works basically like a very fine grid) and so we cannot store \( B(X) \)
- Particle filtering is a main technique
Particle Filter Localization (Sonar)

Global localization with sonar sensors
Robot Mapping

- **SLAM: Simultaneous Localization And Mapping**
  - We do not know the map or our location
  - State consists of position AND map!
  - Main techniques: Kalman filtering (Gaussian HMMs) and particle methods

DP-SLAM, Ron Parr
Particle Filter SLAM – Video 1
Dynamic Bayes Nets
Dynamic Bayes Nets (DBNs)

- We want to track multiple variables over time, using multiple sources of evidence.
- Idea: Repeat a fixed Bayes net structure at each time.
- Variables from time $t$ can condition on those from $t-1$.

Dynamic Bayes nets are a generalization of HMMs.
Exact Inference in DBNs

- Variable elimination applies to dynamic Bayes nets
- Procedure: “unroll” the network for T time steps, then eliminate variables until $P(X_t|e_{1:T})$ is computed
- Online belief updates: Eliminate all variables from the previous time step; store factors for current time only
DBN Particle Filters

- A particle is a complete sample for a time step

- **Initialize**: Generate prior samples for the t=1 Bayes net
  - Example particle: $G_1^a = (3,3)$ $G_1^b = (5,3)$

- **Elapse time**: Sample a successor for each particle
  - Example successor: $G_2^a = (2,3)$ $G_2^b = (6,3)$

- **Observe**: Weight each *entire* sample by the likelihood of the evidence conditioned on the sample
  - Likelihood: $P(E_1^a | G_1^a) \times P(E_1^b | G_1^b)$

- **Resample**: Select prior samples (tuples of values) in proportion to their likelihood
Most Likely Explanation
HMMs: MLE Queries

- HMMs defined by
  - States $X$
  - Observations $E$
  - Initial distribution: $P(X_1)$
  - Transitions: $P(X \mid X_{-1})$
  - Emissions: $P(E \mid X)$

- New query: most likely explanation: $\arg \max_{x_{1:t}} P(x_{1:t} \mid e_{1:t})$

- New method: the Viterbi algorithm

- Question: Why not just apply filtering and predict most likely value of each variable separately?
State Trellis

- State trellis: graph of states and transitions over time

- Each arc represents some transition $x_{t-1} \rightarrow x_t$
- Each arc has weight $P(x_t|x_{t-1})P(e_t|x_t)$
- Each path is a sequence of states
- The product of weights on a path is that sequence's probability along with the evidence
- Forward algorithm computes sums of all paths to each node, Viterbi computes best paths
- Exponentially many paths, but dynamic programming can find best path in linear time!
Forward / Viterbi Algorithms

\[
\begin{align*}
    f_t[x_t] &= P(x_t, e_{1:t}) \\
    &= P(e_t|x_t) \sum_{x_{t-1}} P(x_t|x_{t-1}) f_{t-1}[x_{t-1}]
\end{align*}
\]

Viterbi Algorithm (Max)

\[
\begin{align*}
    m_t[x_t] &= \max_{x_{1:t-1}} P(x_{1:t-1}, x_t, e_{1:t}) \\
    &= P(e_t|x_t) \max_{x_{t-1}} P(x_t|x_{t-1}) m_{t-1}[x_{t-1}]
\end{align*}
\]
Speech Recognition
Speech Recognition in Action
Digitizing Speech
- Speech input is an acoustic waveform

Figure: Simon Arnfield, http://www.psyc.leeds.ac.uk/research/cogn/speech/tutorial/
Spectral Analysis

- Frequency gives pitch; amplitude gives volume
  - Sampling at ~8 kHz (phone), ~16 kHz (mic) (kHz=1000 cycles/sec)

- Fourier transform of wave displayed as a spectrogram
  - Darkness indicates energy at each frequency

Human ear figure: depion.blogspot.com
Acoustic Feature Sequence

- Time slices are translated into acoustic feature vectors (~39 real numbers per slice)

- These are the observations $E$, now we need the hidden states $X$
Speech State Space

- **HMM Specification**
  - $P(E|X)$ encodes which acoustic vectors are appropriate for each phoneme (each kind of sound)
  - $P(X|X')$ encodes how sounds can be strung together

- **State Space**
  - We will have one state for each sound in each word
  - Mostly, states advance sound by sound
  - Build a little state graph for each word and chain them together to form the state space $X$
States in a Word

Word Model:

Observation Sequence (spectral feature vectors):
Transitions with a Bigram Model

Training Counts

<table>
<thead>
<tr>
<th>Count</th>
<th>Phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>198015222</td>
<td>the first</td>
</tr>
<tr>
<td>194623024</td>
<td>the same</td>
</tr>
<tr>
<td>168504105</td>
<td>the following</td>
</tr>
<tr>
<td>158562063</td>
<td>the world</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>14112454</td>
<td>the door</td>
</tr>
<tr>
<td>23135851162</td>
<td>the *</td>
</tr>
</tbody>
</table>

\[
\hat{P}(\text{door}|\text{the}) = \frac{14112454}{23135851162} = 0.0006
\]
Decoding

- Finding the words given the acoustics is an HMM inference problem
- Which state sequence $x_{1:T}$ is most likely given the evidence $e_{1:T}$?

$$x_{1:T}^* = \arg \max_{x_{1:T}} P(x_{1:T} | e_{1:T}) = \arg \max_{x_{1:T}} P(x_{1:T}, e_{1:T})$$

- From the sequence $x$, we can simply read off the words
Next Time: Value of Information