Probabilistic Reasoning over Time

AIMA Chapter 15

[Adapted from slides by Dan Klein and Pieter Abbeel at UC Berkeley]

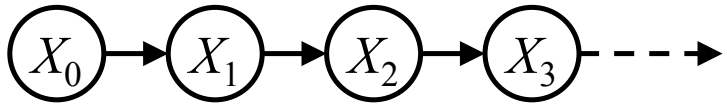
Uncertainty and Time

- Often, we want to reason about a ***sequence*** of observations
  - Speech recognition
  - Robot localization
  - Medical monitoring
  - User attention eg. 浏览网页(个性化推荐)
- Need to introduce time into our models

Markov Models (aka Markov chain/process)

↗ Time / stage discrete 离散

- Value of X at a given time is called the ***state*** (usually discrete, finite)

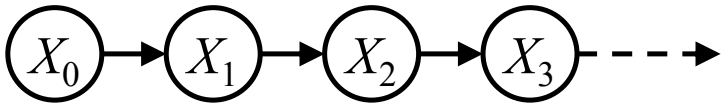


- The ***transition model*** $P(X_t \mid X_{t-1})$ specifies how the state evolves over time
- ***Stationarity*** assumption: same transition probabilities at all time steps

只与前一个时刻的状态有关.

- Joint distribution $P(X_0, \ldots, X_T) = P(X_0) \prod_t P(X_t \mid X_{t-1})$

Quiz: are Markov models a special case of Bayes nets?

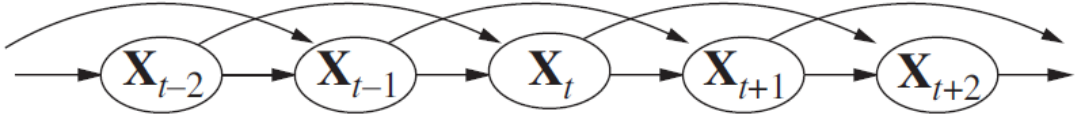


- Yes and no!
- Yes:
  - Directed acyclic graph, joint = product of conditionals
- No:
  - Infinitely many variables (unless we truncate)
  - Repetition of transition model not part of standard Bayes net syntax

Ittrankit 剪枝

Markov Assumption: Conditional Independence

- ***Markov*** assumption: X_{t+1}, … are independent of X_0,…, X_{t-1} given X_t
  - Past and future independent given the present
  - Each time step only depends on the previous
- This is a ***first-order*** Markov model
- A kth-order model allows dependencies on k earlier steps



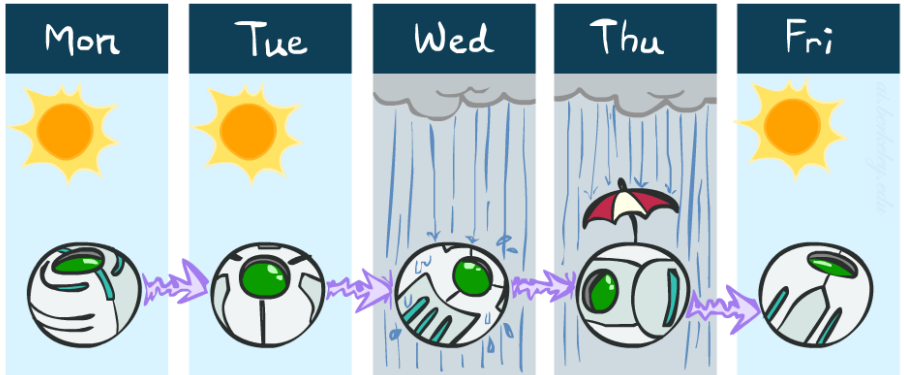
Example: Weather

- States {rain, sun}
- Initial distribution $P(X_0)$

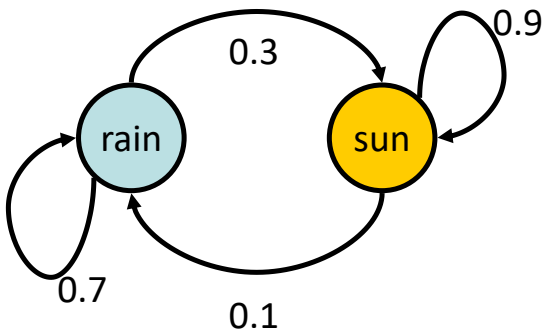
$P(X_0)$	
sun	rain
0.5	0.5

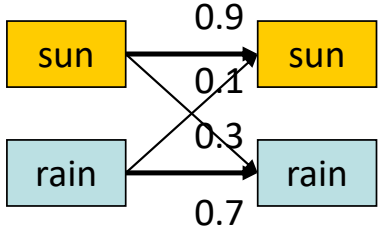
- Transition model $P(X_t \mid X_{t-1})$

X_{t-1}	$P(X_t \mid X_{t-1})$	
	sun	rain
sun	0.9	0.1
rain	0.3	0.7



Two new ways of representing the same CPT

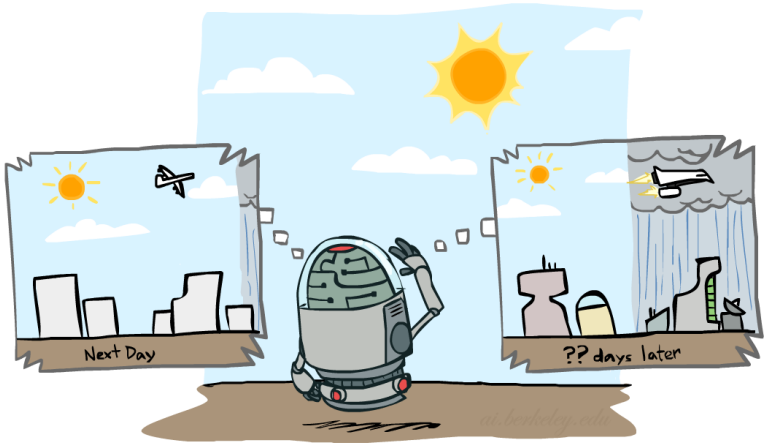




Weather prediction

- Time 0: <0.5,0.5>

X_{t-1}	$P(X_t \mid X_{t-1})$	
	sun	rain
sun	0.9	0.1
rain	0.3	0.7



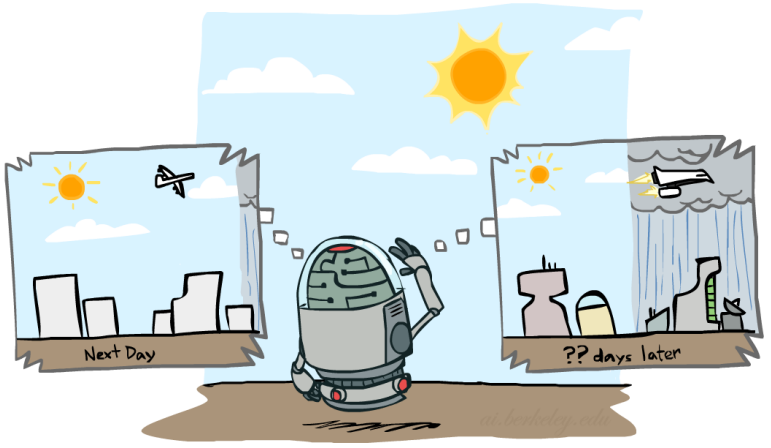
- What is the weather like at time 1?
  - $P(X_1) = \sum_{x_0} P(X_1, X_0=x_0)$
  - $= \sum_{x_0} P(X_0=x_0)\, P(X_1 \mid X_0=x_0)$
  - $= 0.5\langle 0.9,0.1\rangle + 0.5\langle 0.3,0.7\rangle = \langle 0.6,0.4\rangle$

X_1: { sun_t1: $0.5 \times 0.9 + 0.5 \times 0.3 = 0.6$
rain_t1: $0.5 \times 0.1 + 0.5 \times 0.7 = 0.4$

Weather prediction, contd.

- Time 1: <0.6,0.4>

X_{t-1}	$P(X_t \mid X_{t-1})$	
	sun	rain
sun	0.9	0.1
rain	0.3	0.7

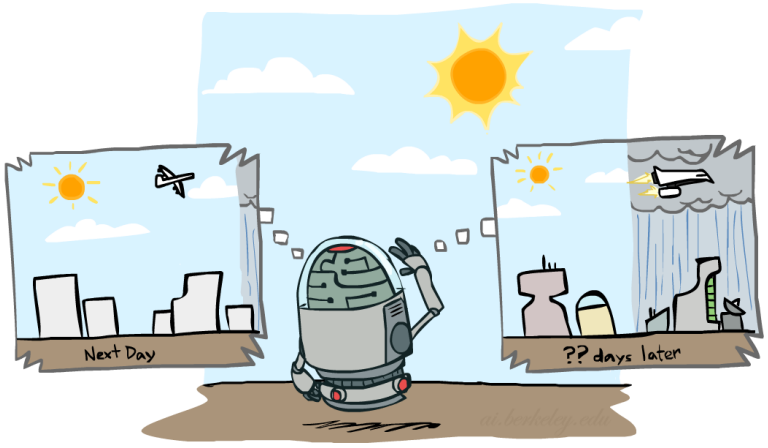


- What is the weather like at time 2?
  - $P(X_2) = \sum_{x_1} P(X_2, X_1=x_1)$
  - $= \sum_{x_1} P(X_1=x_1)\, P(X_2 \mid X_1=x_1)$
  - $= 0.6\langle 0.9,0.1\rangle + 0.4\langle 0.3,0.7\rangle = \langle 0.66,0.34\rangle$

Weather prediction, contd.

- Time 2: <0.66,0.34>

X_{t-1}	$P(X_t \mid X_{t-1})$	
	sun	rain
sun	0.9	0.1
rain	0.3	0.7



- What is the weather like at time 3?
  - $P(X_3) = \sum_{x_2} P(X_3, X_2=x_2)$
  - $= \sum_{x_2} P(X_2=x_2)\, P(X_3 \mid X_2=x_2)$
  - $= 0.66\langle 0.9,0.1\rangle + 0.34\langle 0.3,0.7\rangle = \langle 0.696,0.304\rangle$

Forward algorithm (simple form)

- What is the state at time t (given an initial distribution $P(X_0)$)?
  - $P(X_t) = \sum_{x_{t-1}} P(X_t, X_{t-1}=x_{t-1})$
  - $= \sum_{x_{t-1}} P(X_{t-1}=x_{t-1})\, P(X_t \mid X_{t-1}=x_{t-1})$

Probability from previous iteration

Transition model

- Iterate this update starting at $t=0$

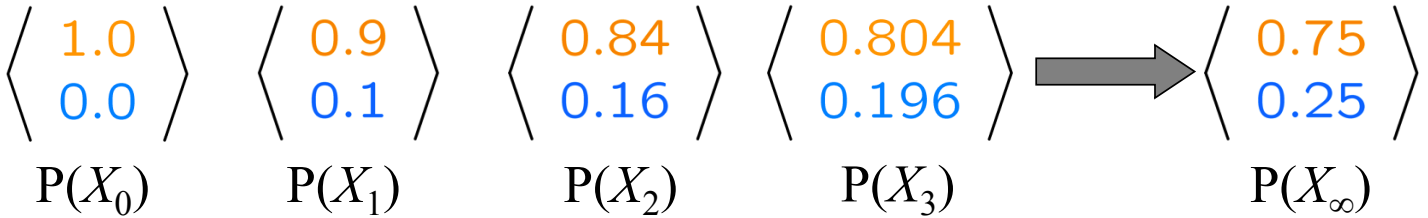
Forward algorithm:

$$P(X_t) = \sum_{x_{t-1}} P(X_t, X_{t-1}=x_{t-1})$$

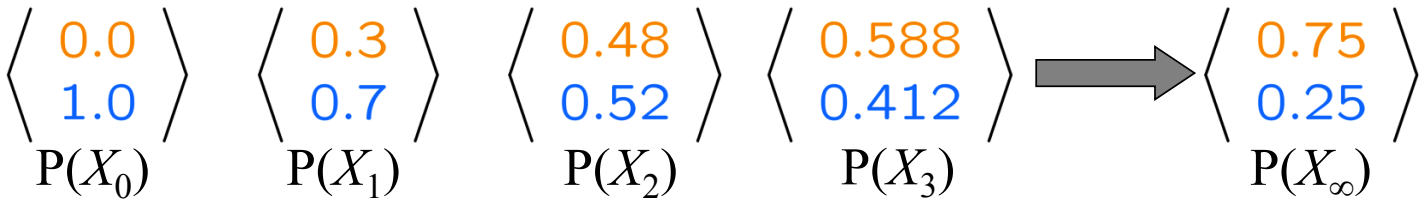
$$= \sum_{x_{t-1}} P(X_{t-1}=x_{t-1})\, P(X_t \mid X_{t-1}=x_{t-1})$$

Example Run of Mini-Forward Algorithm

- From initial observation of sun



- From initial observation of rain



- From yet another initial distribution $P(X_0)$:

X_{t-1}	X_t	$P(X_t \mid X_{t-1})$
sun	sun	0.9
sun	rain	0.1
rain	sun	0.3
rain	rain	0.7

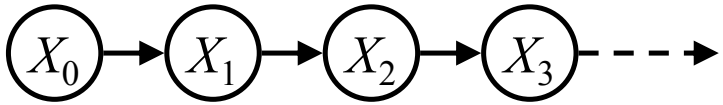
Stationary Distributions

- For most chains:
  - Influence of the initial distribution gets less and less over time.
  - The distribution we end up in is independent of the initial distribution
- Stationary distribution:
  - The distribution we end up with is called the stationary distribution P_∞ of the chain
  - It satisfies

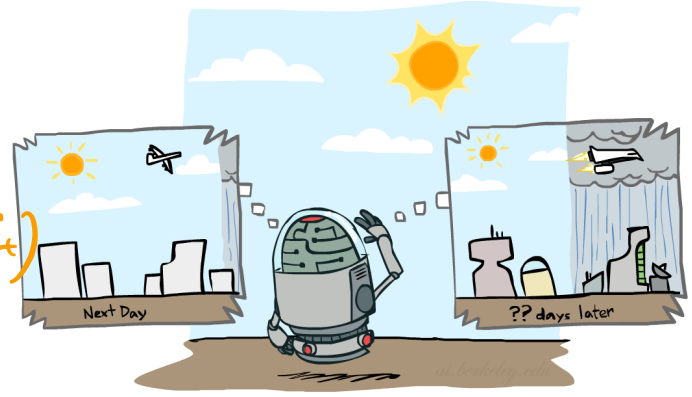
$$P_\infty(X) = P_{\infty+1}(X) = \sum_x P(X|x)P_\infty(x)$$

Example: Stationary Distributions

- Computing the stationary distribution



$$\sum_{x_t} P(x_t) P(X_{t+1}|x_t)$$



$$P_\infty(sun) = P(sun|sun)P_\infty(sun) + P(sun|rain)P_\infty(rain)$$
$$P_\infty(rain) = P(rain|sun)P_\infty(sun) + P(rain|rain)P_\infty(rain)$$

$$P_\infty(sun) = 0.9P_\infty(sun) + 0.3P_\infty(rain)$$
$$P_\infty(rain) = 0.1P_\infty(sun) + 0.7P_\infty(rain)$$

$$P_\infty(sun) = 3P_\infty(rain)$$
$$P_\infty(rain) = 1/3P_\infty(sun)$$

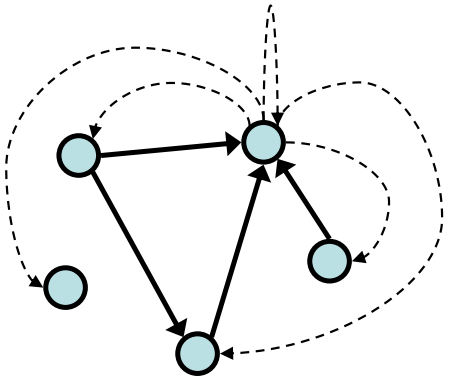
Also: $P_\infty(sun) + P_\infty(rain) = 1$

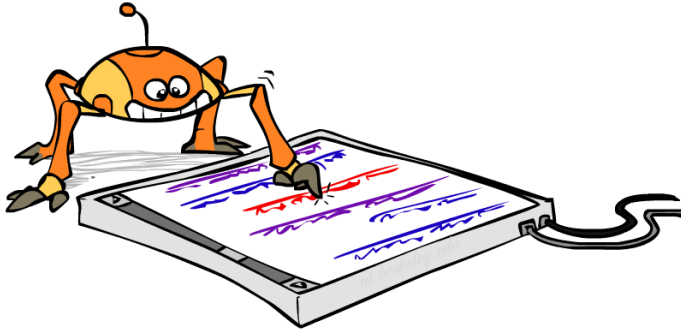
$$P_\infty(sun) = 3/4$$
$$P_\infty(rain) = 1/4$$

X_{t-1}	X_t	$P(X_t \mid X_{t-1})$
sun	sun	0.9
sun	rain	0.1
rain	sun	0.3
rain	rain	0.7

Application of Stationary Distribution: Web Link Analysis

- Web browsing
  - Each web page is a state
  - Initial distribution: uniform over pages
  - Transitions:
    - With prob. c, uniform jump to a random page
    - With prob. 1-c, follow a random outlink
- Stationary distribution: PageRank
  - Will spend more time on highly reachable pages
  - Google 1.0 returned the set of pages containing all your keywords in decreasing rank
  - Now: use link analysis along with many other factors (rank actually getting less important)

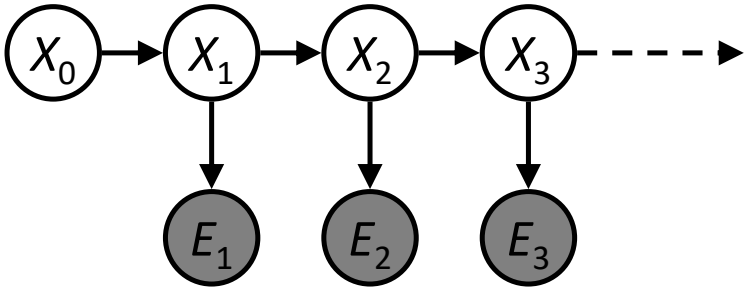




Hidden Markov Models

Hidden Markov Models

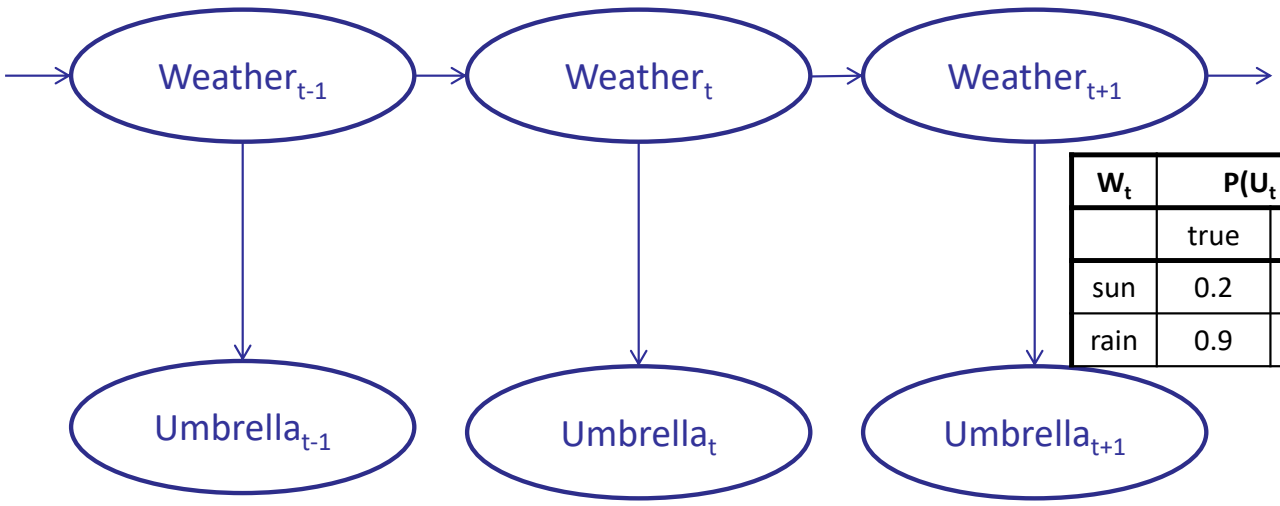
- Usually the true state is not observed directly
  - E.g., you stay indoor and cannot see the weather, but you can see if people come in with umbrella or not.
- Hidden Markov models (HMMs)
  - Underlying Markov chain over states *X*
  - You observe evidence *E* at each time step



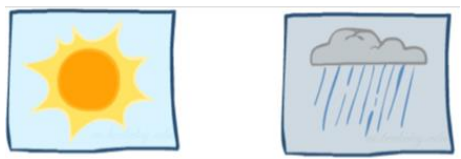
Example: Weather HMM

- An HMM is defined by:
  - Initial distribution: $P(X_0)$
  - Transition model: $P(X_t \mid X_{t-1})$
  - Emission model: $P(E_t \mid X_t)$

W_{t-1}	$P(W_t \mid W_{t-1})$	
	sun	rain
sun	0.9	0.1
rain	0.3	0.7



W_t	$P(U_t \mid W_t)$	
	true	false
sun	0.2	0.8
rain	0.9	0.1

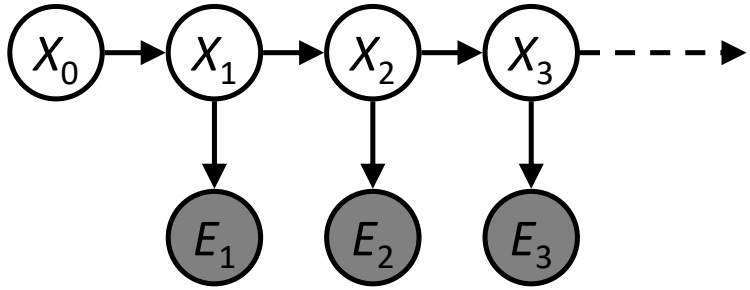


HMM as probability model

- Joint distribution for Markov model: $P(X_0,\ldots, X_T) = P(X_0) \prod_{t=1:T} P(X_t \mid X_{t-1})$
- Joint distribution for hidden Markov model:

  $P(X_0,X_1, E_1,\ldots, X_T,E_T) = P(X_0) \prod_{t=1:T} P(X_t \mid X_{t-1})\, P(E_t \mid X_t)$

- Independence in HMM
  - Future states are independent of the past given the present
  - Current evidence is independent of everything else given the current state



Real HMM Examples

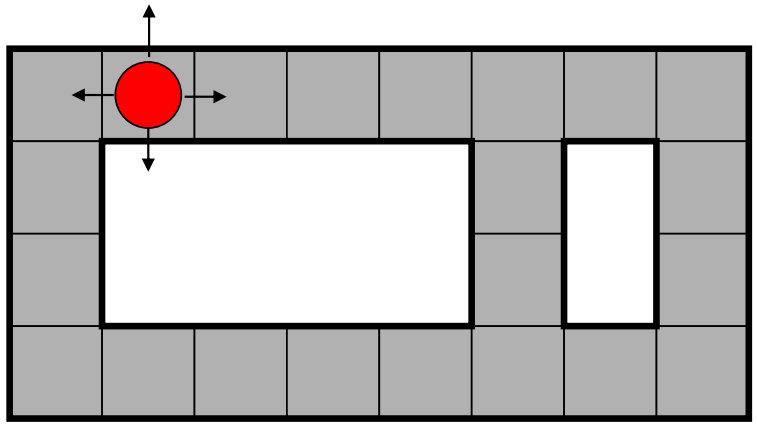
- Speech recognition HMMs:
  - Observations are acoustic signals (continuous valued)
  - States are specific positions in specific words (so, tens of thousands)
- Machine translation HMMs:
  - Observations are words (tens of thousands)
  - States are translation options
- Robot tracking:
  - Observations are range readings (continuous)
  - States are positions on a map (continuous)
- Molecular biology:
  - Observations are nucleotides ACGT
  - States are coding/non-coding/start/stop/splice-site etc.

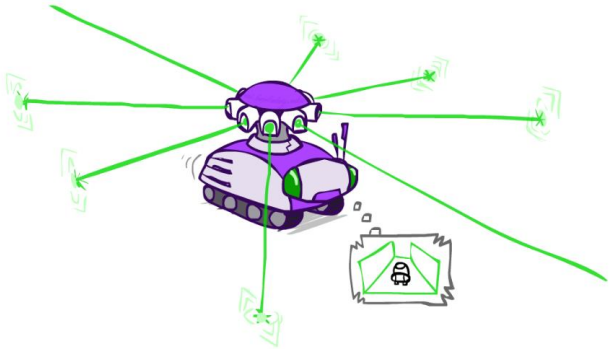
Inference tasks

Notation: $B_t(X_t) = P(X_t | e_{1:t})$
$B_t'(X_t) = P(X_t | e_{1:t-1})$

- Useful notation: $X_{a:b} = X_a, X_{a+1}, \ldots, X_b$
- ***Filtering***: $P(X_t | e_{1:t})$ 当前时刻 $B_t(X) = P(X_t | e_{1:t})$ Belife distribution
  - ***belief state*** — posterior distribution over the most recent state given all evidence
- ***Prediction***: $P(X_{t+k} | e_{1:t})$ for $k > 0$ 未来时刻
  - posterior distribution over a future state given all evidence
- ***Smoothing***: $P(X_k | e_{1:t})$ for $0 \le k < t$ 过去时刻
  - posterior distribution over a past state given all evidence
- ***Most likely explanation***: $\arg\max_{x_{0:t}} P(x_{0:t} \mid e_{1:t})$ 连续0~t时刻对应的distribution分布最似然情况.
  - Ex: speech recognition, decoding with a noisy channel → 不同时刻状态最有可能的情况.

Example: Robot Localization





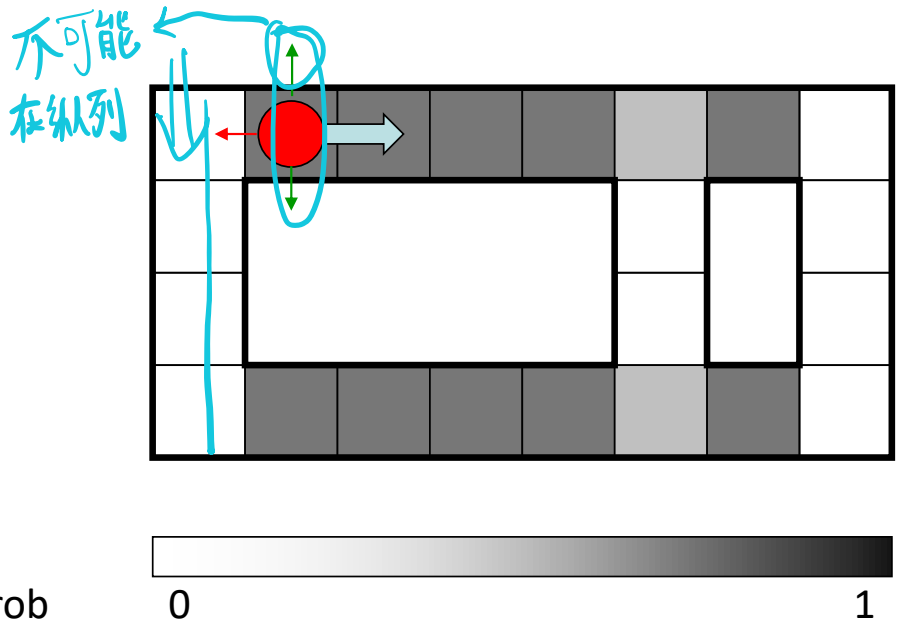
t=1

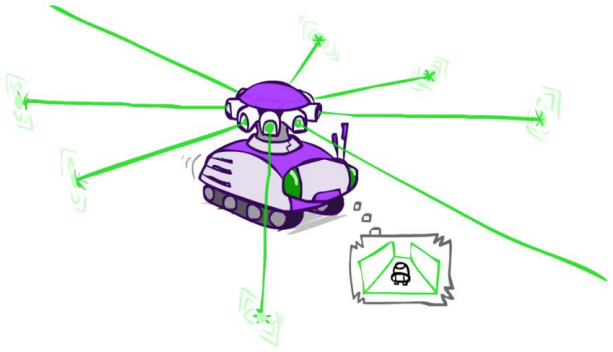
Hidden state: robot location

Sensor model: four bits for wall/no-wall in each direction, never more than 1 mistake

Transition model: action may fail with small prob.

Example: Robot Localization

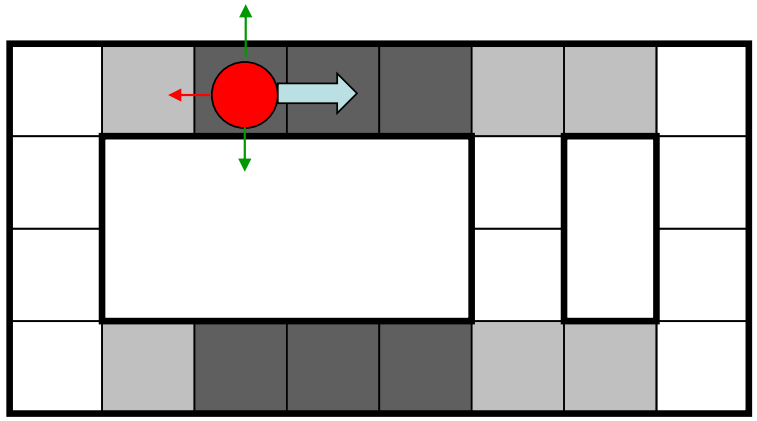


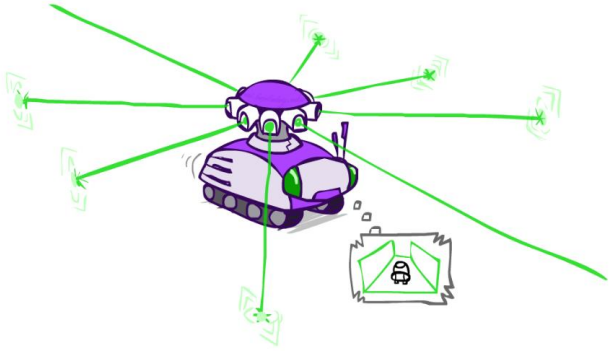


t=1

Lighter grey: was possible to get the reading, but less likely b/c required 1 mistake

Example: Robot Localization

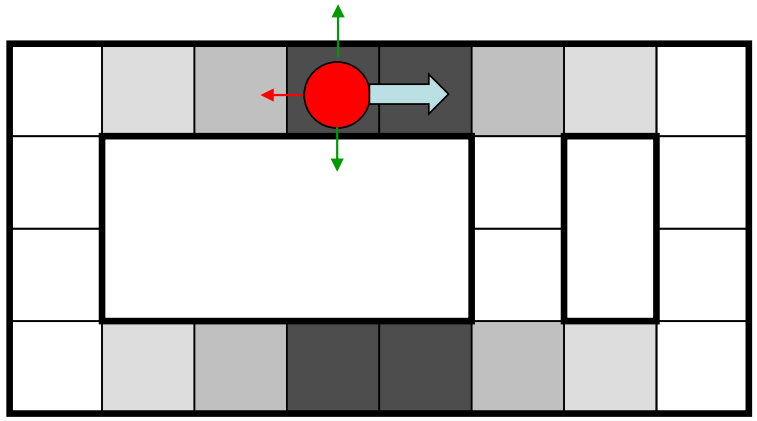


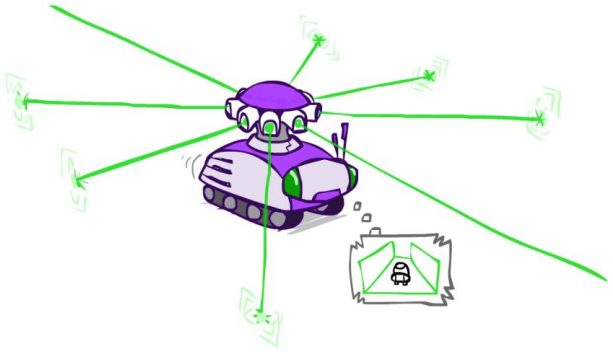


Prob 0 1

t=2

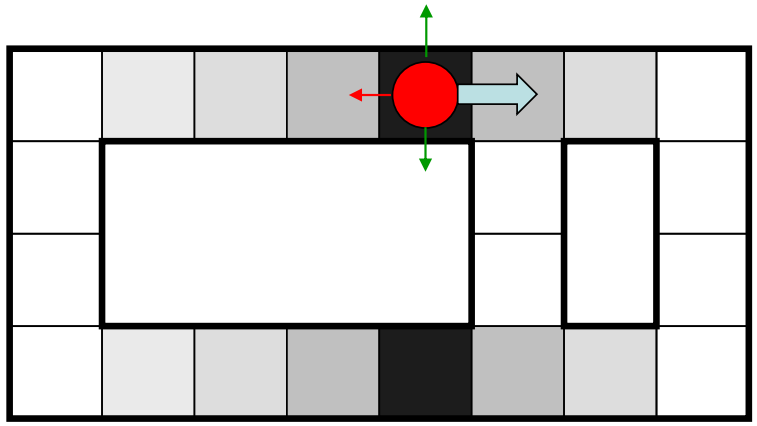
Example: Robot Localization

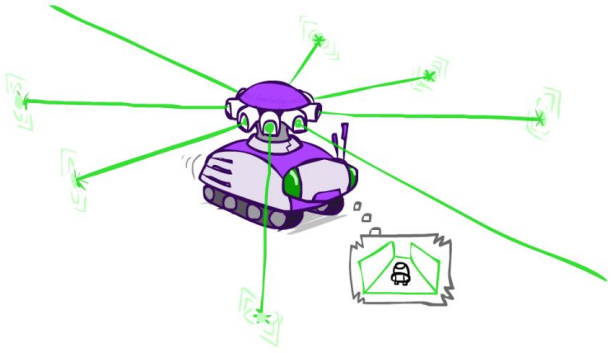




t=3

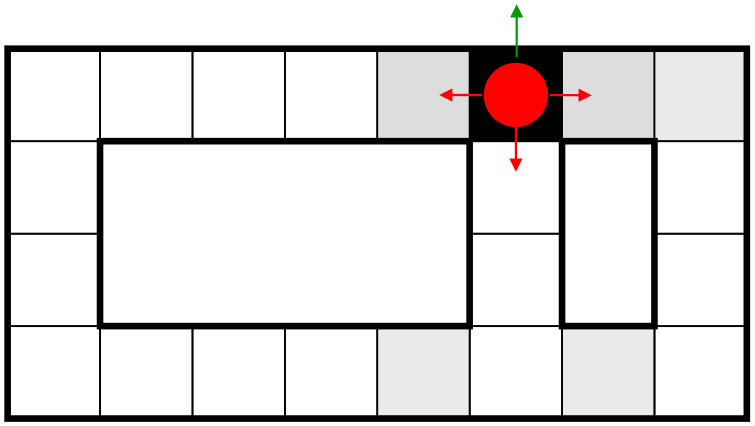
Example: Robot Localization

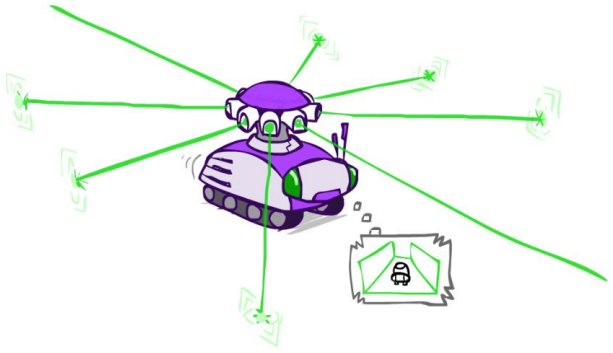




t=4

Example: Robot Localization





t=5

Filtering algorithm

- Filtering: infer current state given all evidence $P(X_t | e_{1:t})$
- Aim: a ***recursive filtering*** algorithm of the form or aka. $P(X_{t+1} | e_{1:t+1})$
  - $P(X_{t+1} | e_{1:t+1}) = g(e_{t+1}, P(X_t | e_{1:t}))$
    递归函数

Apply Bayes' rule

- $P(X_{t+1} | e_{1:t+1}) = P(X_{t+1} | e_{1:t}, e_{t+1})$
- $= \alpha\, P(e_{t+1} | X_{t+1}, e_{1:t})\, P(X_{t+1} | e_{1:t})$

$\alpha = 1 / P(e_{t+1} | e_{1:t})$

$= \alpha\, P(X_{t+1}, e_{1:t}, e_{t+1})$

$= \alpha\, P(X_{t+1}, e_{t+1} | e_{1:t})$

$= \alpha\, P(X_{t+1} | e_{1:t}) \cdot P(e_{t+1} | X_{t+1}, e_{1:t})$

↳ $\frac{P(e_{t+1}, X_{t+1} | e_{1:t})}{P(e_{t+1} | e_{1:t})} = P(X_{t+1} | e_{1:t}, e_{t+1})$

Filtering algorithm

- Filtering: infer current state given all evidence
- Aim: a ***recursive filtering*** algorithm of the form
  - $P(X_{t+1} | e_{1:t+1}) = g(e_{t+1}, P(X_t | e_{1:t}))$

the e_{t+1} is independent with others given X_{t+1}.

- $P(X_{t+1} | e_{1:t+1}) = P(X_{t+1} | e_{1:t}, e_{t+1})$
- $= \alpha\, P(e_{t+1} | X_{t+1}, e_{1:t})\, P(X_{t+1} | e_{1:t})$ — Apply conditional independence
- $= \alpha\, P(e_{t+1} | X_{t+1})\, P(X_{t+1} | e_{1:t})$

Normalize | Update | Predict

Filtering algorithm

- Filtering: infer current state given all evidence
- Aim: a ***recursive filtering*** algorithm of the form
  - $P(X_{t+1} | e_{1:t+1}) = g(e_{t+1}, P(X_t | e_{1:t}))$

$P(X_{t+1} | e_{1:t})$

$= \sum_{x_t} P(x_t, X_{t+1} | e_{1:t})$

$= \sum_{x_t} P(x_t | e_{1:t}) P(X_{t+1} | x_t, e_{1:t})$

- $P(X_{t+1} | e_{1:t+1}) = P(X_{t+1} | e_{1:t}, e_{t+1})$
- $= \alpha\, P(e_{t+1} | X_{t+1}, e_{1:t})\, P(X_{t+1} | e_{1:t})$
- $= \alpha\, P(e_{t+1} | X_{t+1})\, P(X_{t+1} | e_{1:t})$
- $= \alpha\, P(e_{t+1} | X_{t+1}) \sum_{x_t} P(x_t | e_{1:t})\, P(X_{t+1} | x_t, e_{1:t})$

$\alpha = \frac{1}{P(e_{t+1} | e_{1:t})}$

无关

Condition on X_t

$\sum_{x_t} P(X_{t+1}, x_t | e_{1:t})$

Filtering algorithm

- Filtering: infer current state given all evidence
- Aim: a ***recursive filtering*** algorithm of the form
  - $P(X_{t+1} | e_{1:t+1}) = g(e_{t+1}, P(X_t | e_{1:t}))$

$P(X_{t+1} | e_{1:t+1}) = P(X_{t+1} | e_{1:t}, e_{t+1})$

$= \alpha\, P(e_{t+1} | X_{t+1}, e_{1:t})\, P(X_{t+1} | e_{1:t})$

$= \alpha\, P(e_{t+1} | X_{t+1})\, P(X_{t+1} | e_{1:t})$

$= \alpha\, P(e_{t+1} | X_{t+1}) \sum_{x_t} P(x_t | e_{1:t})\, P(X_{t+1} | x_t, e_{1:t})$

$= \alpha\, P(e_{t+1} | X_{t+1}) \sum_{x_t} P(x_t | e_{1:t})\, P(X_{t+1} | x_t)$

$P(X_{t+1} | e_{1:t})$

$P = \sum_{x_t} P(x_t | e_{1:t})\, P(X_{t+1} | x_t, e_{1:t})$

X_{t+1}(Future) is independent with the past.

Apply conditional independence

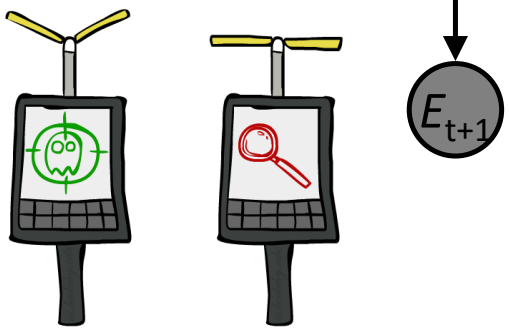
Filtering algorithm

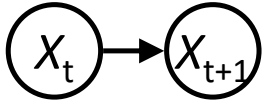
- $P(X_{t+1} | e_{1:t+1}) = \alpha\, P(e_{t+1} | X_{t+1}) \sum_{x_t} P(x_t \mid e_{1:t})\, P(X_{t+1} | x_t)$

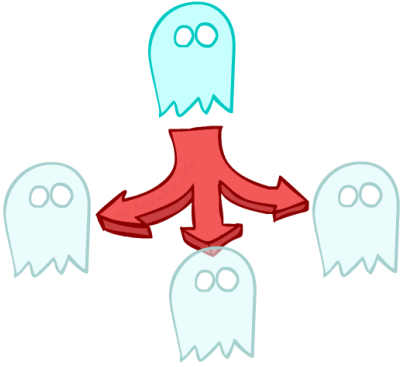
Normalize

Update

Predict







Filtering algorithm

- $P(X_{t+1} | e_{1:t+1}) = \alpha\, P(e_{t+1} | X_{t+1}) \sum_{x_t} P(x_t \mid e_{1:t})\, P(X_{t+1} | x_t)$

Normalize | Update | Predict

- $\boldsymbol{f}_{1:t+1} = \text{FORWARD}(\boldsymbol{f}_{1:t}, e_{t+1})$
- We start with $\boldsymbol{f}_{1:0} = P(X_0)$ and then iterate
- Cost per time step: $O(|X|^2)$ where $|X|$ is the number of states

$B_{t+1}(X) = \alpha\, P(e_{t+1} | X_{t+1}) \sum_{x_t} B_t(x_t)\, P(X_{t+1} | x_t)$

X种情况(对 x_{t+1})

X种情况(对 X_{t+1})

Do some math

$$P(X_{t+1}|e_{1:t+1}) = P(X_{t+1}|e_{1:t}, e_{t+1})$$

$$= P(X_{t+1}|e_{1:t})P(e_{t+1}|X_{t+1}, e_{1:t}) \frac{1}{P(e_{t+1}|e_{t:1})}$$

$$= P(e_{t+1}|X_{t+1}) \cdot \alpha \cdot P(X_{t+1}|e_{t:1})$$

$$= \alpha P(e_{t+1}|X_{t+1}) \cdot \sum_{x_t} P(x_t|e_{t:1}) \cdot \frac{P(X_{t+1}|x_t, e_{1:t})}{P(X_{t+1}|x_t)}$$

$\therefore$ $P(X_{t+1}|e_{1:t+1}) = g(e_{t+1}, P(X_t|e_{1:t}))$

$f_{1:t+1} = \text{Forward}(f_{1:t}, e_{t+1})$

(并不是特别关心它的值)

normalization

$f_{1:0} = P(X_0)$ $\quad f_{1:1} = P(X_1|e_1) = \alpha P(e_1|X_1) \sum_{x_0} P(x_0)P(X_1|x_0)$

实际上是对 $P(X_{t+1}, e_{1:t}, e_{t+1})$ 规范化

$= P(e_{t+1}|X_{t+1})P(X_{t+1}, e_{1:t})$

$= P(e_{t+1}|X_{t+1}) \sum_{x_t} P(X_{t+1}, x_t, e_{1:t})$

$= \sum_{x_1 \cdots x_t} P(x_1, x_2 \cdots x_t, X_{t+1}, e_{1:t+1})$ 进行 Elimination

$f_{1:3} = P(X_3|e_{3:1})$

$= \alpha P(e_3|X_3) \sum_{x_2} P(x_2|e_{2:1}) P(X_3|x_2, e_{1:2})$

$f_{1:2}$

$f_{1:2} = P(X_2|e_{1:2})$

$= \alpha P(e_2|x_2) \sum_{x_1} P(x_1|e_1) P(X_2|x_1, e_1)$

$f_{1:1}$

$f_{0:1}$

时间复杂度分析:

① $\sum_{x_t} P(X_{t+1}|x_t)P(x_t|e_{1:t})$

② X_{t+1} 包含了 $|N|$ 情况

③ 单步时间复杂度为 $|N|^2$

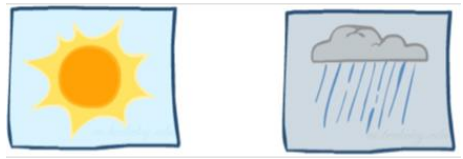
④ Ask For $f_{1:t}$: $|T|$

⑤ $\therefore$ total: $|T|\,|N|^2$

Belief Updating = the forward algorithm broken down into two steps and with normalization

- Forward algorithm: $P(x_t, e_{1:t}) = P(e_t|x_t) \sum_{x_{t-1}} P(x_t|x_{t-1}) P(x_{t-1}, e_{1:t-1})$
- Can break this down into:
  - Time update: $P(x_t, e_{1:t-1}) = \sum_{x_{t-1}} P(x_t|x_{t-1}) P(x_{t-1}, e_{1:t-1})$
  - Observation update: $P(x_t, e_{1:t}) = P(e_t|x_t) P(x_t, e_{1:t-1})$
- Normalizing in the observation update gives:
  - Time update: $P(x_t|e_{1:t-1}) = \sum_{x_{t-1}} P(x_t|x_{t-1}) P(x_{t-1}|e_{1:t-1})$
  - Observation update: $P(x_t|e_{1:t}) \propto P(e_t|x_t) P(x_t|e_{1:t-1})$
- Notation: $B_t(x_t) = P(x_t|e_{1:t}), \quad B'_t(x_t) = P(x_t|e_{1:t-1})$
  - Time update: $B'_t(x_t) = \sum_{x_{t-1}} P(x_t|x_{t-1}) B_{t-1}(x_{t-1})$
  - Observation update: $B_t(x_t) = P(e_t|x_t) B'_t(x_t)$

Example: Weather HMM



$\sum_{x_t} P(X_t | e_{1:t}) P(X_{t+1} | X_t)$

predict

0.6
0.4

$P(s) = 0.5 \times 0.9 + 0.5 \times 0.3 = 0.6$
$P(r) = 0.5 \times 0.1 + 0.5 \times 0.7 = 0.4$

update & normalize

$\times P(e_{t+1} | X_{t+1})$

f(sun) = 0.5
f(rain) = 0.5

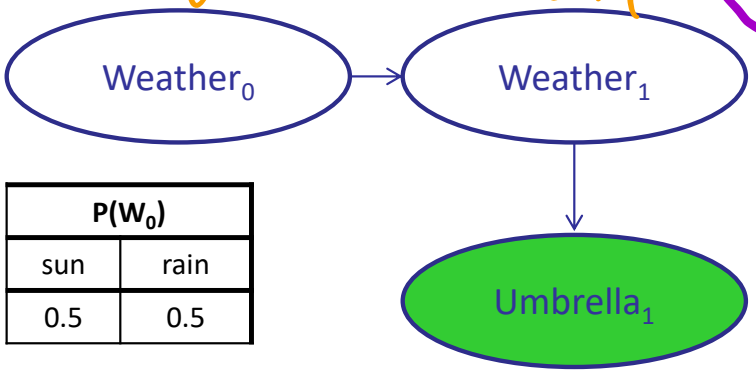
f(sun) = 0.25
f(rain) = 0.75

$P(s|u) \propto 0.6 \times 0.2 = 0.12$
$P(r|u) \propto 0.4 \times 0.9 = 0.36$

normalization

t

t+1



$P(W_0)$	
sun	rain
0.5	0.5

$P(X_{t+1} | e_{1:t+1}) = \alpha\, P(e_{t+1} | X_{t+1}) \sum_{x_t} P(x_t \mid e_{1:t})\, P(X_{t+1} | x_t)$

Normalize | Update | Predict

调整权重

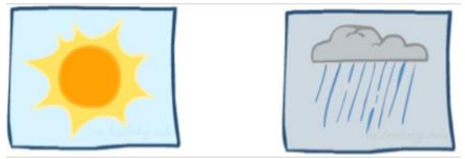
W_{t-1}	$P(W_t \mid W_{t-1})$	
	sun	rain
sun	0.9	0.1
rain	0.3	0.7

W_t	$P(U_t \mid W_t)$	
	true	false
sun	0.2	0.8
rain	0.9	0.1

$P(S|u) = f_{0:1}$

$= \alpha P(u|S) \sum_{x_0} P(x_0) P(S | x_0)$

Example: Weather HMM



$$P(X_{t+1}|e_{1:t+1}) = \alpha\, P(e_{t+1}|X_{t+1}) \sum_{x_t} P(x_t \mid e_{1:t})\, P(X_{t+1}|x_t)$$

Normalize | Update | Predict

predict

0.6
0.4

update & normalize

f(sun) = 0.5
f(rain) = 0.5

predict

f(sun) = 0.25
f(rain) = 0.75

update & normalize

0.45
0.55

f(sun) = 0.154
f(rain) = 0.846

$0.25 \times 0.9 + 0.75 \times 0.3 = 0.45$

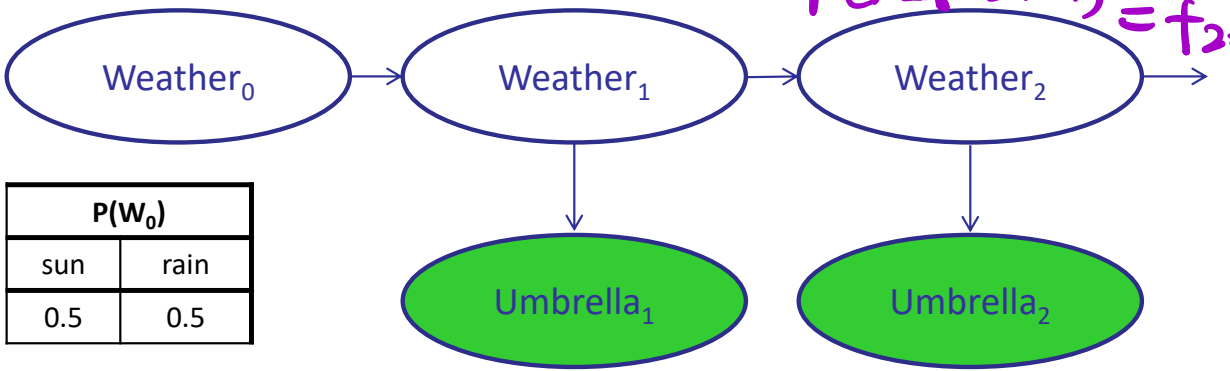
$0.25 \times 0.1 + 0.75 \times 0.7 = 0.55$

$\in \begin{cases} 0.45 \times 0.2 = 0.09 \\ 0.55 \times 0.9 \end{cases}$

$= 0.495$

W_{t-1}	$P(W_t \mid W_{t-1})$	
	sun	rain
sun	0.9	0.1
rain	0.3	0.7

$P(X_2|e_2,e_1) = f_{2:1} = \alpha P(e_2|X_2) \sum_{x_1} P(x_1|e_1) P(X_2|x_1, e_1)$

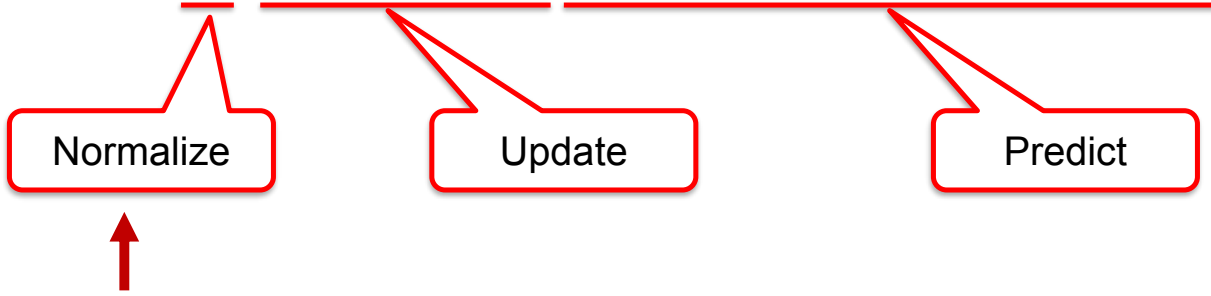


$P(W_0)$	
sun	rain
0.5	0.5

W_t	$P(U_t \mid W_t)$	
	true	false
sun	0.2	0.8
rain	0.9	0.1

Filtering algorithm

- $P(X_{t+1} | e_{1:t+1}) = \alpha\, P(e_{t+1} | X_{t+1}) \sum_{x_t} P(x_t \mid e_{1:t})\, P(X_{t+1} | x_t)$

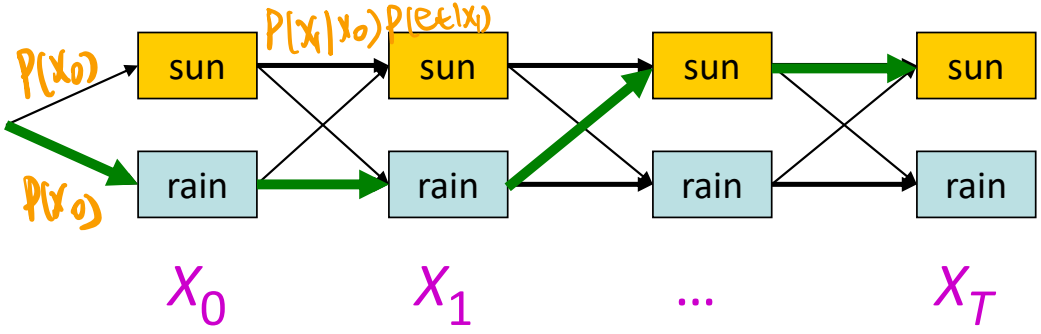


α is a constant. So if we only want to compute $P(x_t \mid e_{1:t})$, then we can skip normalization when computing $P(x_1 \mid e_1)$, $P(x_2 \mid e_{1:2})$, …, $P(x_{t-1} \mid e_{1:t-1})$

最后一步归一化就ok了

Another view of the algorithm

- ***State trellis***: graph of states and transitions over time

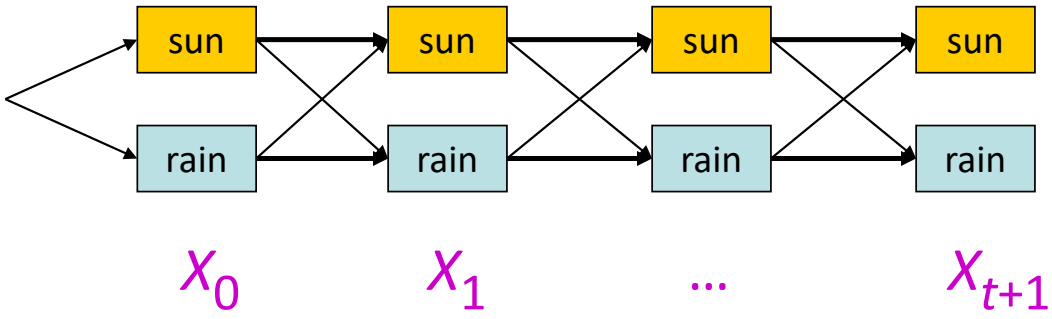


- Each arc represents some transition $x_{t-1} \rightarrow x_t$
- Each arc has weight $P(x_t \mid x_{t-1})\, P(e_t \mid x_t)$ (arcs to initial states have weight $P(x_0)$)
- Each path is a sequence of states
- The ***product*** of weights on a path is proportional to that state sequence's probability

$$P(x_0) \prod_t P(x_t \mid x_{t-1})\, P(e_t \mid x_t) = P(x_{1:t}, e_{1:t}) \propto P(x_{1:t} \mid e_{1:t})$$

$$\log P(x_0) + \sum \log P(x_t \mid x_{t-1}) + \sum \log P(e_t \mid x_t)$$

Another view of the algorithm



对应的就是一条path

$P(X_0)\prod_t P(X_t|X_{t-1})\,P(e_t|X_t)$

- Forward algorithm computes sum over all possible paths

  $P(X_{t+1}|e_{1:t+1}) = \sum_{x_{1:t}} P(x_{1:t+1} \mid e_{1:t+1}) \simeq \sum_{x_1,x_2,\cdots x_t} P(x_1,x_2,\cdots x_{t+1}|e_1,\cdots e_{t+1})$

- It uses dynamic programming to sum over all paths
  - For each state at time t, keep track of the total probability of all paths to it

    $\boldsymbol{f}_{1:t+1} = \text{FORWARD}(\boldsymbol{f}_{1:t}, e_{t+1})$

    $= \alpha\, P(e_{t+1}|X_{t+1}) \sum_{x_t} P(X_{t+1}| x_t)\, \boldsymbol{f}_{1:t}[x_t]$

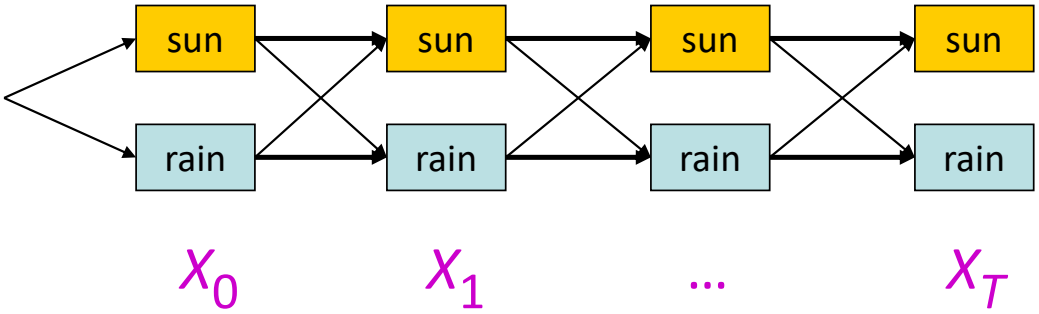
Most Likely Explanation

Inference tasks

- ***Filtering***: $P(X_t | e_{1:t})$
  - ***belief state***—input to the decision process of a rational agent
- ***Prediction***: $P(X_{t+k} | e_{1:t})$ for $k > 0$
  - evaluation of possible action sequences; like filtering without the evidence
- ***Smoothing***: $P(X_k | e_{1:t})$ for $0 \leq k < t$
  - better estimate of past states, essential for learning
- ***Most likely explanation***: $\arg\max_{x_{0:t}} P(x_{0:t} \mid e_{1:t})$
  - speech recognition, decoding with a noisy channel

Most likely explanation = most probable path

- ***State trellis***: graph of states and transitions over time



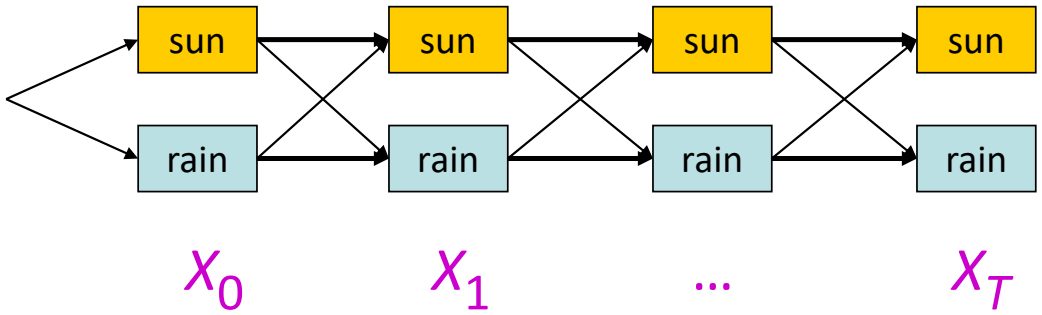
- The ***product*** of weights on a path is proportional to that state sequence's probability

  $P(x_0) \prod_t P(x_t \mid x_{t-1}) P(e_t \mid x_t) = P(x_{0:t}, e_{1:t}) \propto P(x_{0:t} \mid e_{1:t})$

- ***Viterbi algorithm*** computes best paths

  $\arg\max_{x_{0:t}} P(x_{0:t} \mid e_{1:t})$

Forward / Viterbi algorithms



Viterbi Algorithm (max)

For each state at time t, keep track of the ***maximum probability of any path*** to it

$\boldsymbol{m}_{1:t+1} = \text{VITERBI}(\boldsymbol{m}_{1:t}, e_{t+1})$

$= P(e_{t+1}|X_{t+1}) \max_{x_t} P(X_{t+1}|x_t)\, \boldsymbol{m}_{1:t}[x_t]$

Forward Algorithm (sum)

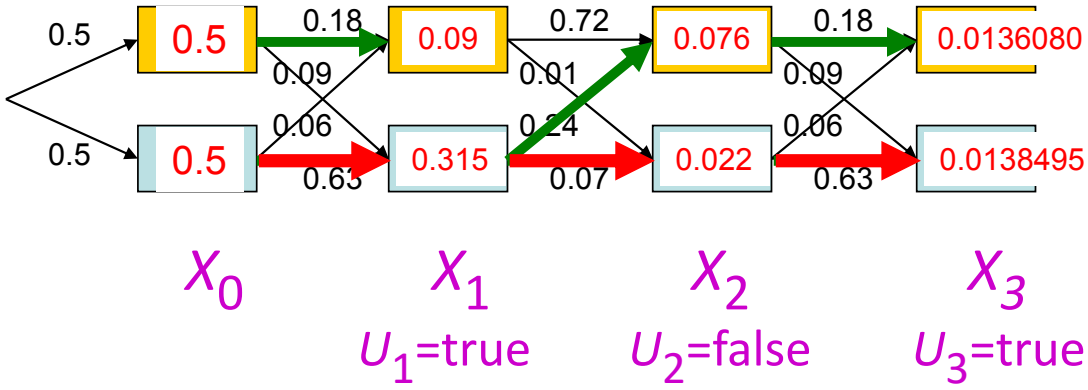
For each state at time t, keep track of the ***total probability of all paths*** to it

$\boldsymbol{f}_{1:t+1} = \text{FORWARD}(\boldsymbol{f}_{1:t}, e_{t+1})$

$= \alpha\, P(e_{t+1}|X_{t+1}) \sum_{x_t} P(X_{t+1}|x_t)\, \boldsymbol{f}_{1:t}[x_t]$

Viterbi algorithm contd.

$P(W_0)$	
sun	rain
0.5	0.5



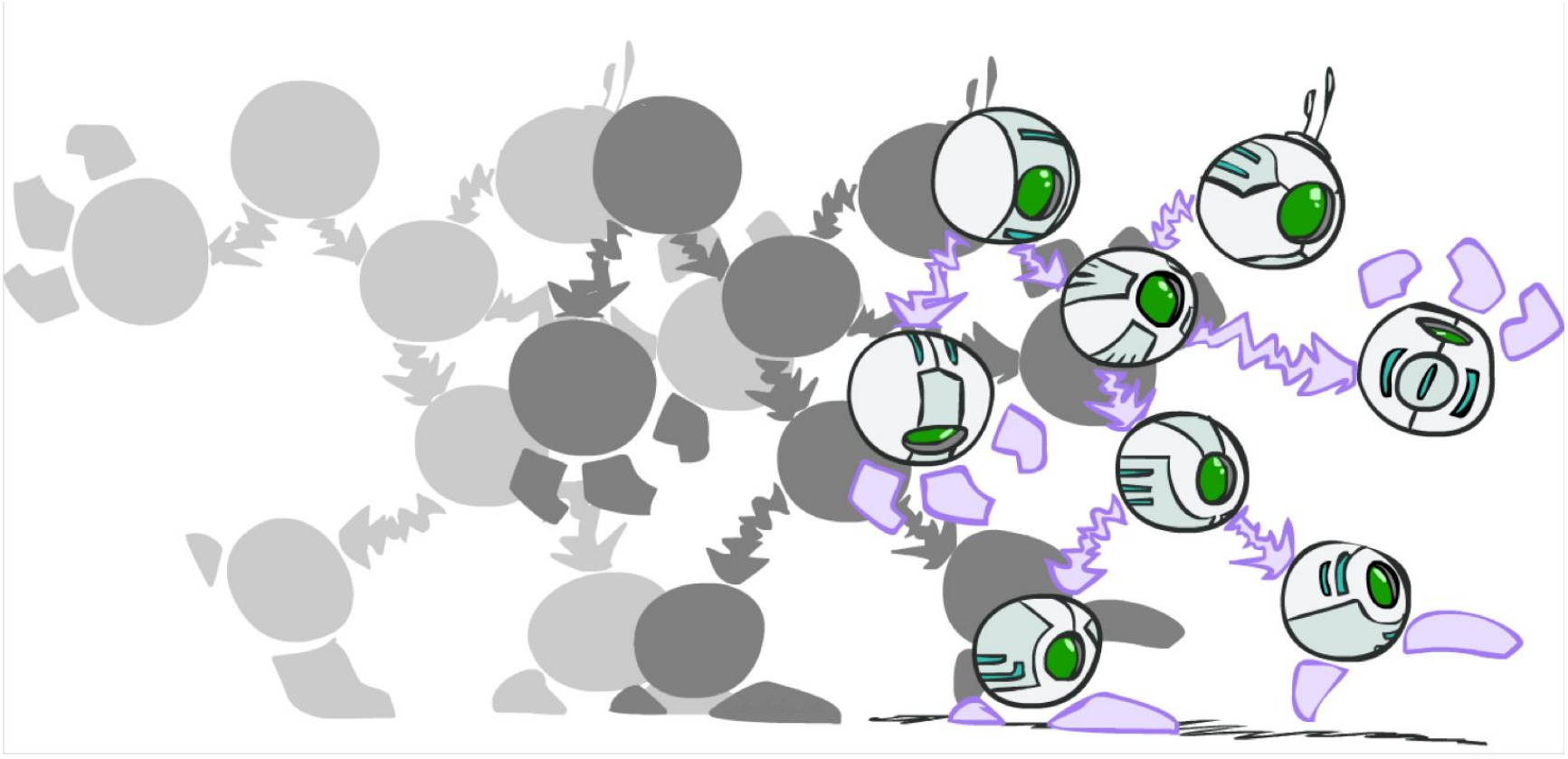
W_{t-1}	$P(W_t \mid W_{t-1})$	
	sun	rain
sun	0.9	0.1
rain	0.3	0.7

W_t	$P(U_t \mid W_t)$	
	true	false
sun	0.2	0.8
rain	0.9	0.1

- $\boldsymbol{m}_{1:t+1} = P(e_{t+1} \mid X_{t+1}) \max_{x_t} P(X_{t+1} \mid x_t)\, \boldsymbol{m}_{1:t}[x_t]$

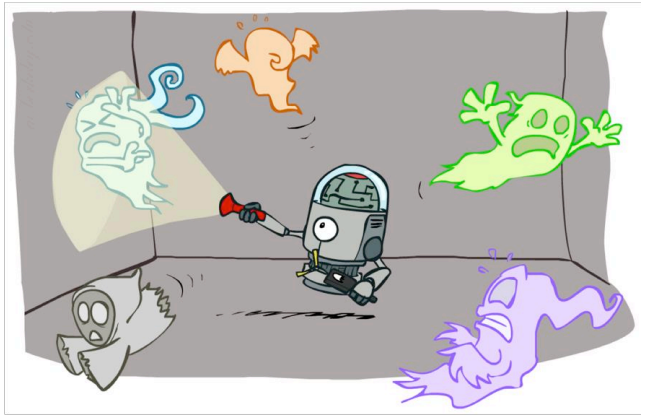
- Time complexity: **O(|X|² T)**
- Space complexity: **O(|X| T)** dp 1x1T

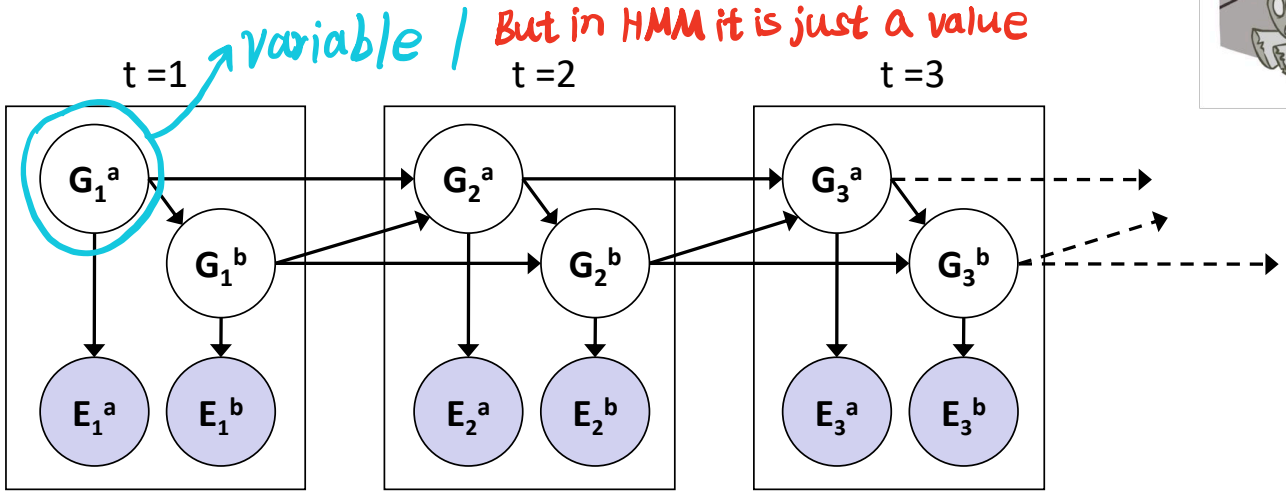
Dynamic Bayes Nets



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*





t=2的数据集是在t=1的数据集上获得的

DBNs and HMMs

- Every HMM is a DBN
- Every discrete DBN can be represented by a HMM
  - Each HMM state is Cartesian product of DBN state variables
    - E.g., 3 binary state variables => one state variable with 2^3 possible values
  - Advantage of DBN vs. HMM?
    - Sparse dependencies => exponentially fewer parameters
    - E.g., 20 binary state variables, 2 parents each;
      DBN has 20 x 2^{2+1} = 160 parameters, HMM has 2^{20} x 2^{20} =~ 10^{12} parameters

指的是转移矩阵

DBN: $n * 2^{(k+1)}$ → 每个状态变量只依赖k个父结点，每个状态变量取两个值(binary)，有n个状态变量，父结点自身还有两种情况

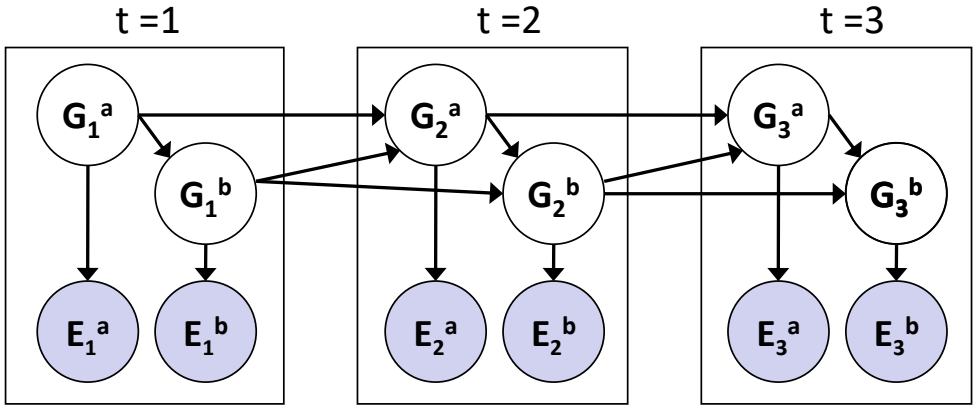
HMM: $2^n \cdot 2^n$

P_1 又 sun / rain

P_2

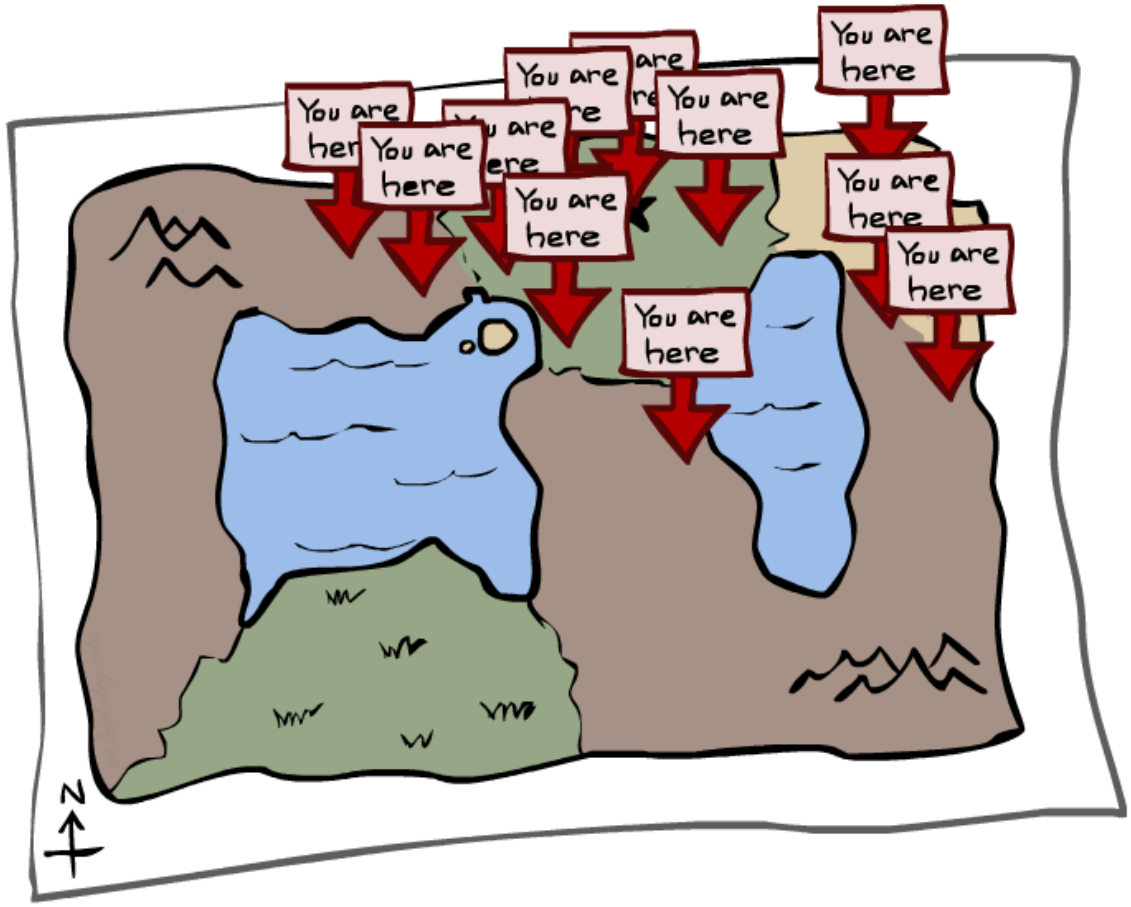
Exact Inference in DBNs

- Variable elimination applies to dynamic Bayes nets
- Offline: "unroll" the network for T time steps, then eliminate variables to find $P(X_T | e_{1:T})$
  - Problem: results in very large BN



- Can we do better?
  - Do we need to unroll for many steps? What is the best variable order of elimination?
- Online: unroll as we go, eliminate all variables from the previous time step

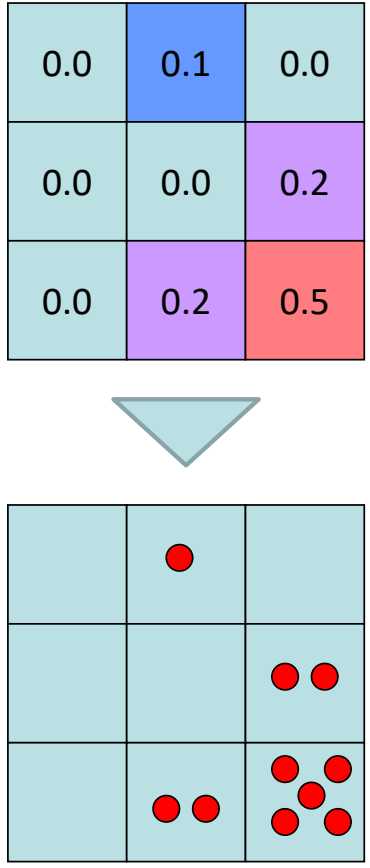
Particle Filtering



Particle Filtering 近似推断

sample

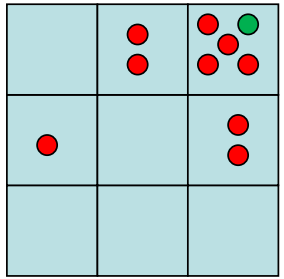
- Filtering: approximate solution
- Sometimes |X| is too big to use exact inference
- Solution: approximate inference
  - Track samples of X, not all values
  - Samples are called ***particles***
- Our representation of P(X) is now a list of N particles (samples)
  - P(x) approximated by number of particles with value x
    - So, many x may have P(x) = 0
  - Generally, N << |X|
    - More particles, more accuracy; but a large N would defeat the point.



Representation: Particles

- At first, all particles have a weight of 1

$$P(2,3) = \frac{1}{5}$$



Particles:
(3,3)
(2,3)
(3,3)
(3,2)
(3,3)
(3,2)
(1,2)
(3,3)
(3,3)
(2,3)

Particle Filtering: Propagate forward

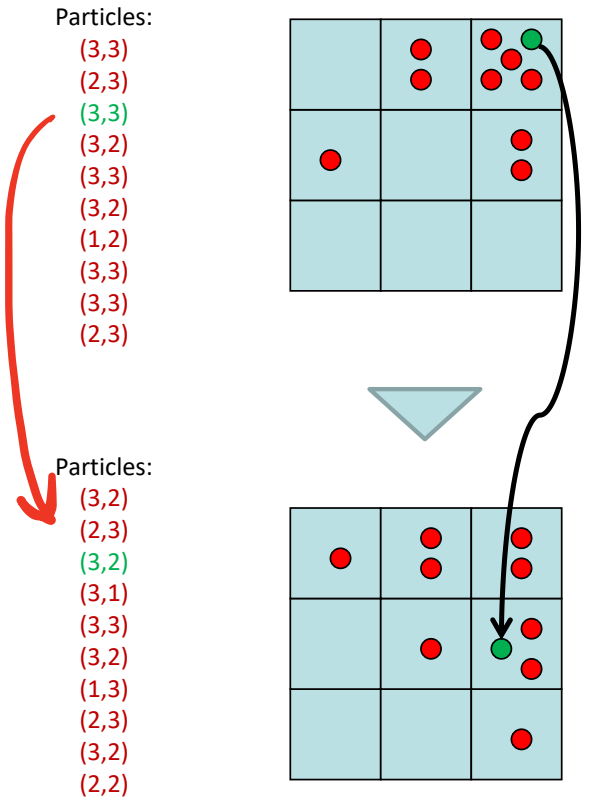
- Each particle is moved by sampling its next position from the transition model:
  - $x_{t+1} \sim P(X_{t+1} \mid x_t)$

$x_{t+1} \sim \text{Sample}(P(X_{t+1} \mid x_t))$

- This captures the passage of time
  - If enough samples, close to exact probabilities (consistent)

$x_t = (3,3)$

eg. sample (x_{t+1}: (3,2), (3,3) ; $P(x_{t+1} \mid x_t)$: 0.2, 0.8)



Particle Filtering: Observe

Tricker: Don't sample observation, fix it.

- Similar to likelihood weighting, weight samples based on the evidence
  - $W = P(e_t \mid x_t)$ 我们只需要包含 e_t 的
  - Particles that fit the evidence better get higher weights, others get lower weights

$f_{1:t+1} \propto P(e|X)\, f_t(X)$

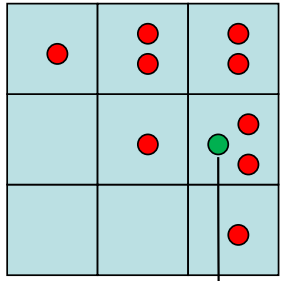
这个是需要加的权重

- What happens if we repeat the Propagate-Observe procedure over time?
  - It is exactly likelihood weighting (if we multiply the weights)
  - Weights drop quickly…

问题：但是有很多权重为零的点，原因为它们的跳转可能与 evidence 并不一致。

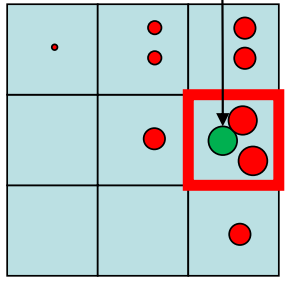
我们并不想让他们继续存在

Particles:
(3,2)
(2,3)
(3,2)
(3,1)
(3,3)
(3,2)
(1,3)
(2,3)
(3,2)
(2,2)



Vote

Particles:
(3,2) w=.9
(2,3) w=.2
(3,2) w=.9
(3,1) w=.4
(3,3) w=.4
(3,2) w=.9
(1,3) w=.1
(2,3) w=.2
(3,2) w=.9
(2,2) w=.4



Particle Filtering: Resample

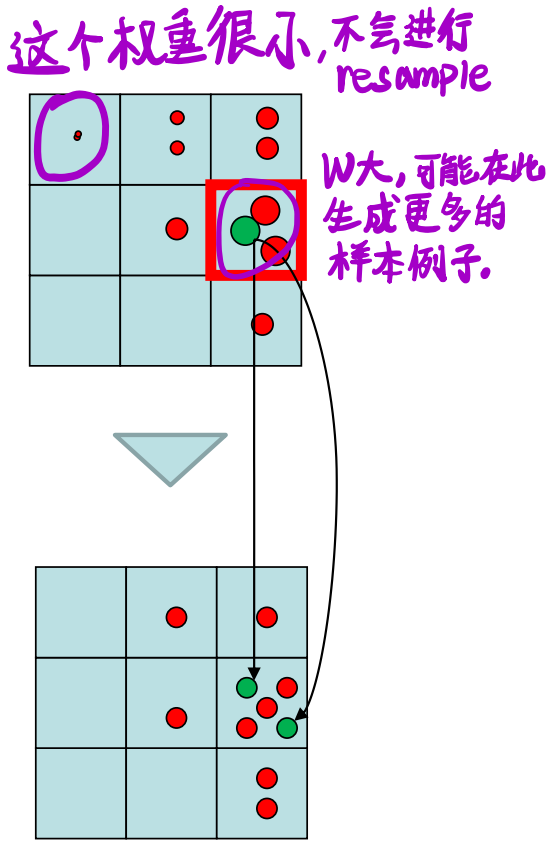
- Rather than tracking weighted samples, we ***resample***
  - Generate *N* new samples from our weighted sample distribution
  - Each new sample is selected from the current population of samples; the probability is proportional to its weight.
  - The new samples have weight of 1

  ☆权重为隐式的

- Now the update is complete for this time step, continue with the next one

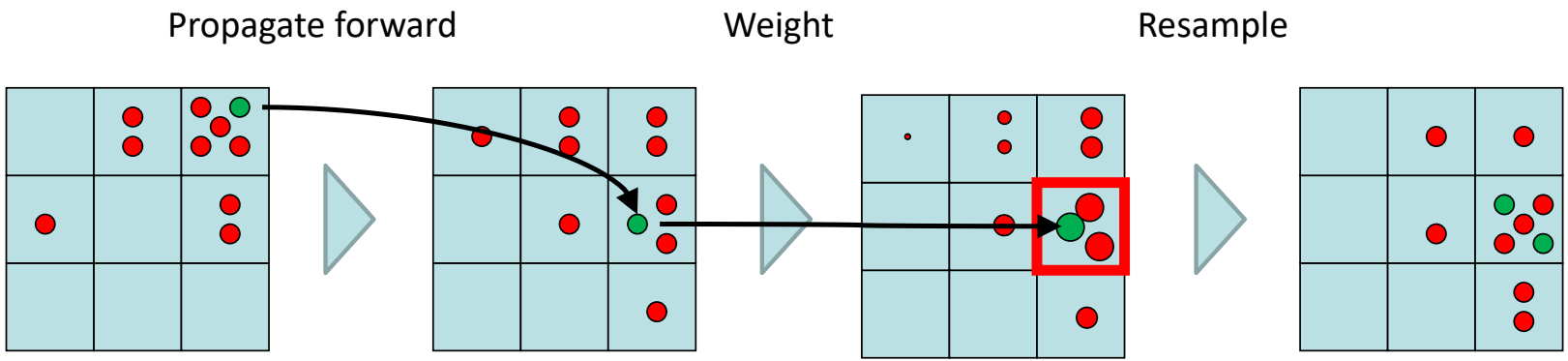
Particles:
(3,2) w=.9
(2,3) w=.2
(3,2) w=.9
(3,1) w=.4
(3,3) w=.4
(3,2) w=.9
(1,3) w=.1
(2,3) w=.2
(3,2) w=.9
(2,2) w=.4

(New) Particles:
(3,2)
(2,2)
(3,2)
(2,3)
(3,3)
(3,2)
(1,3)
(2,3)
(3,2)
(3,2)



Summary: Particle Filtering

- Particles: track samples of states rather than an explicit distribution

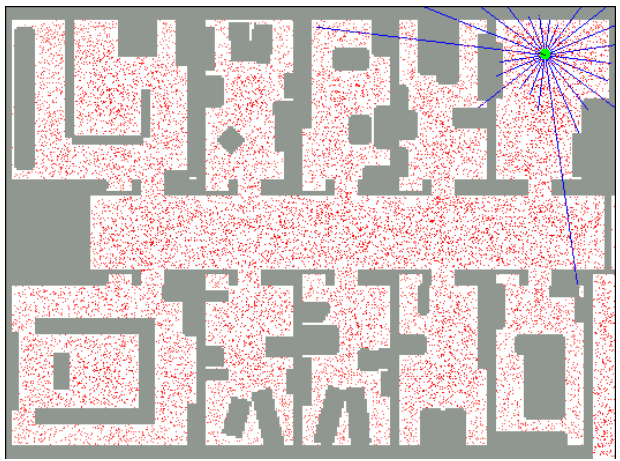


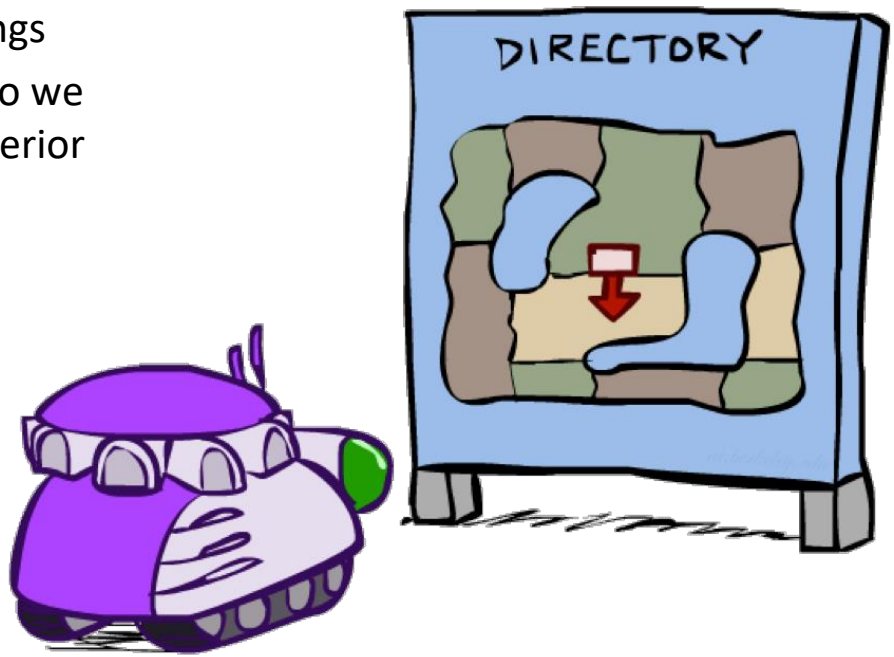
Particles:	Particles:	Particles:	(New) Particles:
(3,3)	(3,2)	(3,2) w=.9	(3,2)
(2,3)	(2,3)	(2,3) w=.2	(2,2)
(3,3)	(3,2)	(3,2) w=.9	(3,2)
(3,2)	(3,1)	(3,1) w=.4	(2,3)
(3,3)	(3,3)	(3,3) w=.4	(3,3)
(3,2)	(3,2)	(3,2) w=.9	(3,2)
(1,2)	(1,3)	(1,3) w=.1	(1,3)
(3,3)	(2,3)	(2,3) w=.2	(2,3)
(3,3)	(3,2)	(3,2) w=.9	(3,2)
(2,3)	(2,2)	(2,2) w=.4	(3,2)

Consistency: see proof in AIMA Ch. 15

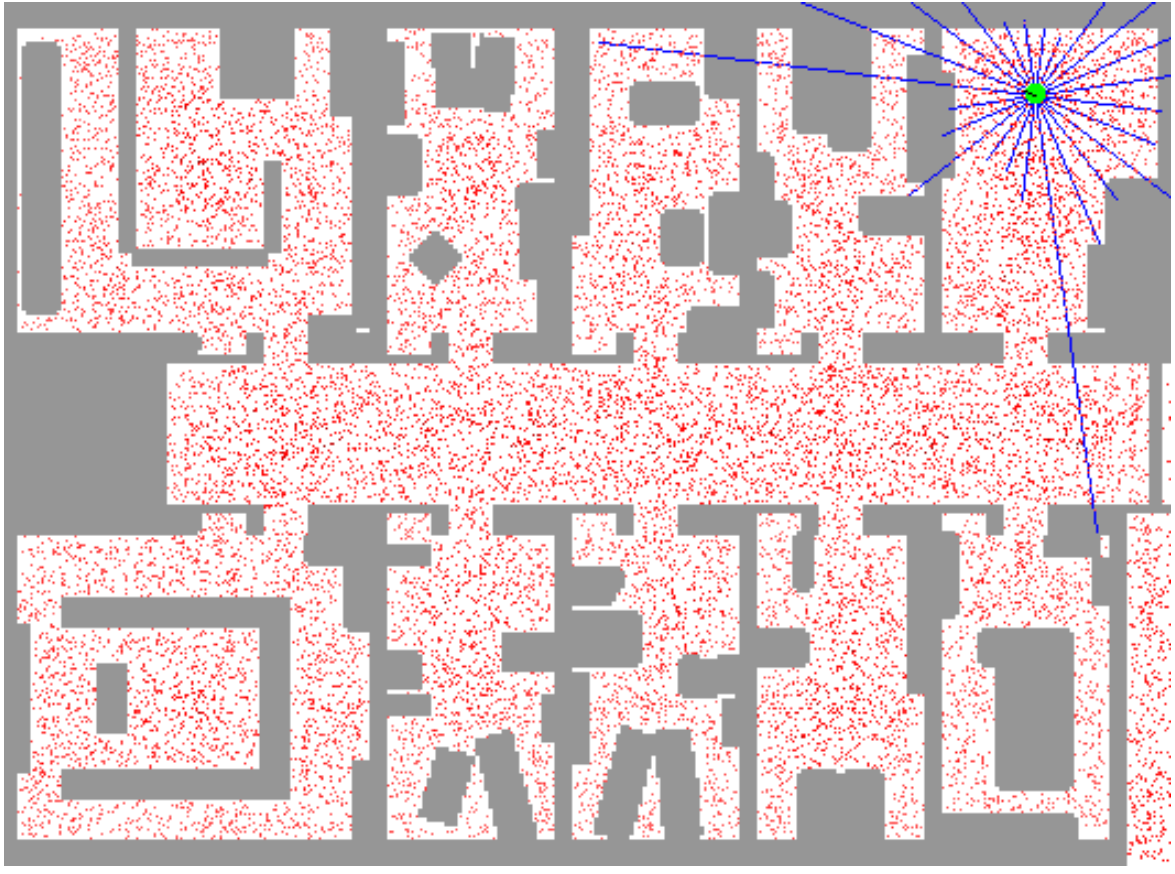
Robot Localization

- 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 so we cannot usually represent or compute an exact posterior
  - Particle filtering is a main technique





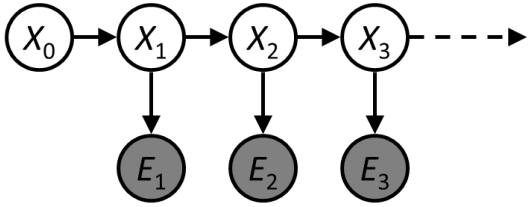
Particle Filter Localization (Laser)

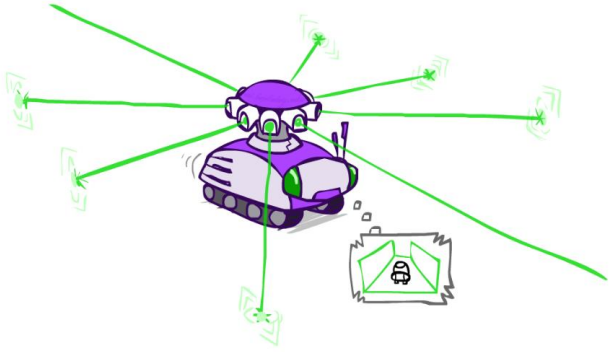


[Dieter Fox, et al.]

Summary

- Probabilistic temporal models
  - Markov model
  - Hidden Markov model
    - Filtering: forward algorithm
    - MLE: Viterbi algorithm
  - Dynamic Bayesian network
  - Approximate inference by particle filtering





CS 188 Introduction to Artificial Intelligence
Fall 2022 Note 18

These lecture notes are heavily based on notes originally written by Nikhil Sharma.

Last updated: October 25, 2022

Particle Filtering

Recall that with Bayes' nets, when running exact inference was too computationally expensive, using one of the sampling techniques we discussed was a viable alternative to efficiently approximate the desired probability distribution(s) we wanted. Hidden Markov Models have the same drawback - the time it takes to run exact inference with the forward algorithm scales with the number of values in the domains of the random variables. This was acceptable in our current weather problem formulation where the weather can only take on 2 values, $W_i \in \{sun, rain\}$, but say instead we wanted to run inference to compute the distribution of the actual temperature on a given day to the nearest tenth of a degree.

The Hidden Markov Model analog to Bayes' net sampling is called **particle filtering**, and involves simulating the motion of a set of particles through a state graph to approximate the probability (belief) distribution of the random variable in question. This solves the same question as the Forward Algorithm: it gives us an approximation of $P(X_N|e_{1:N})$.

Instead of storing a full probability table mapping each state to its belief probability, we'll instead store a list of n **particles**, where each particle is in one of the d possible states in the domain of our time-dependent random variable. Typically, n is significantly smaller than d (denoted symbolically as $n << d$) but still large enough to yield meaningful approximations; otherwise the performance advantage of particle filtering becomes negligible. Particles are just the name for samples in this algorithm.

Our belief that a particle is in any given state at any given timestep is dependent entirely on the number of particles in that state at that timestep in our simulation. For example, say we indeed wanted to simulate the belief distribution of the temperature T on some day i and assume for simplicity that this temperature can only take on integer values in the range $[10, 20]$ ($d = 11$ possible states). Assume further that we have $n = 10$ particles, which take on the following values at timestep i of our simulation:

$$[15, 12, 12, 10, 18, 14, 12, 11, 11, 10]$$

By taking counts of each temperature that appears in our particle list and diving by the total number of particles, we can generate our desired empirical distribution for the temperature at time i:

T_i	10	11	12	13	14	15	16	17	18	19	20
$B(T_i)$	0.2	0.2	0.3	0	0.1	0.1	0	0	0.1	0	0

Now that we've seen how to recover a belief distribution from a particle list, all that remains to be discussed is how to generate such a list for a timestep of our choosing.

Particle Filtering Simulation

Particle filtering simulation begins with particle initialization, which can be done quite flexibly - we can sample particles randomly, uniformly, or from some initial distribution. Once we've sampled an initial list of particles, the simulation takes on a similar form to the forward algorithm, with a time elapse update followed by an observation update at each timestep:

- *Time Elapse Update* - Update the value of each particle according to the transition model. For a particle in state t_i, sample the updated value from the probability distribution given by $P(T_{i+1}|t_i)$. Note the similarity of the time elapse update to prior sampling with Bayes' nets, since the frequency of particles in any given state reflects the transition probabilities.
- *Observation Update* - During the observation update for particle filtering, we use the sensor model $P(F_i|T_i)$ to weight each particle according to the probability dictated by the observed evidence and the particle's state. Specifically, for a particle in state t_i with sensor reading f_i, assign a weight of $P(f_i|t_i)$. The algorithm for the observation update is as follows:

  1. Calculate the weights of all particles as described above. 计算所有棋子的权重
  2. Calculate the total weight for each state. 每个 State 所有棋子权重相加
  3. If the sum of all weights across all states is 0, reinitialize all particles. (weight=0重采样)
  4. Else, normalize the distribution of total weights over states and resample your list of particles from this distribution.

  Note the similarity of the observation update to likelihood weighting, where we again downweight samples based on our evidence.

Let's see if we can understand this process slightly better by example. Define a transition model for our weather scenario using temperature as the time-dependent random variable as follows: for a particular temperature state, you can either stay in the same state or transition to a state one degree away, within the range $[10, 20]$. Out of the possible resultant states, the probability of transitioning to the one closest to 15 is 80% and the remaining resultant states uniformly split the remaining 20% probability amongst themselves.

Our temperature particle list was as follows:

$$[15, 12, 12, 10, 18, 14, 12, 11, 11, 10]$$

To perform a time elapse update for the first particle in this particle list, which is in state $T_i = 15$, we need the corresponding transition model:

T_{i+1}	14	15	16
$P(T_{i+1}\|T_i = 15)$	0.1	0.8	0.1

In practice, we allocate a different range of values for each value in the domain of T_{i+1} such that together the ranges entirely span the interval $[0, 1)$ without overlap. For the above transition model, the ranges are as follows:

1. The range for $T_{i+1} = 14$ is $0 \leq r < 0.1$.
2. The range for $T_{i+1} = 15$ is $0.1 \leq r < 0.9$.

3. The range for $T_{i+1} = 16$ is $0.9 \leq r < 1$.

In order to resample our particle in state $T_i = 15$, we simply generate a random number in the range $[0, 1)$ and see which range it falls in. Hence if our random number is $r = 0.467$, then the particle at $T_i = 15$ remains in $T_{i+1} = 15$ since $0.1 \leq r < 0.9$. Now consider the following list of 10 random numbers in the interval $[0, 1)$:

$$[0.467, 0.452, 0.583, 0.604, 0.748, 0.932, 0.609, 0.372, 0.402, 0.026]$$

If we use these 10 values as the random value for resampling our 10 particles, our new particle list after the full time elapse update should look like this:

$$[15, 13, 13, 11, 17, 15, 13, 12, 12, 10]$$

Verify this for yourself! The updated particle list gives rise to the corresponding updated belief distribution $B(T_{i+1})$:

T_i	10	11	12	13	14	15	16	17	18	19	20
$B(T_{i+1})$	0.1	0.1	0.2	0.3	0	0.2	0	0.1	0	0	0

Comparing our updated belief distribution $B(T_{i+1})$ to our initial belief distribution $B(T_i)$, we can see that as a general trend the particles tend to converge towards a temperature of $T = 15$.

Next, let's perform the observation update, assuming that our sensor model $P(F_i|T_i)$ states that the probability of a correct forecast $f_i = t_i$ is 80%, with a uniform 2% chance of the forecast predicting any of the other 10 states. Assuming a forecast of $F_{i+1} = 13$, the weights of our 10 particles are as follows:

Particle	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}
State	15	13	13	11	17	15	13	12	12	10
Weight	0.02	0.8	0.8	0.02	0.02	0.02	0.8	0.02	0.02	0.02

Then we aggregate weights by state:

State	10	11	12	13	15	17
Weight	0.02	0.02	0.04	2.4	0.04	0.02

Summing the values of all weights yields a sum of 2.54, and we can normalize our table of weights to generate a probability distribution by dividing each entry by this sum:

State	10	11	12	13	15	17
Weight	0.02	0.02	0.04	2.4	0.04	0.02
Normalized Weight	0.0079	0.0079	0.0157	0.9449	0.0157	0.0079

The final step is to resample from this probability distribution, using the same technique we used to resample during the time elapse update. Let's say we generate 10 random numbers in the range $[0, 1)$ with the following values:

$$[0.315, 0.829, 0.304, 0.368, 0.459, 0.891, 0.282, 0.980, 0.898, 0.341]$$

This yields a resampled particle list as follows:

$$[13, 13, 13, 13, 13, 13, 13, 15, 13, 13]$$

With the corresponding final new belief distribution:

T_i	10	11	12	13	14	15	16	17	18	19	20
$B(T_{i+1})$	0	0	0	0.9	0	0.1	0	0	0	0	0

Observe that our sensor model encodes that our weather prediction is very accurate with probability 80%, and that our new particles list is consisistent with this since most particles are resampled to be $T_{i+1} = 13$.

Summary

In this note, we covered two new types of models:

- *Markov models*, which encode time-dependent random variables that possess the Markov property. We can compute a belief distribution at any timestep of our choice for a Markov model using probabilistic inference with the mini-forward algorithm.
- *Hidden Markov Models*, which are Markov models with the additional property that new evidence which can affect our belief distribution can be observed at each timestep. To compute the belief distribution at any given timestep with Hidden Markov Models, we use the forward algorithm.

Sometimes, running exact inference on these models can be too computationally expensive, in which case we can use particle filtering as a method of approximate inference.