Filtering

CS 480
Intro to Artificial Intelligence

Reasoning with uncertainty and time

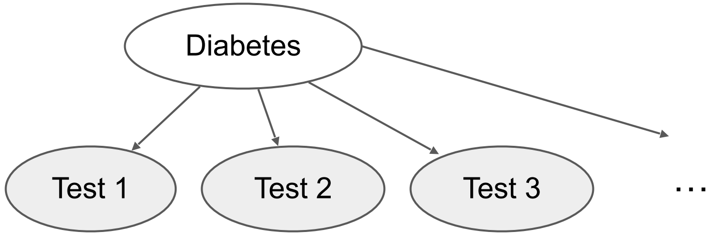
We’ve seen how Bayes Nets can help us **reason** about state that we can’t directly measure: Apply Bayes Rule so we can use probabilities we can measure

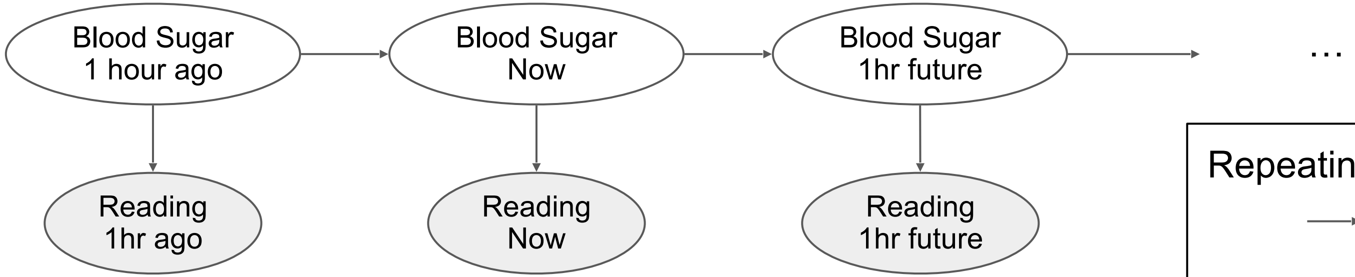
$$p(Hidden|Evidence) = \frac{p(Evidence|Hidden)p(Hidden)}{p(Evidence)}$$

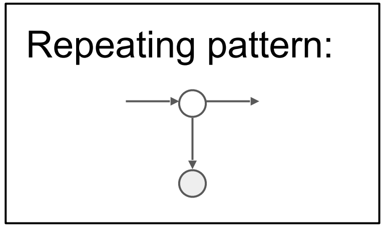
How can we leverage Bayes Nets to **reason** about how state may change over time?

In partially observable environments, what’s the most likely **sequence of states** given a **sequence of sensor readings**?

Compare: diagnosis vs management







Temporal Random Variables - Notation

Unobserved random variable at time t

$$X_t$$

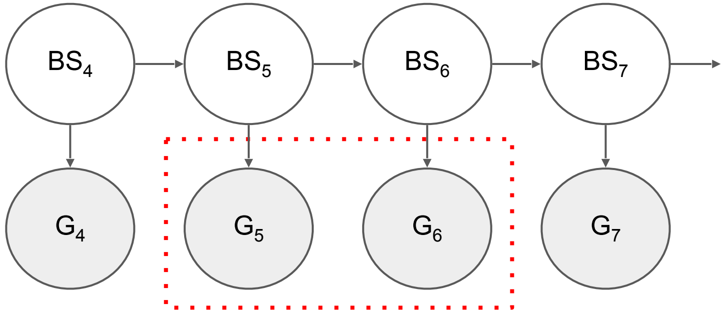
Observed random variable at time t

$$E_t$$

Time series from a to b, a **set** of random variables

$$Y_{a:b} = \{Y_a, Y_{a+1}, \ldots, Y_{b-1}, Y_b\}$$

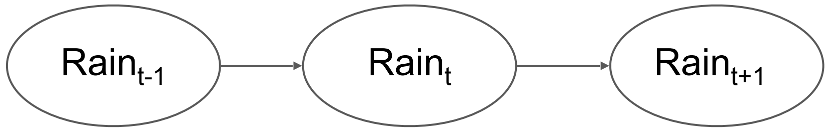
Example: $G_{5:6}$ (glucose readings between time 5 and 6)



Markov Assumptions

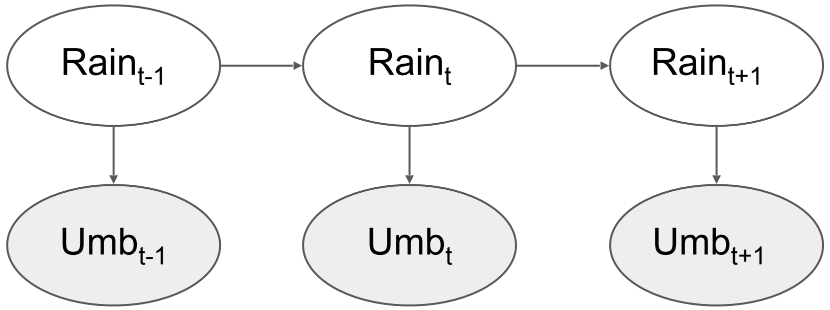
First Order Markov Assumption
X_t depends on X_{t-1} **only**

$$p(X_t \mid X_{1:t-1}) = p(X_t \mid X_{t-1})$$



Sensor Markov Assumption
E_t depends on X_t **only**

$$p(E_t \mid X_{1:t}, E_{1:t-1}) = p(E_t \mid X_t)$$



Defining necessary probabilities

Transition Model

$$p(X_t \mid X_{t-1})$$

Probability of rain today given rain yesterday

	$p(R_t=T \mid R_{t-1})$	$p(R_t=F \mid R_{t-1})$
$R_{t-1} = T$	0.7	0.3
$R_{t-1} = F$	0.3	0.7

Sensor Model

$$p(E_t \mid X_t)$$

Probability of seeing an umbrella given raining

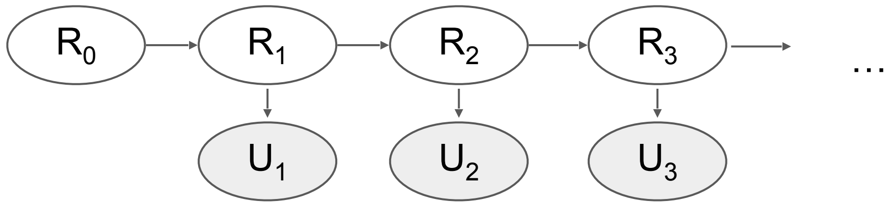
	$p(U_t=T \mid R_t)$	$p(U_t=F \mid R_t)$
$R_t=T$	0.9	0.1
$R_t=F$	0.2	0.8

Prior

$$p(X_0)$$

Prior probability of rain

$p(R_0=T)$	$p(R_0=F)$
0.5	0.5


$$\begin{tikzcd} R_0 \arrow[r] & R_1 \arrow[r] \arrow[d] & R_2 \arrow[r] \arrow[d] & R_3 \arrow[r] \arrow[d] & \ldots \\ & U_1 & U_2 & U_3 & \end{tikzcd}$$

What can we learn using this framework?

Two types of questions we can ask:

- **Filtering** (estimation): probability of current state given sequence of evidence

$$p(X_t \mid E_{1:t})$$

- **Prediction**: distribution over **future** states

$$p(X_{t+k} \mid E_{1:t})$$

Notation:
$p(e_t) := p(E_t=e_t)$

Exact filtering (1)

Split $(e_{1:t+1})$ into $(e_{t+1}, e_{1:t})$

$$p(X_{t+1} \mid e_{1:t+1}) = p(X_{t+1} \mid e_{t+1}, e_{1:t})$$

Bayes Rule

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

Sensor Markov assumption

$$= \alpha \cdot p(e_{t+1} \mid X_{t+1}) \cdot p(X_{t+1} \mid e_{1:t})$$

$$= \alpha \cdot p(e_{t+1} \mid X_{t+1}) \cdot \sum_h p(X_{t+1}, X_t = h \mid e_{1:t})$$

Marginalize to introduce X_t

Exact filtering (2)

Def. of cond. prob.

$$\underline{p(X_{t+1} \mid e_{1:t+1})} = \alpha \cdot p(e_{t+1} \mid X_{t+1}) \cdot \sum_h p(X_{t+1}, X_t = h \mid e_{1:t})$$

$$= \alpha \cdot p(e_{t+1} \mid X_{t+1}) \cdot \sum_h p(X_{t+1} \mid X_t = h, e_{1:t}) \cdot p(X_t = h \mid e_{1:t})$$

First order Markov assumption

$$= \alpha \cdot \underline{p(e_{t+1} \mid X_{t+1})} \cdot \sum_h \underline{p(X_{t+1} \mid X_t = h)} \cdot \underline{p(X_t = h \mid e_{1:t})}$$

Recurrence!

Sensor Model

Transition Model

Base case: **Prior**
$p(X_0|e_{1:0}) = p(X_0)$

Exact filtering (3)

Let’s rewrite this probability so we can compute it iteratively as the agent moves forward in time, rather than recursively.

$$p(X_t \mid e_{1:t}) = \alpha \cdot p(e_t \mid X_t) \sum_h p(X_t \mid X_{t-1} = h) p(X_{t-1} = h \mid e_{1:t-1})$$

$$\underline{\text{belief}_t(X_t = s_i)} = \alpha \cdot p(e_t \mid X_t = s_i) \sum_h p(X_t = s_i \mid X_{t-1} = h) \underline{\text{belief}_{t-1}(X_{t-1} = h)}$$

New, updated belief (for each state)

For finite state space, can represent belief(X_t=s_i) as a table (like we did for utility)

Belief at the previous time step

Exact filtering algorithm

```
def exact_filtering_agent(sensor_model, transition_model, prior):
    belief = {s:prior(s) for s in states}
    while (not_done()):
        new_belief = apply_transition_model(transition_model,belief)
        e = get_sensors()
        if not(e is None):
            new_belief = apply_sensor_model(sensor_model,e,new_belief)
        belief = new_belief
```

Note: we can **decouple** the transition model and sensor model

```
def apply_transition_model(transition_model,belief):
    new_belief = {}
    for s in states:
        for s_prime in states:
            new_belief[s_prime] = belief[s]*transition_model(s,s_prime)
    return new_belief.normalize()
```

Note: double for loop, pick order that's most efficient!

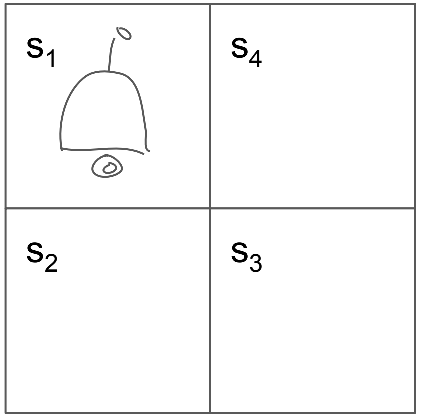
```
def apply_sensor_model(sensor_model,e,belief):
    new_belief = {}
    for s in states:
        new_belief[s] = belief[s]*sensor_model(s,e)
    return new_belief.normalize()
```

Exact filtering example (1)

Transition model
80% intended, 20% perp.

Prior
Uniform, 25% each

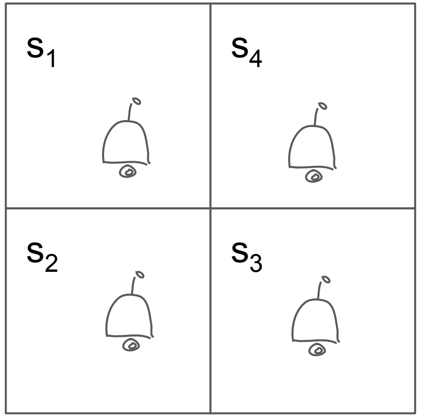
Wall sensor
Detects presence or absence of wall with 90% accuracy:
$p(E=\text{'LU'} \mid X=s_1) = 0.9*0.9*0.9*0.9$
$p(E=\text{'D'} \mid X=s_4) = 0.9*0.1*0.1*0.1$



Exact filtering example (2)

Initialize belief

	belief(s_i)
s_1	0.25
s_2	0.25
s_3	0.25
s_4	0.25



Exact filtering (3)

Apply transition model

$$\text{belief}(X_t = s_i) = \alpha \cdot \sum_h p(X_t = s_i \mid X_{t-1} = h)\text{belief}(X_{t-1} = h)$$

Took action: RIGHT. Apply transition model

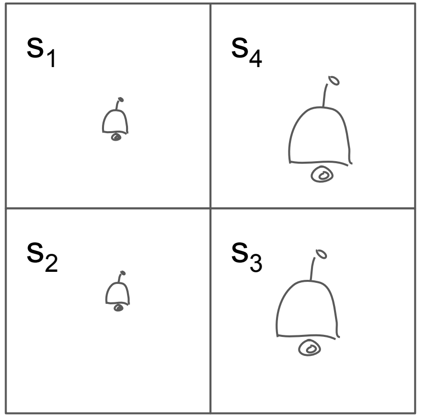
$b(s_1) = p(s_1|s_1)b(s_1) + p(s_1|s_2)b(s_2) + p(s_1|s_3)b(s_3)$
$\quad + p(s_1|s_4)b(s_4)$
$\quad = (0.1)*(0.25)+(0.1)*(0.25)+0+0 = 0.05$

$b(s_2) = p(s_2|s_1)b(s_1) + p(s_2|s_2)b(s_2) + p(s_2|s_3)b(s_3)$
$\quad + p(s_2|s_4)b(s_4) = 0.05$

$b(s_3) = p(s_3|s_1)b(s_1) + p(s_3|s_2)b(s_2) + p(s_3|s_3)b(s_3)$
$\quad + p(s_3|s_4)b(s_4)$
$\quad = 0+(0.8)*(0.25)+(0.9)*(0.25) + (0.1)*(0.25)$
$\quad = 0.45$

$b(s_4) = 0.45$

	belief(s_i)
s_1	0.05
s_2	0.05
s_3	0.45
s_4	0.45



Exact filtering (4)

Apply sensor model

$$\text{belief}(X_t = s_i) = \alpha \cdot p(E_t = e_t \mid X_t = s_i) \cdot \text{belief}(X_t = s_i)$$

Received sensor reading “U” (one north wall).
Apply sensor model

$b(s_1) = \alpha\ p(\text{“U”}|s_1)*b(s_1) = (.9^3*.1)*(0.05)=.0036$

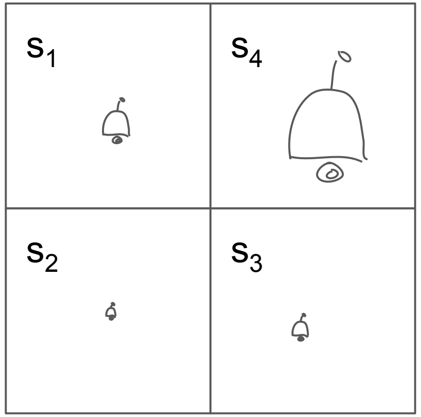
$b(s_2) = \alpha\ p(\text{“U”}|s_2)*b(s_2) = (.9*.1^3)*(0.05)=4.5\text{x}10^{-5}$

$b(s_3) = \alpha\ p(\text{“U”}|s_3)*b(s_3) = (.9*.1^3)*(0.45)=4\text{x}10^{-4}$

$b(s_4) = \alpha\ p(\text{“U”}|s_4)*b(s_4) = (.9^3*.1)*(0.45)=.0328$

$\alpha = 1/0.0368$

	belief(s_i)
s_1	0.0978
s_2	0.0012
s_3	0.0107
s_4	0.8913



Exact filtering notes

Exact filtering can be expensive!

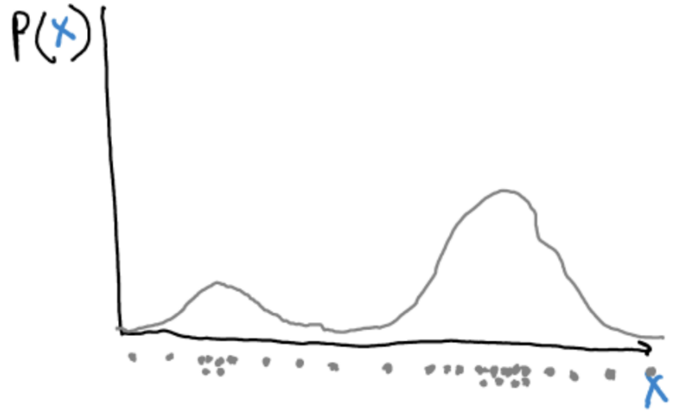
$$\text{belief}_t(X_t = s_i) = \alpha \cdot p(e_t \mid X_t = s_i) \boxed{\sum_h p(X_t = s_i \mid X_{t-1} = h) \cdot \text{belief}_{t-1}(X_{t-1} = h)}$$

What if our state space is continuous? Sums become integrals!

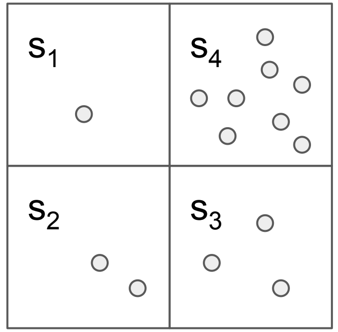
It would be nice if we could come up with an approximation where we could trade off speed and accuracy with a single parameter

A new approach: **Particle Filter**

Estimating distributions with samples



$p(X_t = s_i)$ = # samples in s_i / # samples



Complexity and accuracy are proportional to the number of samples we use!

We need two things to do this:

A way to apply the **transition model** to particles (samples)

A way to apply the **sensor model** to particles

Particle filter algorithm

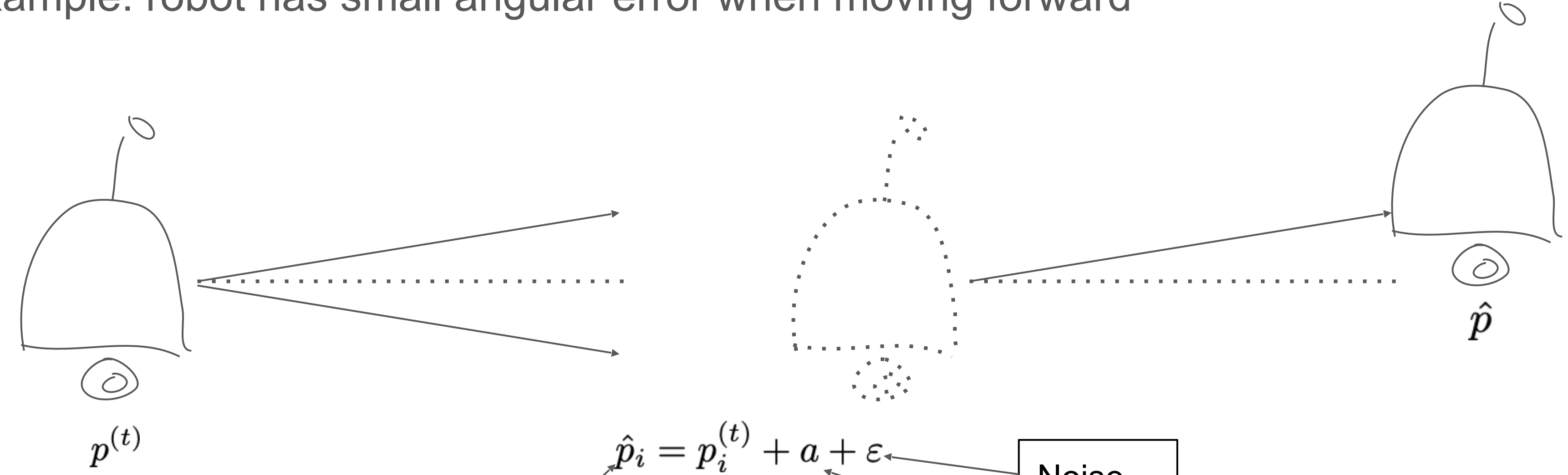
Notation:
x ~ p(X) : "Sample x from the distribution of X"

```
Initialize N particles from prior    $p^{(0)} = \{p_1^{(0)}, p_2^{(0)}, \ldots, p_N^{(0)}\}, \quad p_i^{(0)} \sim p(X_0)$
For each timestep t
        For each particle $p_i$
                Sample $p_i$ from transition model
                    $\hat{p}_i \sim p\left(X_t \mid X_{t-1} = p_i^{(t-1)}\right)$
                Re-weight $p_i$ according to sensor model
                    $w_i = p\left(e_t \mid X_t = \hat{p}_i\right)$
        Re-sample particles according to weight
            $p_i^{(t)} \sim p(X_t \mid w), \quad p(X_t = \hat{p}_i \mid w) = \frac{w_i}{\sum_j w_j}$
```

Sampling from transition model

Easy, even for continuous distributions! Just “simulate” a transition:

Example: robot has small angular error when moving forward

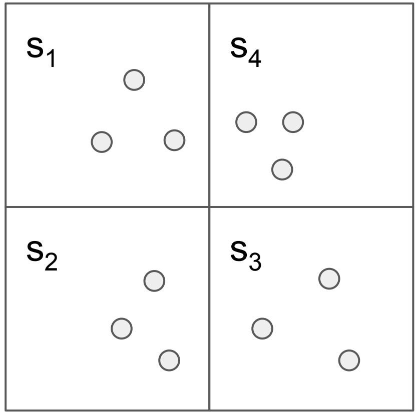


$$\underbrace{\hat{p}_i}_{\text{New particle}} = \underbrace{p_i^{(t)}}_{\text{Original}} + \underbrace{a}_{\text{Action taken}} + \underbrace{\varepsilon}_{\text{Noise}}$$

Particle Filter example (1)

Initialize

- Number of particles: 12
- Prior probability: uniform
- $p^{(0)} = [s_1,s_1,s_1,s_2,s_2,s_2,s_3,s_3,s_3,s_4,s_4,s_4]$

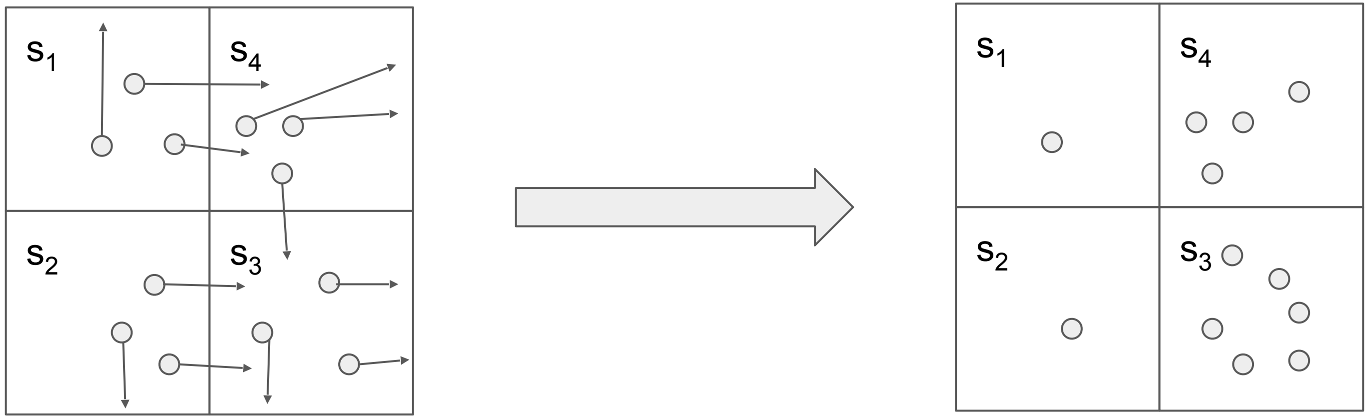


Part	1	2	3	4	5	6	7	8	9	10	11	12
State	s_1	s_1	s_1	s_2	s_2	s_2	s_3	s_3	s_3	s_4	s_4	s_4
Weight	.083	.083	.083	.083	.083	.083	.083	.083	.083	.083	.083	.083

Transition model:
80% go in intended direction, 20% perpendicular

Particle Filter example (2)

Apply transition model: Took action "RIGHT"



Part	1	2	3	4	5	6	7	8	9	10	11	12
State	s_1	**s_4**	**s_4**	s_2	**s_3**	**s_3**	s_3	s_3	s_3	s_4	s_4	**s_3**
Weight	.083	.083	.083	.083	.083	.083	.083	.083	.083	.083	.083	.083

Sensor model:
$p(e|s_i) = (0.9)$^(# of correct walls) * (0.1)^(# of incorrect walls)

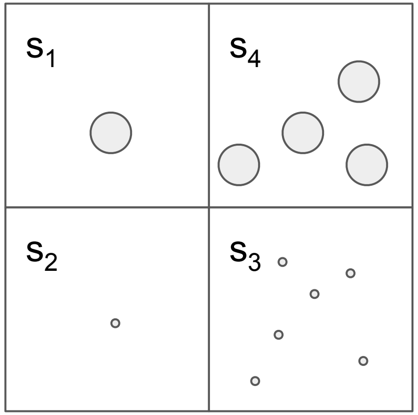
Particle Filter example (3)

Apply sensor model

Received sensor reading: “One wall to the north”

$p(e|s_1) = p(e|s_4) = 0.9^3 * 0.1 = 0.073$

$p(e|s_2) = p(e|s_3) = 0.1^3 * 0.9 = 0.0009$



Part	1	2	3	4	5	6	7	8	9	10	11	12
State	s_1	s_4	s_4	s_2	s_3	s_3	s_3	s_3	s_3	s_4	s_4	s_3
Weight	**.073**	**.073**	**.073**	**$9e^{-4}$**	**$9e^{-4}$**	**$9e^{-4}$**	**$9e^{-4}$**	**$9e^{-4}$**	**$9e^{-4}$**	**.073**	**.073**	**$9e^{-4}$**

Particle Filter example (4)

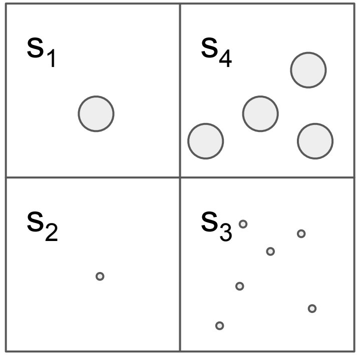
Resample according to weights

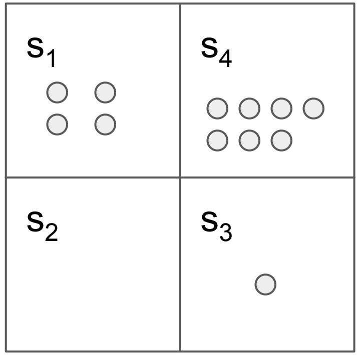
Normalize: α = 2.7…

Compute Cumulative Distribution Function (CDF):

CDF[i] = $w_1+w_2+\ldots+w_i$

```
For number of particles to generate:
    Pick random number p between 0 and 1
    Find first bin in CDF such that p<CDF[i]
    Copy particle i into new set of particles
```





Part	1	2	3	4	5	6	7	8	9	10	11	12
State	s_1	s_1	s_1	s_1	s_3	s_4	s_4	s_4	s_4	s_4	s_4	s_4
Weight	.083	.083	.083	.083	.083	.083	.083	.083	.083	.083	.083	.083

Particle Filter notes

Can break into two steps like exact filtering and have multiple sensor readings per move, or multiple moves between sensor readings

- Apply transition model when actions occur
- Apply sensor model when sensors arrive

Edge cases

- All particles in one state
- All particles have weight 0

Solution? Re-initialize, or “inject” random particles at each timestep

Summary and preview

Wrapping up

- We can use Bayes nets to think about how state random variables are related **through time**, and to answer questions about the current state (**filtering**) and future states (**prediction**)
- **Exact Filtering**: iteratively update a **belief** vector/array using **transition model** and **sensor model**
- **Particle Filtering**: iteratively update a **set of particles** as an estimate of belief

Up next: Smoothing and Viterbi