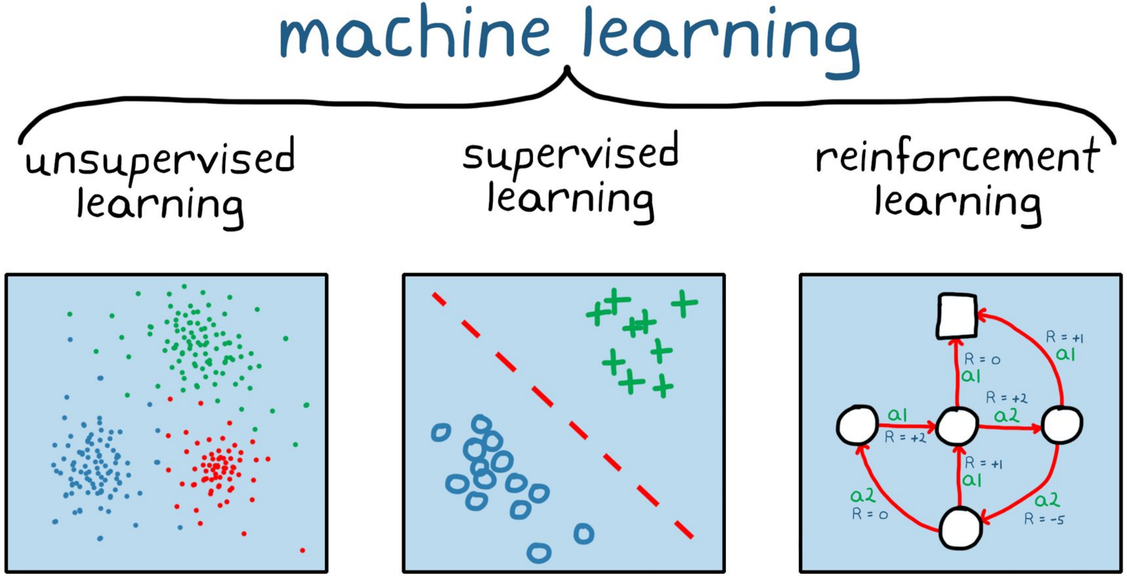
Principles of Reinforcement Learning: A Speedrun

Tarun Amarnath

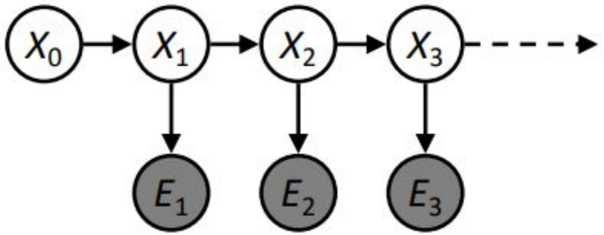


Note: Many examples derived from CS 188 content

Part 1:
Hidden Markov Models

Hidden Markov Models

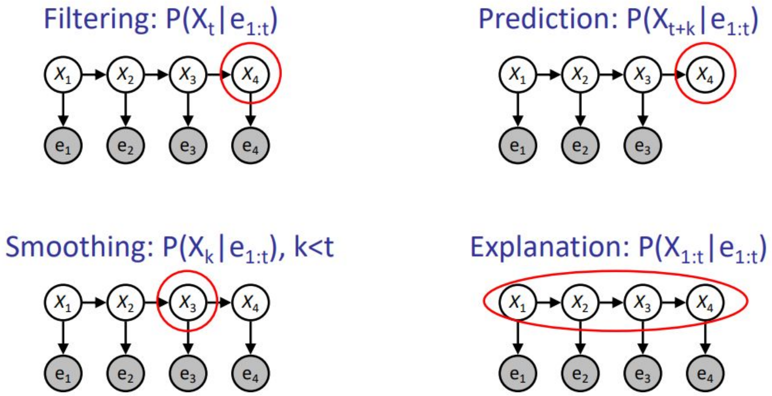
- Similar to what we covered with SLAM (but in a discrete space)
- Underlying Markov chain over states
- We can only observe evidence variable at each time step



- Example of robot localization:
  - We know where the robot's been
  - We can read sensor data
  - Can we figure out a probability distribution for where we are now?

What We Have

- Given in an HMM:
  - Initial distribution P(X_0)
  - Transition model P(X_t | X_(t-1))
  - Sensor model P(E_t | X_t)
- We can do a ton of things with this!

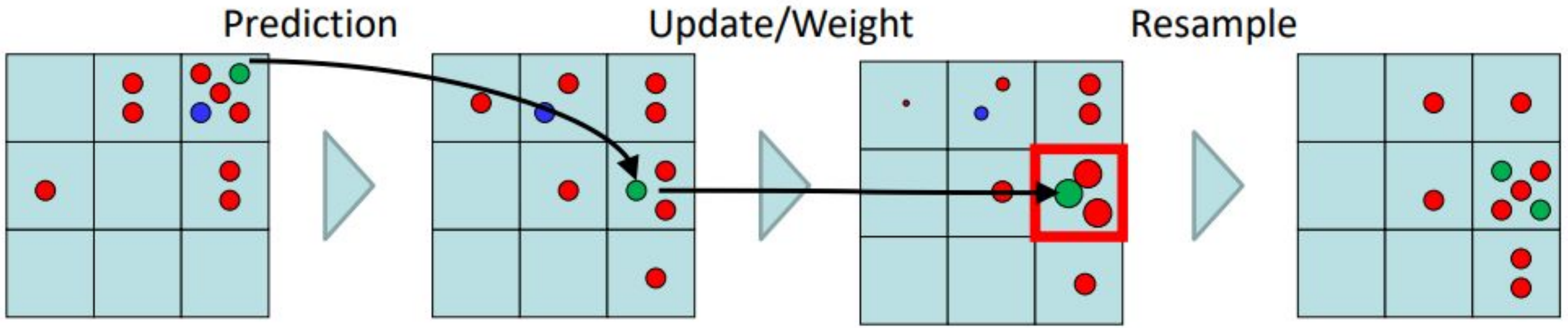


- The process for filtering has a 1) prediction and 2) update step, just as we saw in SLAM

Aside: Particle Filtering

- Solving HMMs might be computationally expensive
  - Could have too many states and transition probabilities
  - Unknown starting location
  - Ex. massive maze where the robot could be
- Instead use particle filtering
  1) Assign n particles to positions randomly
  2) Elapse time based on their transition probabilities
  3) Find updated distribution of state based on positions of particles
  4) Resample particles (reassign locations)

Particle Filtering Process

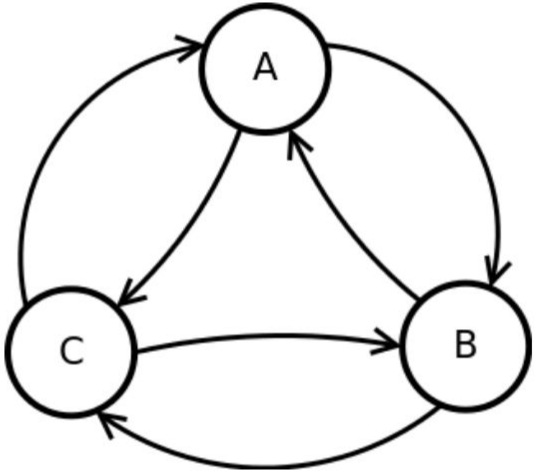


Part 2:
Markov Decision Processes

Markov Decision Processes

- Set of **states**
- Set of **actions**
  - Each with some probability of going to another state (**transition model**)
- **Reward** for each transition
- **Start state**
- **Terminal state(s)**
- **Discount factor** to prefer sooner rewards

- **Question:** What action should we take from each state?



Discount Factor, $\gamma = 0.5$

s	a	s'	T(s,a,s')	R(s,a,s')
A	Clockwise	B	1.0	0.0
A	Counterclockwise	C	1.0	-2.0
B	Clockwise	A	0.4	-1.0
B	Clockwise	C	0.6	2.0
B	Counterclockwise	A	0.6	2.0
B	Counterclockwise	C	0.4	-1.0
C	Clockwise	A	0.6	2.0
C	Clockwise	B	0.4	2.0
C	Counterclockwise	A	0.4	2.0
C	Counterclockwise	B	0.6	0.0

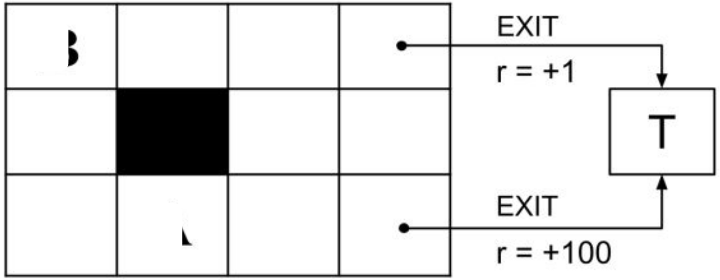
Value Iteration

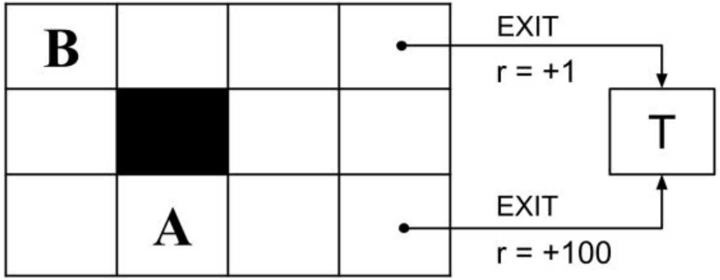
- A technique to find the best possible value we can get from a state
- Perform a **Bellman update** to get the utility that can be derived from starting at a particular node
- Find the **action that maximizes expected utility**, and **then set the value** of the node to that utility

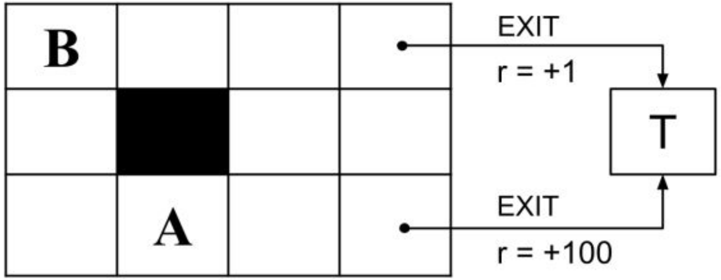
$$U_{k+1}(s) \leftarrow \max_a \sum_{s'} P(s' \mid a,s) \left[R(s,a,s') + \gamma U_k(s')\right]$$

- Do this for many iterations until convergence

Quick Value Iteration Example







Source: CS 188 Exam Prep 12

Q Values and Q Iteration

- A **value** is the best expected reward you can get if you start at a particular state **(maximized over all actions)**
- A **Q-value** is the maximum expected reward if you start at a state and perform a *particular* action = Q(s, a)
  - **This allows us to get the max expected rewards for each action we can perform at a state**
- Bellman equation looks very similar:

$$
\begin{aligned}
Q(s,a) &= \sum_{s'} P(s' \mid s,a)[R(s,a,s') + \gamma U(s')] \\
&= \sum_{s'} P(s' \mid s,a)[R(s,a,s') + \gamma \max_{a'} Q(s',a')]
\end{aligned}
$$

Policies and Policy Iteration

- **Policy:** what action should you take at a particular state?
- **Policy extraction** (from values or Q-values):

$$\pi_U(s) = \text{argmax}_a \sum_{s'} P(s' \mid a,s)\,[R(s,a,s') + \gamma \boldsymbol{U}(s')\,]$$

- **Policy iteration:**
  - Perform value iteration, but also **keep track of which action** you select
  - If the **actions selected don't change**, then the policy has **converged** (you know what you should do at some state, but not necessarily what reward it'll get you)
- **Policy evaluation:** follow through on the policy from some state and **see what reward** it gets you

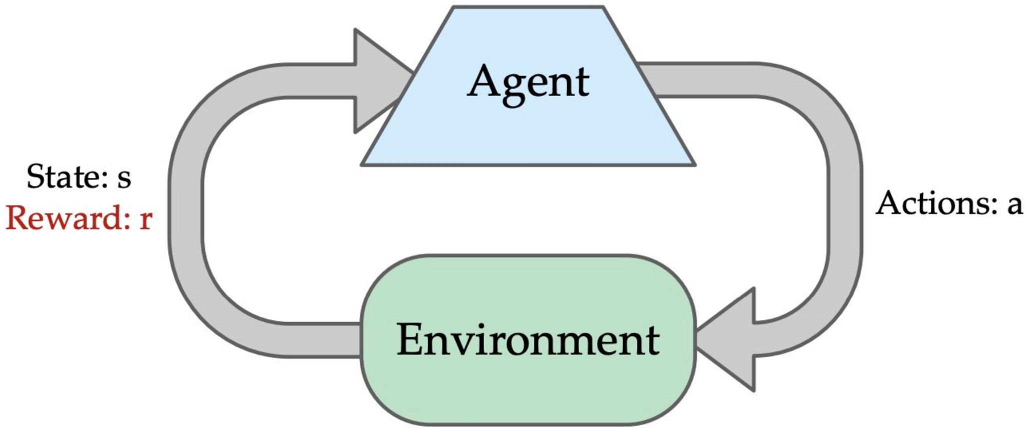
Part 3:
Reinforcement Learning

Reinforcement Learning

- Huge idea in CS/robotics right now!
- In Markov Decision Processes, we had states, actions, transitions, and rewards
- We can think of Reinforcement Learning as a similar framework, but we **don’t know transition probabilities or rewards**
- So we just try stuff to learn what works!
- And then we reinforce the good behaviors!

The Framework

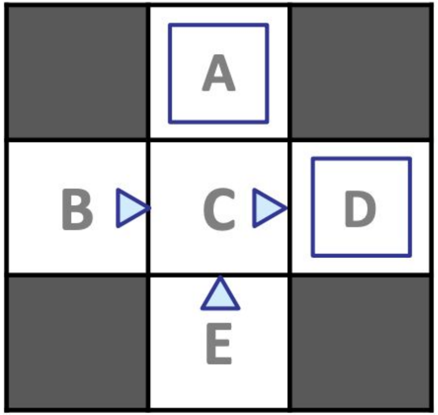
- Perform an action and get some reward for it
  - Sample → observe result
- Learn to maximize expected rewards



Model-Based RL

- Figure out the resulting state (s’) you reach from a given state and action (s, a)
- Find transition probabilities for each (s, a, s’)
- Find rewards for each (s, a, s’)
- Learn the model, and then solve this MDP

Input Policy π



Assume: $\gamma = 1$

Observed Episodes (Training)

Episode 1

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 2

B, east, C, -1
C, east, D, -1
D, exit, x, +10

Episode 3

E, north, C, -1
C, east, D, -1
D, exit, x, +10

Episode 4

E, north, C, -1
C, east, A, -1
A, exit, x, -10

Learned Model

$\hat{T}(s, a, s')$

T(B, east, C) = 1.00
T(C, east, D) = 0.75
T(C, east, A) = 0.25
...

$\hat{R}(s, a, s')$

R(B, east, C) = -1
R(C, east, D) = -1
R(D, exit, x) = +10
...

Example taken from CS 188 Fa21 Lecture 10

Model-Free RL

- Model-based has the limitation of having to construct the MDP
  - Need to figure out what the resulting state is for every action
- Instead, we can learn the control **policy** directly without having to determine which state we will reach
  - Decide what to do from a particular state to get the highest reward

Temporal Difference Q-Learning

- Start at a state
- Take some action
- Get some reward

$$sample = R(s, a, s') + \gamma \max_{a'} Q(s', a')$$

- Update Q-value for (s, a)
- Keep a running average

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + (\alpha)\,[sample]$$

- Converges to optimal policy! As long as you keep going for long enough

Part 4:
Generalizing to Continuous Spaces

State Space Representation

- With discrete spaces, it’s easy to represent our states
- Real-world robots, however, have many more variables
- Instead use state vectors that we are used to

- **Question:** How can we do Q-learning if we don’t have a consistent (s, a, s’) because of uncertainties in the real world/continuous spaces?
  - Example: Action to move forward for 2 seconds
  - End up at slightly different final states s’

Deep Reinforcement Learning

- Neural networks can help solve this problem!
- Learn an optimal policy for each state
- Inputs:
  - Current state
  - Some kind of desired movement
- Output:
  - Optimal control policy to achieve desired movement (maximize reward)

- Different RL approaches have different ways of including the action

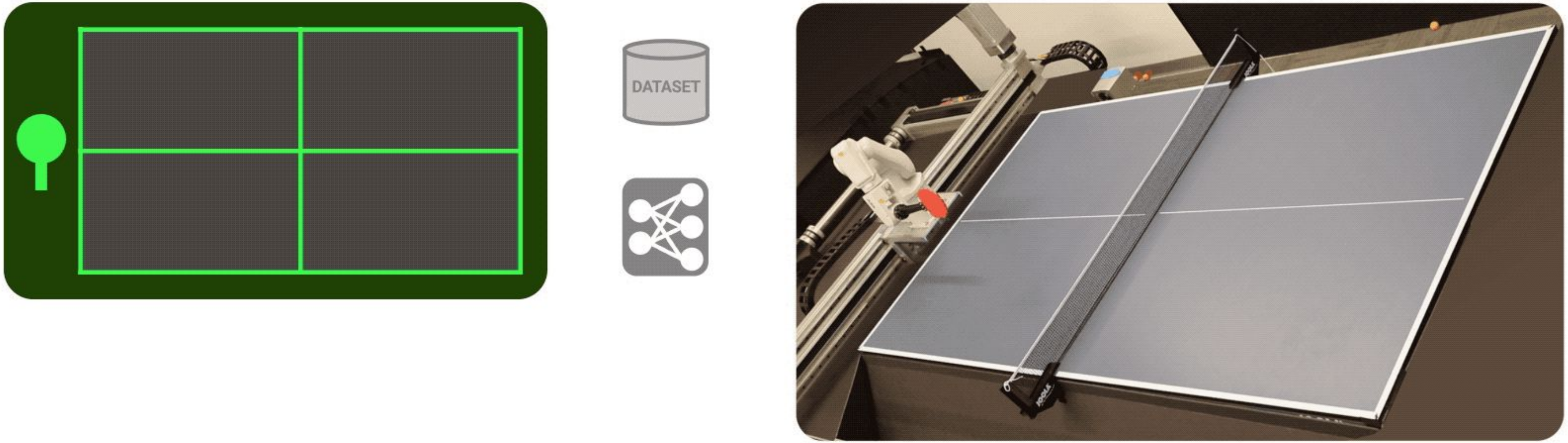
Exploration vs. Exploitation

- We want to both explore different policies and fine-tune them so they can be used effectively
- Explore a new action or choose randomly with some probability (exploration) or follow optimal policy (exploitation)
- Probability of exploration reduces as training time goes on

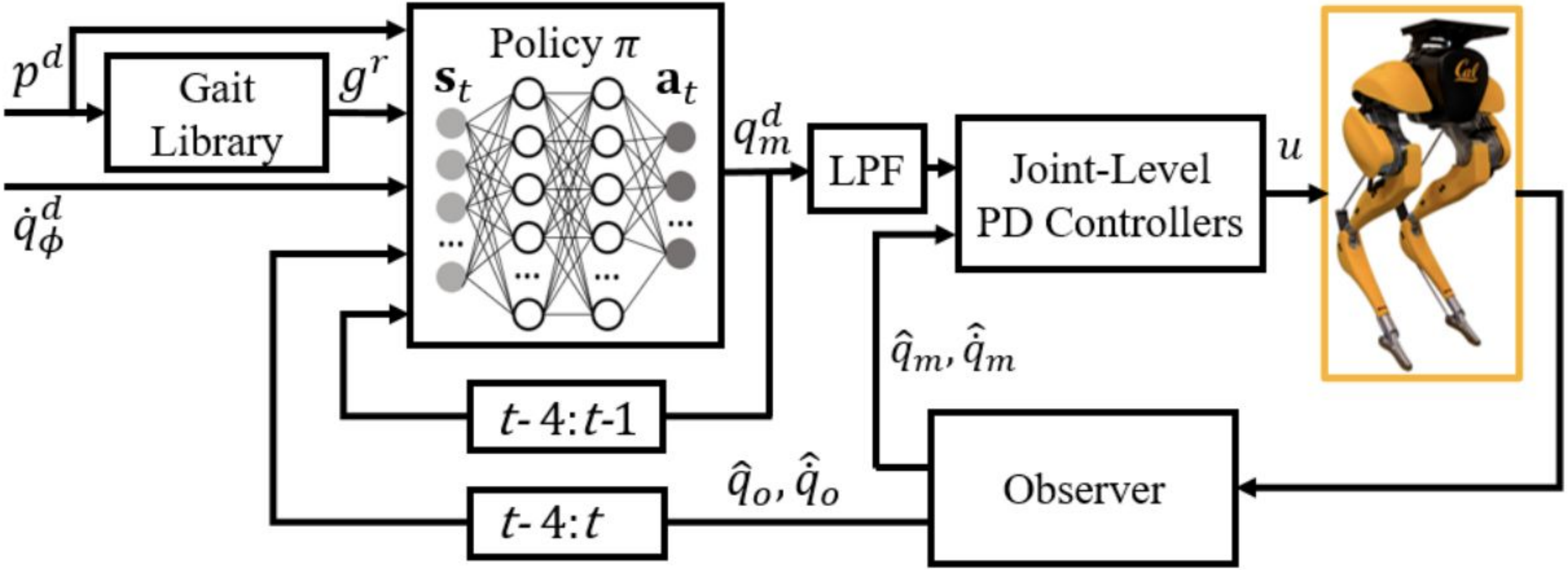
Ping-Pong Learning

- An example of RL used by Google researchers to create a robot that plays table tennis effectively
- RL can be sample-inefficient - doesn't use data as well as it could
- Google technique combines Learning from Play and Goal-Conditioned Supervised Learning to build database and policy simultaneously

Singh et. al., 2022


DATASET

Gait Library to Inform Controller



Li et. al., 2021