COL864: Special Topics in AI
Semester II, 2021-22

Reinforcement Learning

Rohan Paul

Outline

- Last Class
  - Markov Decision Processes
- This Class
  - Basics of Reinforcement Learning
  - Model-based Reinforcement Learning
- Reference Material
  - Please follow the notes as the primary reference on this topic.

Acknowledgements

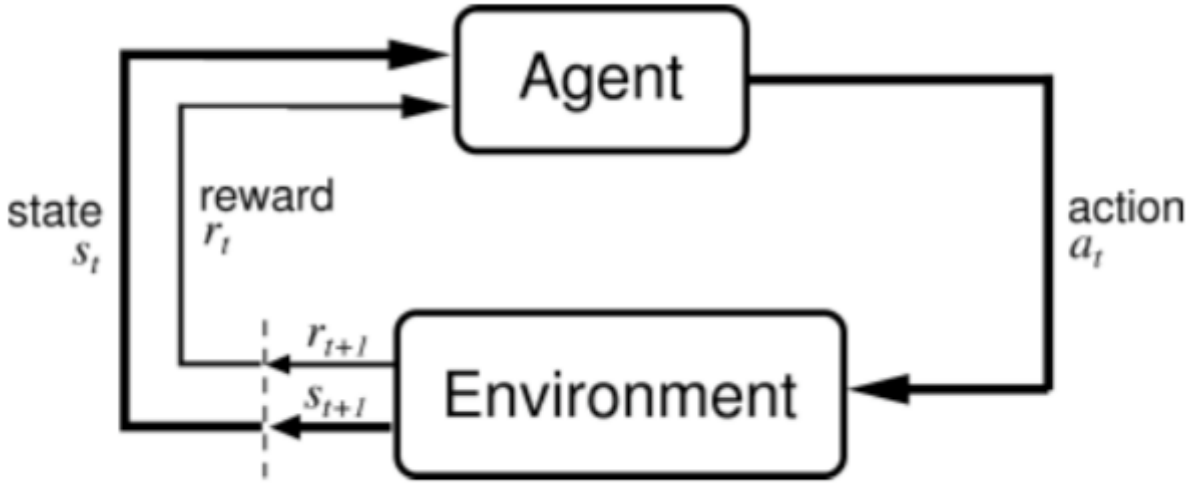
These slides are intended for teaching purposes only. Some material has been used/adapted from web sources and from slides by Nicholas Roy, Wolfram Burgard, Dieter Fox, Sebastian Thrun, Siddharth Srinivasa, Dan Klein, Pieter Abbeel, Max Likhachev and others.

Learning to Act from Data

- So far we assumed to have an a-priori model of the domain
  - MDP: Tuple of states, actions, transition function, rewards, start state and discount factor
  - Safe to assume that some parts of the model are constant.
    - State and action sets are fixed and are given by the domain.
- In practice
  - Commonly, we don’t know the
    - Transition function (*if I take an action which state I will land up in?*)
    - Reward function (*when I take a transition is it good or bad?)*
  - Experience
    - We can observe transitions and rewards as a function of actions
    - Can we use this data to learn something from this experience?

Learning to Act from Data

- The agent experiences the environment and receives a reward and observes the consequence of the action.
- Does not have the full rewards function or the transition model ahead of time.
- Needs to determine how to act?
- Note: there is evaluative feedback for actions not prescriptive feedback.



$$\pi^* = \arg\max_{\pi} \mathrm{E}[\sum_{t=0}^{H} \gamma^t R_t(S_t, A_t, S_{t+1})|\pi]$$

Reinforcement Learning in relation to other types of Learning

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn function to map $x \rightarrow y$

Apple example:

This thing is an apple.

Unsupervised Learning

Data: x
x is data, no labels!

Goal: Learn underlying structure

Apple example:

This thing is like the other thing.

Reinforcement Learning

Data: state-action pairs

Goal: Maximize future rewards over many time steps

Apple example:

Eat this thing because it will keep you alive.

Figure: Alexander Amini

Biological Motivation

Children learning to walk.

Examples: Learning to Walk

Initial

[Kohl and Stone, ICRA 2004]

Examples: Learning to Walk

Training

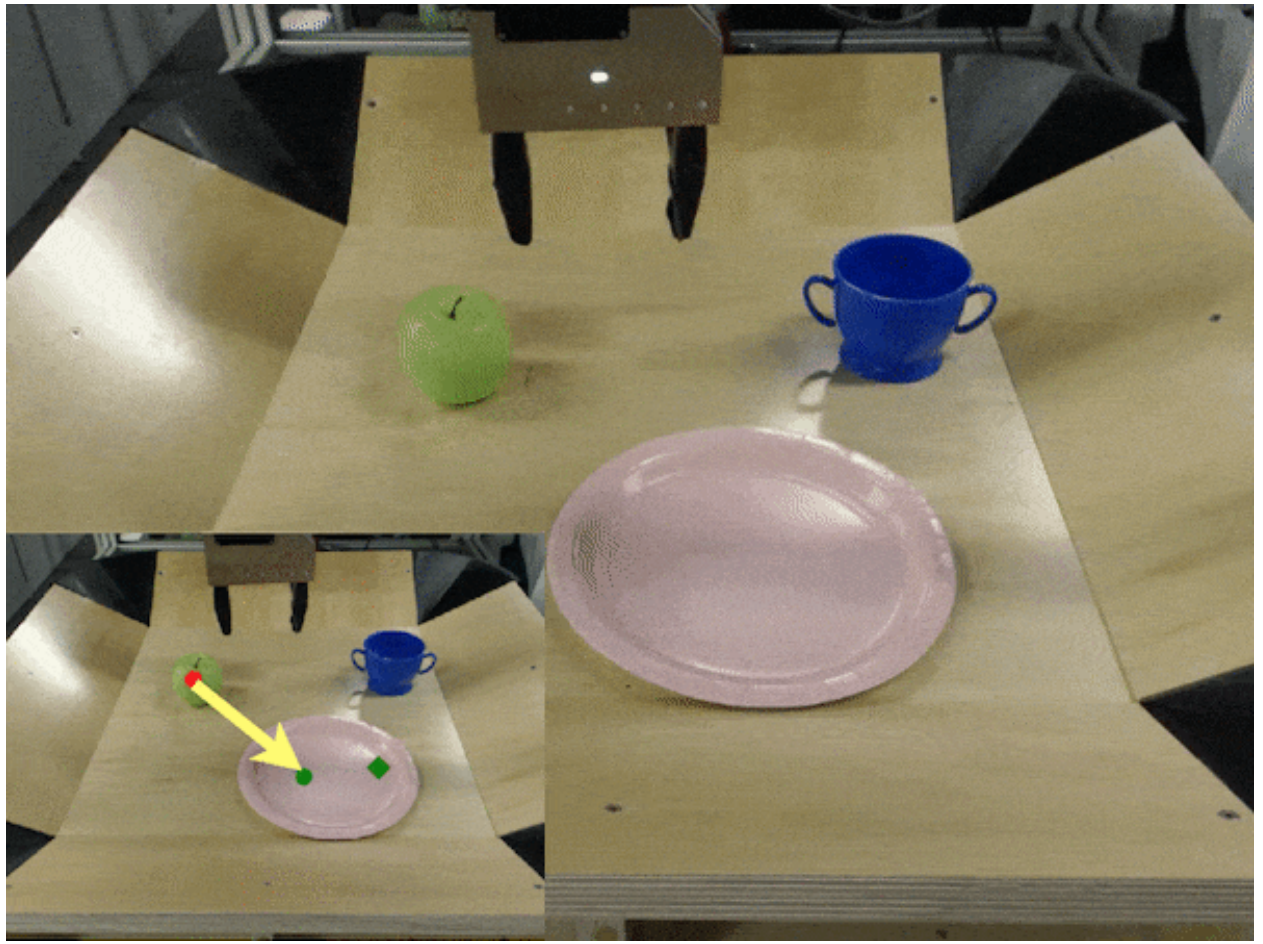
[Kohl and Stone, ICRA 2004]

Examples: Learning to Walk

Finished

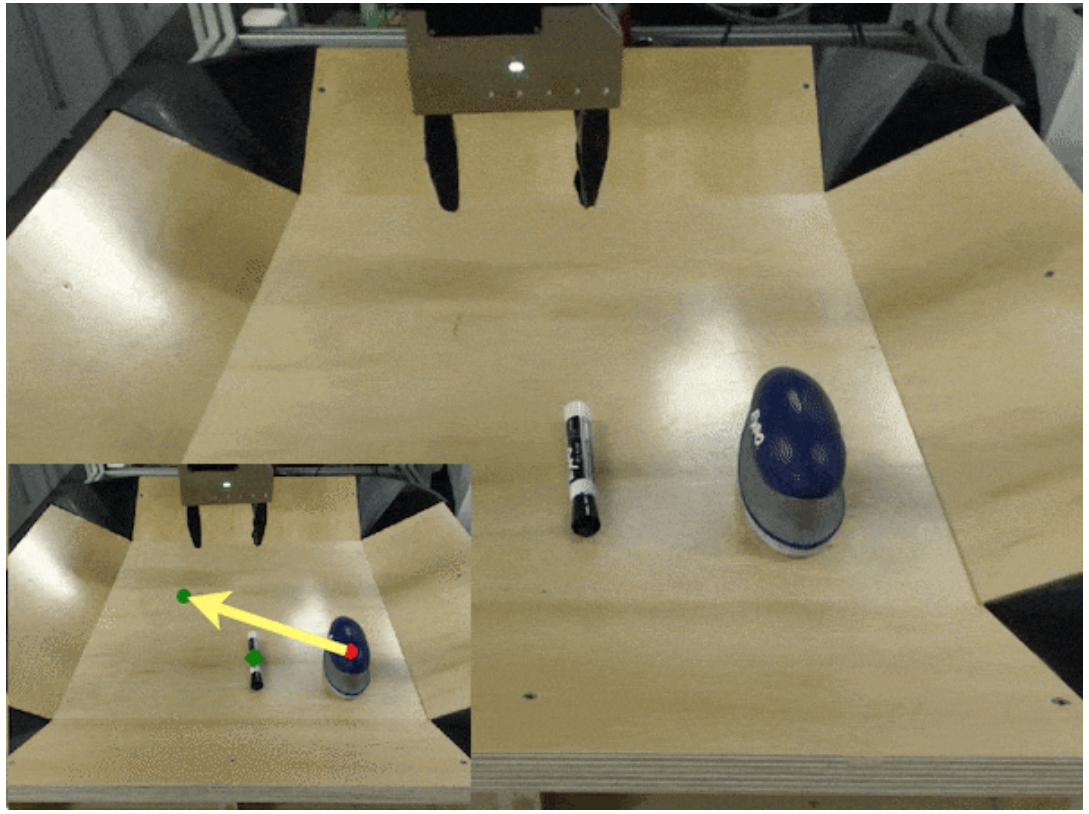
[Kohl and Stone, ICRA 2004]

Examples: Learning to Manipulate



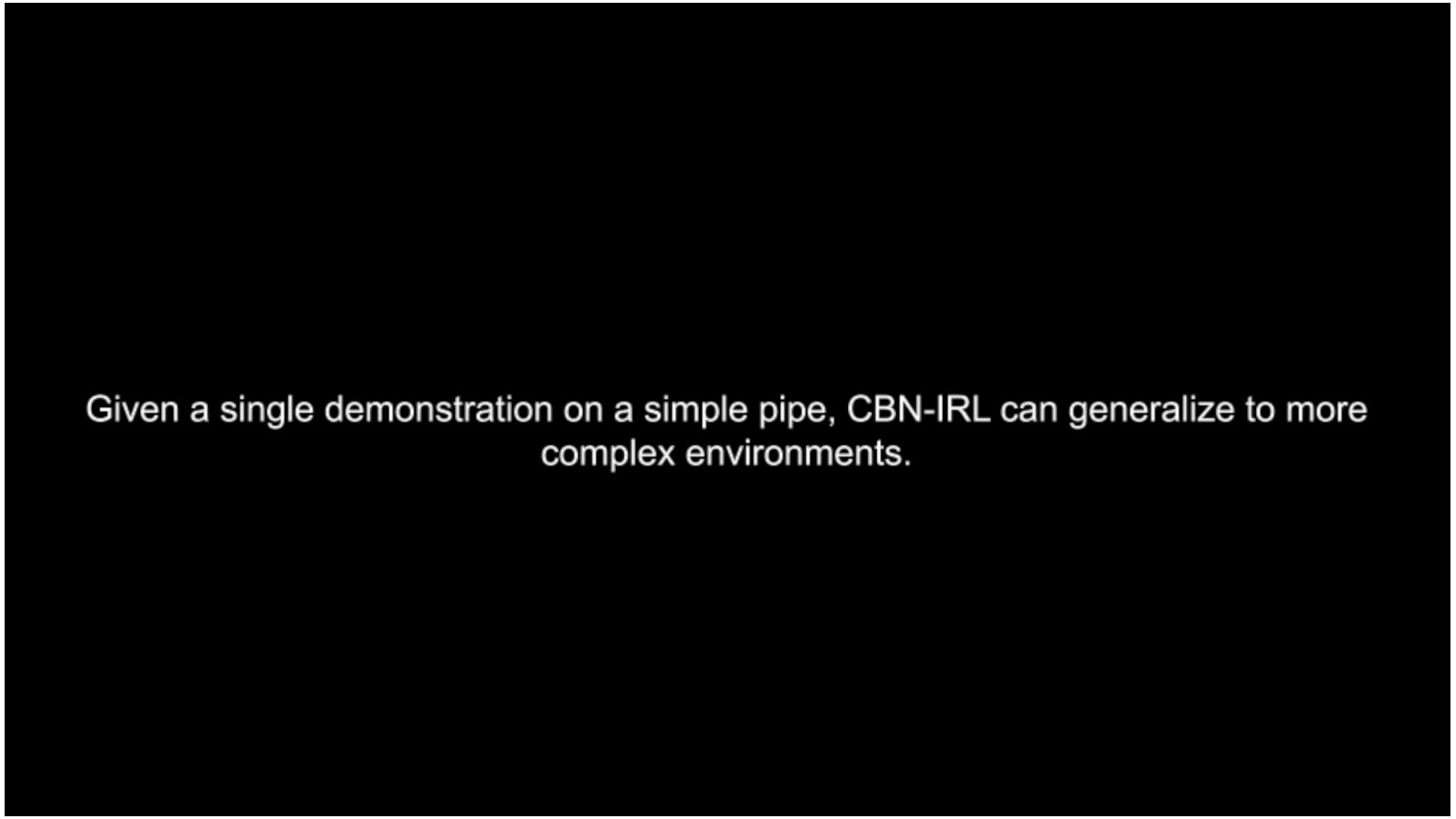
Chelsea Finn
http://bair.berkeley.edu/blog/2018/11/30/visual-rl/

Examples: Learning to Manipulate



Chelsea Finn et al.
http://bair.berkeley.edu/blog/2018/11/30/visual-rl/

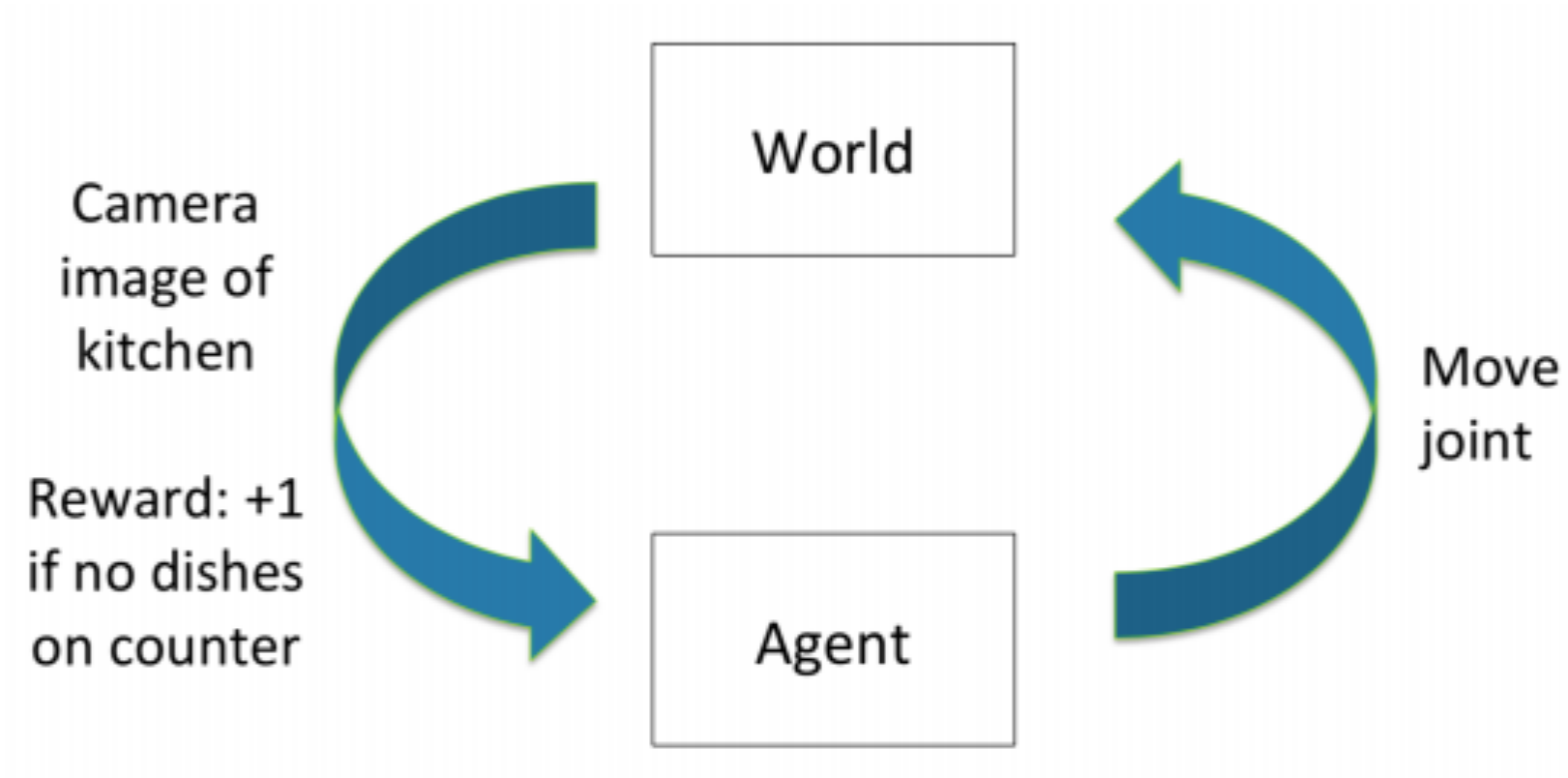
Examples: Learning to Perform Complex Skills



Inverse RL a form of reinforcement learning

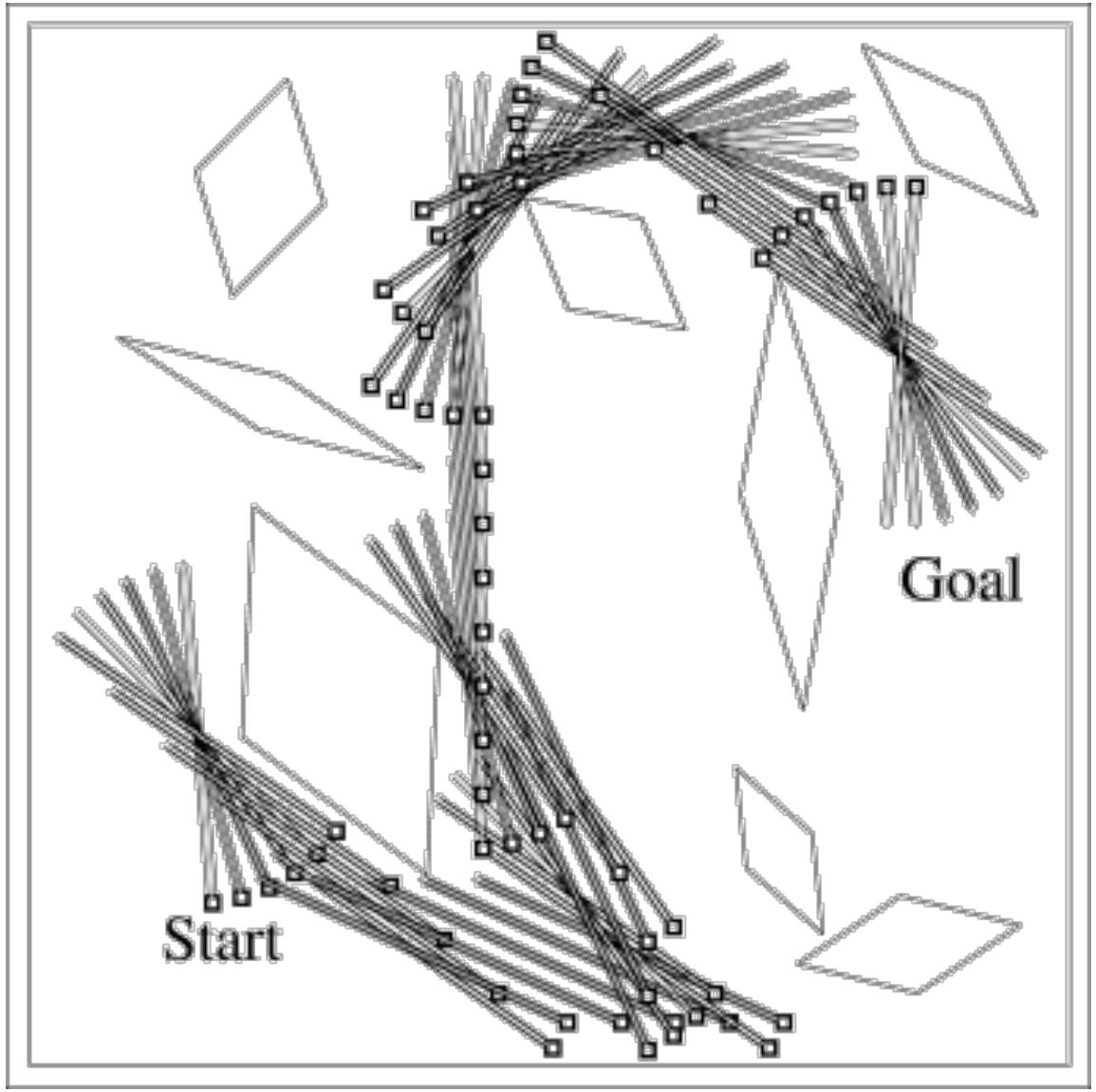
Park, Noseworthy, Paul and Roy. CoRL 2019.

Example


World
Camera image of kitchen
Reward: +1 if no dishes on counter
Agent
Move joint

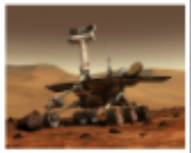
Example

Rod maneuvering from an initial to a goal state solved as an RL task.

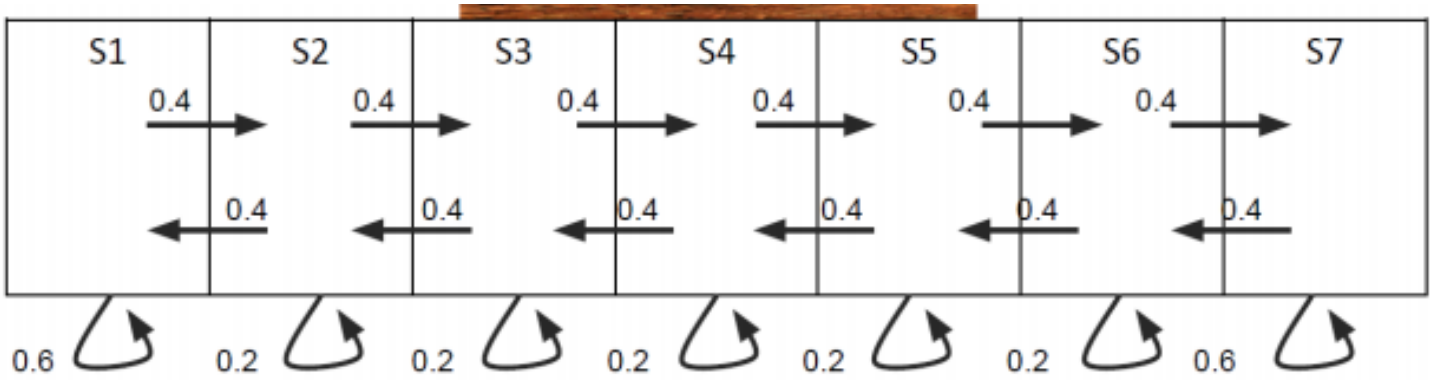


MDP Example: Mars Rover

States and rewards
(encode the agent's goal)

S1	S2	S3	S4	S5	S6	S7
Okay Field Site R=+1	R=0	R=0	 R=0	R=0	R=0	Fantastic Field Site R=+10

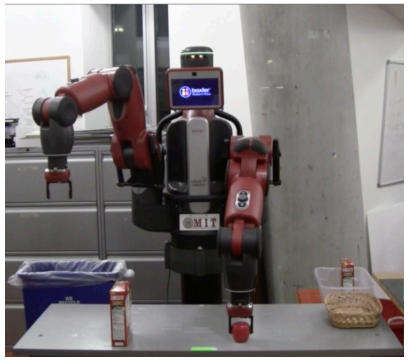
Transition function.



The agent experiences rewards and transitions while acting in the world. It is not provided a-priori.

Characteristics

- Goal is to find an optimal way to act.
- Delayed Reward
- Credit Assignment Problem
  - Which of the actions actually helped in getting to this state?
- Exploration vs. Exploitation
  - Should I *exploit* what I know and find a good policy w.r.t. this knowledge?
    - Risk of missing out on a better reward somewhere else
  - Should I *explore* and look for states with more reward?
    - Risk of wasting time & getting some negative reward

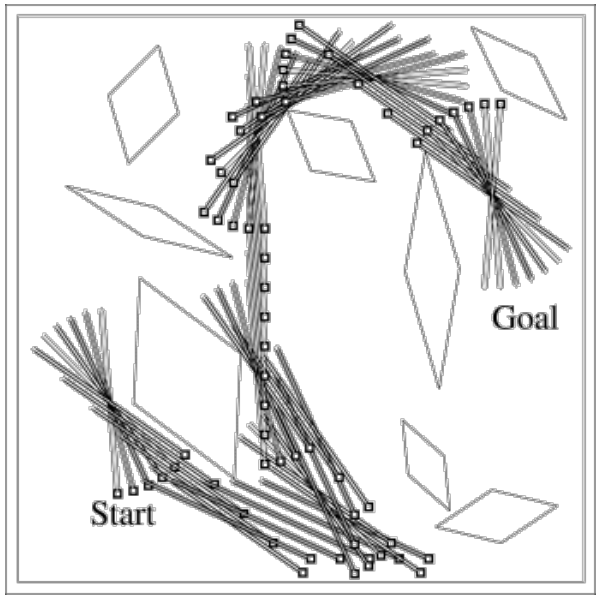


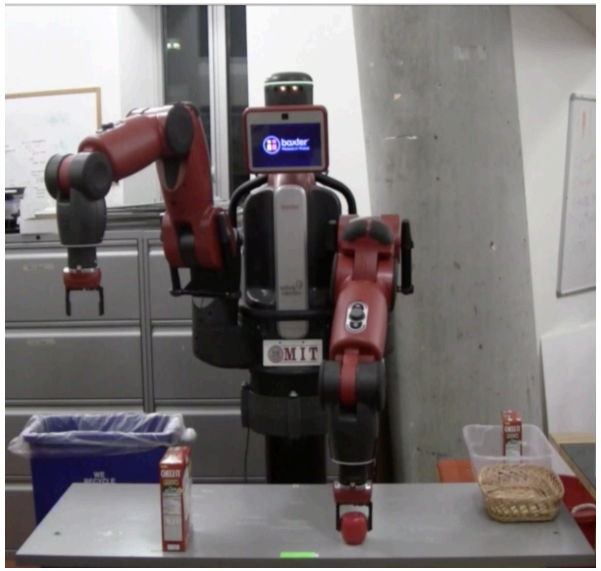
An agent trying to grasp and place at a location.

I'm in state 43,	reward = 0,	action = 2
" " " 39,	" = 0,	" = 4
" " " 22,	" = 0,	" = 1
" " " 21,	" = 0,	" = 1
" " " 21,	" = 0,	" = 1
" " " 13,	" = 0,	" = 2
" " " 54,	" = 0,	" = 2
" " " 26,	" = 100,	

RL Characteristics: Optimization and Delayed Rewards

- Goal is to find an optimal way to make decisions
- Explicit notion of utility of decisions





An agent trying to grasp and place at a location.

The “Credit Assignment” Problem

I’m in state 43, reward = 0, action = 2

I’m in state 43, reward = 0, action = 2
“ “ “ 39, “ = 0, “ = 4

I’m in state 43, reward = 0, action = 2
“ “ “ 39, “ = 0, “ = 4
“ “ “ 22, “ = 0, “ = 1

I’m in state 43, reward = 0, action = 2
“ “ “ 39, “ = 0, “ = 4
“ “ “ 22, “ = 0, “ = 1
“ “ “ 21, “ = 0, “ = 1
“ “ “ 21, “ = 0, “ = 1
“ “ “ 13, “ = 0, “ = 2
“ “ “ 54, “ = 0, “ = 2

The ”Credit Assignment” Problem

Which of the actions actually helped in getting to this state?

I'm in state 43,	reward = 0,	action = 2
" " " 39,	" = 0,	" = 4
" " " 22,	" = 0,	" = 1
" " " 21,	" = 0,	" = 1
" " " 21,	" = 0,	" = 1
" " " 13,	" = 0,	" = 2
" " " 54,	" = 0,	" = 2
" " " 26,	" = 100,	

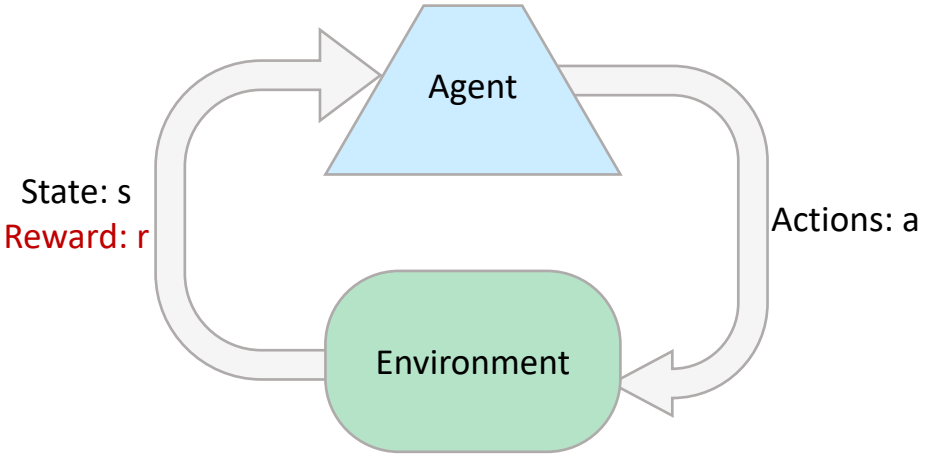
RL Characteristics Exploration vs. Exploitation

- **Example:**
  - You have visited part of the state space and found a reward of +100
  - *Question: Is this the best that the agent can get in the environment?*
- **Exploitation**
  - Should I *exploit* what I know and find a good policy w.r.t. this knowledge?
  - Risk of missing out on a better reward somewhere else
- **Exploration**
  - Should I *explore* and look for states with more reward?
  - Risk of wasting time & getting some negative reward

Learning to Act from Data

- What can we learn from experience?
  - Could learn the **model** parameters and compute the value function
  - Could learn the long-term (possibly discounted) expected return or **value** of each state action pair Q(s, a) function and extract the optimal policy from this.
  - Could learn a **policy** (map from states directly to actions) that will max. the long-term expected return.

Lead to different RL algorithms



Model-Based Reinforcement Learning

- Model-Based Idea
  - Learn an approximate model (R(), T()) based on experiences
  - Compute the value function using the learned model (as if it were correct).

- Learn an empirical MDP model
  - Count outcomes s' for each s, a
  - Normalize to give an estimate of $\hat{T}(s, a, s')$
  - Discover each $\hat{R}(s, a, s')$ when we experience (s, a, s')

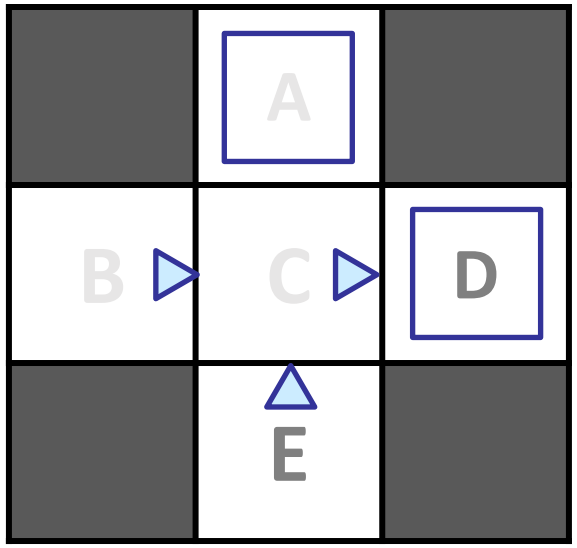
- Solve the learned MDP
  - For example, use value iteration, to obtain the final policy.
  - Plug in the estimated T and R in the following equation:

$$\pi^*(s) = \arg\max_a \sum_{s'} \hat{T}(s, a, s')[\hat{R}(s, a, s') + \gamma V^*(s')]$$

Note: going through an intermediate stage of learning the model. In contrast, "**model-free**" approaches do not learn the intermediate model.

Model-Based Learning: Example

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
…

Certainty Equivalence

- Assume some prior policy π.
- Follow π and record state transitions (s_t to s_{t+1}, etc.), and reward $r(s_t, a_t, s_{t+1})$ at each time step.
- Estimate model parameters, e.g.,

$$p(s^k|s^i, a^j) = \frac{n(s^i, a^j, s^k))}{\sum_s n(s^i, a^j, s)} \quad (1)$$

$$R(s^i, a^j, s^k) = \frac{\sum_t r(s^i = s_t, a^j = a_t, s^k = s_{t+1})}{n(s^i, a^j, s^k)} \quad (2)$$

where $n(s^i, s^j, a^k)$ is the number of times in state from s^i to s^k based after taking action a^j.

Given a good coverage of states and actions, the the estimated model can be believed to be correct and can solve for the optimal policy.

Balanced Wandering

- Problem
  - Note: we needed a policy to generate the data
  - What if the policy is really bad and does not explore the space well?
    - Could end up never seeing many states of transitions
- Exploration
  - Good exploration policies are critical to learning a good policy.
    - Need to ensure that we have seen all states and transitions often enough.
- Balanced Wandering
  - Assuming we are in state s
    - If s is a state we have never seen before, take any action with uniform probability.
    - If have been at s before then take the action that has been tried the fewest times from s (breaking ties randomly).

- Assume some prior policy π.
- Follow π and record state transitions (s_t to s_{t+1}, etc.), and reward $r(s_t, a_t, s_{t+1})$ at each time step.
- Estimate model parameters, e.g.,

$$p(s^k|s^i, a^j) = \frac{n(s^i, a^j, s^k))}{\sum_s n(s^i, a^j, s)} \quad (1)$$

$$R(s^i, a^j, s^k) = \frac{\sum_t r(s^i = s_t, a^j = a_t, s^k = s_{t+1})}{n(s^i, a^j, s^k)} \quad (2)$$

where $n(s^i, s^j, a^k)$ is the number of times in state from s^i to s^k based after taking action a^j.

Use Balance Wandering to collect enough data to know the model parameters with confidence.

Simulation Lemma: Relation between model approx. and value function approx.

- If we're going to compute a policy from a model, we have to make sure we can guarantee we "know" the model
- The *Simulation Lemma* says that if an approximate model $\hat{M}$ is an $O((\epsilon/(NTG_{max}^T))^2)$-approximation to a model M, then the error in the resulting approximate value function is at most ϵ
  - An α-approximation to a model means that the error in the approximate reward function $R(\cdot,\cdot)$ is at most α and the error in any part of the transition function $p(\cdot|\cdot,\cdot)$ is at most α.
  - N is the number of states in the MDP
  - G_{max}^T is the maximum possible T-step return
  - $G_{max}^T \leq TR_{max}$ in the discounted case.
- The Simulation Lemma allows us to determine how many visits to a state s are required to "know" its transition distribution $p(\cdot|s,a)$ and reward $R(s,\cdot,\cdot)$ well enough to ensure that any error in the computed policy that results from an error in the model of s is bounded by ϵ, with probability $1-\delta$

Near-optimal Reinforcement Learning in Polynomial Time " [Kearns and Singh].

Advanced material from a paper. The basic ideas are covered in class. Students are expected to know only the high-level ideas. The full technical details reside in the paper.

States become eventually known during balanced wandering

- A state is "known" if it has been visited $m_{known} = O(((NTG^T_{max})/\epsilon)^4 Var_{max}(\log(1/\delta))$ times
- After $N(m_{known} - 1) + 1$ steps of balanced wandering, by the Pigeonhole Principle, some state guaranteed to become known.
  - Is the worst case, in terms of the time required for at least one state to become known.
  - Need to pick ϵ, δ, T, need to know N, G^T_{max}, Var_{max} which is the maximum variance in the reward function.
- Once we have visited each state sufficiently often, can compute a policy and know that it is ϵ correct with probability $1 - \delta$.
- Once the state is known then it does not participate in balanced wandering (better to focus on unexplored parts of the state space).

Near-optimal Reinforcement Learning in Polynomial Time " [Kearns and Singh]

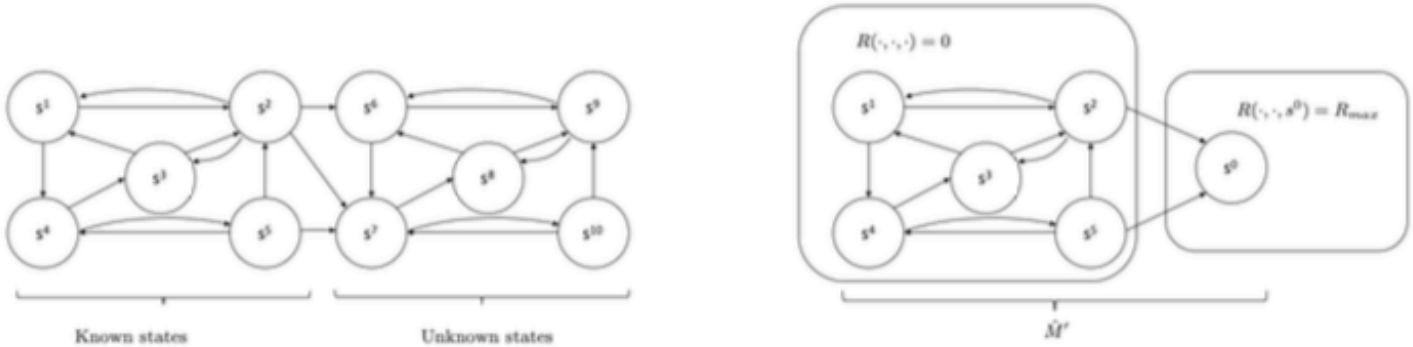
Explicit Explore or Exploit (E^3) Algorithm

- A Model-Based RL technique.
- Provides guarantee on the quality of the model and the policy that is acquired.
- Makes use of the Simulation Lemma in its analysis.
- Key Idea: Exploration Policy
  - Uses an exploration policy that drives the agent to unknown states as quickly as possible.
  - The exploration policy is “optimistic” in the face of uncertainty.

Near-optimal Reinforcement Learning in Polynomial Time ” [Kearns and Singh]

E^3 Algorithm

- Construct an MDP $\hat{M}'_S$ consisting of the known states and add an extra abstract state s_0 that represents all the unknown states.
- The algorithm starts with no states being in the set of known states. As the states are explored in balanced wandering and become "known" they get added to the set of known states.
  - Set $R(s, a, s') = 0$ for all states except $R(s, a, s_0) = R_{max}$
  - Set $p(s_0|s, a) = 1 - \sum_{s'} p(s'|s, a)$
- Solve for the policy $\hat{\pi}'$ for T steps in $\hat{M}'$
  - The output policy π' is our exploration policy
  - The exploration policy is designed to drive the agent to unknown states as quickly as possible, and guaranteed to have probability at least $\epsilon/(2G^T_{max})$ of leaving the set S.
- A good exploration policy is to be "optimistic" in the face of uncertainty (the algorithm thinks that there is a high reward in the unknown states).



E^3 Algorithm

- Algorithm:
  1. (Initialization) the set S of known states as empty.
  2. (Balanced Wandering) Perform balanced wandering, keeping track of number of counts of each state transition we experience, and keeping track of the reward we get from each state.
  3. (Discovery of New Known States) Any time a state s has been visited m_{known} times during balanced wandering, it becomes known. The state that becomes known enters known set S, gets incorporated in $\hat{M}'_S$, and no longer participates in balanced wandering.
  4. (Determine and Use a New Exploration Policy) Recompute exploration policy $\hat{\pi}'$. Follow $\hat{\pi}'$ until encountering an unknown state. Anytime the current state is not in S the algorithm resumes balanced wandering.
  5. (Balanced Wandering) Any time an attempted exploitation or attempted exploration visits a state not in S that is a new state, the algorithm immediately resumes balanced wandering.

Takeaways

- RL is bout learning to identify a good (ideally optimal) policy from interacting with the environment (receiving immediate rewards) via experienced state transitions.
- Good exploration policies are critical to learning a good policy.
  - A good exploration policy is to be optimistic in the face of uncertainty.
- Model-based methods attempt to first learn a model and then use the model to determine the policy.
  - A relation can be established between the quality of the model approximation and how good the resulting policy is.
- E^3 is a Model-based RL algorithm.