Passive Reinforcement Learning

Pradipta Biswas

Associate Professor

Indian Institute of Science

Pradipta@iisc.ac.in, https://cambum.net/PB/

Reinforcement Learning

- Learning to interact with an environment
  - Robots, games, process control
  - With limited human training
  - Where the 'right thing' isn't obvious

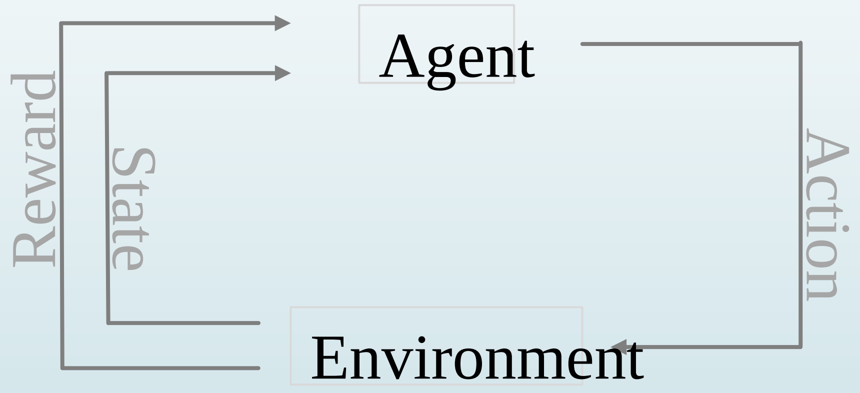
- Supervised Learning:
  - Goal: $f(x) = y$
  - Data: $[<x_1, y_1>, \ldots, <x_n, y_n>]$

- Reinforcement Learning:
  - Goal:

    Maximize $\sum_{i=1}^{\infty} Reward(State_i, Action_i)$

  - Data:

    $Reward_i, State_{i+1} = Interact(State_i, Action_i)$



Supervised vs. Unsupervised Learning

- Supervised learning (classification)
  - Supervision: The training data (observations, measurements, etc.) are accompanied by labels indicating the class of the observations
  - New data is classified based on the training set
- Unsupervised learning (clustering)
  - The class labels of training data is unknown
  - Given a set of measurements, observations, etc. with the aim of establishing the existence of classes or clusters in the data

How Reinforcement Learning is Different

- Delayed Reward
- Agent chooses training data
- Explore vs Exploit (Life long learning)
- Very different terminology (can be confusing)
  - Reward
  - Utility
  - Policy
  - State-Action Mapping

Reinforcement Learning

- Reinforcement learning:
  - Still assume an MDP:
    - A set of states $s \in S$
    - A set of actions (per state) A
    - A model T(s,a,s')
    - A reward function R(s,a,s')
    - A discount factor γ (could be 1)
  - Still looking for a policy $\pi(s)$

  - New twist: don't know T or R
    - i.e. don't know which states are good or what the actions do
    - Must actually try actions and states out to learn

Reinforcement Learning

M = 0.8 in direction you want to go

0.2 in perpendicular < 0.1 left, 0.1 right

Policy: mapping from states to actions

An optimal policy for the stochastic environment:

3	→	→	→	+1
2	↑		↑	-1
1	↑	←	←	←
	1	2	3	4

utilities of states:

3	0.812	0.868	0.912	+1
2	0.762		0.660	-1
1	0.705	0.655	0.611	0.388
	1	2	3	4

Environment < Observable (accessible): percept identifies the state; Partially observable

Markov property: Transition probabilities depend on state only, not on the path to the state.

Markov decision problem (MDP).

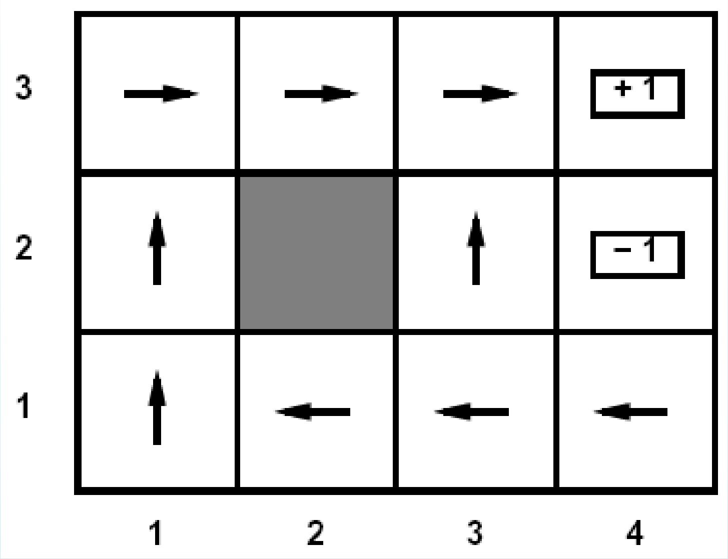
Partially observable MDP (POMDP): percepts does not have enough info to identify transition probabilities.

Passive vs. Active learning

- Passive learning
  - The agent has a fixed policy and tries to learn the utilities of states by observing the world go by
  - Analogous to policy evaluation
  - Often serves as a component of active learning algorithms
  - Often inspires active learning algorithms
- Active learning
  - The agent attempts to find an optimal (or at least good) policy by acting in the world
  - Analogous to solving the underlying MDP, but without first being given the MDP model

Passive Learning

- Simplified task
  - You don't know the transitions T(s,a,s')
  - You don't know the rewards R(s,a,s')
  - You are given a policy $\pi(s)$
  - Goal: learn the state values
  - … what policy evaluation did



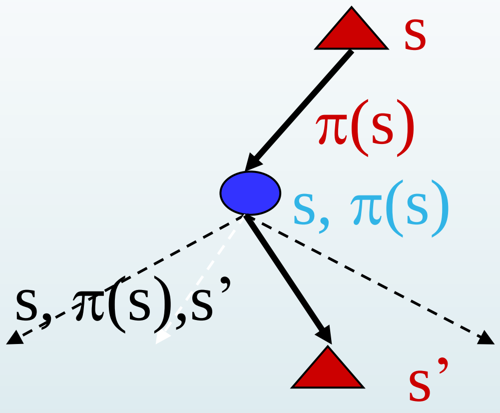
- In this case:
  - Learner "along for the ride"
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - We'll get to the active case soon
  - This is NOT offline planning! You actually take actions in the world and see what happens…

Model-Based vs. Model-Free RL

- *Model based approach to RL:*
  - learn the MDP model, or an approximation of it
  - use it for policy evaluation or to find the optimal policy
- *Model free approach to RL:*
  - derive the optimal policy without explicitly learning the model
  - useful when model is difficult to represent and/or learn

Model-Based Learning

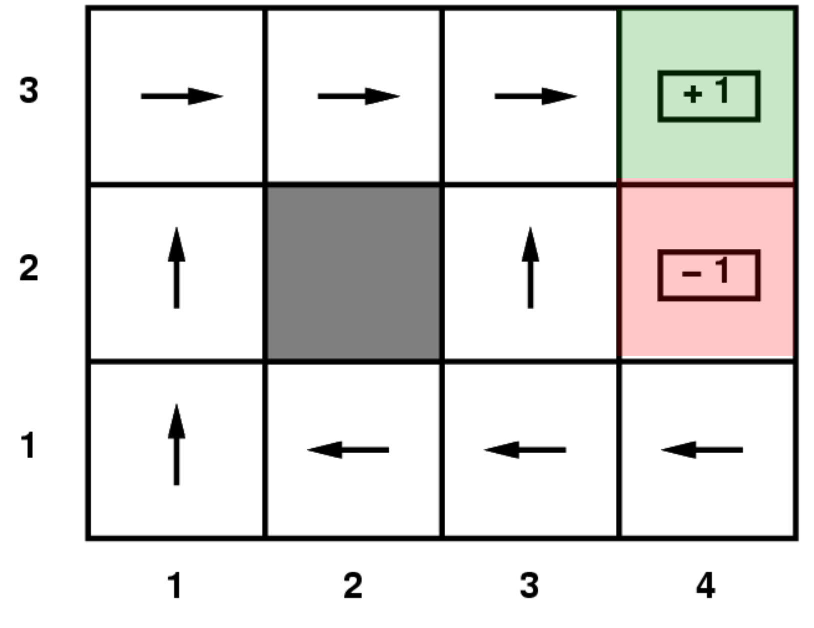
- Idea:
  - Learn the model empirically through experience
  - Solve for values as if the learned model were correct
- Simple empirical model learning
  - Count outcomes for each s,a
  - Normalize to give estimate of **T(s,a,s')**
  - Discover **R(s,a,s')** when we experience (s,a,s')
- Solving the MDP with the learned model
  - Iterative policy evaluation, for example



$$U_{i+1}^{\pi}(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma U_i^{\pi}(s')]$$

Passive RL

- Estimate $U^{\pi}(s)$
- Not given
  - transition matrix, nor
  - reward function!
- Follow the policy for many epochs giving training sequences.



(1,1)→(1,2)→(1,3)→(1,2)→(1,3)→(2,3)→(3,3) →(3,4) +1
(1,1)→(1,2)→(1,3)→(2,3)→(3,3)→(3,2)→(3,3)→(3,4) +1
(1,1)→(2,1)→(3,1)→(3,2)→(4,2) -1

- Assume that after entering +1 or -1 state the agent enters zero reward terminal state
  - So we don't bother showing those transitions

Approach 1: Direct Estimation

- Direct estimation (also called Monte Carlo)
  - Estimate $U^{\pi}(s)$ as average total reward of epochs containing s (calculating from s to end of epoch)
- ***Reward to go*** of a state s

  the sum of the (discounted) rewards from that state until a terminal state is reached
- Key: use observed ***reward to go*** of the state as the direct evidence of the actual expected utility of that state
- Averaging the reward-to-go samples will converge to true value at state
- Convert the problem into supervised learning problem
  - Learn / Estimate utilities from the list of utilities in training sequences
  - Can average utilities for one sequence if the same state appears multiple times

Direct Estimation

- Converge very slowly to correct utilities values (requires more sequences than perhaps necessary)

- Doesn't exploit Bellman constraints on policy values

$$U^{\pi}(s) = R(s) + \beta \sum_{s'} T(s, \pi(s), s')\, U^{\pi}(s')$$

  - It is happy to consider value function estimates that violate this property badly.

How can we incorporate the Bellman constraints?

Approach 2: Adaptive Dynamic Programming (ADP)

- ADP is a model based approach
  - Follow the policy for awhile
  - Estimate transition model based on observations
  - Learn reward function
  - Use estimated model to compute utility of policy

$$U^{\pi}(s) = R(s) + \beta \sum_{s'} T(s,a,s')\, U^{\pi}(s')$$

learned

- How can we estimate transition model T(s,a,s')?
  - **Simply the fraction of times we see s' after taking a in state s.**

Adaptive DP (ADP)

Use the constraints (state transition probabilities) between states to speed learning.

Solve

$$U(i) = R(i) + \sum_j M_{ij} U(j)$$

= value determination.

No maximization over actions because agent is passive unlike in value iteration.

using DP

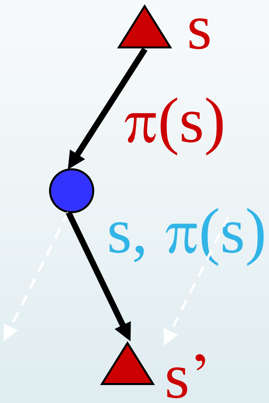
Too many linear equations to solve

One equation for each state →Large state space

e.g. Backgammon: 10^{50} equations in 10^{50} variables

Temporal-Difference Learning

- Use the observed transitions to adjust the values of the observed states so that they agree with the constraint equations.
- Learn from every experience!
  - Update U(s) each time we experience (s,a,s',r)
  - Likely s' will contribute updates more often
- Temporal difference learning
  - Policy still fixed!
  - Move values toward value of whatever successor occurs: running average!



Sample of U(s): $sample = R(s, \pi(s), s') + \gamma U^{\pi}(s')$

Update to U(s): $U^{\pi}(s) \leftarrow (1 - \alpha) U^{\pi}(s) + (\alpha) sample$

Same update: $U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(sample - U^{\pi}(s))$

Temporal Difference (TD) Learning

Do ADP backups on a per move basis, not for the whole state space.

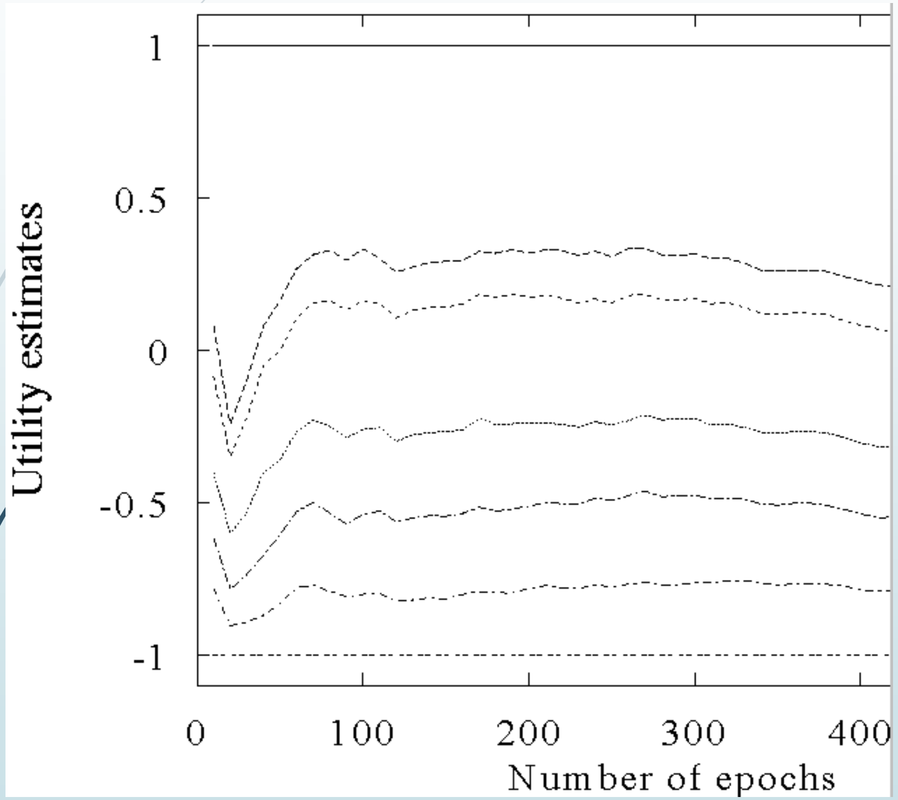
$$U(i) \leftarrow U(i) + \alpha[R(i) + U(j) - U(i)]$$

Thrm: Average value of U(i) converges to the correct value.

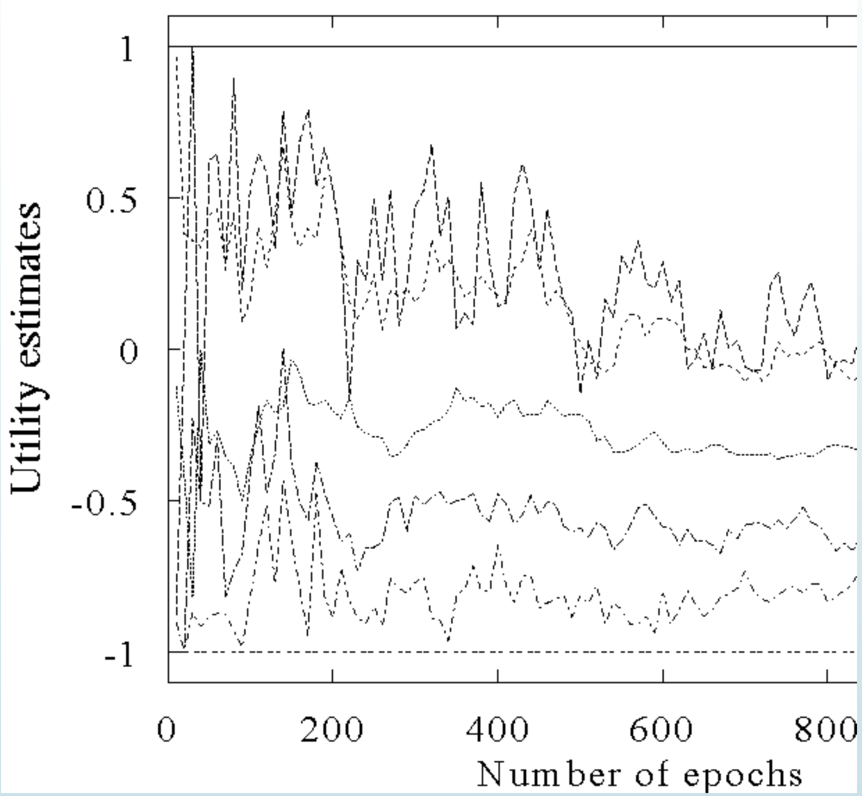
Thrm: If α is appropriately decreased as a function of times a state is visited ($\alpha=\alpha[N[i]]$), then U(i) itself converges to the correct value

Comparing Convergence between ADP and TD

ADP



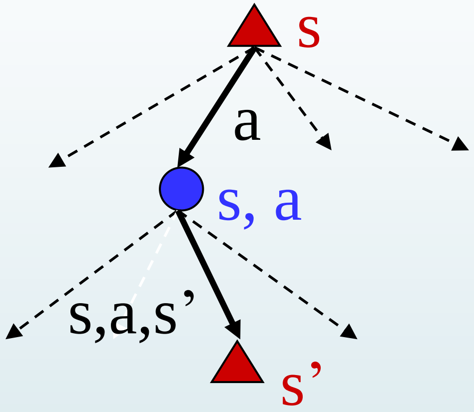
TD



- **Tradeoff:** requires more training experience (epochs) than ADP but much less computation per epoch
- Choice depends on relative cost of experience vs. computation

Problems with TD Value Learning

- TD value leaning is a model-free way to do policy evaluation
- We want to turn values into a (new) policy



$$\pi(s) = \arg\max_a Q^*(s,a)$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma \mathrm{U}^*(s') \right]$$

- Idea: learn Q-values directly
- Makes action selection model-free too!

Passive RL: Comparisons

- Monte-Carlo Direct Estimation (model free)
  - Simple to implement
  - Each update is fast
  - Does not exploit Bellman constraints
  - Converges slowly
- Adaptive Dynamic Programming (model based)
  - Harder to implement
  - Each update is a full policy evaluation (expensive)
  - Fully exploits Bellman constraints
  - Fast convergence (in terms of updates)
- Temporal Difference Learning (model free)
  - Update speed and implementation similiar to direct estimation
  - Partially exploits Bellman constraints---adjusts state to ‘agree’ with observed successor
    - Not ***all*** possible successors as in ADP
  - Convergence in between direct estimation and ADP