



Introduction to Reinforcement Learning

Pradipta Biswas
Associate Professor
Indian Institute of Science

pradipta@iisc.ac.in, <https://cambum.net/PB/>

Introduction

Passive Reinforcement Learning

- Direct Utility Estimation
- Adaptive Dynamic Programming
- Temporal Difference Learning
- Comparison

Active Reinforcement Learning

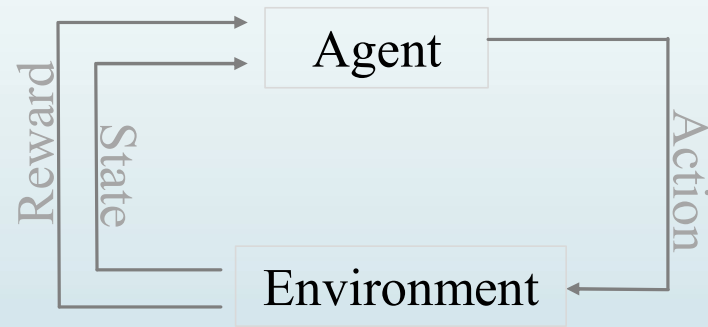
- Q-Learning
- Multi-arm Bandit Problem

Advanced Reinforcement Learning

- Deep Q-Learning
- Inverse Reinforcement Learning
- Demonstrations

Introduction

- ▶ Learning to interact with an environment
 - ▶ Robots, games, process control
 - ▶ With limited human training
 - ▶ Where the 'right thing' is not obvious
- ▶ Supervised Learning:
 - ▶ Goal: $f(x) = y$
 - ▶ Data: [$\langle x_1, y_1 \rangle, \dots, \langle x_n, y_n \rangle$]
- ▶ Reinforcement Learning:
 - ▶ Goal:
 $\text{Maximize } \sum_{i=1}^{\infty} \text{Reward}(\text{State}_i, \text{Action}_i)$
 - ▶ Data:
 $\text{Reward}_i, \text{State}_{i+1} = \text{Interact}(\text{State}_i, \text{Action}_i)$



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

5

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

$R(s)$

Reinforcement Learning – Set Up

$M = 0.8$ in direction you want to go
 0.2 in perpendicular $\left\{ \begin{array}{l} 0.1 \text{ left} \\ 0.1 \text{ right} \end{array} \right.$
 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 $\left\{ \begin{array}{l} \text{Observable (accessible): percept identifies the state} \\ \text{Partially observable} \end{array} \right.$

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.

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

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

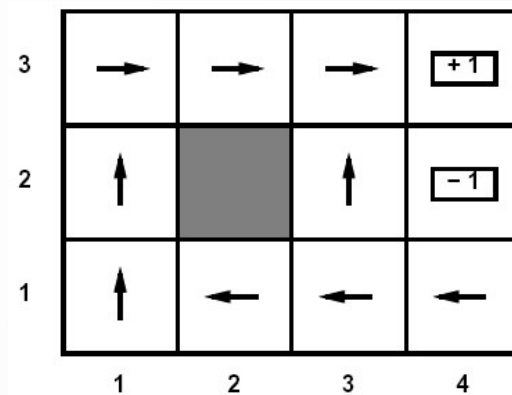


Passive Reinforcement Learning

Passive Learning

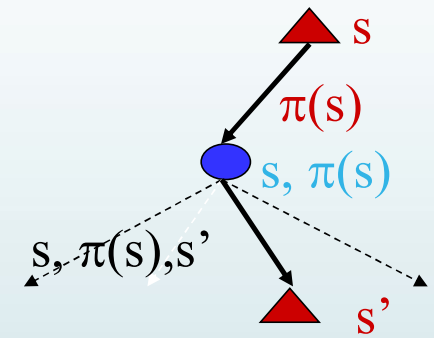
11

- 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
 - This is NOT offline planning! You actually take actions in the world and see what happens...



Model-Based Learning

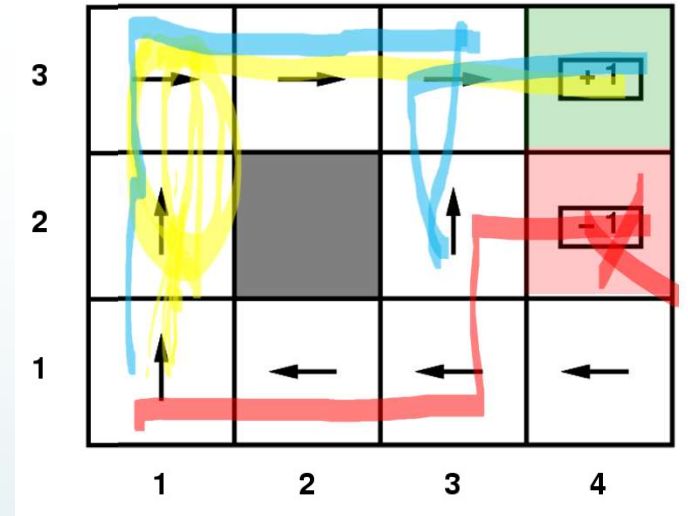
- Idea:
 - Learn the model empirically through experience
 - Solve for values as if the learned model were correct
- Simple empirical model learning
 - Count outcomes for each s, a
 - Normalize to give estimate of $\mathbf{T}(s, \mathbf{a}, s')$
 - Discover $\mathbf{R}(s, \mathbf{a}, s')$ when we experience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



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

Model Estimation

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



$(1,1) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (3,3) \rightarrow (3,4) \text{ +1}$
 $(1,1) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (2,3) \rightarrow (3,3) \rightarrow (3,2) \rightarrow (3,3) \rightarrow (3,4) \text{ +1}$
 $(1,1) \rightarrow (2,1) \rightarrow (3,1) \rightarrow (3,2) \rightarrow (4,2) \text{ -1}$

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

Training Samples

~~Handwritten notes and scribbles:~~
 0.92
 0.88
 0.84
 $J_{2,3}$
 $J_{3,3}$
 $J_{1,2}$

3				
2				
1				
	1	2	3	4

$(1, 1)_{-.04} \rightsquigarrow (1, 2)_{-.04} \rightsquigarrow (1, 3)_{-.04} \rightsquigarrow (1, 2)_{-.04} \rightsquigarrow (1, 3)_{-.04} \rightsquigarrow (2, 3)_{-.04} \rightsquigarrow (3, 3)_{-.04} \rightsquigarrow (4, 3)_{+1}$

~~Handwritten annotations for the first sequence:~~
 0.80 0.84 0.88 0.92 0.96

$(1, 1)_{-.04} \rightsquigarrow (1, 2)_{-.04} \rightsquigarrow (1, 3)_{-.04} \rightsquigarrow (2, 3)_{-.04} \rightsquigarrow (3, 3)_{-.04} \rightsquigarrow (3, 2)_{-.04} \rightsquigarrow (3, 3)_{-.04} \rightsquigarrow (4, 3)_{+1}$

~~Handwritten annotations for the second sequence:~~
 0.88 0.92 0.96

$(1, 1)_{-.04} \rightsquigarrow (2, 1)_{-.04} \rightsquigarrow (3, 1)_{-.04} \rightsquigarrow (3, 2)_{-.04} \rightsquigarrow (4, 2)_{-1}$

~~Handwritten annotations for the third sequence:~~
 -1.04 -1.04

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
- ▶ 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 .

Handwritten notes in red:
 $S \rightarrow a \rightarrow S' \rightarrow R'$
 $P(\mathcal{E} | s, a)$ utilization

Adaptive DP (ADP)

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

Solve
using DP

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

= value determination.

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

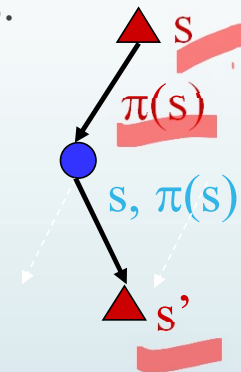
Too many linear equations to solve

One equation for each state → Large state space

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

Approach 3: 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.

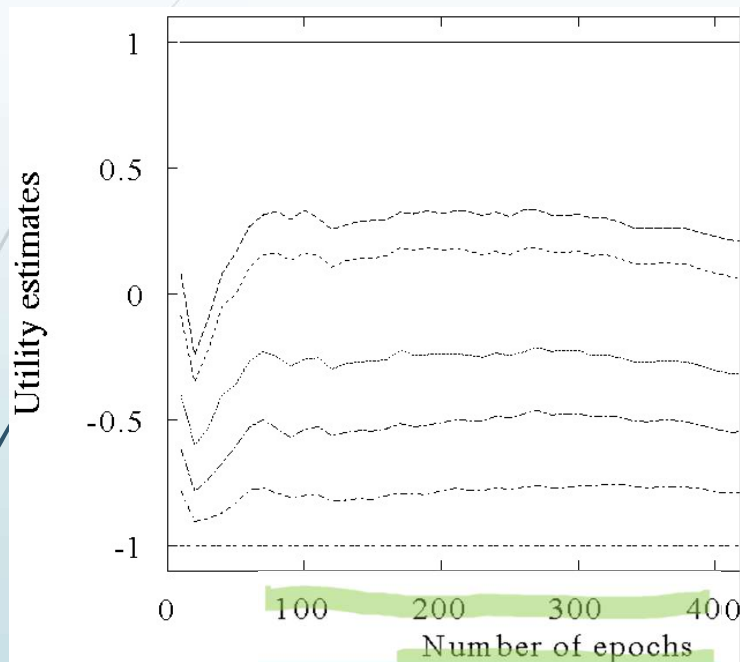
$$U(i) \leftarrow U(i) + \alpha [R(i) + U(j) - U(i)]$$

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

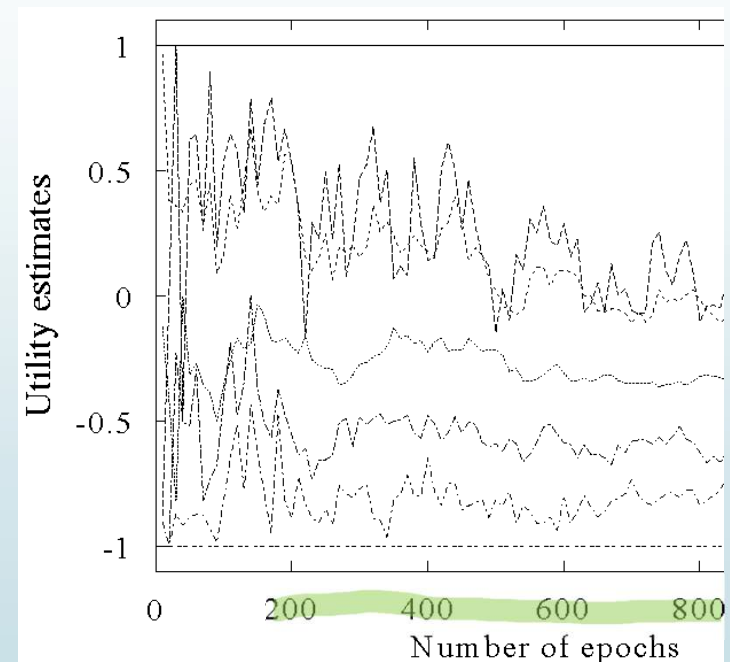
Thm: If α is appropriately decreased as a function of times a state is visited ($\alpha = \alpha[N[i]]$), then $U(i)$ itself converges to the correct value

Comparing Convergence between ADP and TD

ADP



TD



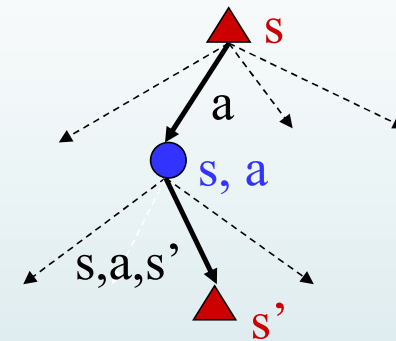
- **Tradeoff:** TD requires more training experience (epochs) than ADP but much less computation per epoch
- Choice depends on relative cost of experience vs. computation

Problems with TD Value Learning

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

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

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



- ▶ 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 similar 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