

Markov Decision Processes

Resource: David Silver's Lecture 2

<https://www.youtube.com/watch?v=lfHX2hHRMVQ&list=PLzvuYNsE1EZAXYR4FJ75jcJseBmo4KQ9-&index=2>

- Markov Processes: 6:25 (chains of Markov states)
- Markov Reward Processes: 13:00 (chains of Markov states with reward)
- Bellman Equation: 29:10
- Markov Decision Processes: 43:00 (add actions)
- Policy: 46:25



Where we are in the Textbook

Let's look at the textbook Table of Contents

David Silver Qlearning

https://www.youtube.com/watch?v=0g4j2k_Ggc4&list=PLzvuYNsE1EZAXYR4FJ75jcJseBmo4KQ9-&index=5

Qlearning: 1:29

Q-Learning Deep Dive

- Question: Why does our CliffWalking Example converge on the shortest path?
- In grid-based worlds, there is a similarity in structure between the Q-table and the grid world itself (we can animate the learning of the Q function in grid-based worlds)
- Discussion
 - How does the reward (besides cliff) affect the eventual path?
 - Negative reward? 0 reward? Positive reward?
 - How does the initialization of the qtable affect convergence?
 - Randomized? Initialize to zero?
 - Do we set the action-values of the terminal state to zero?



SARSA

- An implementation of the SARSA algorithm

```
# SARSA algorithm from Sutton textbook
#Algorithm parameters: step size alpha in (0, 1], small epsilon > 0
#Initialize Q(s,a), for all s in S+,a in A(s), arbitrarily except that
Q(terminal, ·) = 0
#Loop for each episode:
#    Initialize S
#    Choose A from S using policy derived from Q (e.g., epsilon-greedy)
#    Loop for each step of episode:
#        Take action A, observe R, S_prime
#        Choose A_prime from S_prime using policy derived from Q (e.g.,
#        epsilon-greedy)
#        Q(S, A) = Q(S, A) + alpha * (R + gamma * Q(S_prime, A_prime) - Q(S,
#        A))
#        S = S_prime; A = A_prime;
#    until S is terminal
```

On-policy vs Off-policy

- SARSA is an on-policy control method and Q-learning is an off-policy control method
- Why, what is the difference?
- In both, we have an implicit policy (epsilon-greedy):

```
# act randomly sometimes to allow exploration
if np.random.uniform() < epsilon:
    action = env.action_space.sample()
# otherwise select max action in Qtable (act greedy)
else:
    action = qtable[state].index(max(qtable[state]))
```
- The difference is magnified if we set $\text{epsilon} = 1$ (totally random policy)
 - SARSA update the qtable using the value of the random action
 - Q-Learning update the qtable using the action with max value

Reinforcement Learning: what we know so far

- There is an agent and an environment
- Repeatedly,
 - The agent performs an action which affects the environment
 - The environment enters a resulting state
 - The agent receives the new state and a scalar reward

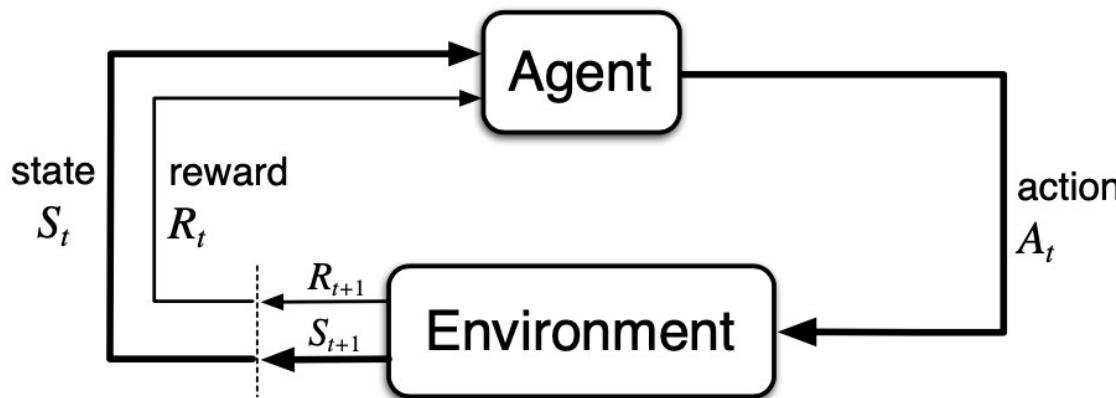


Figure 3.1: The agent–environment interaction in a Markov decision process.

Reinforcement Learning: what we know so far

- The agent learns by interacting with the environment
- The goal of the agent is to maximize reward
 - The agent learns how to maximize reward
 - The agent takes actions to maximize reward
- Reward Hypothesis (from Sutton):

That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).

RL: High-level Programmer's methodology

When applying RL to a problem in a domain, the programmer needs to

- Identify the problem to be solved as a subset of the domain
 - Example domain: AlphaGo playing Go
 - Was the physical act of placing the stones considered part of the problem? No, but it could have been. DeepMind did not identify that aspect of the domain as part of the problem to be solved.
- Given the problem, determine
 - What is the environment and what are the states? (simulated or actual)
 - What is the agent and what are the actions? (simulated or actual)
 - What is the reward function? (Implement it)
 - Big one: How is the agent going to learn to get better at maximizing reward?



Agent – Environment distinction

Read Sutton, last paragraph of Page 50, and Page 51

Examples of what you will read:

- The MDP framework is abstract and flexible and can be applied to many different problems in many different ways.
- In particular, the boundary between agent and environment is typically not the same as the physical boundary of a robot's or animal's body.
- The general rule we follow is that anything that cannot be changed arbitrarily by the agent is considered to be outside of it and thus part of its environment.
- The agent-environment boundary can be located at different places for different purposes.
- [RL] proposes that whatever the details of the sensory, memory, and control apparatus, and whatever objective one is trying to achieve, any problem of learning goal-directed behavior can be reduced to three signals passing back and forth between an agent and its environment: one signal to represent the choices made by the agent (the actions), one signal to represent the basis on which the choices are made (the states), and one signal to define the agent's goal (the rewards).
- Of course, the particular states and actions vary greatly from task to task, and how they are represented can strongly affect performance. In reinforcement learning, as in other kinds of learning, such representational choices are at present more art than science.



Rewards + Goals

Sutton, Pages 53-4:

- We must provide rewards in such a way that in maximizing them the agent will also achieve our goals. It is thus critical that the rewards we set up truly indicate what we want accomplished. In particular, the reward signal is not the place to impart to the agent prior knowledge about how to achieve what we want it to do.
- Do not design the reward around subgoals
- The reward signal is your way of communicating to the robot **what** you want it to achieve, not **how** you want it achieved. (compare with Declarative programming)
- Do not base rewards on previous actions (unless the action sequence IS the goal, like maybe dance moves?)



Returns and Episodes

- Some processes/tasks have a terminal state (episodic tasks, episodes):
 - A single play of a game
 - A run through a maze, or race around a track (finish line)
 - Making a cup of coffee
 - These processes have a terminal state
 - The time-step of termination, T , is a random variable that normally varies from episode to episode.
- Other processes never finish (continuing tasks, $T=\infty$):
 - Controlling a power plant
 - Home thermostat controlling humidity, temperature

Returns and Episodes

- We seek to maximize expected return, where return, G_t could be defined as

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \dots + R_T$$

but this is a problem for continuing tasks, because return blows up.

- To address this problem, there is discounting, with a discount rate γ :

$$\begin{aligned} G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

As γ approaches 1, the return objective takes future rewards into account more strongly; the agent becomes more farsighted

- For episodic tasks, to make this work, there is a special state ("absorbing state") that always transitions to itself, with a reward of 0

Epochs

In RL, we have episodes

Sometimes (Lab 1) you'll see a notion of epochs in a RL context. An Epoch is a single pass through the dataset, but RL has no such dataset!

Policies and Value Functions

- A ***Policy*** is a mapping (a function) from states to probabilities of selecting each possible action: $\pi(a | s) = P[A_t = a | S_t = s]$
- A deterministic policy is a mapping from states to actions: $\pi(s) = a$
- ***Value function*** of a state (or state-action pair):
 - Gives the expected return when starting at a state
 - Different policies result in different returns

Policies and Value Functions

- State-value function of a state under policy π :

$$v_\pi(s) \doteq E_\pi[G_t | S_t = s]$$

- Action-value function where you take action a :

$$q_\pi(s,a) \doteq E_\pi[G_t | S_t = s, A_t = a]$$

- $E_\pi[\cdot]$ denotes the expected value of a random variable given that the agent follows policy π , and t is any time step

Time to check your learning!

Let's see how many key concepts you recall by answering the following questions!

- What is a return?
- What is the expression for the expected return at timestep t?
- What is the meaning of a state-value function? An action-value function?
- What is a policy in the context of RL?
- What is an episode the context of RL?

