Reinforcement Learning: Basics and DQN

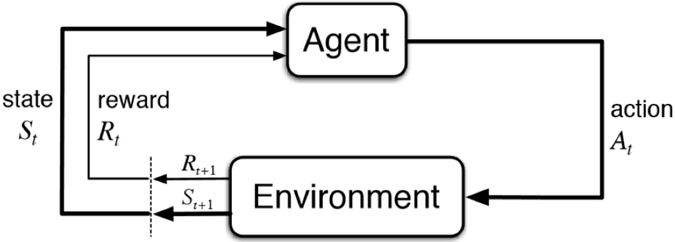
Changyou Chen

Department of Computer Science and Engineering
Universitpy at Buffalo, SUNY
changyou@buffalo.edu

April 23, 2019

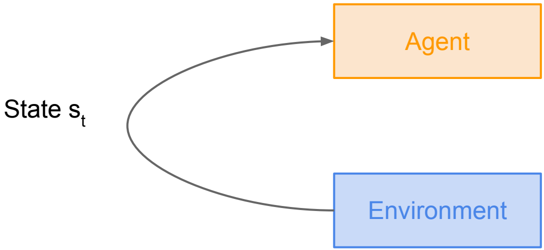
Reinforcement Learning

- Problems involving an agent interacting with an environment, which provides numeric reward signals.
- **Goal**: Learn how to take actions in order to maximize reward.

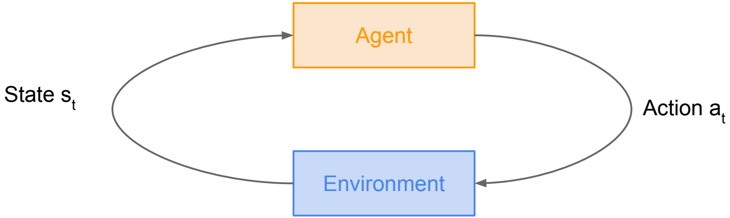


Reinforcement Learning

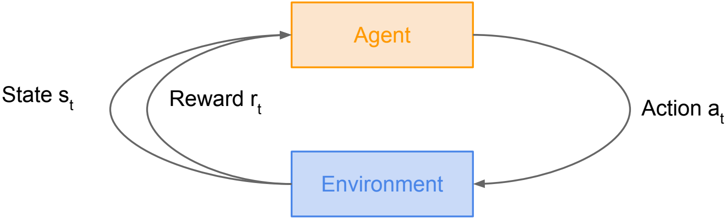
Reinforcement Learning



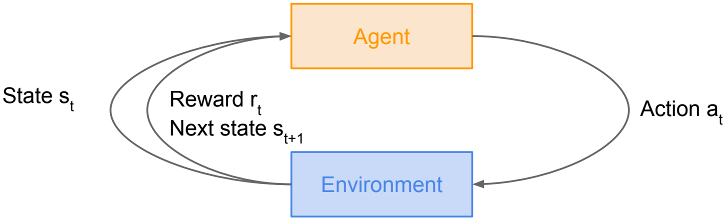
Reinforcement Learning



Reinforcement Learning

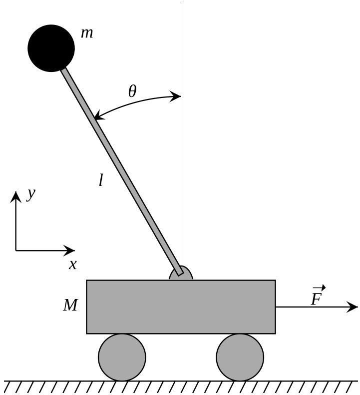


Reinforcement Learning



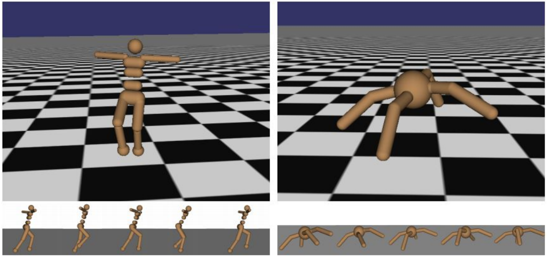
Cart-Pole Problem

- **Objective**: Balance a pole on top of a movable cart.
- **State**: angle, angular speed, position, horizontal velocity.
- **Action**: horizontal force applied on the cart.
- **Reward**: 1 at each time step if the pole is upright.



Robot Locomotion

- **Objective**: Make the robot move forward.
- **State**: Angle and position of the joints.
- **Action**: Torques applied on joints.
- **Reward**: 1 at each time step upright + forward movement.

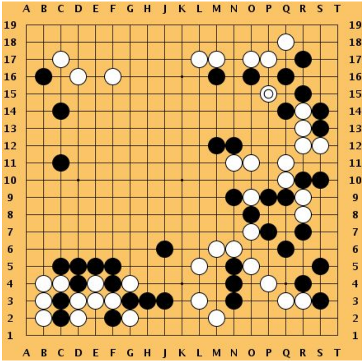


Atari Games

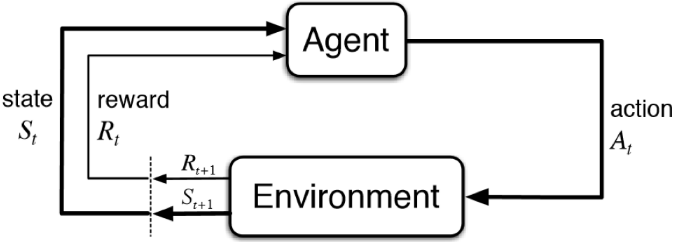
- **Objective**: Complete the game with the highest score.
- **State**: Raw pixel inputs of the game state.
- **Action**: Game controls e.g. Left, Right, Up, Down.
- **Reward**: Score increase/decrease at each time step.

Go

- **Objective**: Win the game.
- **State**: Position of all pieces.
- **Action**: Where to put the next piece down.
- **Reward**: 1 if win at the end of the game, 0 otherwise.



How to Mathematically Formalize the RL Problem?



Markov Decision Process

- Mathematical formulation of the RL problem.
- **Markov property**: Current state completely characterizes the state of the world.

$\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

- $\mathcal{S}$: set of possible states.
- $\mathcal{A}$: set of possible actions.
- $\mathcal{R}$: distribution of reward given (state, action) pairs.
- $\mathbb{P}$: transition probability, *i.e.*, distribution over next state given a (state, action) pair.
- γ: discount factor.

Markov Decision Process

1. At time step $t = 0$, environment samples initial state $s_0 \sim p(s_0)$.
2. Then, for $t = 0$ until done:
   - Agent selects an action a_t.
   - Environment samples reward $r_t \sim \mathcal{R}(\cdot|s_t, a_t)$.
   - Environment samples next state $s_{t+1} \sim \mathbb{P}(\cdot|s_t, a_t)$.
   - Agent receives reward r_t and next state s_{t+1}.
3. A policy π is a function from $\mathcal{S}$ to $\mathcal{A}$ that specifies what action to take in each state:
   - usually it is modeled as a conditional distribution of action given state.
4. **Objective**: find policy π^* that maximizes cumulative discounted reward: $\sum_{t \geq 0} \gamma^t r_t$.

Markov Decision Process

1. At time step $t = 0$, environment samples initial state $s_0 \sim p(s_0)$.
2. Then, for $t = 0$ until done:
   - Agent selects an action a_t.
   - Environment samples reward $r_t \sim \mathcal{R}(\cdot | s_t, a_t)$.
   - Environment samples next state $s_{t+1} \sim \mathbb{P}(\cdot | s_t, a_t)$.
   - Agent receives reward r_t and next state s_{t+1}.
3. A policy π is a function from $\mathcal{S}$ to $\mathcal{A}$ that specifies what action to take in each state:
   - usually it is modeled as a conditional distribution of action given state.
4. **Objective**: find policy π^* that maximizes cumulative discounted reward: $\sum_{t \geq 0} \gamma^t r_t$.

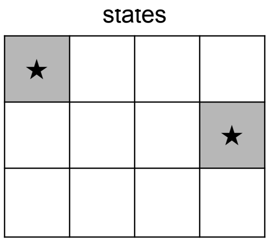
We need to learn the policy π^* and sometimes (parts of) the MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$.

A Simple MDP: Policy for Grid World

actions = {

1. right
2. left
3. up
4. down

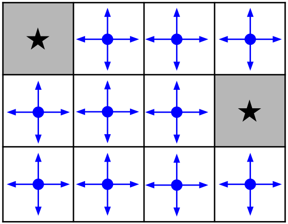
}



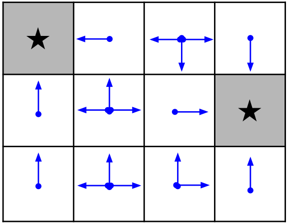
Set a negative "reward" for each transition (e.g. r = -1)

Objective: reach one of terminal states (greyed out) in least number of actions

A Simple MDP: Policy for Grid World



Random Policy



Optimal Policy

The Optimal Policy

1. We want to find optimal policy π^* that maximizes the sum of rewards.
2. Directly summing over rewards endows randomness, *e.g.*, initial state, transition probability, reward probability.
3. We should maximize the *expected total reward*:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t\right],$$
$$\text{with } s_0 \sim p(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)$$

Exact Methods and Monte Carlo Approximation[1]

[1] Adapted from: https://drive.google.com/file/d/0BxXI_RttTZAhVXBlMUVkQ1BVVDQ/view, https://drive.google.com/file/d/0BxXI_RttTZAhREJKRGhDT25OOTA/view

Exact Methods

Optimal control

- Given an MDP: $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$, find the optimal policy π^*.

Exact methods

- Value iteration.
- Policy iteration.

Exact Methods

Optimal control

- Given an MDP: $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$, find the optimal policy π^*.

Exact methods

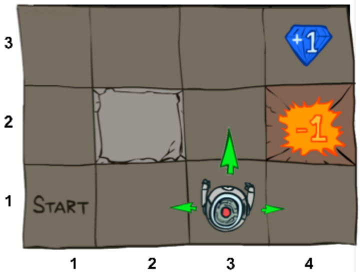
- Value iteration.
- Policy iteration.

Optimal Value Function

- Define the optimal value function $V^*(s)$ as the sum of discounted rewards when starting from state s and acting optimally:

$$V^*(s) \triangleq \max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) | \pi, s_0 = s\right]$$
$$= \max_{\pi} \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t R(s_t, a_t, s_{t+1}) | \pi, s_0 = s, H \to \infty\right]$$

- What are the values $V^*(1,1)$, $V^*(1,2)$, $\cdots$, $V^*(3,4)$?



Value Iteration: Dynamic Programming

- $V_0^*(s)$: optimal value for state s when $H = 0$:

$$V_0^*(s) = 0, \forall s$$

- $V_1^*(s)$: optimal value for state s when $H = 1$:

$$V_1^*(s) = \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_0^*(s'))$$

- $V_2^*(s)$: optimal value for state s when $H = 2$:

$$V_2^*(s) = \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_1^*(s'))$$

- $V_k^*(s)$: optimal value for state s when $H = k$:

$$V_k^*(s) = \max_a \sum_{s'} P(s'|s,a)(R(s,a,s') + \gamma V_{k-1}^*(s'))$$

Value Iteration

Algorithm:

Start with $V_0^*(s) = 0$ for all s.

For k = 1, … , H:

For all states s in S:

$$V_k^*(s) \leftarrow \max_a \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$$

$$\pi_k^*(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s,a) \left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$$

This is called a value update or Bellman update/back-up

Value Iteration

Theorem

Value iteration converges. At convergence, we have found the optimal value function V^* for the discounted infinite horizon problem, which satisfies the Bellman equations:

$$\forall s, V^*(s) = \max_a \sum_{s'} P(s'|s) \left[R(s, a, s') + \gamma V^*(s')\right]$$

Now we know how to act for infinite horizon with discounted rewards:

- Run value iteration until convergence.
- This produces V^*, which tells us the optimal policy:

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s'|s) \left[R(s, a, s') + \gamma V^*(s')\right]$$

Q-Values

- $Q^*(s,a)$: expected reward starting in s, taking action a, and (thereafter) acting optimally.
- Bellman equation:

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right)$$

- Q-value iteration:

$$Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right)$$

$$\pi_{k+1}(s) = \arg\max_a \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right)$$

$$= \arg\max_a Q_{k+1}(s,a)$$

Exact Methods

Optimal control

- Given an MDP: $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$, find the optimal policy π^*.

Exact methods

- Value iteration.
- Policy iteration.

Policy Evaluation

- Recall value iteration:

$$V_k^*(s) = \max_a \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma V_{k-1}^*(s')\right)$$

- Policy evaluation for a given π:

Deterministic π: $V_i^\pi(s) = \sum_{s'} P(s'|s,\pi(s))\left(R(s,\pi(s),s') + \gamma V_{i-1}^*(s')\right)$

Stochastic π: $V_i^\pi(s) = \mathbb{E}_{a\sim\pi(\cdot|s)} \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma V_{i-1}^*(s')\right)$

- At convergence:

$$\forall s,\, V^\pi(s) = \sum_{s'} P(s'|s,\pi(s))\left(R(s,\pi(s),s') + \gamma V^*(s')\right)$$

Policy Iteration

One iteration of policy iteration:

- Policy evaluation for current policy π_k :
  - Iterate until convergence

$$V_{i+1}^{\pi_k}(s) \leftarrow \sum_{s'} P(s'|s, \pi_k(s)) \left[R(s, \pi(s), s') + \gamma V_i^{\pi_k}(s')\right]$$

- Policy improvement: find the best action according to one-step look-ahead

$$\pi_{k+1}(s) \leftarrow \arg\max_a \sum_{s'} P(s'|s, a) \left[R(s, a, s') + \gamma V^{\pi_k}(s')\right]$$

Policy Iteration

Theorem

Policy iteration is guaranteed to converge. At convergence, the current policy and its value function are the optimal policy and the optimal value function.

Limitations:

- Iteration over / storage for all states and actions: requires small, discrete state-action space:
  - sampling-based approximations
- Update equations require access to dynamics mode.

Policy Iteration

Theorem

Policy iteration is guaranteed to converge. At convergence, the current policy and its value function are the optimal policy and the optimal value function.

Limitations:

- Iteration over / storage for all states and actions: requires small, discrete state-action space:
  - sampling-based approximations
- Update equations require access to dynamics mode.

Sampling-Based Approximation

- Q-value iteration?
- Value iteration?
- Policy iteration:
  - policy evaluation?
  - policy improvement?

Sampling-Based Approximation

- Q-value iteration?
- Value iteration?
- Policy iteration:
  - policy evaluation?
  - policy improvement?

Recap: Q-Values

- $Q^*(s, a)$: expected reward starting in s, taking action a, and (thereafter) acting optimally.
- Bellman equation:

$$Q^*(s, a) = \sum_{s'} P(s'|s, a) \left(R(s, a, s') + \gamma \max_{a'} Q^*(s', a') \right)$$

- Q-value iteration:

$$Q_{k+1}(s, a) = \sum_{s'} P(s'|s, a) \left(R(s, a, s') + \gamma \max_{a'} Q_k(s', a') \right)$$

(Tabular) Q-Learning

$$Q_{k+1}(s,a) = \sum_{s'} P(s'|s,a)\left(R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right)$$

$$\Rightarrow Q_{k+1}(s,a) = \mathbb{E}_{s' \sim P(s'|s,a)}\left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]$$

(Tabular) Q-Learning: replace expectation by samples:

- For an state-action pair (s,a), sample $s' \sim P(s'|s,a)$.
- Calculate the new sample estimate based on the old estimate $Q_k(s,a)$:

$$\text{target}(s') = R(s,a,s') + \gamma \max_{a'} Q_k(s',a')$$

- Update the new sample estimate by a running average:

$$Q_{k+1}(s,a) = (1-\alpha)Q_k(s,a) + \alpha \times \text{target}(s')$$

(Tabular) Q-Learning

Algorithm:

Start with $Q_0(s, a)$ for all s, a.

Get initial state s

For k = 1, 2, … till convergence

Sample action a, get next state s'

If s' is terminal:

$$\text{target} = R(s, a, s')$$

Sample new initial state s'

else:

$$\text{target} = R(s, a, s') + \gamma \max_{a'} Q_k(s', a')$$

$$Q_{k+1}(s, a) \leftarrow (1 - \alpha) Q_k(s, a) + \alpha \, [\text{target}]$$

$$s \leftarrow s'$$

How to Sample Actions?

- Random actions.
- Action that maximizes $Q_k(s, a)$ (greedy).
- ϵ-Greedy: choose random actions with prob.ϵ; otherwise choose actions greedily.
- Caveats:
  - You have to explore enough
  - You have to eventually make the learning rate small enough, but not decrease it too quickly

Sampling-Based Approximation

- Q-value iteration?
- Value iteration?
- Policy iteration:
  - policy evaluation?
  - policy improvement?

Value Iteration

$$V_{k+1}^*(s) = \max_a \mathbb{E}_{s' \sim P(s'|s,a)} \left[R(s, a, s') + \gamma V_k^*(s') \right]$$

- V^* does not depend on actions, have to integrate it out.
- Unclear how to draw samples through max.

Sampling-Based Approximation

- Q-value iteration?
- Value iteration?
- Policy iteration:
  - policy evaluation?
  - policy improvement?

Policy Iteration

One iteration of policy iteration:

- Policy evaluation for current policy π_k :
  - Iterate until convergence

$$V_{i+1}^{\pi_k}(s) \leftarrow \mathbb{E}_{s' \sim P(s'|s,\pi_k(s))}[R(s,\pi_k(s),s') + \gamma V_i^{\pi_k}(s')]$$

Can be approximated by samples
This is called Temporal Difference (TD) Learning

- Policy improvement: find the best action according to one-step look-ahead

$$\pi_{k+1}(s) \leftarrow \arg\max_a \mathbb{E}_{s' \sim P(s'|s,a)}[R(s,a,s') + \gamma V^{\pi_k}(s')]$$

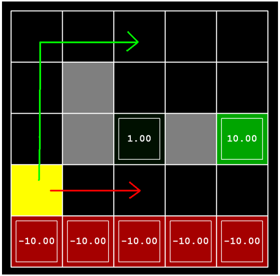
Unclear what to do with the max (for now)

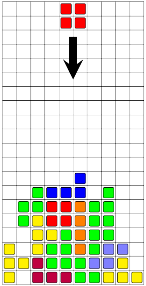
Limitation

Limitations:

- Iteration over / storage for all states and actions: requires small, discrete state-action space:
  - sampling-based approximations
- Update equations require access to dynamics mode:
  - Q/V function fitting.

Can tabular methods scale?





Gridworld
10^1

Tetris
10^60

Atari
10^308 (ram) 10^16992 (pixels)

Approximate Q-Learning

- Instead of a table, we have a parametrized Q-function $Q_\theta(s, a)$:
  - Can be a linear function in features:

$$Q_\theta(s, a) = \theta_0 f_0(s, a) + \theta_1 f_1(s, a) + \cdots + \theta_n f_n(s, a)$$

  - Or a complicated neural network.
- Learning rule:

$$\theta_{k+1} = \theta_k - \alpha \nabla_\theta \left[\frac{1}{2} \left(Q_\theta(s, a) - \text{target}(s') \right)^2 \right] |_{\theta=\theta_k}$$

$$\text{target}(s') \triangleq R(s, a, s') + \gamma \max_{a'} Q_{\theta_k}(s', a')$$

Connection to Tabular Q-Learning

- Suppose $\theta \in \mathbb{R}^{|S|\times|A|}, \quad Q_\theta(s,a) \equiv \theta_{sa}$

$$\begin{aligned}
&\nabla_{\theta_{sa}}\left[\frac{1}{2}(Q_\theta(s,a) - \text{target}(s'))^2\right] \\
&= \nabla_{\theta_{sa}}\left[\frac{1}{2}(\theta_{sa} - \text{target}(s'))^2\right] \\
&= \theta_{sa} - \text{target}(s')
\end{aligned}$$

- Plug into update: $\theta_{sa} \leftarrow \theta_{sa} - \alpha(\theta_{sa} - \text{target}(s'))$

$$= (1-\alpha)\theta_{sa} + \alpha[\text{target}(s')]$$

- Compare with Tabular Q-Learning update:

$$Q_{k+1}(s,a) \leftarrow (1-\alpha)Q_k(s,a) + \alpha\left[\text{target}(s')\right]$$

Convergence of Approximate Q-Learning

- It is not guaranteed to converge, even if the function approximation is expressive enough to represent the true Q-function:
  - The approximation is sequential, so if each time there induces a large enough error, the aggregate error might be exploded.