Reinforcement Learning

Prof. Seungchul Lee
Industrial AI Lab.

Source

- David Silver's Lecture (DeepMind)
  - UCL homepage for slides (http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html)
  - DeepMind for RL videos (https://www.youtube.com/watch?v=2pWv7GOvuf0)
  - An Introduction to Reinforcement Learning, Sutton and Barto pdf

- CMU by Zico Kolter
  - http://www.cs.cmu.edu/~zkolter/course/15-780-s14/lectures.html
  - https://www.youtube.com/watch?v=un-FhSC0HfY&hd=1

- Deep RL Bootcamp by Rocky Duan
  - https://sites.google.com/view/deep-rl-bootcamp/home
  - https://www.youtube.com/watch?v=qO-HUo0LsO4

- Stanford Univ. by Serena Yeung
  - https://www.youtube.com/watch?v=lvoHnicueoE&list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv&index=15&t=1337s

Markov Decision Process

$$M = (S, A, P, R)$$

- S: set of states
- A: set of actions
- P: $S \times A \times S \rightarrow [0, 1]$: transition probability distribution $P(s' \mid s, a)$
- $R : S \rightarrow \mathbb{R}$: reward function, where $R(S)$ is reward for state s
- γ: discount factor

- Policy $\pi : S \rightarrow A$ is a mapping from states to actions

- The RL twist: we do not know P or R,
- They are too big to enumerate (only have the ability to act in MDP, observe states and rewards)

Solving MDP

- (Policy evaluation) Determine value of policy π

$$\begin{aligned} v_\pi(s) &= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid s_0 = s\right] \\ &= R(s) + \gamma \sum_{s' \in S} P\left(s' \mid s, \pi(s)\right) v_\pi\left(s'\right) \end{aligned}$$

accomplished via the iteration (similar to a value iteration, but for a fixed policy)

$$v_\pi(s) \leftarrow R(s) + \gamma \sum_{s' \in S} P\left(s' \mid s, \pi(s)\right) v_\pi\left(s'\right), \quad \forall s \in S$$

- (Value iteration) Determine value of optimal policy

$$v_*(s) = R(s) + \gamma \sum_{s' \in S} P(s' \mid s, a) v_*\left(s'\right)$$

accomplished via value iteration:

$$v(s) \leftarrow R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P\left(s' \mid s, a\right) v\left(s'\right), \quad \forall s \in S$$

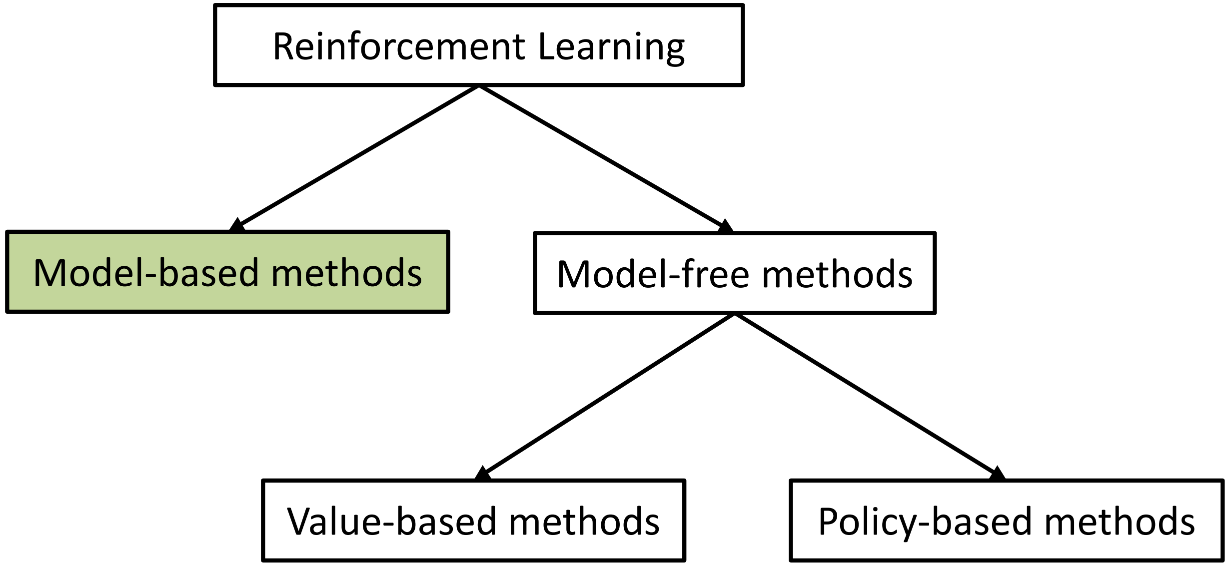
Optimal Policy

- Optimal policy π_* is then

$$\pi_*(s) = \arg\max_{a \in A} \sum_{s' \in S} P(s' \mid s, a)\, v_*(s')$$

- How can we compute these quantities when P and R are unknown?
  - model-based RL
  - model-free RL

Overview of RL



Model-based RL

- A simple approach: just estimate the MDP from data (known as Monte Carlo method)
  - Agent acts in the work (according to some policy), observes episodes of experience

$$s_1, r_1, a_1, s_2, r_2, a_2, \cdots, s_m, r_m, a_m$$

  - We form the empirical estimate of the MDP via the counts

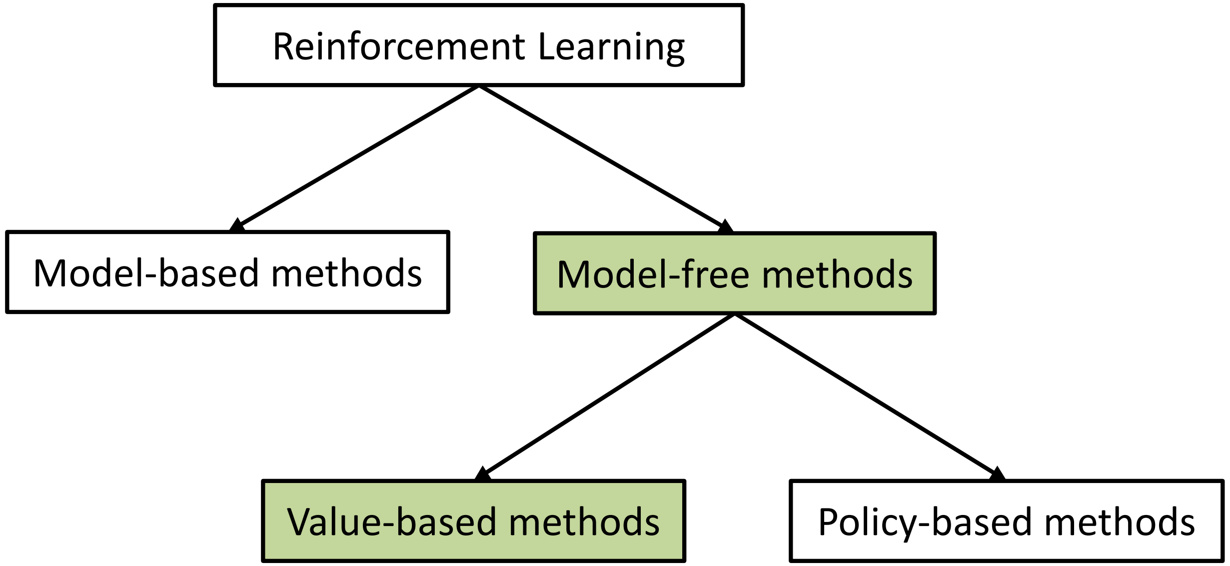
$$\hat{P}\left(s' \mid s, a\right) = \frac{\sum_{i=1}^{m-1} \mathbf{1}\left\{s_i = s, a_i = a, s_{i+1} = s'\right\}}{\sum_{i=1}^{m-1} \mathbf{1}\{s_i = s, a_i = a\}}$$

$$\hat{R}(s) = \frac{\sum_{i=1}^{m-1} \mathbf{1}\{s_i = s\} r_i}{\sum_{i=1}^{m-1} \mathbf{1}\{s_i = s\}}$$

Model-based RL

- Will converge to correct MDP (and hence correct value function/policy) given enough samples of each state
- How can we ensure we get the "right" samples? (a challenging problem for all methods we present here)
- Advantages (informally): makes "efficient" use of data
- Disadvantages: requires we build the actual MDP models, not much help if state space is too large

Overview of RL



Model-free RL

- Temporal difference methods (TD, SARSA, Q-learning):
  - directly learn value function v_π or v_*

- Direct policy search:
  - directly learn optimal policy π_*

Temporal Difference (TD) Methods (1/2)

- Let's consider computing the value function for a fixed policy via the iteration

$$v_\pi(s) \;\leftarrow\; R(s) + \gamma \sum_{s' \in S} P\left(s' \mid s, \pi(s)\right) v_\pi\left(s'\right), \quad \forall s \in S$$

- Suppose we are in some state s_t, receive reward R_t, take action $a_t = \pi(s_t)$ and end up in state s_{t+1}
- We cannot update v_π for all $s \in S$, but can we update just for s_t ?

$$v_\pi(s_t) \;\leftarrow\; R_t + \gamma \sum_{s' \in S} P\left(s' \mid s_t, a_t\right) v_\pi\left(s'\right)$$

- No, because we still do not know $P(s'|s_t, a_t)$ for all $s' \in S$

Temporal Difference (TD) Methods (2/2)

$$v_\pi(s_t) \;\leftarrow\; R_t + \gamma \sum_{s' \in S} P\left(s' \mid s_t, a_t\right) v_\pi\left(s'\right)$$

- But, s_{t+1} is a sample from the distribution $P(s'|s_t, a_t)$, so we could perform the update

$$v_\pi(s_t) \;\leftarrow\; R_t + \gamma v_\pi(s_{t+1})$$

  - It is too "harsh" assignment if we assume that s_{t+1} is the only possible next state;
  - Instead "smooth" the update using some $\alpha < 1$

$$v_\pi(s_t) \;\leftarrow\; (1-\alpha)\;(v_\pi(s_t)) + \alpha\;(R_t + \gamma v_\pi(s_{t+1}))$$

- This is the temporal difference (TD) algorithm. Its mathematical background will be briefly discussed later.

Issue with Traditional TD Algorithms

- TD lets us learn the value function of a policy π directly, without ever constructing the MDP.

- But is this really that helpful?

- Consider trying to execute greedy policy with respect to estimated v_π

$$\pi'(s) = \arg\max_{a \in A} \sum_{s' \in S} P(s' \mid s, a) \, v_\pi(s')$$

- We need a model $P(s'|s, a_t)$ anyway.

Entering Q Function (= State-Action Value Function)

- Q function is a value of starting state s, taking action a, and then acting according to π (or optimally for Q_*)

$$Q_\pi(s,a) = R(s) + \gamma \sum_{s' \in S} P\left(s' \mid s,a\right) Q_\pi\left(s', \pi\left(s'\right)\right)$$

$$Q_*(s,a) = R(s) + \gamma \sum_{s' \in S} P\left(s' \mid s,a\right) \max_{a'} Q_*\left(s', a'\right)$$

$$= R(s) + \gamma \sum_{s' \in S} P\left(s' \mid s,a\right) v_*\left(s'\right)$$

- Optimal policy

$$\pi_*(s) = \arg\max_a \sum_{s'} P\left(s' \mid s,a\right) v_*\left(s'\right) \quad \text{or}$$

$$\pi_*(s) = \arg\max_a Q_*(s,a) \quad \text{without knowing dynamics}$$

SARSA and Q-learning

- Q function leads to new TD-like methods.
- As with TD, observe state s, reward r, take action a (but not necessarily $a = \pi(s)$), observe next sate s'

- SARSA: estimate $Q_\pi(s, a)$ for expectation

$$Q_\pi(s,a) = R(s) + \gamma \sum_{s' \in S} P(s' \mid s,a)\, Q_\pi(s', \pi(s'))$$

TD

$$\boxed{Q_\pi(s,a) \leftarrow (1-\alpha)(Q_\pi(s,a)) + \alpha\left(R_t + \gamma Q_\pi(s', \pi(s'))\right)}$$

- Q-learning: estimate $Q_*(s, a)$ for optimality

$$Q_*(s,a) = R(s) + \gamma \sum_{s' \in S} P(s' \mid s,a) \max_{a'} Q_*(s', a')$$

TD

$$\boxed{Q_*(s,a) \leftarrow (1-\alpha)(Q_*(s,a)) + \alpha\left(R_t + \gamma \max_{a'} Q_*(s', a')\right)}$$

Note: State–Action–Reward–State–Action (SARSA)

SARSA and Q-learning

- The advantage of this approach is that we can now select actions without a model of MDP

- SARSA, greedy policy with respect to $Q_\pi(s,a)$

$$\pi'(s) = \arg\max_a Q_\pi(s,a)$$

- Q-learning, optimal policy

$$\pi_*(s) = \arg\max_a Q_*(s,a)$$

Solving Q-Value

- Q-value iteration

$$
\begin{aligned}
Q_{k+1}(s,a) &\leftarrow R(s) + \gamma \sum_{s'} P\left(s' \mid s,a\right) \max_{a'} Q_k\left(s',a'\right) \\
&\leftarrow 1 \cdot R(s) + \gamma \sum_{s'} P\left(s' \mid s,a\right) \max_{a'} Q_k\left(s',a'\right) \\
&\leftarrow \sum_{s'} P\left(s' \mid s,a\right) \cdot R(s) + \gamma \sum_{s' \in S} P\left(s' \mid s,a\right) \max_{a'} Q_k\left(s',a'\right) \\
&\leftarrow \sum_{s'} P\left(s' \mid s,a\right) \left[R(s) + \gamma \max_{a'} Q_k\left(s',a'\right) \right]
\end{aligned}
$$

$$
Q_{k+1}(s,a) \leftarrow \mathbb{E}_{s' \sim P(s' \mid s,a)} \left[R(s) + \gamma \max_{a'} Q_k\left(s',a'\right) \right] \qquad \text{Rewrite as expectation}
$$

- Replace expectation by samples

Q-Learning Algorithm (1/2)

- Replace expectation by samples

1) For an state-action pair (s, a), receive: $s' \sim P\left(s' \mid s, a\right)$

2) Consider your old estimate: $Q_k(s, a)$

3) Consider your new sample estimate:

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

4) Incorporate the new estimate into a running average [Temporal Difference or learning incrementally]:

$$\begin{aligned}
Q_{k+1}(s, a) & \leftarrow Q_k(s, a)+\alpha\left(\operatorname{target}\left(s'\right)-Q_k(s, a)\right) \\
& \leftarrow(1-\alpha) Q_k(s, a)+\alpha \operatorname{target}\left(s'\right) \\
& \leftarrow(1-\alpha) Q_k(s, a)+\alpha\left(R(s)+\gamma \max_{a'} Q_k\left(s', a'\right)\right)
\end{aligned}$$

How to Sample Actions (Exploration vs. Exploitation) ?

- All the methods we discussed so far had some condition like “assuming we visit each state enough”, or “taking actions according to some policy”

- A fundamental question: if we don’t know the system dynamics, should we take exploratory actions that will give us more information, or exploit current knowledge to perform as best we can?

- Example: a model-based procedure that does not work
  - Use all past experience to build model $\hat{P}$ and $\hat{R}$
  - Find optimal policy for MDP $\widehat{M} = \left(S, A, \hat{P}, \hat{R}, \gamma\right)$ using e.g. value iteration and act according to this policy

  - Initial bad estimates may lead policy into sub-optimal region, and never explores further

Exploration: ε-Greedy

- Key idea: instead of acting according to the "best" policy based upon the current MDP estimate, act according to a policy that will *explore* less visited state-action pairs until we get a "good estimate"

- Choose random actions? Or
- Choose action that maximizes $Q_s(s, a)$ (i.e. greedily)?
- ε-Greedy: choose random action with probability ε, otherwise choose action greedily

$$\pi(s) = \begin{cases} \max_{a \in A} Q_k(s, a) & \text{with probability } 1 - \varepsilon \\ \text{random action} & \text{otherwise} \end{cases}$$

exploitation

exploration

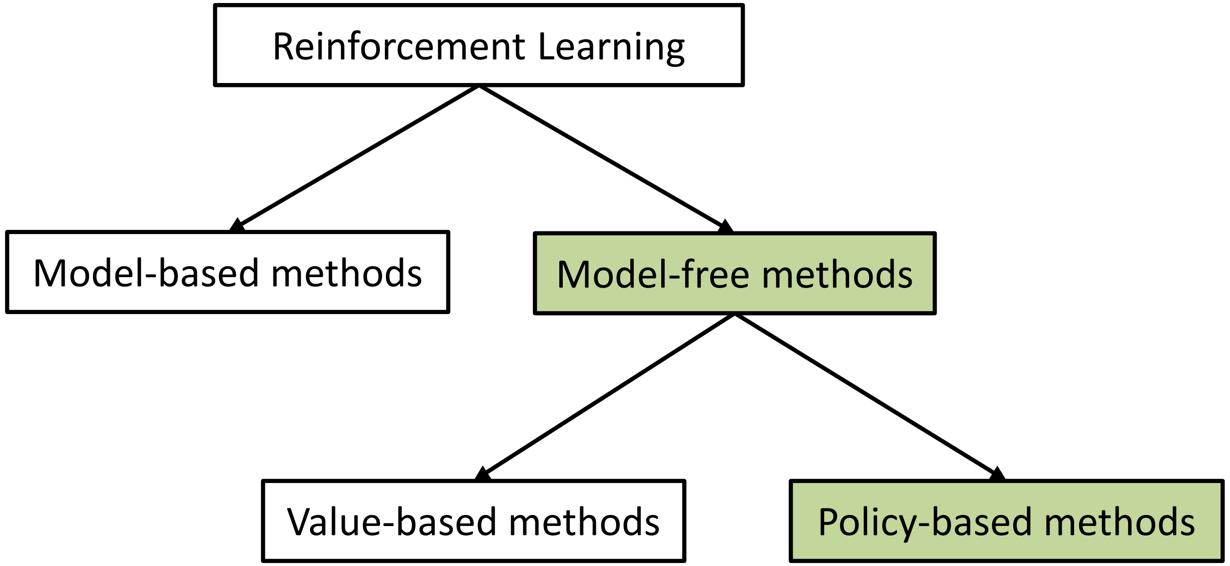
- Want to decrease ε as we see more examples

Q-Learning Algorithm (2/2)

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize s
  Repeat (for each step of episode):
    Choose a from s using policy derived from Q (e.g., ε greedy)
    Take action a, observe R, s'
    $Q_*(s,a) \leftarrow (1-\alpha)\left(Q_*(s,a)\right) + \alpha\left(R_t + \gamma \max_{a'} Q_*\left(s', a'\right)\right)$
    $s \leftarrow s'$
until s is terminal

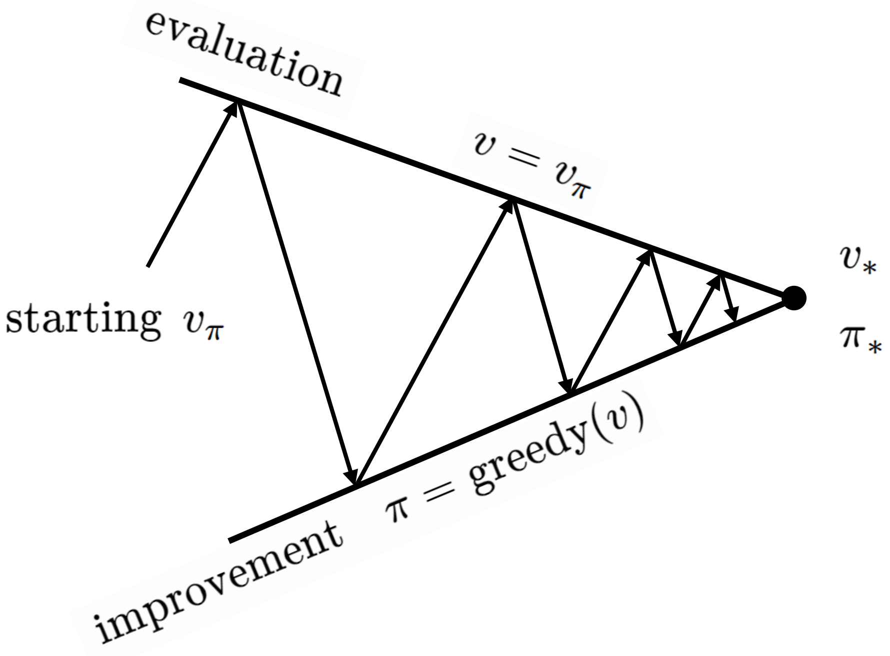
- Q-Learning Properties
  - Amazing result: Q-learning converges to optimal policy if all state-action pairs seen frequently enough
  - With Q-learning, we can learn optimal policy without model of MDP

Overview of RL



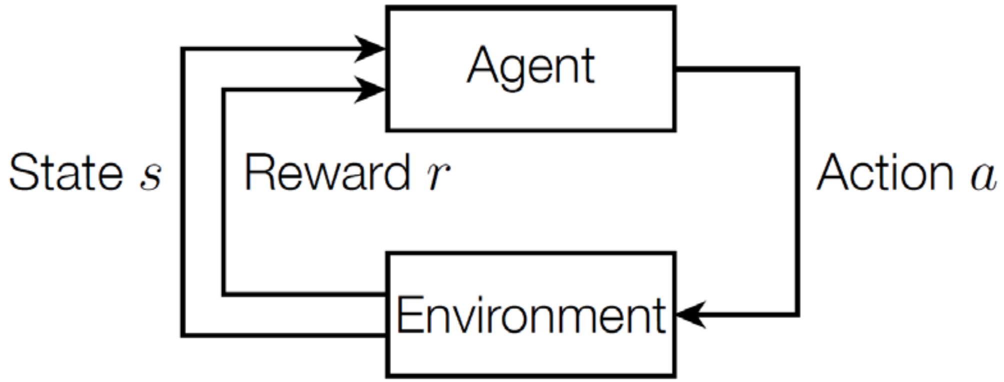
Iterative Policy Evaluation

- Given a policy π, then evaluate the policy π
- Improve the policy by acting greedily with respect to v_π



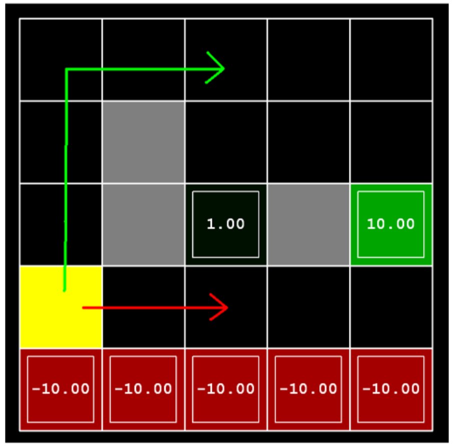
Q-Learning with Gym Environment

- Agent interaction with environment

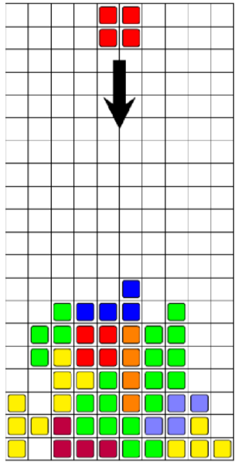


- OpenAI Gym
  - A Python API for RL environments
  - A set of tools to measure agent performance
  - Read https://gym.openai.com/docs/

- Examples



Gridworld

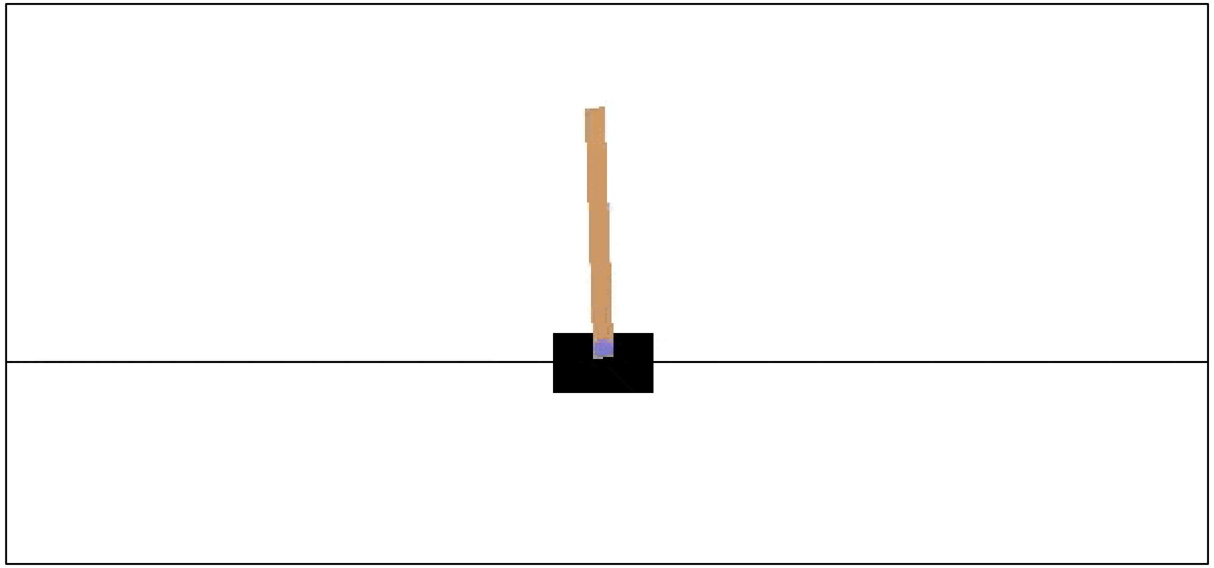


Tetris

Atari

CartPole-v1

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



Q-Learning

Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
        Choose a from s using policy derived from Q (e.g., ε greedy)
        Take action a, observe R, s'
        $Q_*(s,a) \leftarrow (1-\alpha)\left(Q_*(s,a)\right) + \alpha\left(R_t + \gamma \max_{a'} Q_*\left(s', a'\right)\right)$
        $s \leftarrow s'$
until s is terminal

```
# Exploration vs. Exploitation
epsilon = 0.5 * (1 / (episode + 1))
if np.random.random(1)[0] < epsilon:
    action = np.random.randint(2)
else:
    action = np.argmax(Q_table[idx_state])
```

```
# Temporal Difference Update
Q_table[idx_state, action] = (1-LR)*Q_table[idx_state, action] + LR*(reward + gamma*np.max(Q_table[new_idx_state,:]))
```