Machine Learning

AIAA 5046, Spring 2024

Sihong Xie

AI Thrust

HKUST-GZ

<u>Goals:</u>

- MDP
- Value functions

<u>Readings:</u>

RL Chap 3-6

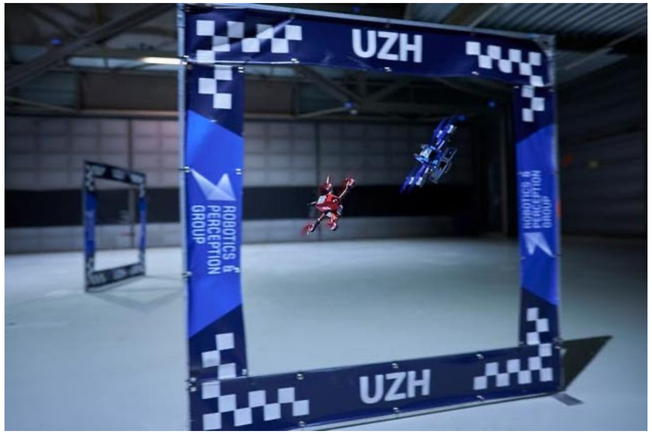
Sequential decision making

Sequential decision making:
- an agent interacts with an environment, and
- takes a sequence of actions to reach a goal.

Different from supervised learning:
- data are generated via interactions and can change with the agent's behavior;
- data are not I.I.D.;
- no one-shot prediction and immediate feedback.

Different from unsupervised learning:
- finding clustering patterns can't solve sequential decision making problems.
- but can help find representation of the environment to help.

Drone control



Game

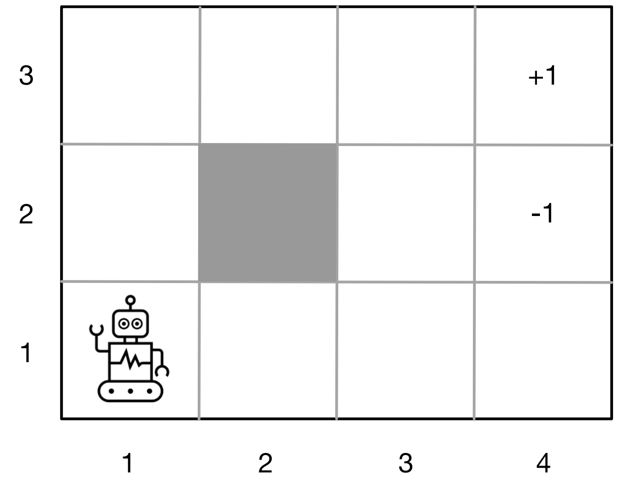
Markov Decision Processes

Markov Decision Processes: a mathematical description of sequential decision making.

- We use *t* = 1,2, … to denote the steps of decision-making.

An MDP has five components.

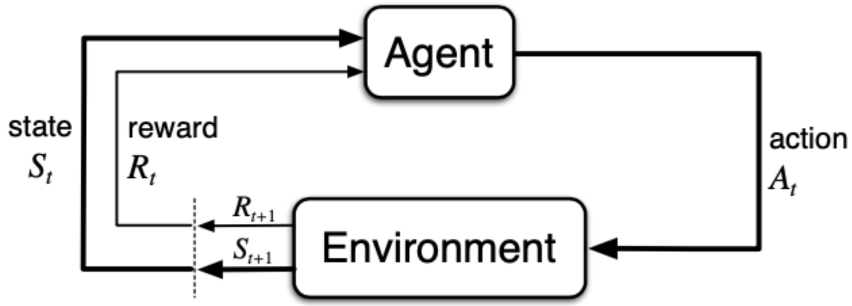
- A set of states $\mathcal{S}$ and current state $S_t \in \mathcal{S}$
- Action space $\mathcal{A}(S_t)$
- Reward R_t
- Dynamics $\Pr(S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a)$
- Discounting factor $0 \leq \gamma \leq 1$



Markov Decision Processes

Learning from interactions:

The agent interacts with the environment and learns from the interaction experiences.



A trajectory or experience is denoted as

$$S_0, A_0, R_0, S_1, A_1, R_1, S_2, A_2, R_2, \ldots$$

These are all random variables, as the agent selects actions stochastically, and the MDP dynamics (rewards and state transition) are stochastic.

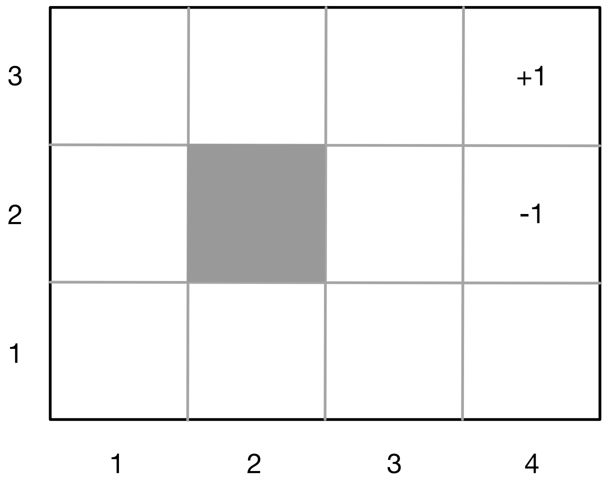
Markov Decision Processes

Goal of reinforcement learning:

- maximizing the expectation of the discounted cumulative rewards (called ``return''):

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

(γ can ensure convergence of the series)



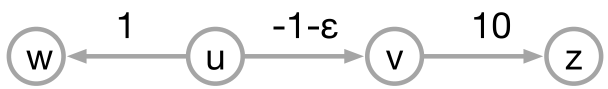
An example trajectory:

State, Action, reward

(1,1), Up, -0.04,
(1,2), Up, -0.04,
(1,3), Right, -0.04,
(2,3), Right, -0.04,
(3,3), Right, -0.04,
(4,3), NoAction, 1.

Return = 0.8 with γ=1.

The following MDP shows that it is important to maximize return, not immediate reward. Starting from state u, going left has a higher immediate reward than going right, which can lead to a high reward when reaching z.

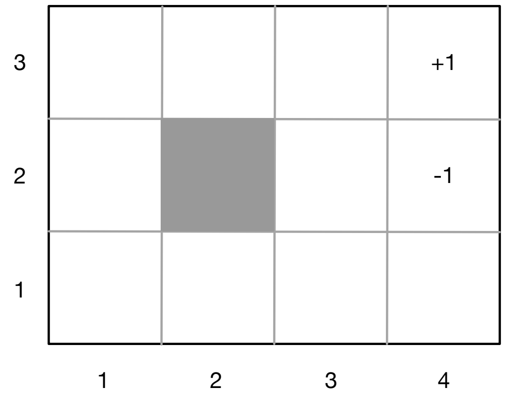


Policy

The agent interacts with the environment using a policy, which decides how to choose an action in a state.

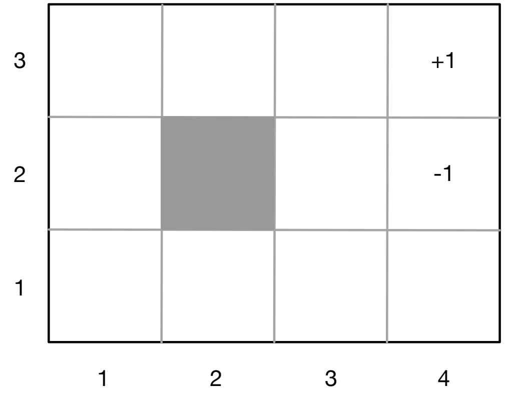
$$0 \le \pi(a|S_t) \le 1, \sum_{a \in \mathcal{A}(S_t)} \pi(a|S_t) = 1$$

- Example on the right:
  - a state is a position in the maze,
  - an action is a direction to go next.



A deterministic sequence of actions will fail in the face of environment uncertainty:

- there is a non-zero chance that the fixed action sequence (U, U, R, R, R) can lead to the undesirable state (4,2).



State-value functions

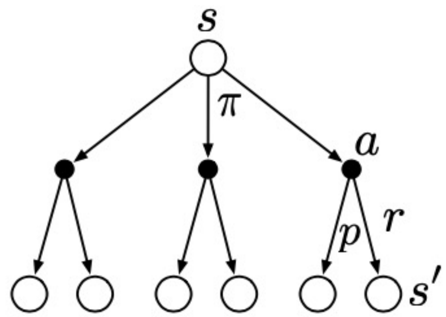
To optimize the policy, a policy is first evaluated by the state-value functions:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \ldots |S_t = s]$$

Expectation taken over all possible trajectories sampled according to the policy and the environment dynamics.

More explicitly,

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots |S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r \Pr(s', r|s, a)[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r \Pr(s', r|s, a)[r + \gamma v_\pi(s')] \\ &= \sum_a \sum_{s'} \sum_r \pi(a|s) \Pr(s', r|s, a)[r + \gamma v_\pi(s')]. \end{aligned}$$

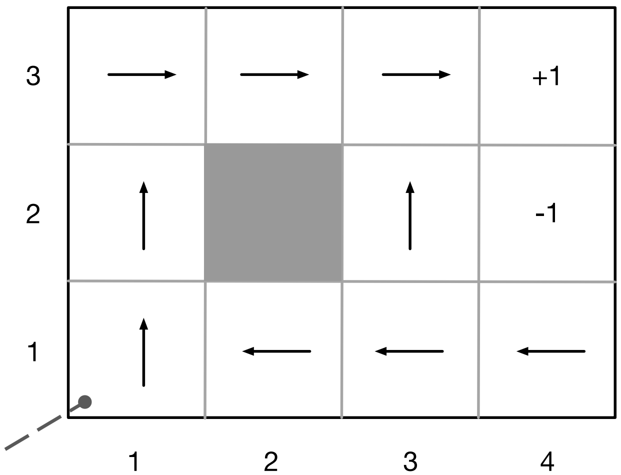


backup diagram

Bellman equation: $v_\pi(s) = \sum_r \Pr_\pi(r|s) r + \gamma \sum_{s'} \Pr_\pi(s'|s) v_\pi(s')$

State-value functions

For one policy π, there is one state-value function $v_\pi(s)$, defined on the state space $\mathcal{S}$.



A special case: each cell has an action according to a *deterministic* policy.

3	0.812	0.868	0.918	+1
2	0.762		0.660	-1
1	0.705	0.655	0.611	0.388
	1	2	3	4

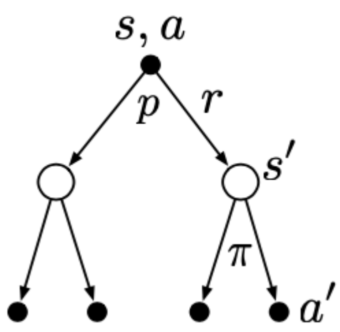
each cell s has $v_\pi(s)$

Action-value function

A related value function: expected return when taking action a at state s.

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots |S_t = s, A_t = a]$$

$$\begin{aligned}
q_\pi(s,a) &= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots |S_t = s, A_t = a] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a] \\
&= \sum_{s'}\sum_{r} \Pr(s', r|s,a)[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s', S_t = s, A_t = a] \\
&= \sum_{s'}\sum_{r} \Pr(s', r|s,a)[r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a')] \\
&= \sum_{r} \Pr(r|s,a) r + \gamma \sum_{s'}\sum_{a'} \Pr(s'|s,a)\pi(a'|s') q_\pi(s', a')].
\end{aligned}$$



backup diagram

Optimal policies

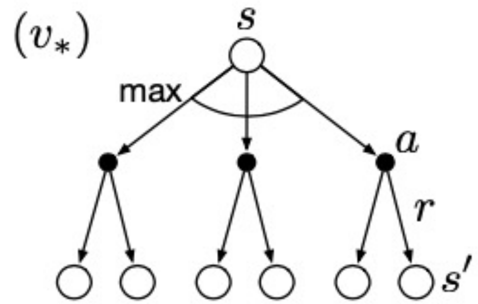
Intuitively, no matter where the agent is, the optimal policy always leads to the best expected return.

The agent wants to find the optimal policy π^*, defined as

$$v_{\pi^*}(s) \geq v_\pi(s), \quad \forall s \in \mathcal{S}, \quad \forall \pi.$$

Bellman optimality equation

$$v_{\pi^*}(s) = \max_{a \in \mathcal{A}(s)} \sum_{s',r} \Pr(s', r|s, a)[r + \gamma v_{\pi^*}(s')]$$



At any state s, the optimal policy must select the best action,

then follow the optimal policy from the successor state s', considered as a subproblem.

The value function $v_{\pi^*}(s')$ caches an optimal values of the subproblems.

The optimal action selection stores an optimal solutions.

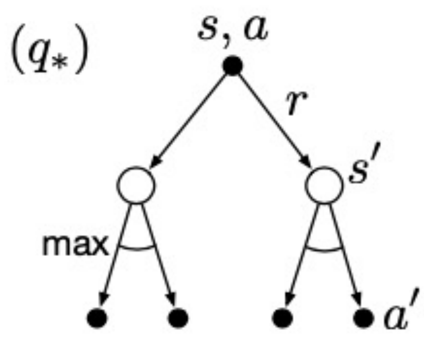
Optimal value function

Bellman optimality equation for the action-value function

$$q_{\pi^*}(s,a) = \sum_{s',r} \Pr(s',r)[r + \gamma \max_{a' \in \mathcal{A}(s')} q_{\pi^*}(s',a')]$$

Select the optimal action locally.

follow the same optimal policy in the future.



Relation to dynamic programming

Find the shortest path from the start node to the goal on a graph.

A dynamic programming algorithm will take advantage of

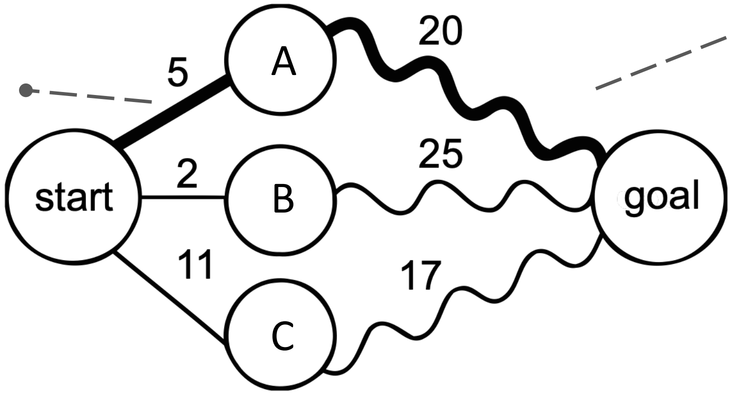
- optimal substructure
- overlapping subproblems

$$v_{\pi^*}(s) = \max_{a \in \mathcal{A}(s)} \sum_{s',r} \Pr(s', r | s, a)[r + \gamma v_{\pi^*}(s')]$$

Immediate "stochastic" cost:

$$-\sum_{s'} \sum_{r} \Pr(s', r | s, a) r$$

$$v_{\pi^*}(s')$$



Optimal substructure
This must be the shortest distance from node A to the goal. Otherwise, one could have found a shorter path.

Overlapping subproblem
This shortest path can be used in solving multiple larger problems.

Machine Learning

AIAA 5046, Spring 2024

Sihong Xie

AI Thrust

HKUST-GZ

<u>Goals:</u>

- Dynamic programming
- Policy iteration
- Value iteration

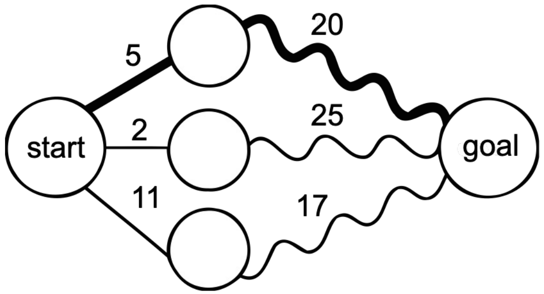
<u>Readings:</u>

RL Chap 3-6

Dynamic programming

If the model $\Pr(S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a)$ is known, use DP to

- evaluate a policy
- optimize a policy



Dynamic programming: optimization (*programming*) for

sequential decision making (*dynamic*).

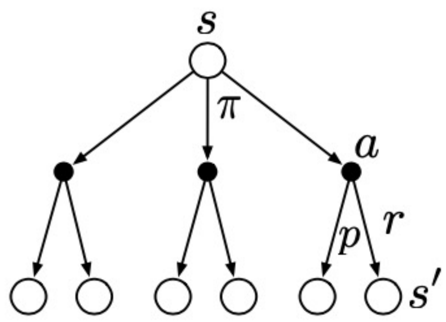
This is not reinforcement learning yet:

- RL learns from experiences of interactions.
- RL needs to explore the environment.

Assumptions:

1. known environment dynamics.
2. optimal substructure
   - principle of optimality
3. overlapping sub-problems.

Policy evaluation



Problem: find state-value function $v_\pi(s)$ of a policy π.

Solution: iteratively compute $v_\pi(s)$ using the Bellman equation.

$$v_\pi(s) = \sum_r \Pr_\pi(r|s)r + \gamma \sum_{s'} \Pr_\pi(s'|s)v_\pi(s')$$

```
for k=1, 2, …
        for all state s
```

$$v_{k+1}(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} \Pr(s',r|s,a)[r + \gamma v_k(s')]$$

Approximation

Synchronous update:

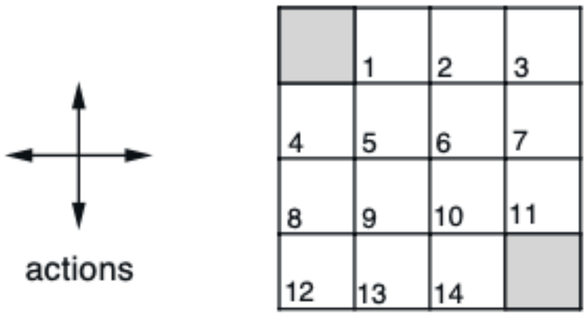
- maintain two arrays for the value function.
- compute the new values only after all old ones are done (two graphs for shortest path problem).
- one sweep can take long with many states.

Asynchronous update:

- update in-place: save space and more practical.
- use the latest value *immediately* to compute new values (one graph for shortest path problem).
- it is the foundation of Monte Carlo methods.

Policy evaluation

Example



$R_t = -1$
on all transitions

v_k for the random policy

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

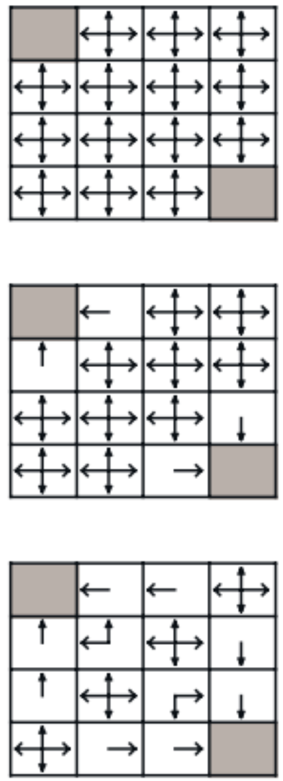
$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

greedy policy w.r.t. v_k



random policy

Excuting the following for each state per iteration:

$$v_{k+1}(s) \leftarrow \sum_{a} \pi(a|s) \sum_{s',r} \boxed{\Pr(s',r|s,a)} \left[r + \gamma v_k(s')\right]$$

Assumed deterministic

Policy evaluation

Example (continued)

$k = 3$

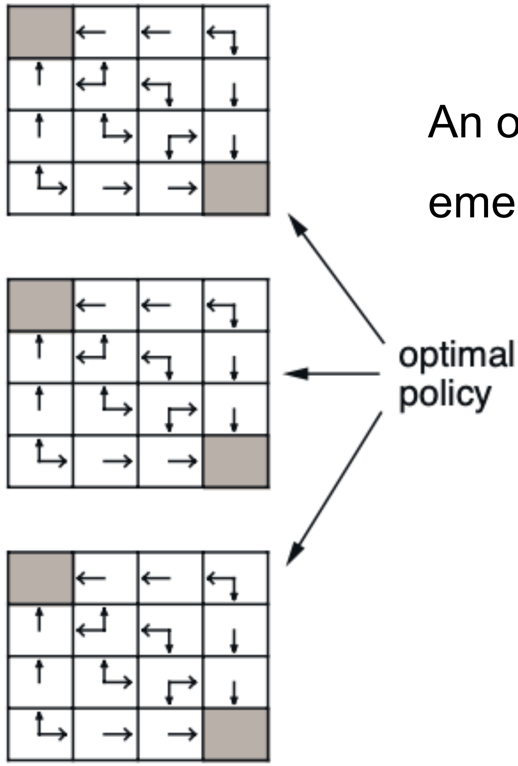
0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



An optimal policy emerges earlier on.

Converged value function for the random policy: each value is the negative of the expected number of steps to the absorbing states.

Policy improvement

After evaluating a policy π, the next step is to upgrade it to a better policy π'.

Assume deterministic policies $a = \pi(s)$.

Policy improvement will find the action

Environment dynamics must be known.

$$a \triangleq \pi'(s) = \arg\max_a q_\pi(s,a) = \arg\max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]$$

$$\Longrightarrow \quad q_\pi(s,\pi'(s)) = \max_a q_\pi(s,a) \geq \sum_a \pi(a|s) q_\pi(s,a) = v_\pi(s).$$

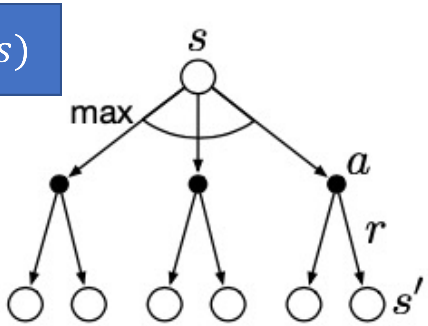
This is the value function of the previous policy π .

This update can be done at all state simultaneously.

Policy improvement

It can be proved that the new policy π' has a better state-value function than π .

$v_\pi(s)$



$$
\begin{aligned}
v_\pi(s) &\le q_\pi(s, \pi'(s)) \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \\
&\le \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}, A_{t+1} = \pi'(S_{t+1})] | S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) | S_t = s] \\
&\le \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 q_\pi(S_{t+2}, \pi'(S_{t+2})) | S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) | S_t = s] \\
&\le \cdots \\
&\le \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s] \\
&= v_{\pi'}(s).
\end{aligned}
$$

Policy iteration

Given an improved policy π', we can repeat the evaluation-improvement cycle to obtain a better policy π'', until convergence:

$$\pi_0 \xrightarrow{\mathrm{E}} v_{\pi_0} \xrightarrow{\mathrm{I}} \pi_1 \xrightarrow{\mathrm{E}} v_{\pi_1} \xrightarrow{\mathrm{I}} \pi_2 \xrightarrow{\mathrm{E}} \cdots \xrightarrow{\mathrm{I}} \pi_* \xrightarrow{\mathrm{E}} v_*.$$

Evaluation: given a policy, evaluate its value function.

Improvement: given an updated value function, select a greedy policy.

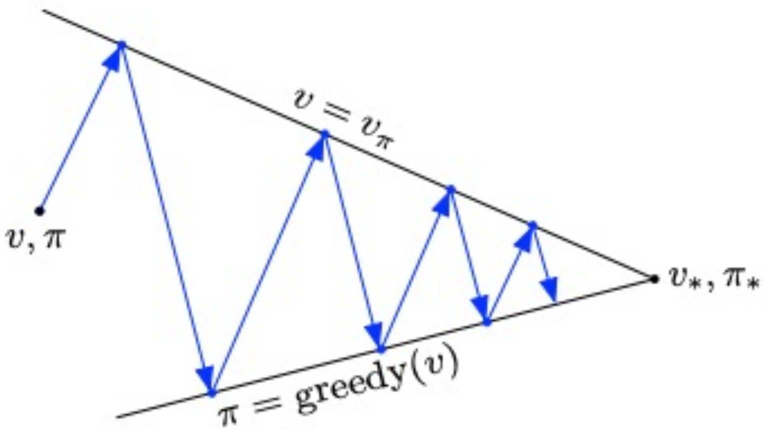
- Policy improvement theorem guarantees better policies.
- Contraction Mapping Theorem guarantees convergence.

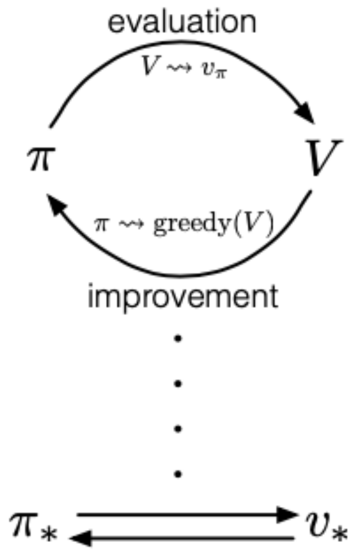
At convergence where $v_\pi(s)$ does not change, we must have the Bellman optimality equation:

$$v_{\pi^*}(s) = \max_{a \in \mathcal{A}(s)} \sum_{s',r} \Pr(s', r|s, a)[r + \gamma v_{\pi^*}(s')]$$

Generalized Policy Improvement

Generalized Policy Improvement (GPI):
any iterative and alternative policy evaluation and policy improvement.





Almost all reinforcement learning methods can be described as GPI.

Policy iteration

The policy evaluation step in policy iteration can take too long to converge. Policy improvement is possible even with a rough estimation of the value function.

Principle of optimality
$$v_{\pi^*}(s) = \underbrace{\max_{a \in \mathcal{A}(s)}}_{\text{Improve } v_{\pi*}(s)} \underbrace{\sum_{s',r} \Pr(s', r|s, a)[r + \gamma v_{\pi^*}(s')]}_{\text{Evaluating } q_{\pi*}(s,a)}$$

Turned into synchronous policy iteration equation:

$$\begin{aligned} v_{k+1}(s) \quad &\leftarrow \quad \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a] \\ &= \quad \max_a \sum_{s',r} \Pr(s', r|s, a)[r + \gamma v_k(s')]. \end{aligned}$$

May have not converged yet.

Turned into asynchronous policy iteration equation

$$v(s) = \max_a \sum_{s',r} \Pr(s', r|s, a)[r + \gamma v(s')].$$

Machine Learning
AIAA 5046, Spring 2024

Sihong Xie
AI Thrust
HKUST-GZ

Goals:
- Monte Carlo methods

Readings:
RL Chap 3-6

MC methods

Starting from here on, we are officially on reinforcement learning.

The dynamics $\Pr(S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a)$ is unavailable:

- blackjack: the player does not know the distribution of the cards;
- portfolio management (finance): market dynamics is unknown;
- Go games: unknown opponent's action that leads to the next state.

Monte Carlo methods estimate value functions using data (experiences / trajectories).

- agent interacts with the environment following some policy.
- the environment responds with a reward and a next state – a sample of the dynamics.

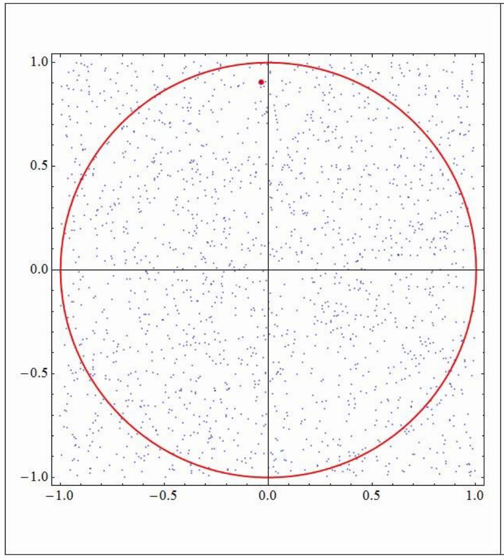
MC methods

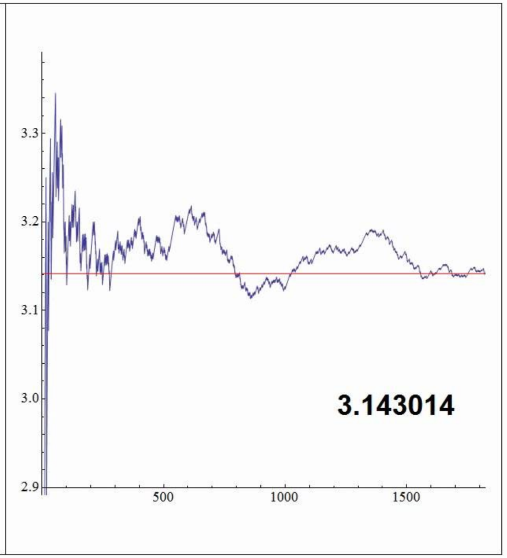
Monte Carlo is a computational tools to compute quantities that are hard to calculate in closed form.

Estimate the constant π:

1. draw a circle with radius = 1
2. generate *n* points in the square (area = 4) uniformly.
3. ratio of areas=π/4 ≈ N_inside / N_outside

The larger the n, the more accurate the estimation.





MC methods

Want to estimate the expectation $\mathbb{E}[X]$ using Monte Carlo: $\mathbb{E}[X] \approx \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$

Why?

- The distribution of X is unknown or hard to compute; or
- The expectation involves integrations of continuous variable.

For reinforcement learning, without the dynamics $\Pr(S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a)$

the state and action value functions are hard to compute.

- $v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s]$
- $q_\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s, A_t = a].$

We need them for policy improvement.

MC prediction (policy evaluation)

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

  $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

  $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

  Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$ ← Sampled under π and the unknown $\Pr(S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a)$

  $G \leftarrow 0$

  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

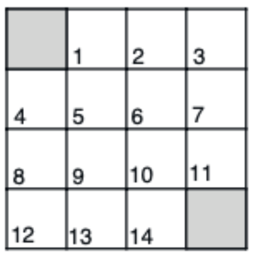
    $G \leftarrow \gamma G + R_{t+1}$

    Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$: ← Count first-visits only.

      Append G to $Returns(S_t)$

      $V(S_t) \leftarrow \text{average}(Returns(S_t))$ ← Approximate the expectation, using average.

actions



$R_t = -1$
on all transitions

Sample trajectories:

2, Left, -1, 1, Left, -1

6, Left, -1, 5, Up, -1, 1, Left, -1

14, Up, -1, 10, Down, -1, 14, Right, -1

MC prediction (policy evaluation)

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated

Initialize:

  $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$

  $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):

  Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$

  $G \leftarrow 0$

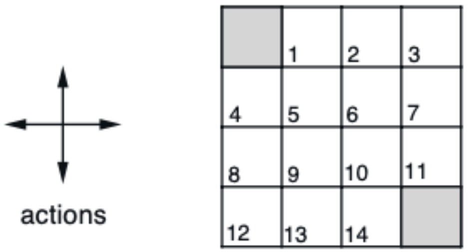
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

    $G \leftarrow \gamma G + R_{t+1}$

    ~~Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:~~ count every visit.

      Append G to $Returns(S_t)$

      $V(S_t) \leftarrow \text{average}(Returns(S_t))$



$R_t = -1$
on all transitions

Sample trajectories:

2, Left, -1, 1, Left, -1

6, Left, -1, 5, Up, -1, 1, Up, -1

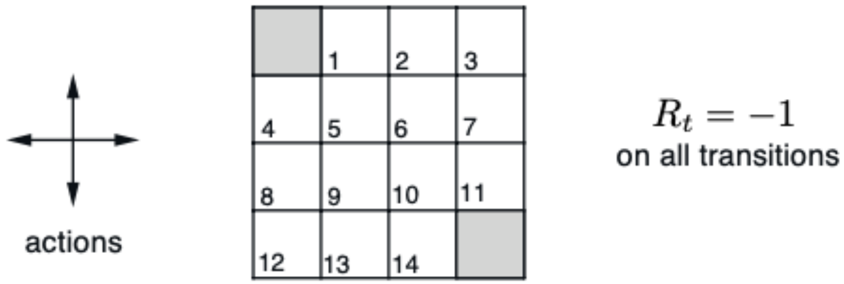
14, Up, -1, 10, Down, -1, 14, Right, -1

MC prediction (policy evaluation)

Remarks

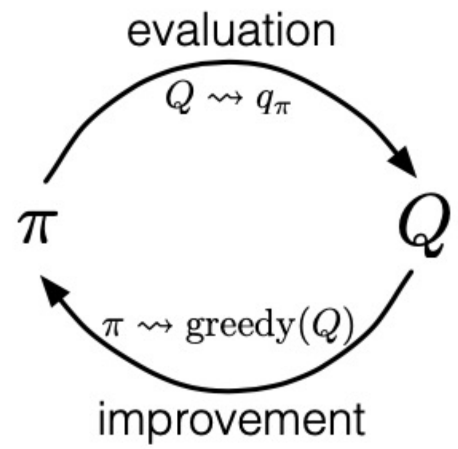
- advantage:
  - no exact environment dynamics is needed;
- disadvantages:
  - works only on episodic MDPs (finite T).
  - needs to wait until an episode ends to compute G.

  (in temporal difference, we relax these disadvantages).



Sample trajectories:
2, Left, -1, 1, Left, -1
6, Left, -1, 5, Up, -1, 1, Up, -1
14, Up, -1, 10, Down, -1, 14, Right, -1

MC control (policy optimization)



Use the Generalized Policy Iteration (GPI)

$$\pi_0 \xrightarrow{\text{E}} q_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} q_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} q_*.$$

MC prediction for the E-step:

- don't need to wait for evaluation to converge: too many trajectories are needed.
- have to work with the action-value function $q_\pi(s, a)$, since one needs to …

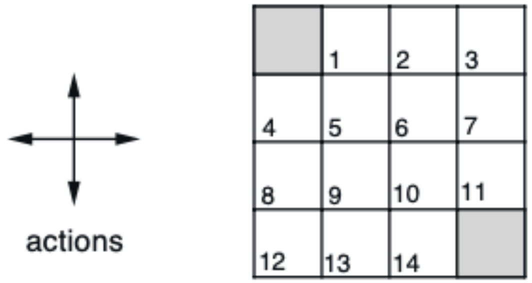
In the I-step, pick the optimal action at each state greedily to improve the policy.

- $a = \max_{a'} q_\pi(s, a')$
- Problematic: some action may need more data to get its true value reliably, and too many actions in reality

Exploration

Without visiting a state or a state-action pair, its value can’t be estimated.

- Need exploration to visit all states or state-action pairs to evaluate value functions.



$R_t = -1$
on all transitions

Sample trajectories:
2, Left, -1, 1, Left, -1, 0
6, Left, -1, 5, Up, -1, 1, Up, -1, 0
14, Up, -1, 10, Down, -1, 14, Right, -1, 0

Cannot estimate $q_\pi(1, right)$ since it is not sampled.

- How about starting from each of all states? Not easy for large state spaces.
- More importantly, some states (e.g., walls for a robot) are dangerous to visit.

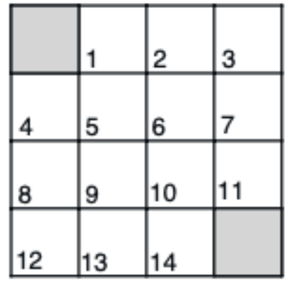
MC control with exploring starts (ES)

Monte Carlo ES (Exploring Starts), for estimating $\pi \approx \pi_*$

Initialize:
    $\pi(s) \in \mathcal{A}(s)$ (arbitrarily), for all $s \in \mathcal{S}$
    $Q(s, a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
    $Returns(s, a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Loop forever (for each episode):
    Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0
    Generate an episode from S_0, A_0, following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
            Append G to $Returns(S_t, A_t)$
            $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
            $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$

actions



$R_t = -1$
on all transitions

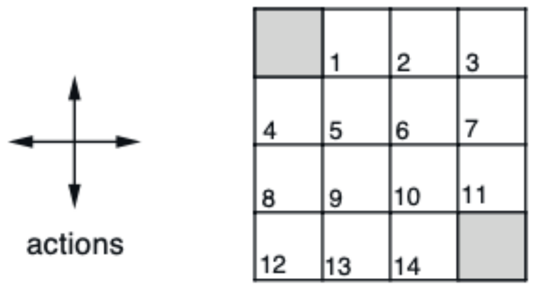
Sample trajectories:
2, Left, -1, 1, Left, -1, 0
6, Left, -1, 5, Up, -1, 1, Up, -1, 0
14, Up, -1, 10, Down, -1, 14, Right, -1, 0

ES is not realistic for large state-action space, since it needs to start from every (s, a) pair.

MC control with ε-greedy exploration

ε-greedy policies:

$$\pi'(a|s) = \begin{cases} \frac{\epsilon}{|\mathcal{A}|} + (1-\epsilon) & \text{if } a = \arg\max_{a'} q_\pi(a'|s), \\ \frac{\epsilon}{|\mathcal{A}|} & \text{for other actions.} \end{cases}$$



$R_t = -1$
on all transitions

So long as all states are visited, there is a non-zero probability that all (s, a) pairs will be visited in the sampled trajectories.

MC control with ε-greedy exploration

An ε-greedy policy improves the prior policy.

For any state s, the following inequality holds:

$$
\begin{aligned}
\mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] &= \sum_a \pi'(a|s) q_\pi(a, s) \\
&= \frac{\epsilon}{|\mathcal{A}|} \sum_a q_\pi(s, a) + (1 - \epsilon) \max_a q_\pi(s, a) \\
&\geq \frac{\epsilon}{|\mathcal{A}|} \sum_a q_\pi(s, a) + (1 - \epsilon) \sum_a \frac{\pi(a|s) - \epsilon/|\mathcal{A}|}{1 - \epsilon} q_\pi(s, a) \\
&= \sum_a \pi(a|s) q_\pi(s, a) \\
&= v_\pi(s).
\end{aligned}
$$

MC control with ε-greedy exploration

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$

Initialize:

$\pi \leftarrow$ an arbitrary ε-soft policy

$Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):

Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$

$G \leftarrow 0$

Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:

$G \leftarrow \gamma G + R_{t+1}$

Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:

Append G to $Returns(S_t, A_t)$

$Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$

$A^* \leftarrow \text{argmax}_a\, Q(S_t, a)$ (with ties broken arbitrarily)

For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

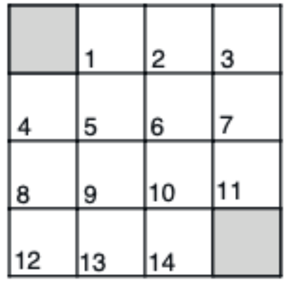
ε-greedy

Remarks:

- It is an on-policy MC control algorithm.
  - The policies being optimized and used to select actions are the same.
- ε must converge to 0 to obtain an optimal policy.
  - But if ε goes to 0 too fast, one loses exploration capability.

Off-policy MC control

actions



$R_t = -1$
on all transitions

On-policy control can lack exploration capability

- ε may go to 0 too soon before (almost) all state-action pairs are visited.

Use a behavior policy $b(a|s)$ that keeps on exploring

while learning the target policy $\pi(a|s)$

This is called "off-policy" reinforcement learning: can use data collected before.

Sample trajectories from $b(a|s)$ are biased samples of the distribution from $\pi(a|s)$

$$\sum_{i=1}^{m} G_t^{(i)} \approx \mathbb{E}_b[G_t|S_t = s, A_t = a] \neq \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$

Importance sampling (IS)

Estimate the mean of a random variable under one distribution,

using sample from another distribution.

This holds true always:

$$\sum_x P(x)x = \mathbb{E}_P[X] = \mathbb{E}_Q\left[\frac{P(X)}{Q(X)}X\right] = \sum_x Q(x)\frac{P(x)}{Q(x)}x$$

Monte Carlo version:

$$\underbrace{\sum_{i=1}^{m} x^{(i)}}_{\text{Sample from } P} \approx \mathbb{E}_P[X] = \mathbb{E}_Q\left[\frac{P(X)}{Q(X)}X\right] \approx \underbrace{\sum_{i=1}^{m} \frac{P(x)}{Q(x)}x^{(i)}}_{\text{Sample from } Q}$$

Importance sampling (IS)

Probability of a trajectory generated by $\pi(a|s)$

$$\Pr(S_t, A_t, R_{t+1}, S_{t+1}, \ldots, S_{T-1}, A_{T-1}, R_T) = \prod_{k=t}^{T-1} \pi(A_t|S_t)\Pr(S_{t+1}, R_{t+1}|S_t, A_t)$$

Probability of a trajectory generated by $b(a|s)$

$$\Pr(S_t, A_t, R_{t+1}, S_{t+1}, \ldots, S_{T-1}, A_{T-1}, R_T) = \prod_{k=t}^{T-1} b(A_t|S_t)\Pr(S_{t+1}, R_{t+1}|S_t, A_t)$$

Weight of the trajectory $\rho_{t:T-1} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$

The product of multiple probabilities leads to high variance: IS is more common in TD (discussed later) rather than MC.

Incremental update of value functions

Assuming action-value function is estimated using n returns;

with the $(n+1)$ -th return, the value function can be updated as:

$$
\begin{aligned}
q_\pi^{(n+1)}(s,a) &\approx \frac{1}{n+1}\sum_{i=1}^{n+1} g_{t(i)} \\
&= \frac{1}{n+1}\left(\sum_{i=1}^{n} g_{t(i)} + g_{t(n+1)}\right) \\
&= \frac{n}{n+1} q_\pi^{(n)}(s,a) + \frac{g_{t(n+1)}}{n+1} \\
&= \left(1 - \frac{1}{n+1}\right) q_\pi^{(n)}(s,a) + \frac{g_{t(n+1)}}{n+1} \\
&= q_\pi^{(n)}(s,a) + \frac{1}{n+1}[g_{t(n+1)} - q_\pi^{(n)}(s,a)]
\end{aligned}
$$

On the left, each observation has weight 1.

More generally, each return can be associated with a weight.

We will use the following general formula:

$$q_\pi(S_t, A_t) \leftarrow q_\pi(S_t, A_t) + \alpha[G_t - q_\pi(S_t, A_t)]$$

Off-policy MC control with IS

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s,a) \in \mathbb{R}$ (arbitrarily)
    $C(s,a) \leftarrow 0$
    $\pi(s) \leftarrow \arg\max_a Q(s,a)$ (with ties broken consistently)

Loop forever (for each episode):
    $b \leftarrow$ any soft policy
    Generate an episode using b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    $W \leftarrow 1$
    Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t,A_t)}[G - Q(S_t, A_t)]$
        $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken consistently)
        If $A_t \neq \pi(S_t)$ then exit inner Loop (proceed to next episode)
        $W \leftarrow W \frac{1}{b(A_t|S_t)}$

Weighted IS

$$q_\pi(s,a) \approx \frac{\sum_{t \in \mathcal{T}(s,a)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s,a)} \rho_{t:T(t)-1}}.$$

Incremental update

$$q_\pi(s,a) \leftarrow q_\pi(s,a) + \alpha([\text{target value}] - q_\pi(s,a))$$

Greedy is good enough and exploration handled by policy b.

Where is $\pi(A_t|S_t)$? It is already in the sampling distribution.

Machine Learning

AIAA 5046, Spring 2024

Sihong Xie

AI Thrust

HKUST-GZ

Goals:

- Temporal difference
- Function approximation

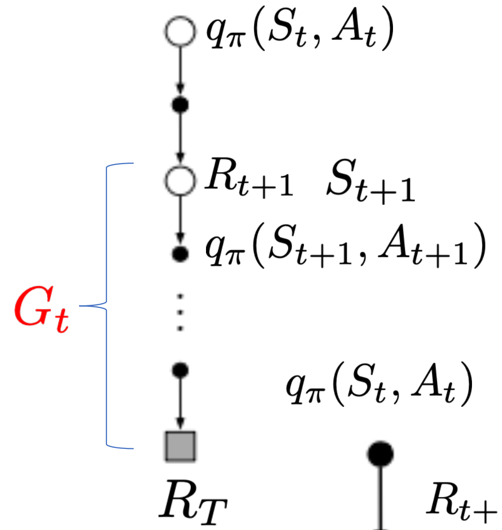
Readings:

RL Chap 3-6

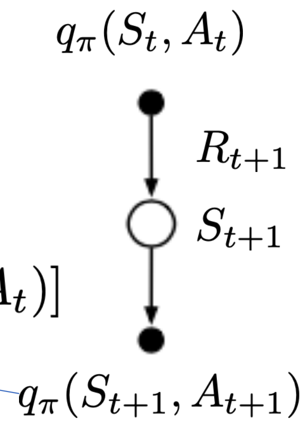
Temporal difference

Use the difference in the value estimation between two consecutive steps for updating value functions.

- does not need to wait until an episode to finish.
- can work with continuous (non-episodic) MDPs.
- in practice converges faster, as updates are seen immediately



In MC, we have $q_\pi(S_t, A_t) \leftarrow q_\pi(S_t, A_t) + \alpha[G_t - q_\pi(S_t, A_t)]$



In TD, we have $q_\pi(S_t, A_t) \leftarrow q_\pi(S_t, A_t) + \alpha[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) - q_\pi(S_t, A_t)]$

Temporal difference

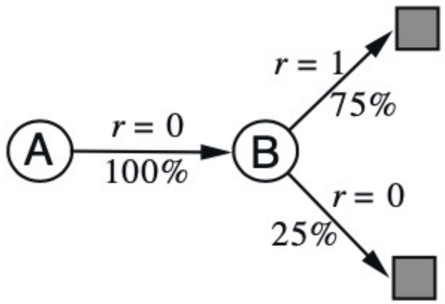
Example: given 8 trajectories sampled from an unknown MDP:

- Each state has only one action. State B: the environment has stochastic response.

Evaluate the policy by estimating the state-value function.

```
B, 0        B, 1
B, 1        B, 1
B, 1        B, 1
B, 1        A,0,B,0
```

TD also estimates this MDP:



- First-visit MC prediction: $q_\pi(B) = \frac{6}{8}$; $q_\pi(A) = 0$.
- TD prediction: $q_\pi(B) = \frac{6}{8}$; $q_\pi(A) = \frac{6}{8}$. (assuming learning rate and discount factor are both 1).
  - Information of action–value function propagate faster.

TD control: SARSA

On-policy TD control: learn from transitions $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize S
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Loop for each step of episode:
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q (e.g., ε-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal

need exploration to visit all (s, a) pairs.

ε must converge to 0

to find an optimal policy.

on-policy

TD control: Q-learning

Off-policy TD control:

- Behavior policy: ε-greedy policy derived from *Q*
- Target policy: the greedy policy derived from *Q*

Why there is no importance sampling weight?

Behavior policy is deterministic and $\pi(a^*|s) = 1$.

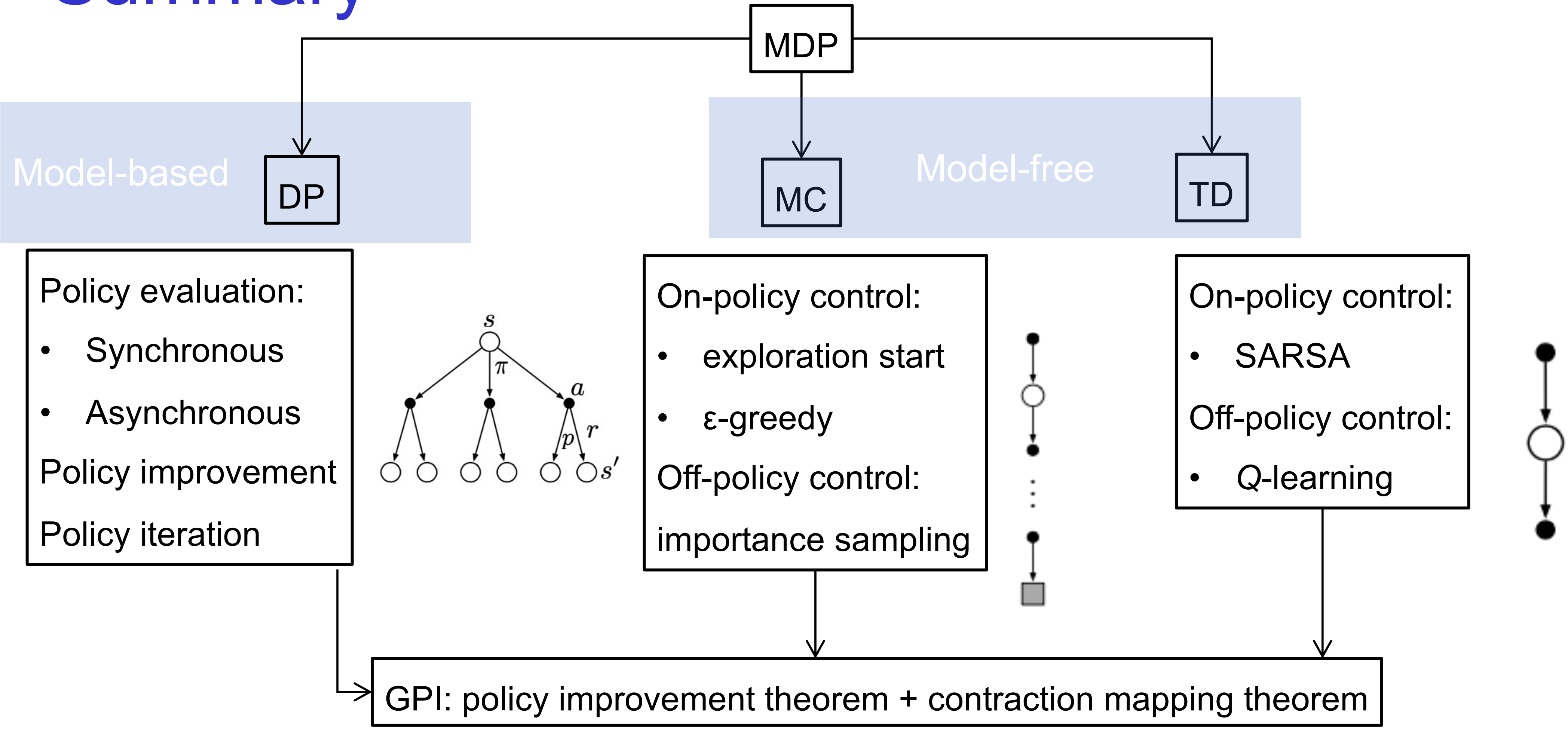
Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using policy derived from Q (e.g., ε-greedy)   the behavior policy
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
    until S is terminal

the target greedy policy

Summary

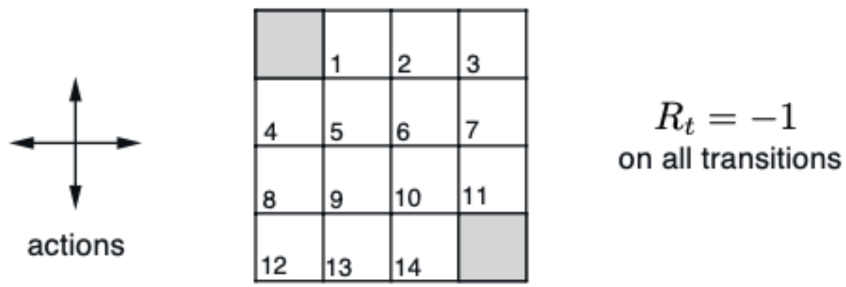


MDP
Model-based
DP
Model-free
MC
TD
Policy evaluation:
• Synchronous
• Asynchronous
Policy improvement
Policy iteration
s
π
a
p
r
s'
On-policy control:
• exploration start
• ε-greedy
Off-policy control:
importance sampling
On-policy control:
• SARSA
Off-policy control:
• Q-learning
GPI: policy improvement theorem + contraction mapping theorem

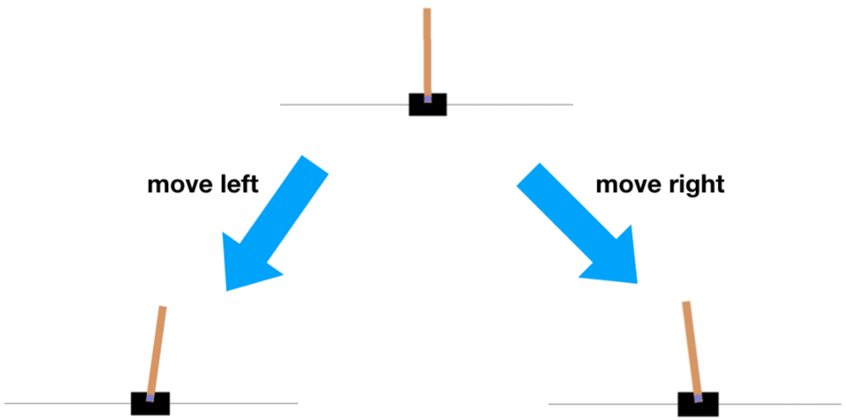
RL with function approximation

Motivations:

- Very large state spaces
  - Go game has 10^{170} states
- Tabular methods can't generalized.



Tabular methods use tables
to keep track of value functions



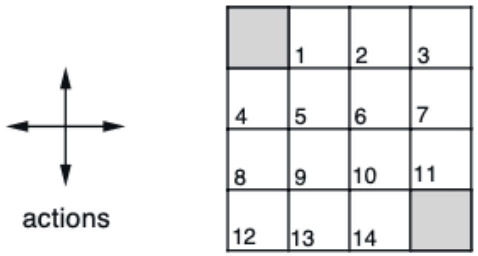
s	v(s)
1	0.1
2	5
…	…

(s, a)	q(s, a)
(1, Left)	0.1
(1, Right)	5
…	…

RL with function approximation

Approximate the value function using some supervised learning models

- capture the input-output relationships;
- can be trained as more input-output are observed;
- should support online learning and forgetting.



$R_t = -1$
on all transitions

what tabular methods will do:

$$v(S_t) \leftarrow R_{t+1} + \gamma v(S_{t+1})$$

s	v(s)
1	-1
2	-2
…	…

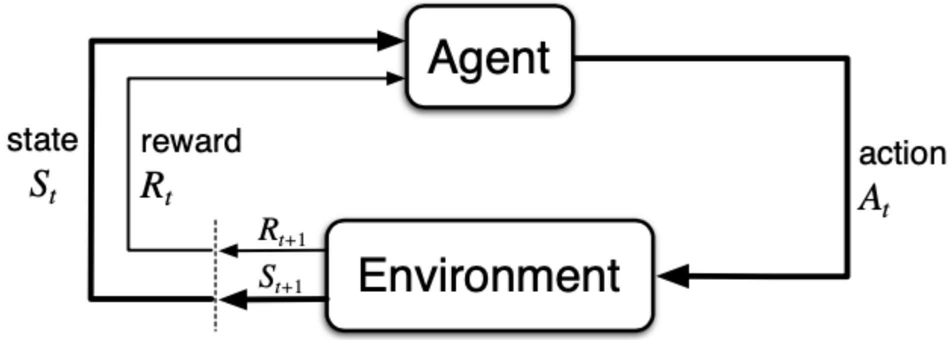
Sample trajectories:
2, Left, -1, 1, Left, -1, 0
6, Left, -1, 5, Up, -1, 1, Up, -1, 0
14, Up, -1, 10, Down, -1, 14, Right, -1, 0

Function approximation methods will minimize the MSE:

$$\sum_t [R_{t+1} + \gamma \hat{v}_\pi(S_{t+1}; \mathbf{w}) - \hat{v}_\pi(S_t; \mathbf{w})]^2$$

Online learning and forgetting

Episodes are generated during interaction



Stochastic gradient descent with MC:

$$\begin{aligned}\mathbf{w} &\leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} [v_\pi(S_t) - \hat{v}_\pi(S_t; \mathbf{w})]^2 \\ &= \mathbf{w} + \alpha [G_t - \hat{v}_\pi(S_t; \mathbf{w})] \nabla_{\mathbf{w}} \hat{v}_\pi(S_t; \mathbf{w})\end{aligned}$$

G_t depends on the value function parameters too.

Stochastic semi-gradient descent with TD:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha [R_{t+1} + \gamma \hat{v}_\pi(S_{t+1}; \mathbf{w}) - \hat{v}_\pi(S_t; \mathbf{w})] \nabla_{\mathbf{w}} \hat{v}_\pi(S_t; \mathbf{w})$$

Issues: non-stationary training data

- dependencies between two consecutive states.
- in GPI, the policy keeps changing.

Some degree of forgetting is necessary.

Linear models

Using linear model $\hat{v}_\pi(s;\mathbf{w}) = \mathbf{w}^\top \mathbf{x}(s)$

Stochastic gradient descent with MC:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha[G_t - \hat{v}_\pi(S_t;\mathbf{w})]\mathbf{x}(S_t)$$

Stochastic semi-gradient descent with TD:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha[R_{t+1} + \gamma\hat{v}_\pi(S_{t+1};\mathbf{w}) - \hat{v}_\pi(S_t;\mathbf{w})]\mathbf{x}(S_t)$$

No need to optimize the parameter so that

$$G_t = \hat{v}_\pi(S_t;\mathbf{w})$$

$$R_{t+1} + \gamma\hat{v}_\pi(S_{t+1};\mathbf{w}) = \hat{v}_\pi(S_t;\mathbf{w})$$

since:

- training data are non-stationary;
- may increase errors in other states.

Control with function approximation

Need to evaluate the action-state value function by minimizing

$$\sum_{s\in\mathcal{S},a\in\mathcal{A}} \mu_\pi(s,a)[q_\pi(s,a) - \hat{q}_\pi(s,a;\mathbf{w})]^2$$

Stochastic gradient descent with MC:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha[G_t - \hat{q}_\pi(S_t, A_t;\mathbf{w})]\nabla\hat{q}(S_t, A_t;\mathbf{w})$$

Stochastic gradient descent with on-policy TD (SARSA):

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha[R_{t+1} + \gamma\hat{q}_\pi(S_{t+1}, A_{t+1};\mathbf{w}) - \hat{q}_\pi(S_t, A_t;\mathbf{w})]\nabla\hat{q}_\pi(S_t, A_t;\mathbf{w})$$

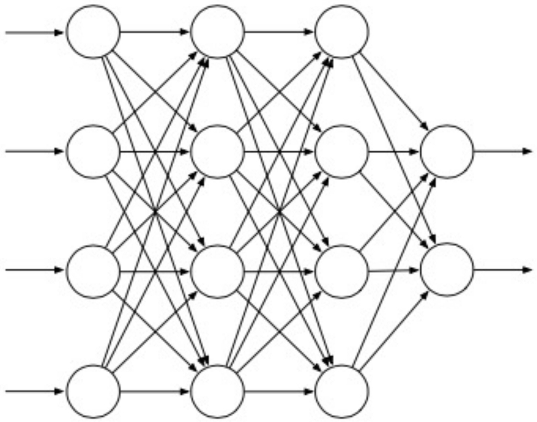
Stochastic gradient descent with off-policy TD (*Q*-learning):

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha[R_{t+1} + \gamma\max_a \hat{q}_\pi(S_{t+1}, a;\mathbf{w}) - \hat{q}_\pi(S_t, A_t;\mathbf{w})]\nabla\hat{q}_\pi(S_t, A_t;\mathbf{w})$$

Deep Q-network (DQN)

Fitting the action-value function using a neural network is not new.

State features



Values for two action

gradient of the output with respect to the network parameter

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha[R_{t+1} + \gamma \max_a \hat{q}_\pi(S_{t+1}, a; \mathbf{w}) - \hat{q}_\pi(S_t, A_t; \mathbf{w})]\nabla \hat{q}_\pi(S_t, A_t; \mathbf{w})$$

Deep Q-network (DQN)

Two challenges

- data dependencies => sample mini-batch from a large buffer of transitions.
- non-stationary target

w keeps moving

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha[R_{t+1} + \gamma \max_a \hat{q}_\pi(S_{t+1}, a; \mathbf{w}) - \hat{q}_\pi(S_t, A_t; \mathbf{w})]\nabla \hat{q}_\pi(S_t, A_t; \mathbf{w})$$

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha[R_{t+1} + \gamma \max_a \hat{q}_\pi(S_{t+1}, a; \mathbf{w}^{-1}) - \hat{q}_\pi(S_t, A_t; \mathbf{w})]\nabla_{\mathbf{w}} \hat{q}_\pi(S_t, A_t; \mathbf{w})$$

Target Q-network that is
not updated too frequent.

Deep Q-network (DQN)

Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For