University at Buffalo
Department of Computer Science and Engineering
School of Engineering and Applied Sciences

TABULAR METHODS & VALUE APPROXIMATION REVIEW

Lecture 18

CSE4/510: Reinforcement Learning

October 24, 2019

MDP

The [redacted] is acting in an [redacted] How the environment reacts to certain actions is defined by a [redacted] which we may or may not know. The agent can stay in one of many [redacted] of the environment, and choose to take one of many [redacted] to switch from one state to another. Which state the agent will arrive in is decided by the [redacted] between states $P(s'|s,a)$. Once an action is taken, the environment delivers a [redacted] as a feedback.

MDP

The agent is acting in an environment. How the environment reacts to certain actions is defined by a [redacted] which we may or may not know. The agent can stay in one of many [redacted] of the environment, and choose to take one of many [redacted] to switch from one state to another. Which state the agent will arrive in is decided by the [redacted] between states $P(s'|s,a)$. Once an action is taken, the environment delivers a [redacted] as a feedback.

MDP

The agent is acting in an environment. How the environment reacts to certain actions is defined by a model which we may or may not know. The agent can stay in one of many [redacted] of the environment, and choose to take one of many [redacted] to switch from one state to another. Which state the agent will arrive in is decided by the [redacted] between states $P(s'|s, a)$. Once an action is taken, the environment delivers a [redacted] as a feedback.

MDP

The agent is acting in an environment. How the environment reacts to certain actions is defined by a model which we may or may not know. The agent can stay in one of many states ($s \in S$) of the environment, and choose to take one of many [redacted] to switch from one state to another. Which state the agent will arrive in is decided by the [redacted] between states $P(s'|s, a)$. Once an action is taken, the environment delivers a [redacted] as a feedback.

MDP

The agent is acting in an environment. How the environment reacts to certain actions is defined by a model which we may or may not know. The agent can stay in one of many states ($s \in S$) of the environment, and choose to take one of many actions ($a \in A$) to switch from one state to another. Which state the agent will arrive in is decided by the between states $P(s'|s, a)$. Once an action is taken, the environment delivers a as a feedback.

MDP

The agent is acting in an environment. How the environment reacts to certain actions is defined by a model which we may or may not know. The agent can stay in one of many states ($s \in S$) of the environment, and choose to take one of many actions ($a \in A$) to switch from one state to another. Which state the agent will arrive in is decided by the transition probabilities between states $P(s'|s, a)$. Once an action is taken, the environment delivers a as a feedback.

MDP

The agent is acting in an environment. How the environment reacts to certain actions is defined by a model which we may or may not know. The agent can stay in one of many states ($s \in S$) of the environment, and choose to take one of many actions ($a \in A$) to switch from one state to another. Which state the agent will arrive in is decided by the transition probabilities between states $P(s'|s, a)$. Once an action is taken, the environment delivers a reward ($r \in R$) as a feedback.

MDP

1. G_t
2. $Q^\pi(s,a)$
3. $\pi(a|s)$
4. $V^\pi(s)$
5. $\pi(s)$

A $\mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | S_t = s]$

B $\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$

C $\mathbb{E}_\pi[R_t | S_t = s]$

D $\mathbb{E}_\pi[R_t | S_t = s, A_t = a]$

E $\mathbb{P}_\pi[A = a | S = s]$

F a

MDP

1. G_t

2. $Q^{\pi}(s, a)$

3. $\pi(a|s)$

4. $V^{\pi}(s)$

5. $\pi(s)$

B $\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$

D $\mathbb{E}_{\pi}[R_t|S_t = s, A_t = a]$

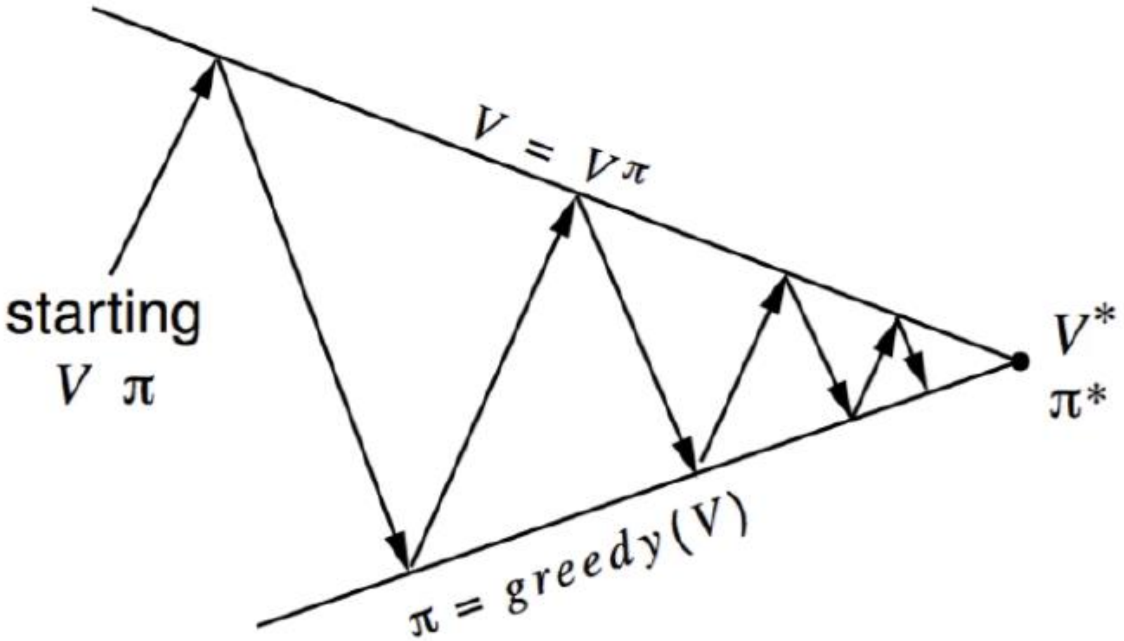
E $\mathbb{P}_{\pi}[A = a|S = s]$

C $\mathbb{E}_{\pi}[R_t|S_t = s]$

A $\mathbb{E}_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|S_t = s]$

F a

Dynamic Programming



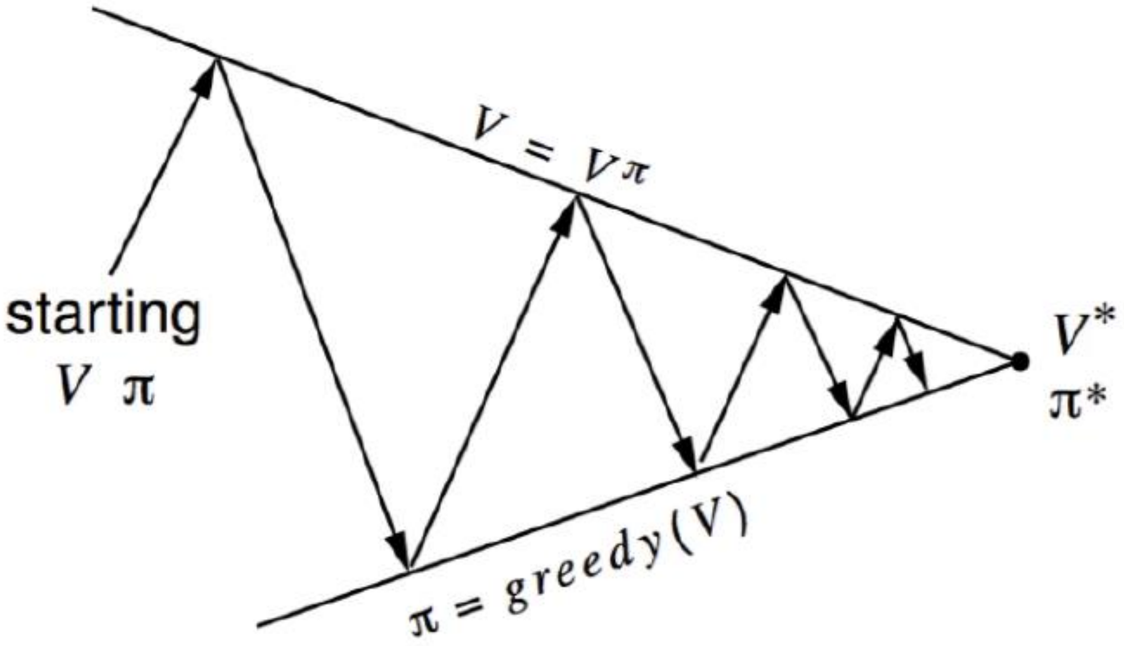
Estimate v_π

Iterative policy evaluation

Generate $\pi' \geq \pi$

Greedy policy improvement

Dynamic Programming



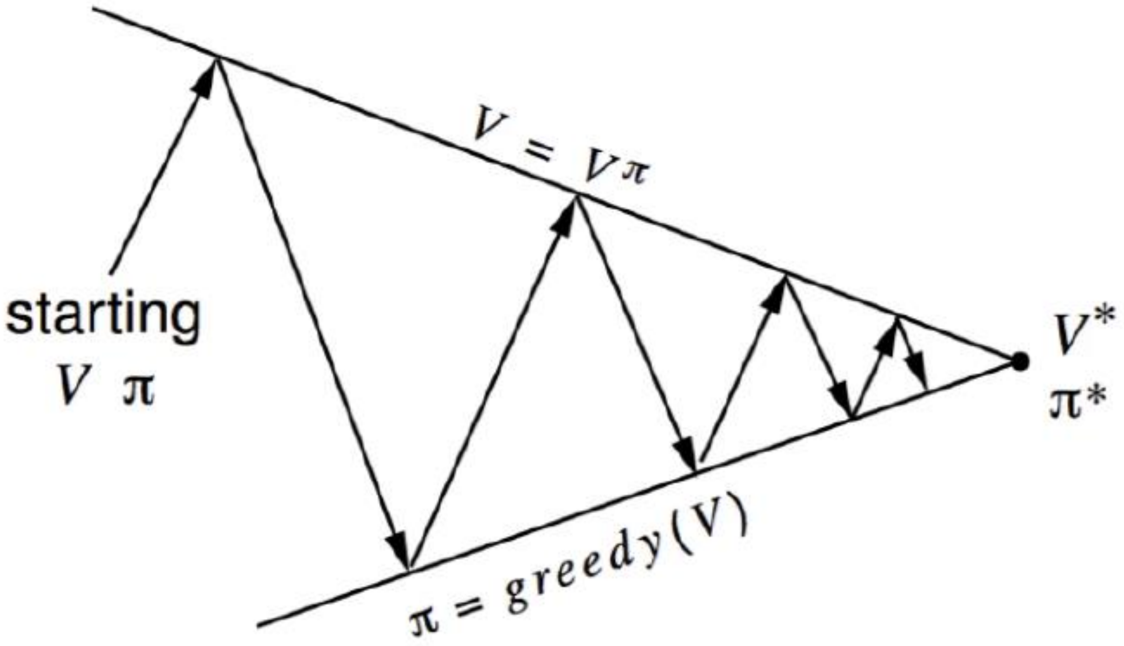
Policy evaluation Estimate v_π

Iterative policy evaluation

Generate $\pi' \geq \pi$

Greedy policy improvement

Dynamic Programming



Policy evaluation Estimate v_π

Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$

Greedy policy improvement

Dynamic Programming

1. Policy Evaluation

2. Policy Improvement

A $\mathrm{argmax}_a \sum_{s',r} p(s', r \,|\, s, a) \left[r + \gamma V(s')\right]$

B $\sum_{s',r} p(s', r \,|\, s, \pi(s)) \left[r + \gamma V(s')\right]$

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
   $\quad \Delta \leftarrow 0$
   $\quad$ Loop for each $s \in \mathcal{S}$:
   $\quad\quad v \leftarrow V(s)$
   $\quad\quad \boxed{V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\big[r + \gamma V(s')\big]}$
   $\quad\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $\quad$ *old-action* $\leftarrow \pi(s)$
   $\quad \boxed{\pi(s) \leftarrow \operatorname{argmax}_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]}$
   $\quad$ If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Dynamic Programming

Policy	Model	Pros	Cons	Applications
On Policy	Model Based	• It finds optimal policies in polynomial time for most cases • Guaranteed to find optimal policy	• Requires the knowledge of the transition probability this is an unrealistic requirement for many problems	Can be applied to environment for which the state transition probability is known

Distribution vs Sample Model

1. Distribution model

A. Produce a single outcome taken according to its probability of occurring

2. Sample model

B. List all possible outcomes and their probabilities

Optimal Functions

1 $\pi'(s)$

2 $V^*(s)$

3 $Q^*(s)$

A $\max_a \mathbb{E}[R_{t+1} + \gamma V^*(S_{t+1}) | S_t = s, A_t = a]$

B $\arg\max_a \mathbb{E}\left[r_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s, A_t = a\right]$

C $\mathbb{E}[R_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a') | S_t = s, A_t = a]$

Optimal Functions

1 $\pi'(s)$ **B** $\arg\max_a \mathbb{E}[r_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s, A_t = a]$

2 $V^*(s)$ **A** $\max_a \mathbb{E}[R_{t+1} + \gamma V^*(S_{t+1}) | S_t = s, A_t = a]$

3 $Q^*(s)$ **C** $\mathbb{E}[R_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a') | S_t = s, A_t = a]$

Update Functions

1. Dynamic Programming
2. Monte Carlo
3. Q-learning
4. Temporal Difference

A. $\alpha\left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$

B. $\sum_a \pi(a|S_t) \sum_{s',r} p(s', r|S_t, a)[r + \gamma V(s')]$

C. $\alpha\left(G_t - V(S_t)\right)$

D. $\alpha\left(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\right)$

Update Functions

1. Dynamic Programming B. $\sum_a \pi(a|S_t) \sum_{s',r} p(s',r|S_t,a)[r + \gamma V(s')]$

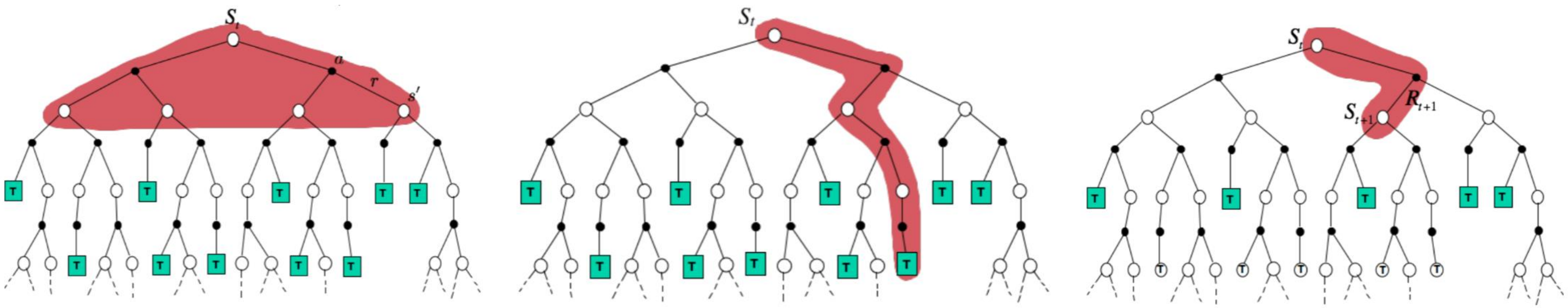
2. Monte Carlo C. $\alpha\left(G_t - V(S_t)\right)$

3. Q-learning D. $\alpha\left(r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\right)$

4. Temporal Difference A. $\alpha\left(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right)$

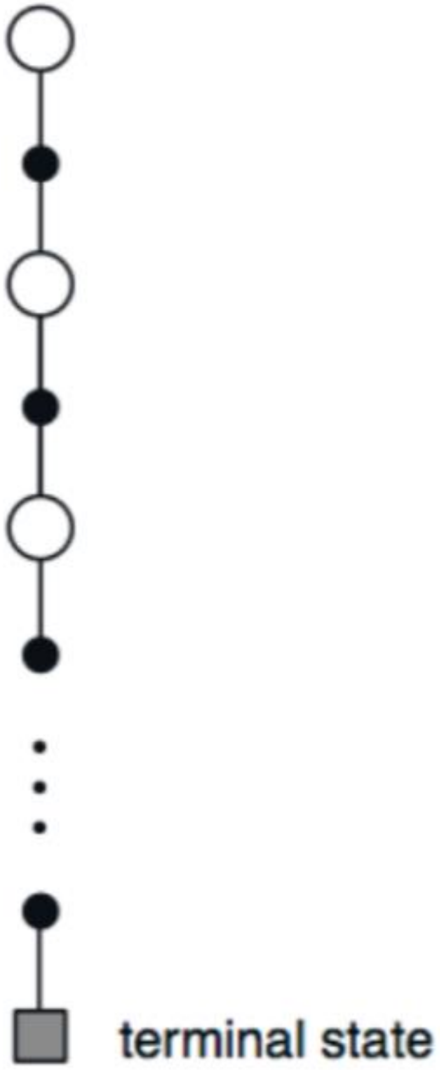
Overview

MC / DP / TD ?



Monte Carlo

- Entire rest of episode included
- Only one choice considered at each state (unlike DP)
  - thus, there will be an explore/exploit dilemma
- Does not bootstrap from successor state's values (unlike DP)
- Value is estimated by mean return
- Time required to estimate one state does not depend on the total number of states



Monte Carlo

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 $S_0, S_1, \ldots, S_{t-1}$:

      Append G to $Returns(S_t)$

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

Monte Carlo

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 $Returns(S_t, A_t)$

$\boxed{\begin{array}{l} Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t)) \\ \pi(S_t) \leftarrow \arg\max_a Q(S_t, a) \end{array}}$

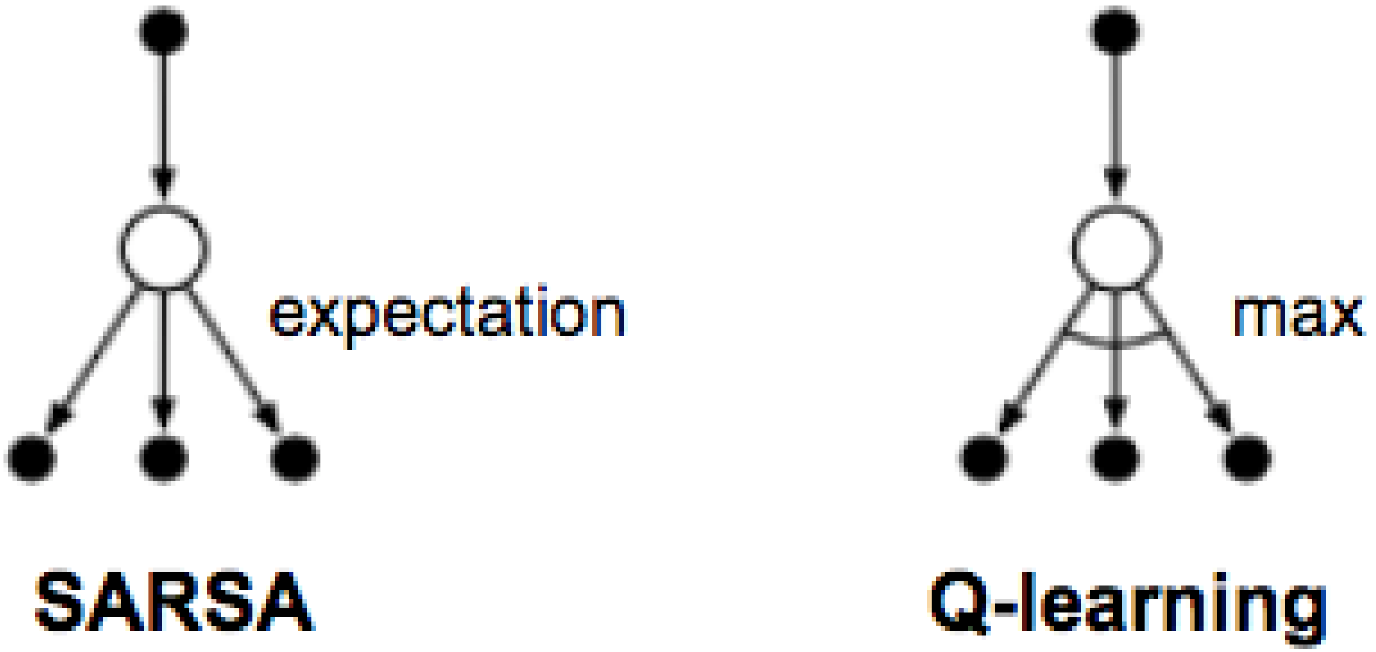
Monte Carlo

Policy	Model	Pros	Cons	Applications

Monte Carlo

Policy	Model	Pros	Cons	Applications
On Policy	Model Free	• Learn optimal behavior directly from interaction with the environment • Can be used to focus on the region of special interest and be accurately evaluated	• Must have the terminal state • Must wait until the end of an episode before return is known. For problems with very long episodes this will become too slow	It couldn't be used on continues task, should be episodic

SARSA vs Q-learning



SARSA

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)
        $\boxed{Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma Q(S', A') - Q(S, A)\big]}$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until S is terminal

SARSA

Policy	Model	Pros	Cons	Applications

Q-learning

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)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[\underbrace{\overbrace{\underset{\text{Immediate Reward}}{R} + \gamma \max_a Q(S', a)}^{\text{Target}} - \overbrace{Q(S, A)}^{\text{Prediction}}}_{\text{loss}}\big]$
        $S \leftarrow S'$
    until S is terminal

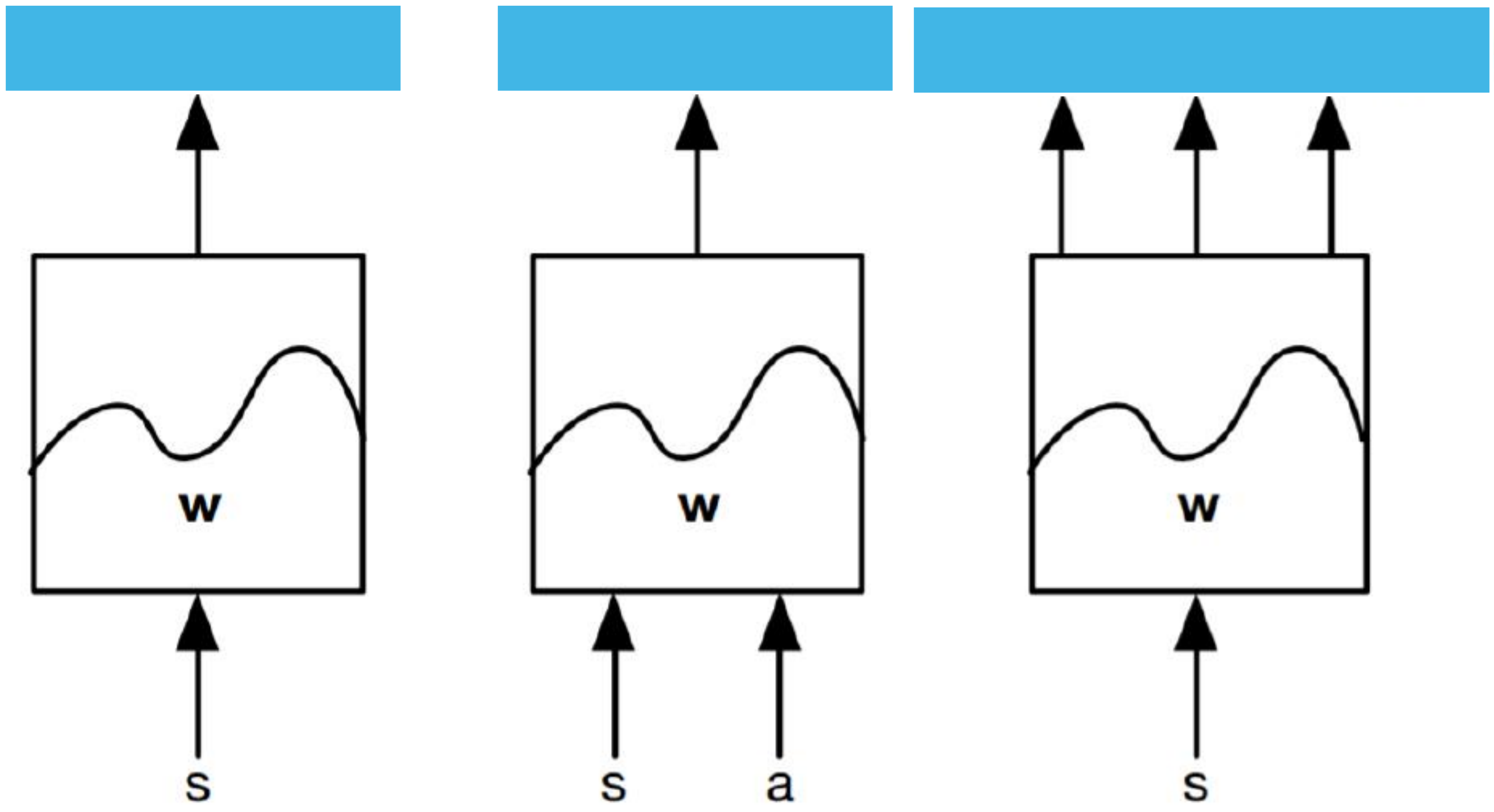
Q-Learning

Policy	Model	Pros	Cons	Applications

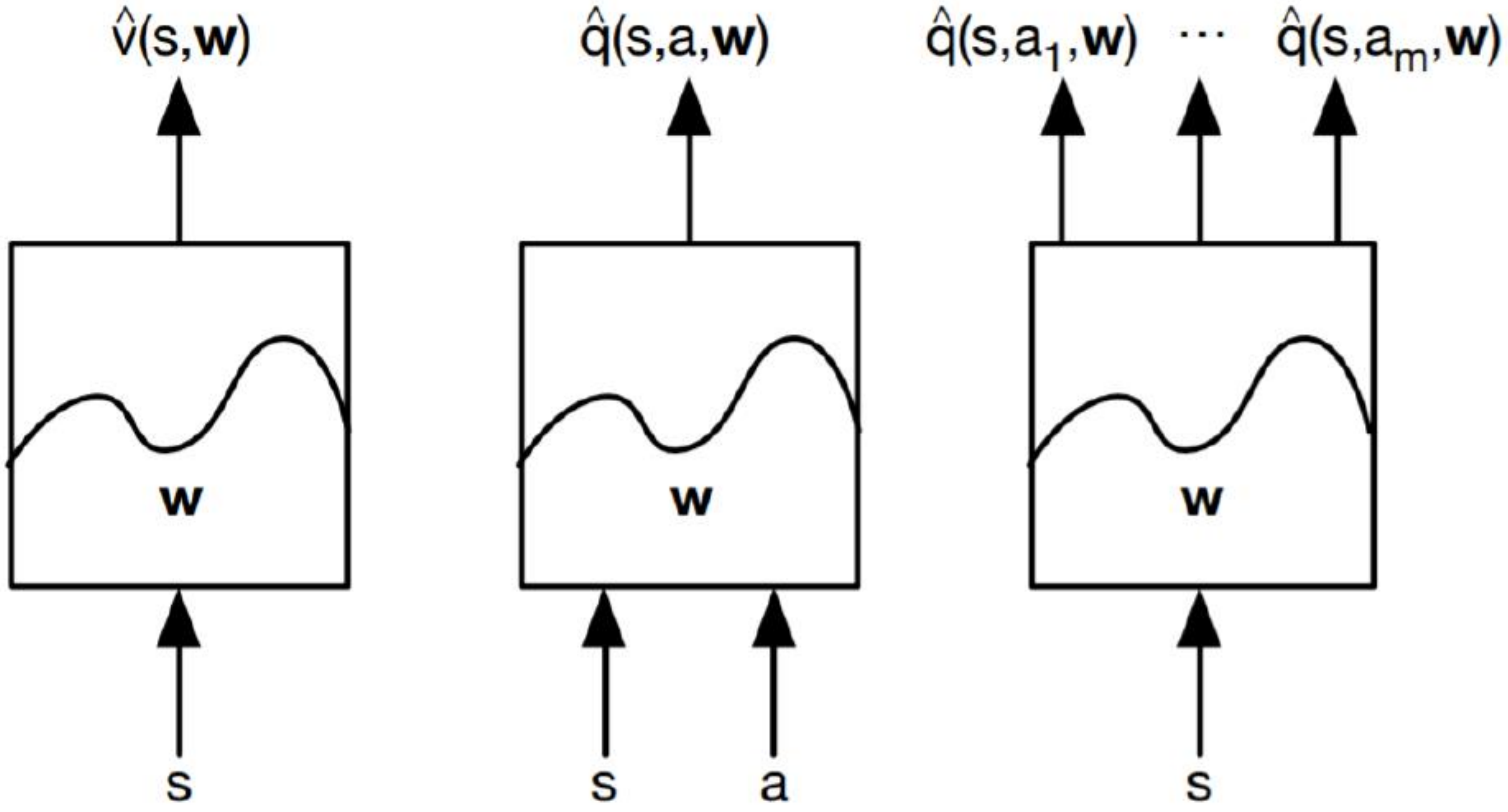
Q-Learning

Policy	Model	Pros	Cons	Applications
Off Policy	Model Free	Easy to implement	• Memory requirement increases with number of states • Does not perform well in stochastic environment	Environment with limited number of states and discrete action spaces

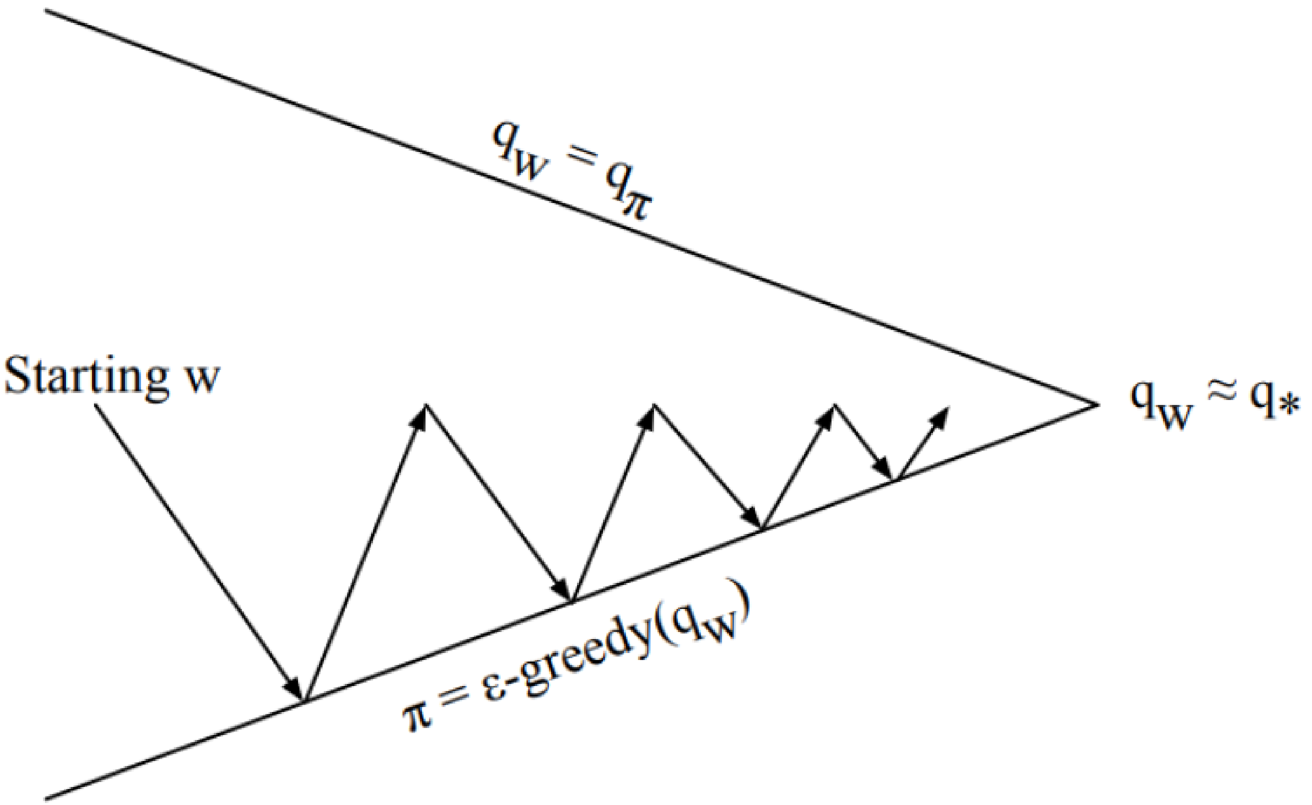
Function Approximation



Function Approximation



Function Approximation



Policy evaluation Approximate policy evaluation, $\hat{q}(\cdot, \cdot, \mathbf{w}) \approx q_\pi$

Policy improvement ϵ-greedy policy improvement

Function Approximation

- Represent value function by a linear combination of features

$$\hat{V}(S, \mathbf{w}) = x(S)^T \mathbf{w} = \sum_{j=1}^{n} x_j(S) \mathbf{w}_j$$

Function Approximation

- Represent value function by a linear combination of features

$$\hat{V}(S, \mathbf{w}) = x(S)^T \mathbf{w} = \sum_{j=1}^{n} x_j(S) \mathbf{w}_j$$

- Objective function is quadratic in parameters $\mathbf{w}$

$$J(\mathbf{w}) = E_\pi \left[(V_\pi(S) - x(S)^T \mathbf{w})^2 \right]$$

Function Approximation

- Represent value function by a linear combination of features

$$\hat{V}(S, \mathbf{w}) = x(S)^T \mathbf{w} = \sum_{j=1}^{n} x_j(S)\mathbf{w}_j$$

- Objective function is quadratic in parameters $\mathbf{w}$

$$J(\mathbf{w}) = E_\pi\left[(V_\pi(S) - x(S)^T\mathbf{w})^2\right]$$

- Stochastic gradient descent converges on global optimum
- Update rule is particularly simple

$$\nabla_{\mathbf{w}} \hat{V}(S, \mathbf{w}) = x(S)$$

$$\Delta\mathbf{w} = \alpha(V_\pi(S) - \hat{V}(S, \mathbf{w}))x(S)$$

Update = step-size x prediction error x feature value

Deep Q-network

- Represent value function by deep Q-network with weights w

$$Q(s, a, w) \approx Q^{\pi}(s, a)$$

- Define objective function

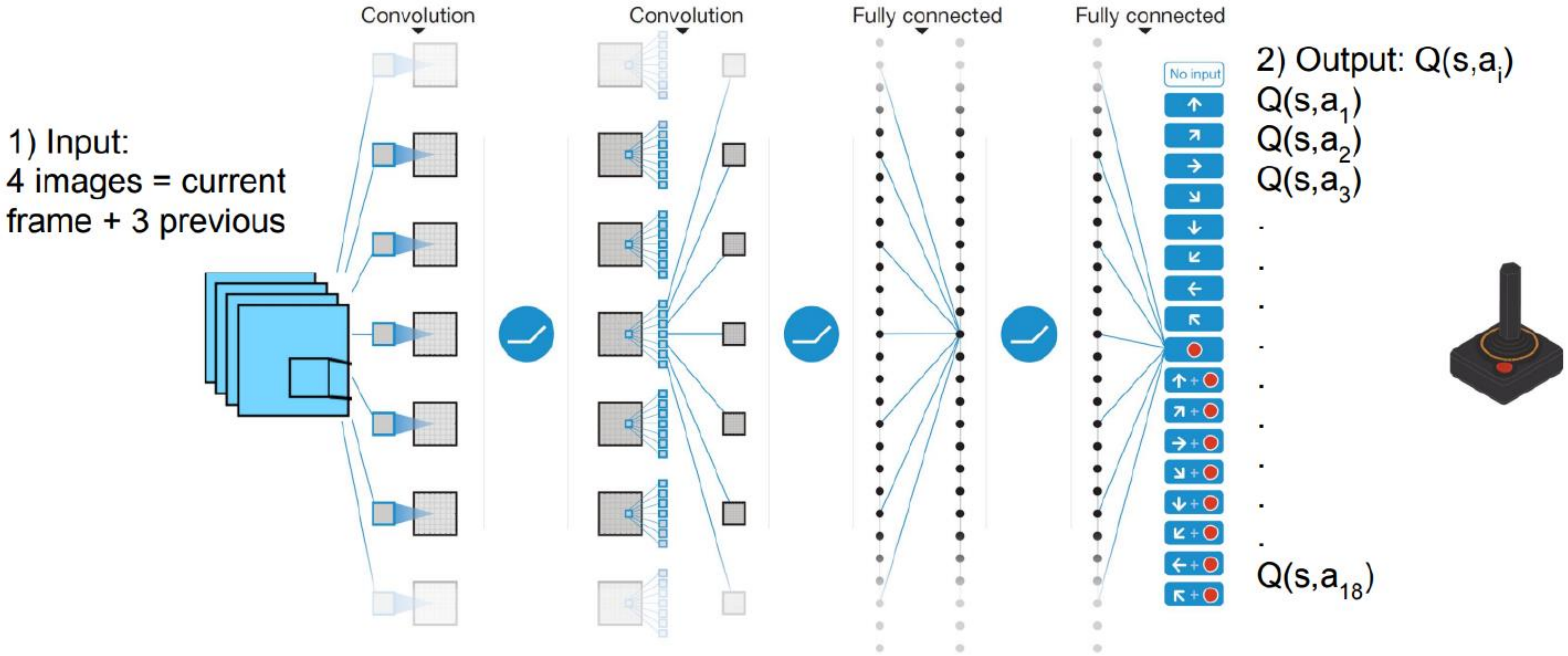
$$\mathcal{L}(w) = \mathbb{E}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a', w)}_{\text{target}} - Q(s, a, w)\right)^2\right]$$

- Leading to the following Q-leaning gradient

$$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w)\right) \frac{\partial Q(s, a, w)}{\partial w}\right]$$

- Optimize objective end-to-end by SGD, using $\frac{\partial L(w)}{\partial w}$

Deep Q-network



Deep Q-network

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 $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$

    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 $\left(\phi_t, a_t, r_t, \phi_{t+1}\right)$ in D

    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For

Deep Q-network (DQN)

Policy	Model	Pros	Cons	Applications

Deep Q-network (DQN)

Policy	Model	Pros	Cons	Applications
Off Policy	Model Free	• Can generalize to unseen states • Input is just a state	• It may over-estimate value • Cannot be applicable to continuous action spaces	Environment with limited number of states and discrete action spaces

Double Deep Q-network

Two estimators:

- Estimator Q_1: Obtain best actions
- Estimator Q_2: Evaluate Q for the above action

$$Q_1(s, a) \leftarrow Q_1(s, a) + \alpha(\text{Target} - Q_1(s, a))$$

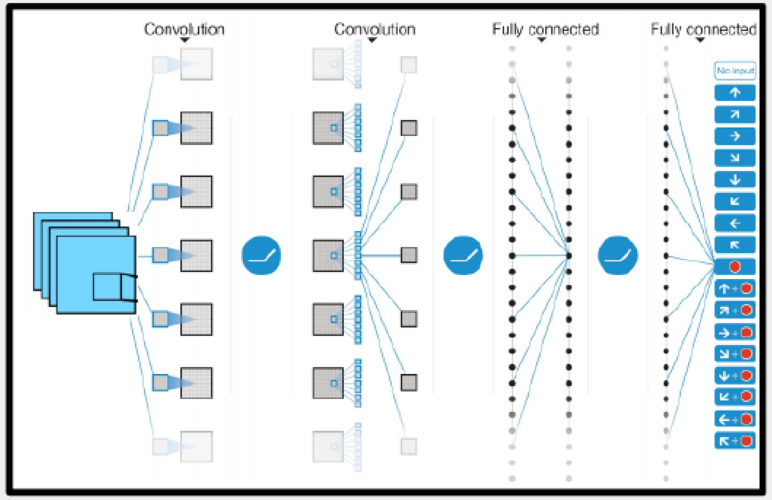
Q Target: $r(s, a) + \gamma \max_{a'} Q_1(s', a')$

Double Q Target: $r(s, a) + \gamma Q_2(s', \arg\max_{a'}(Q_1(s', a')))$

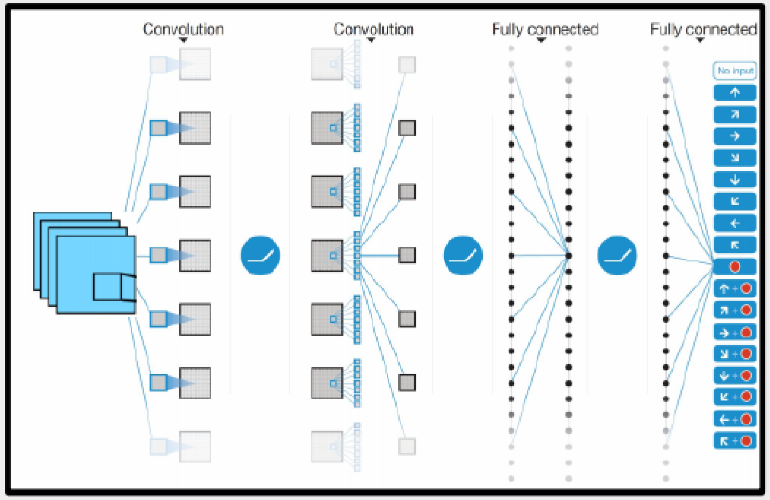
Double Deep Q-network

Two estimators:

- Estimator Q_1: Obtain best actions
- Estimator Q_2: Evaluate Q for the above action



Q1



Q2

Double Deep Q-network

$$Y_t^{\text{DoubleDQN}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \operatorname*{argmax}_a Q(S_{t+1}, a; \boldsymbol{\theta}_t), \boldsymbol{\theta}_t^-)$$

Double Deep Q-network

Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function $\boxed{Q}$ with random weights θ
Initialize target action-value function $\boxed{\hat{Q}}$ 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 $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
    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 $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ \boxed{R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\mathrm{argmax}}\, Q(S_{t+1}, a; \boldsymbol{\theta}_t), \boldsymbol{\theta}_t^{-})} & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Double Deep Q-network (DDQN)

Policy	Model	Pros	Cons	Applications

Double Deep Q-network (DDQN)

Policy	Model	Pros	Cons	Applications
Off Policy	Model Free	• Value estimation is more accurate comparing to DQN • Input is just a state	• It may take longer to train	Environment with limited number of states and discrete action spaces

Dueling Deep Q-network (Dueling DQN)

How can we decompose $Q^{\pi}(s, a)$?

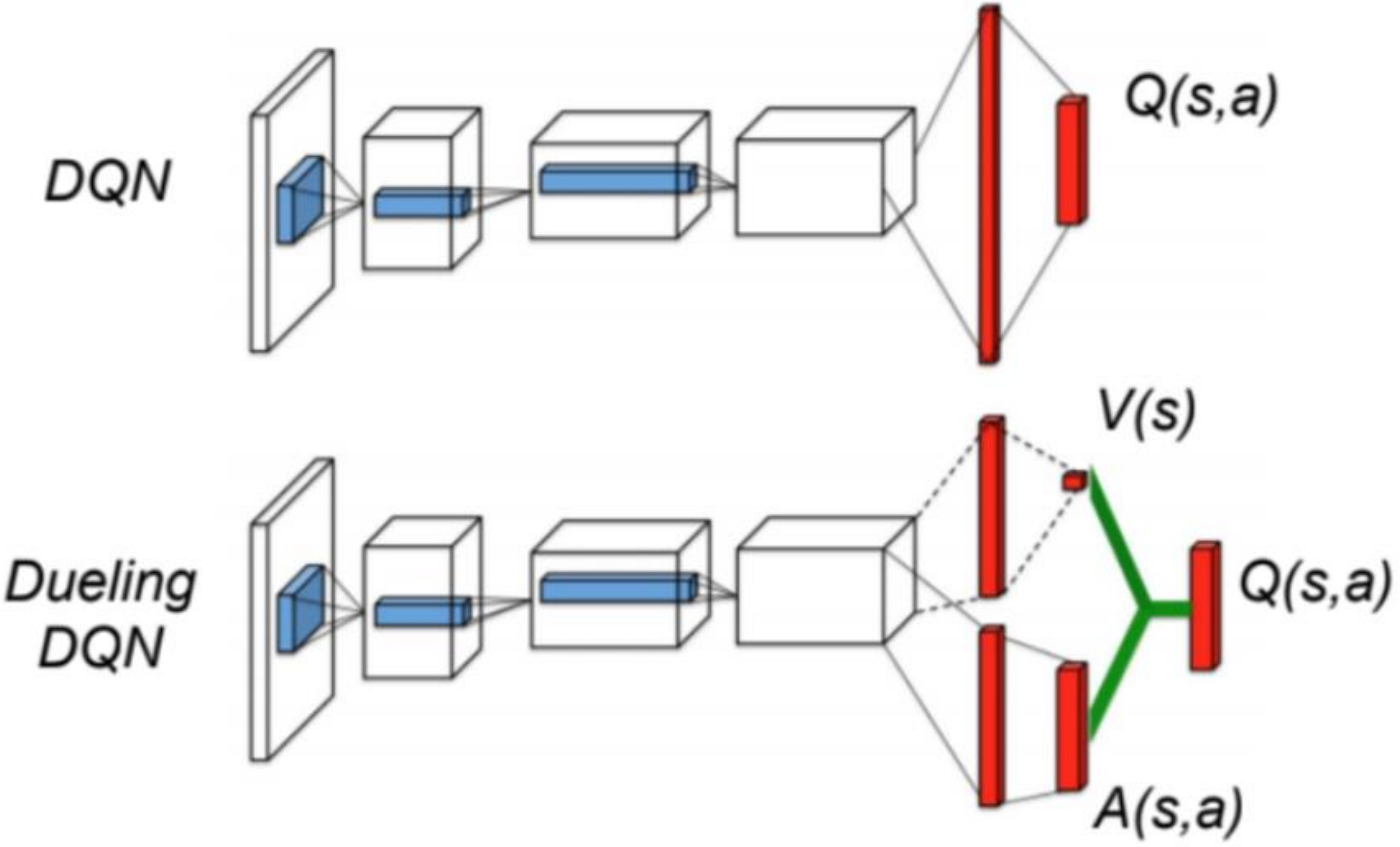
$$Q^{\pi}(s, a) = V^{\pi}(s) + A^{\pi}(s, a)$$
$$V^{\pi}(s) = E_{a \sim \pi(s)}[Q^{\pi}(s, a)]$$

In Dueling DQN, we separate the estimator of these two elements, using two new streams:

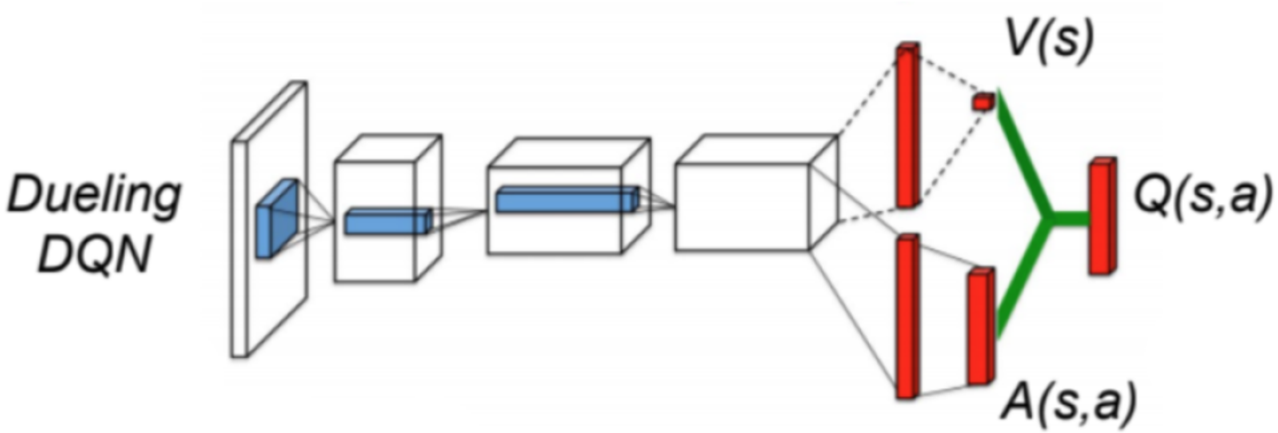
- one estimates the state value $V(s)$
- one estimates the advantage for each action $A(s, a)$

Networks that separately computes the advantage and value functions, and combines back into a single Q-function at the final layer.

Dueling Deep Q-network (Dueling DQN)



Dueling Deep Q-network (Dueling DQN)



- One stream of fully-connected layers output a scalar $V(s;\theta,\beta)$
- Other stream output an $|A|$-dimensional vector $A(s,a;\theta,\alpha)$

Here, θ denotes the parameters of the convolutional layers, while α and β are the parameters of the two streams of fully-connected layers.

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + A(s,a;\theta,\alpha)$$

Dueling Deep Q-network (Dueling DQN)

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha)$$

Problem: Equation is unidentifiable $\rightarrow$ given Q we cannot recover V and A uniquely $\rightarrow$ poor practical performance.

Solutions:

1. Force the advantage function estimator to have zero advantage at the chosen action

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \max_{a' \in |A|} A(s, a'; \theta, \alpha)\right)$$

$$\begin{aligned} a^* &= \arg\max_{a' \in A} Q(s, a'; \theta, \alpha, \beta) \\ &= \arg\max_{a' \in A} A(s, a'; \theta, \alpha) \\ Q(s, a^*; \theta, \alpha, \beta) &= V(s; \theta, \beta) \end{aligned}$$

Dueling Deep Q-network (Dueling DQN)

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha)$$

Problem: Equation is unidentifiable $\rightarrow$ given Q we cannot recover V and A uniquely $\rightarrow$ poor practical performance.

Solutions:

2. Replaces the max operator with an average

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \frac{1}{|A|}\sum_{a'} A(s, a'; \theta, \alpha)\right)$$

It increases the stability of the optimization: the advantages only need to change as fast as the mean, instead of having to compensate any change.

Dueling Deep Q-network (Dueling DQN)

Policy	Model	Pros	Cons	Applications

Prioritized Experience Replay (RER)

Problem: Online RL agents incrementally update their parameters while they observe a stream of experience. In their simplest form, they discard incoming data immediately, after a single update. Two issues are

1. Strongly correlated updates that break the i.i.d. assumption
2. Rapid forgetting of possibly rare experiences that would be useful later on.

Solution: Experience replay

- Break the temporal correlations by mixing more and less recent experience for the updates
- Rare experience will be used for more than just a single update

PER

Algorithm 1 Double DQN with proportional prioritization

Input: minibatch k, step-size η, replay period K and size N, exponents α and β, budget T.
Initialize replay memory $\mathcal{H} = \emptyset$, $\Delta = 0$, $p_1 = 1$
Observe S_0 and choose $A_0 \sim \pi_\theta(S_0)$
for $t = 1$ **to** T **do**
  Observe S_t, R_t, γ_t
  Store transition $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$ in $\mathcal{H}$ with maximal priority $p_t = \max_{i<t} p_i$
  if $t \equiv 0 \mod K$ **then**
    for $j = 1$ **to** k **do**
      Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
      Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
      Compute TD-error $\delta_j = R_j + \gamma_j Q_{\text{target}}(S_j, \arg\max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
      Update transition priority $p_j \leftarrow |\delta_j|$
      Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
    end for
    Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$, reset $\Delta = 0$
    From time to time copy weights into target network $\theta_{\text{target}} \leftarrow \theta$
  end if
  Choose action $A_t \sim \pi_\theta(S_t)$
end for