Advances in Deep Reinforcement Learning

Oleksii Hrinchuk

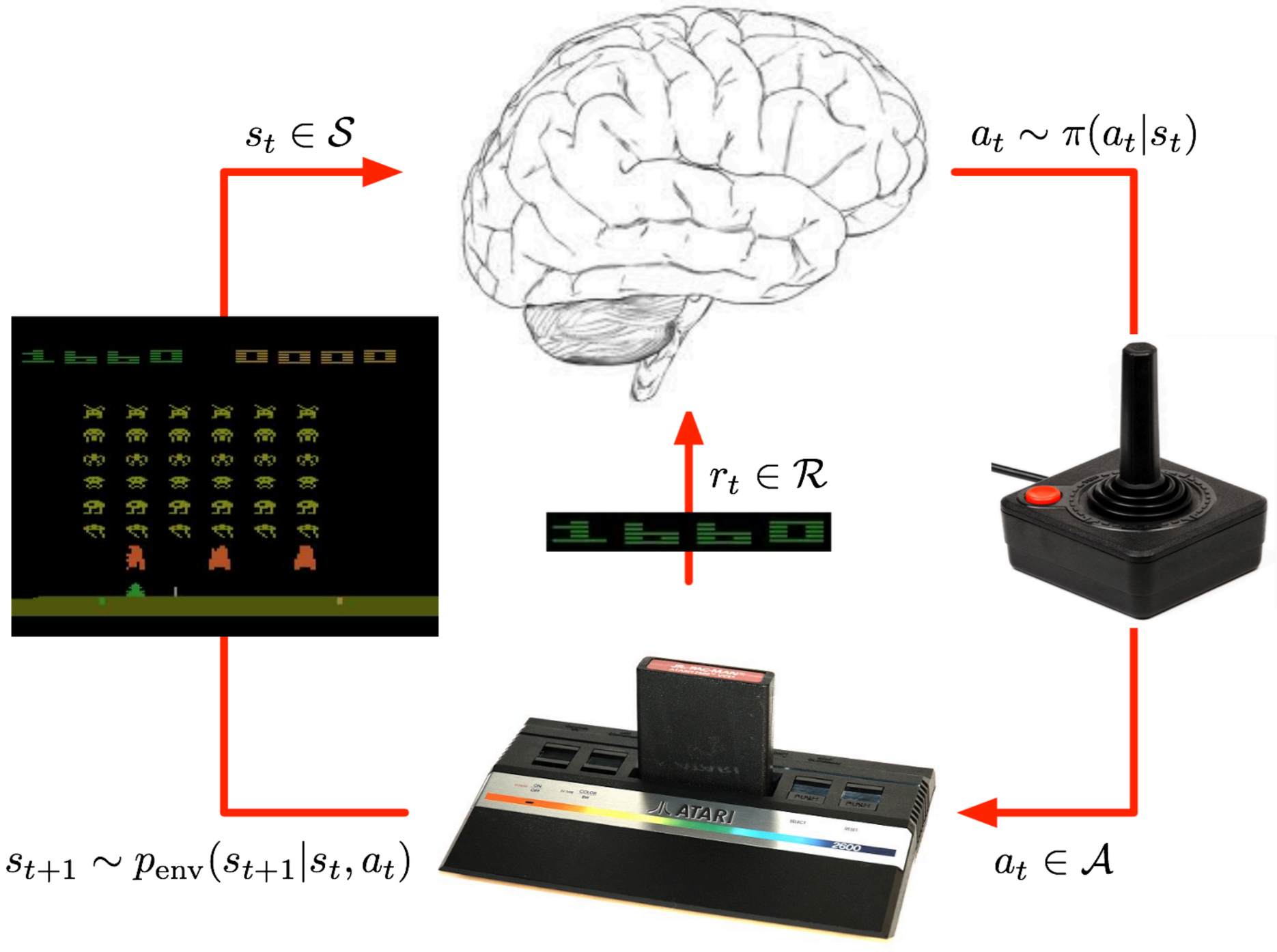
Outline

- **Q-learning methods**
  - Deep Q-network (DQN)
  - Double Q-learning (DDQN)
  - Dueling network for DRL

- **Policy gradient methods**
  - Deep policy gradient (PG) network
  - Asynchronous methods (A3C)
  - Deterministic policy gradient (DDPG)

- **Our ideas**
  - Maxvol exploration
  - One-shot target update

Markov Decision Process



Markov Decision Process

- Environment setting

  — Environment states $s_t \in \mathcal{S} = \mathbb{R}^{W \times H \times F}$

  — Available actions $a_t \in \mathcal{A} = \{a^1, \ldots, a^m\}$

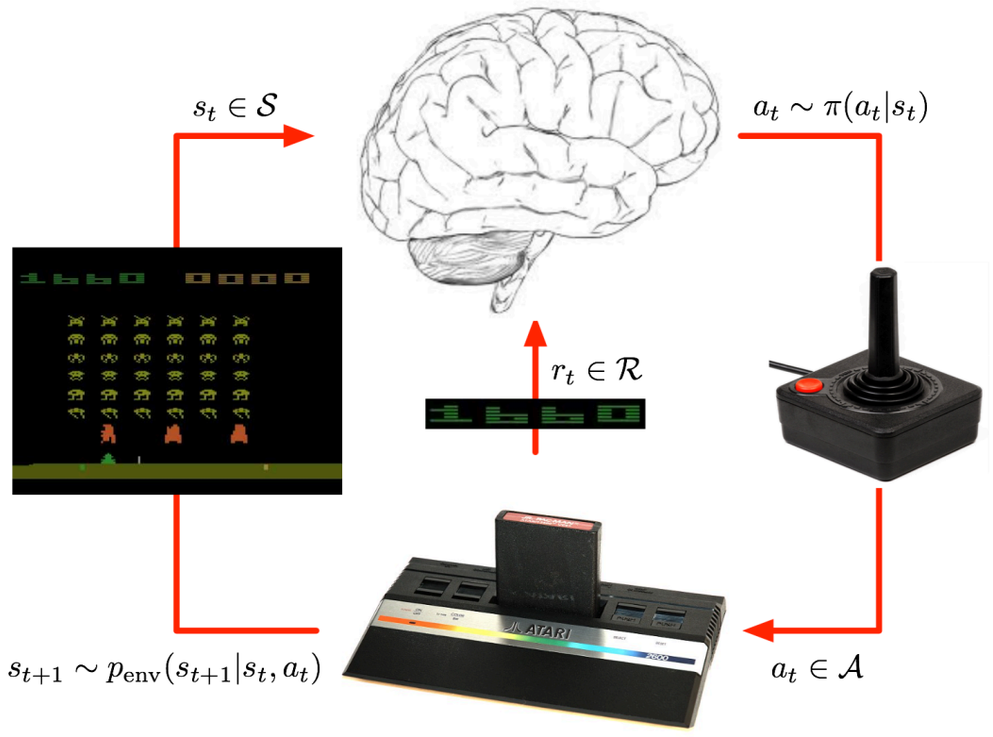
  — Obtained rewards $r_t \in \mathcal{R} = \mathbb{R}$

- Environment model

  $s_{t+1} \sim p_{\text{env}}(s_{t+1}|s_t, a_t)$

- Agent's policy

  $a_t \sim \pi(a_t|s_t)$



Value function

- State value function (value function)

$$V^{\pi}(s_t) = \mathbb{E}_{a_t, s_{t+1}, a_{t+1}, \ldots} \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \right]$$

Value function

- State value function (value function)

$$V^{\pi}(s_t) = \mathbb{E}_{a_t, s_{t+1}, a_{t+1}, \ldots}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots\right]$$

- Optimal value function

$$V^*(s_t) = \max_{\pi} V^{\pi}(s_t)$$

$$V^*(s_t) = \max_{a_t, s_{t+1}, \ldots}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots\right]$$

Value function

- State value function (value function)

$$V^{\pi}(s_t) = \mathbb{E}_{a_t, s_{t+1}, a_{t+1}, \ldots} \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \right]$$

- Optimal value function

$$V^*(s_t) = \max_{\pi} V^{\pi}(s_t)$$

$$V^*(s_t) = \max_{a_t, s_{t+1}, \ldots} \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \right]$$

- Optimal policy

$$\pi^*(a_t|s_t) = \begin{cases} 1, & a_t = \arg\max_{a_t \in \mathcal{A}} \mathbb{E}_{s_{t+1}} V(s_{t+1}) \\ 0, & \text{otherwise} \end{cases}$$

Value function

- State value function (value function)

$$V^{\pi}(s_t) = \mathbb{E}_{a_t, s_{t+1}, a_{t+1}, \ldots}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots\right]$$

- Optimal value function

$$V^*(s_t) = \max_{\pi} V^{\pi}(s_t)$$

$$V^*(s_t) = \max_{a_t, s_{t+1}, \ldots}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots\right]$$

- Optimal policy

$$\pi^*(a_t|s_t) = \begin{cases} 1, & \arg\max_{a_t \in \mathcal{A}} \sum_{s_{t+1} \in \mathcal{S}} V(s_{t+1}) p_{\text{env}}(s_{t+1}|s_t, a_t) \\ 0, & \text{otherwise} \end{cases}$$

Q-function

- State-action value function (Q-function)

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots\right]$$

Q-function

- State-action value function (Q-function)

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots\right]$$

- Optimal Q-function

$$Q^{*}(s_t, a_t) = \max_{\pi} Q^{\pi}(s_t, a_t)$$

$$Q^{*}(s_t, a_t) = \max_{s_{t+1}, a_{t+1}, \cdots}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots\right]$$

Q-function

- State-action value function (Q-function)

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots\right]$$

- Optimal Q-function

$$Q^{*}(s_t, a_t) = \max_{\pi} Q^{\pi}(s_t, a_t)$$

$$Q^{*}(s_t, a_t) = \max_{s_{t+1}, a_{t+1}, \cdots}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots\right]$$

- Optimal policy

$$\pi^{*}(a_t|s_t) = \begin{cases} 1, & a_t = \arg\max_{a_t \in \mathcal{A}} Q(s_t, a_t) \\ 0, & \text{otherwise} \end{cases}$$

Q-learning methods

- Bellman optimality equation

$$Q(s,a) = r(s,a) + \gamma \max_{a' \in \mathcal{A}} Q(s',a')$$

Q-learning methods

- Bellman optimality equation

$$Q(s,a) = r(s,a) + \gamma \max_{a' \in \mathcal{A}} Q(s',a')$$

- Parametric Q-function approximation

$$Q(s,a;\theta) \approx Q(s,a)$$

Q-learning methods

- Bellman optimality equation

$$Q(s,a) = r(s,a) + \gamma \max_{a' \in \mathcal{A}} Q(s',a')$$

- Parametric Q-function approximation

$$Q(s,a;\theta) \approx Q(s,a)$$

- Optimization problem induced by Bellman equation

$$\left[r(s,a) + \gamma \max_{a' \in \mathcal{A}} Q(s',a';\theta) - Q(s,a;\theta) \right]^2 \to \min_{\theta}$$

Q-learning methods

- Bellman optimality equation

$$Q(s,a) = r(s,a) + \gamma \max_{a' \in \mathcal{A}} Q(s',a')$$

- Parametric Q-function approximation

$$Q(s,a;\theta) \approx Q(s,a)$$

- Optimization problem induced by Bellman equation

$$\left[r(s,a) + \gamma \max_{a' \in \mathcal{A}} Q(s',a';\theta) - Q(s,a;\theta) \right]^2 \to \min_{\theta}$$

Target Agent

Learning from samples

- We have no model, only samples (s, a, r, s') where next actions are sampled from some policy (e.g. random) $a \sim \pi(a|s)$ and next states are sampled from the model $s' \sim p_{\text{env}}(s'|s, a)$.

Learning from samples

- We have no model, only samples (s, a, r, s') where next actions are sampled from some policy (e.g. random) $a \sim \pi(a|s)$ and next states are sampled from the model $s' \sim p_{\text{env}}(s'|s, a)$.

- Sample-based estimate of the target

$$y^{\text{target}} = r + \gamma \max_{a'} Q(s', a'; \theta)$$

Learning from samples

- We have no model, only samples (s, a, r, s') where next actions are sampled from some policy (e.g. random) $a \sim \pi(a|s)$ and next states are sampled from the model $s' \sim p_{\text{env}}(s'|s, a)$.

- Sample-based estimate of the target

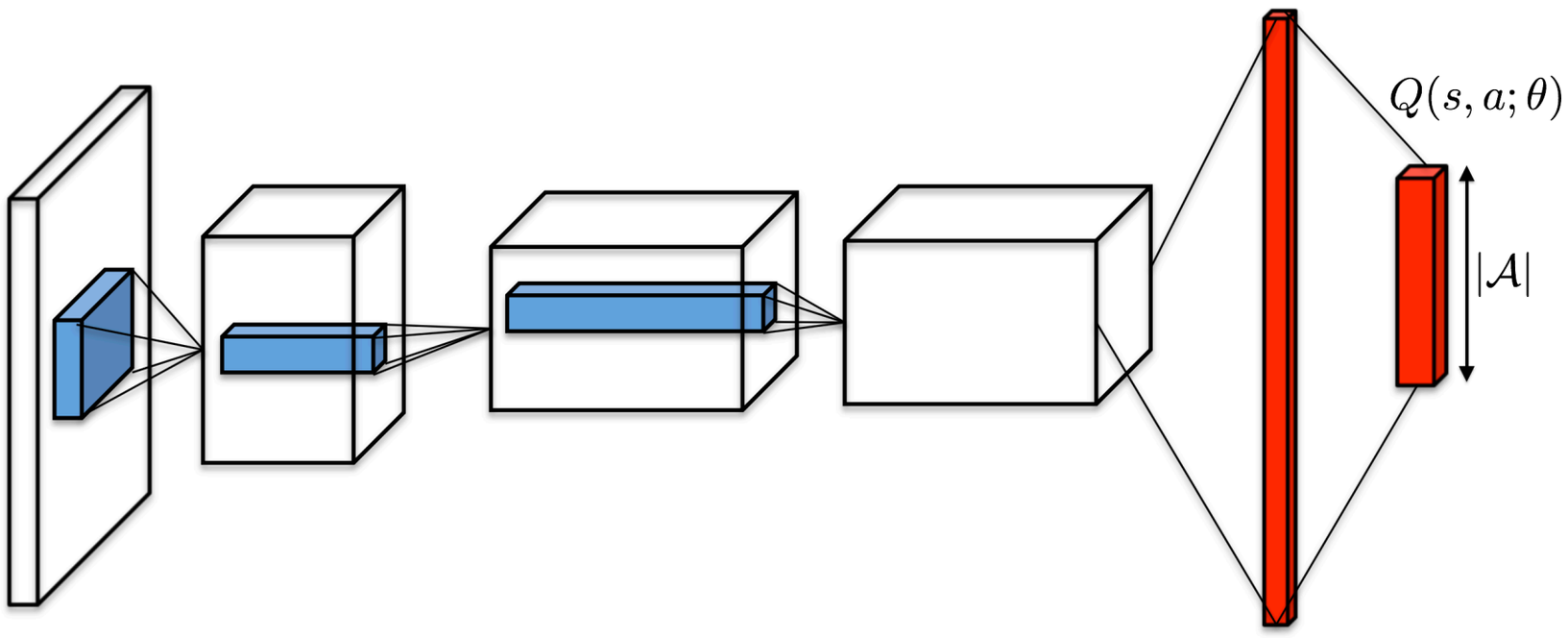
$$y^{\text{target}} = r + \gamma \max_{a'} Q(s', a'; \theta)$$

- Loss function

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{(s,a,r,s')} \left[y^{\text{target}} - Q(s, a; \theta) \right]^2$$

Deep Q-network (DQN)

- Input: environment state $s \in \mathcal{S}$ (image)
- Output: vector of Q-function values for state $s \in \mathcal{S}$ and all possible actions $a \in \mathcal{A}$



Freezing target network

- Use separate target network and update it less frequently than the agent network.

From “Human-level Control Through Deep Reinforcement Learning”, Mnih et al., Nature (2015)

Freezing target network

- Use separate target network and update it less frequently than the agent network.

$$y^{\text{target}} = r + \gamma \max_{a'} Q(s', a'; \theta)$$

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{(s,a,r,s')} \left[y^{\text{target}} - Q(s, a; \theta) \right]^2$$

From “Human-level Control Through Deep Reinforcement Learning”, Mnih et al., Nature (2015)

Freezing target network

- Use separate target network and update it less frequently than the agent network.

$$y^{\mathrm{DQN}} = r + \gamma \max_{a'} Q(s', a'; \theta^-)$$

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{(s,a,r,s')} \left[y^{\mathrm{DQN}} - Q(s, a; \theta) \right]^2$$

From “Human-level Control Through Deep Reinforcement Learning”, Mnih et al., Nature (2015)

Freezing target network

- Use separate target network and update it less frequently than the agent network.

$$y^{\mathrm{DQN}} = r + \gamma \max_{a'} Q(s', a'; \theta^-)$$

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{(s,a,r,s')} \left[y^{\mathrm{DQN}} - Q(s, a; \theta) \right]^2$$

$\theta^- \leftarrow \theta$ after each N training frames

From “Human-level Control Through Deep Reinforcement Learning”, Mnih et al., Nature (2015)

Freezing target network

- Use separate target network and update it less frequently than the agent network.

$$y^{\mathrm{DQN}} = r + \gamma \max_{a'} Q(s', a'; \theta^-)$$

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{(s,a,r,s')} \left[y^{\mathrm{DQN}} - Q(s, a; \theta) \right]^2$$

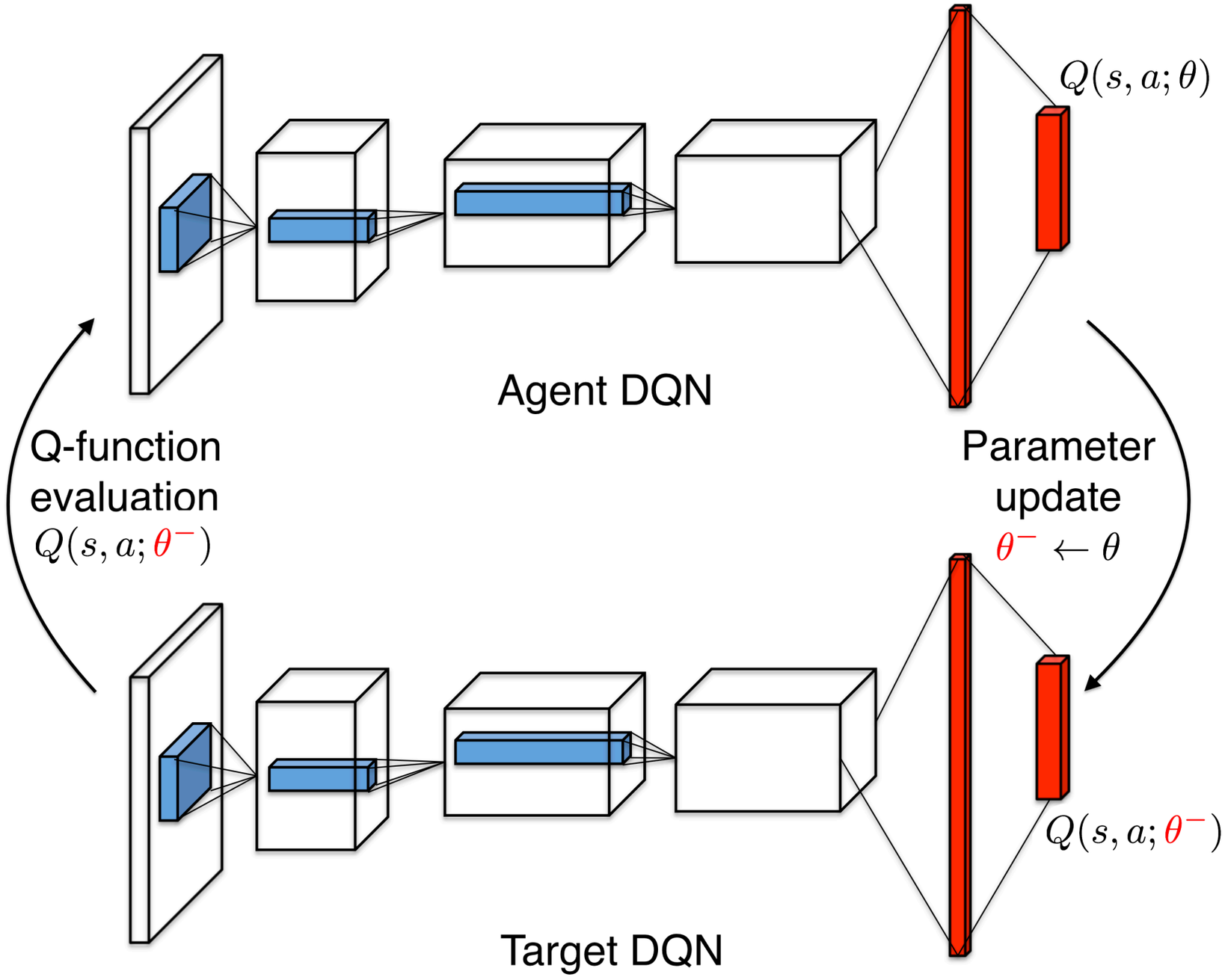
$\theta^- \leftarrow \theta$ after each N training frames

- Update target network frequently, but slowly.

$$\theta^- \leftarrow \tau\theta + (1 - \tau)\theta^-, \; \tau = \frac{1}{N}$$

From “Human-level Control Through Deep Reinforcement Learning”, Mnih et al., Nature (2015)
From “Continuous Control With Deep Reinforcement Learning”, Lillicrap & Hunt et al., ICLR (2016)

Freezing target network

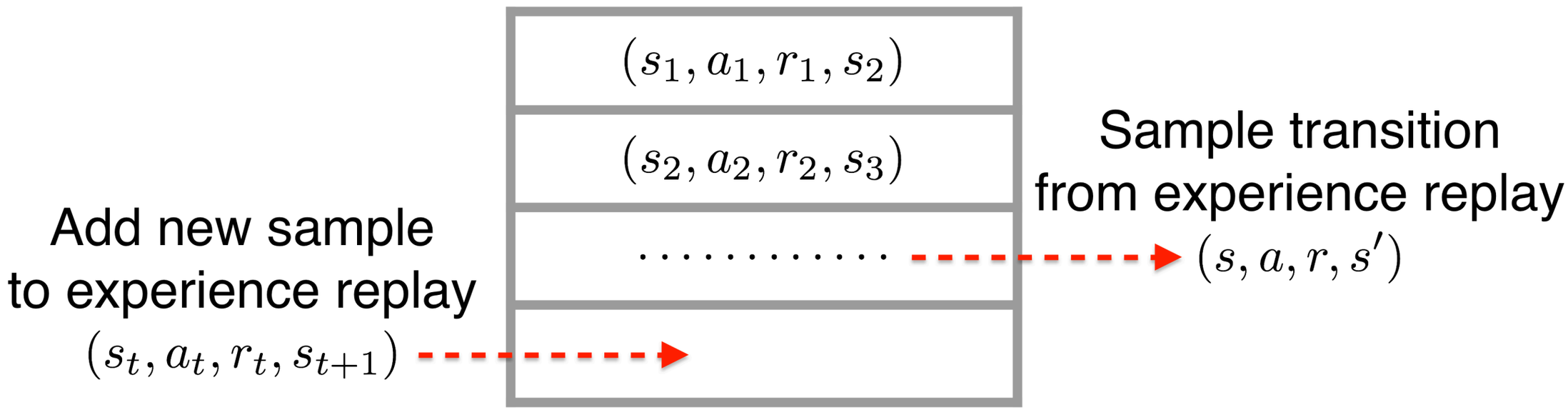


From “Human-level Control Through Deep Reinforcement Learning”, Mnih et al., Nature (2015)

Experience replay

Store most recent agent-environment interactions in experience replay and sample them uniformly when updating model weights.

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{U}(\mathcal{D})} \left[\left(y^{\mathrm{DQN}} - Q(s, a; \theta) \right)^2 \right]$$

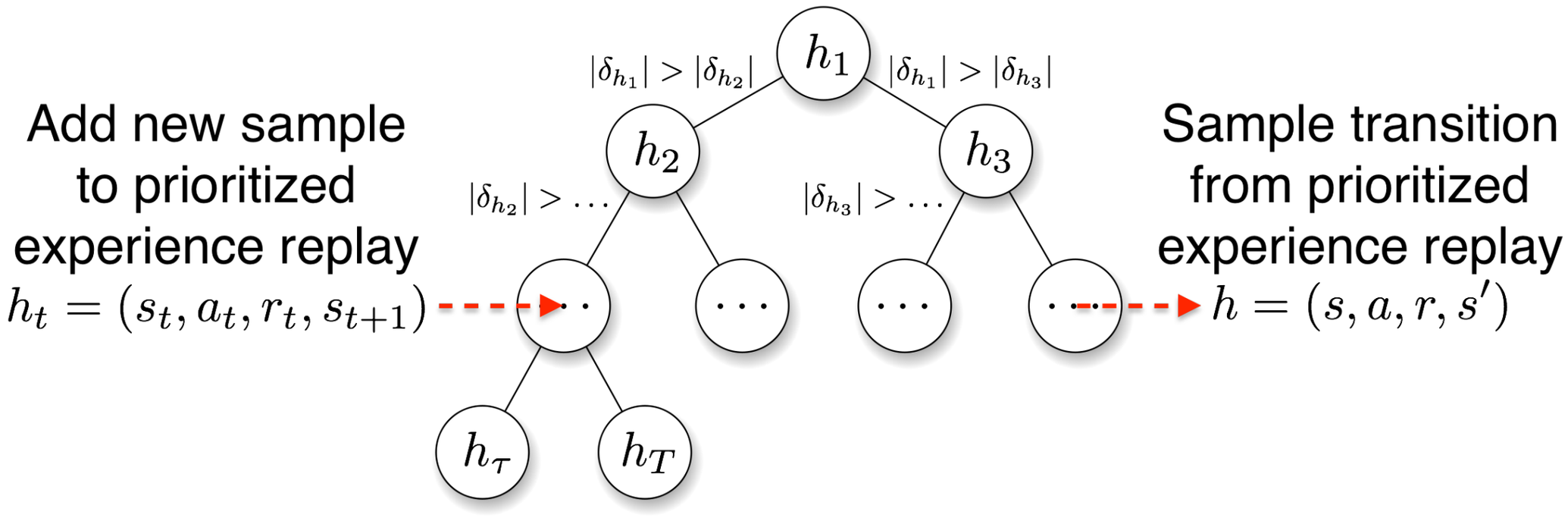


From “Human-level Control Through Deep Reinforcement Learning”, Mnih et al., Nature (2015)

Prioritized experience replay

In prioritized experience replay transitions with high TD error δ_i will be sampled more likely.

$$h_i = (s, a, r, s')_i, \quad p(h_i) = \frac{p_i^\alpha}{\sum_{j=1}^{|\mathcal{D}|} p_j^\alpha}, \quad p_i = \begin{cases} |\delta_i| + \varepsilon \\ \frac{1}{\text{rank}(i)} \end{cases}$$



From “Prioritized Experience Replay”, Schaul et al., ICLR (2016)

Double DQN (DDQN)

- In Q-learning and DQN, the max operator uses the same values to both select and evaluate an action. This can lead to overoptimistic value estimates.

From “Deep Reinforcement Learning with Double Q-learning”, van Hasselt et al., AAAI (2016)

Double DQN (DDQN)

- In Q-learning and DQN, the max operator uses the same values to both select and evaluate an action. This can lead to overoptimistic value estimates.
- Remove upward bias caused by $\max_{a'} Q(s', a'; \theta)$

From “Deep Reinforcement Learning with Double Q-learning”, van Hasselt et al., AAAI (2016)

Double DQN (DDQN)

- In Q-learning and DQN, the max operator uses the same values to both select and evaluate an action. This can lead to overoptimistic value estimates.
- Remove upward bias caused by $\max_{a'} Q(s', a'; \theta)$

$$\max_{a'} Q(s', a'; \theta) = Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta)$$

From “Deep Reinforcement Learning with Double Q-learning”, van Hasselt et al., AAAI (2016)

Double DQN (DDQN)

- In Q-learning and DQN, the max operator uses the same values to both select and evaluate an action. This can lead to overoptimistic value estimates.
- Remove upward bias caused by $\max_{a'} Q(s', a'; \theta)$

$$\max_{a'} Q(s', a'; \theta) = Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta)$$

$$y^{\mathrm{DQN}} = r + \gamma \max_{a'} Q(s', a'; \theta^{-})$$

From “Deep Reinforcement Learning with Double Q-learning”, van Hasselt et al., AAAI (2016)

Double DQN (DDQN)

- In Q-learning and DQN, the max operator uses the same values to both select and evaluate an action. This can lead to overoptimistic value estimates.
- Remove upward bias caused by $\max_{a'} Q(s', a'; \theta)$

$$\max_{a'} Q(s', a'; \theta) = Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta)$$

$$y^{\mathrm{DDQN}} = r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta^{-})$$

From “Deep Reinforcement Learning with Double Q-learning”, van Hasselt et al., AAAI (2016)

Double DQN (DDQN)

- In Q-learning and DQN, the max operator uses the same values to both select and evaluate an action. This can lead to overoptimistic value estimates.
- Remove upward bias caused by $\max_{a'} Q(s', a'; \theta)$

$$\max_{a'} Q(s', a'; \theta) = Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta)$$

$$y^{\mathrm{DDQN}} = r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta^{-})$$

Agent Q-network is used to select actions
Target Q-network is used to evaluate actions

From “Deep Reinforcement Learning with Double Q-learning”, van Hasselt et al., AAAI (2016)

Advantage function

- Value function

$$V^{\pi}(s_t) = \mathbb{E}_{a_t, s_{t+1}, a_{t+1}, \ldots} \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \right]$$

Advantage function

- Value function

$$V^{\pi}(s_t) = \mathbb{E}_{a_t, s_{t+1}, a_{t+1}, \ldots} \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \right]$$

- Q-function

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots} \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \right]$$

Advantage function

- Value function

$$V^{\pi}(s_t) = \mathbb{E}_{a_t, s_{t+1}, a_{t+1}, \ldots} \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \right]$$

- Q-function

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots} \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \right]$$

$$V^{\pi}(s_t) = \mathbb{E}_{a_t} \left[Q^{\pi}(s_t, a_t) \right] = \sum_{a_t \in \mathcal{A}} \pi(a_t | s_t) Q^{\pi}(s_t, a_t)$$

Advantage function

- Value function

$$V^{\pi}(s_t) = \mathbb{E}_{a_t, s_{t+1}, a_{t+1}, \ldots} \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \right]$$

- Q-function

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots} \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \right]$$

$$V^{\pi}(s_t) = \mathbb{E}_{a_t} \left[Q^{\pi}(s_t, a_t) \right] = \sum_{a_t \in \mathcal{A}} \pi(a_t | s_t) Q^{\pi}(s_t, a_t)$$

- Optimal policy case

$$\pi^*(a|s) = \begin{cases} 1, \ a = \arg\max\limits_{a} Q^*(s, a) \\ 0, \ \text{otherwise} \end{cases}$$

Advantage function

- Value function

$$V^{\pi}(s_t) = \mathbb{E}_{a_t, s_{t+1}, a_{t+1}, \ldots} \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \right]$$

- Q-function

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots} \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \right]$$

$$V^{\pi}(s_t) = \mathbb{E}_{a_t} \left[Q^{\pi}(s_t, a_t) \right] = \sum_{a_t \in \mathcal{A}} \pi(a_t | s_t) Q^{\pi}(s_t, a_t)$$

- Optimal policy case

$$\pi^*(a|s) = \begin{cases} 1, \ a = \arg\max\limits_{a} Q^*(s, a) \\ 0, \ \text{otherwise} \end{cases}$$

Advantage function

- Value function

$$V^{\pi}(s_t) = \mathbb{E}_{a_t, s_{t+1}, a_{t+1}, \ldots} \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \right]$$

- Q-function

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots} \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \right]$$

$$V^*(s_t) = \mathbb{E}_{a_t} \left[Q^*(s_t, a_t) \right] = \sum_{a_t \in \mathcal{A}} \pi^*(a_t | s_t) Q^*(s_t, a_t)$$

- Optimal policy case

$$\pi^*(a|s) = \begin{cases} 1, \ a = \arg\max_a Q^*(s, a) \\ 0, \ \text{otherwise} \end{cases}$$

Advantage function

- Value function

$$V^{\pi}(s_t) = \mathbb{E}_{a_t, s_{t+1}, a_{t+1}, \ldots} \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \right]$$

- Q-function

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots} \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \right]$$

$$V^*(s_t) = \mathbb{E}_{a_t} \left[Q^*(s_t, a_t) \right] = 1 \cdot Q^*(s_t, \arg\max_a Q^*(s, a))$$

- Optimal policy case

$$\pi^*(a|s) = \begin{cases} 1, \ a = \arg\max_a Q^*(s, a) \\ 0, \ \text{otherwise} \end{cases}$$

Advantage function

- Value function

$$V^{\pi}(s_t) = \mathbb{E}_{a_t, s_{t+1}, a_{t+1}, \ldots} \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \right]$$

- Q-function

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \ldots} \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \right]$$

$$V^{*}(s_t) = \mathbb{E}_{a_t} \left[Q^{*}(s_t, a_t) \right] = \max_a Q^{*}(s_t, a)$$

- Optimal policy case

$$\pi^{*}(a|s) = \begin{cases} 1, \ a = \arg\max_a Q^{*}(s, a) \\ 0, \ \text{otherwise} \end{cases}$$

Advantage function

- Advantage function for some policy π

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

$$\mathbb{E}_{a\sim\pi} A^{\pi}(s,a) = 0$$

Advantage function

- Advantage function for some policy π

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

$$\mathbb{E}_{a\sim\pi} A^{\pi}(s,a) = 0$$

- Advantage function for optimal policy π^*

$$A^{*}(s,a) = Q^{*}(s,a) - V^{*}(s)$$

$$\max_{a} A^{*}(s,a) = 0$$

Advantage function

- Advantage function for some policy π

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

$$\mathbb{E}_{a\sim\pi} A^{\pi}(s,a) = 0$$

- Advantage function for optimal policy π^*

$$A^{*}(s,a) = Q^{*}(s,a) - V^{*}(s)$$

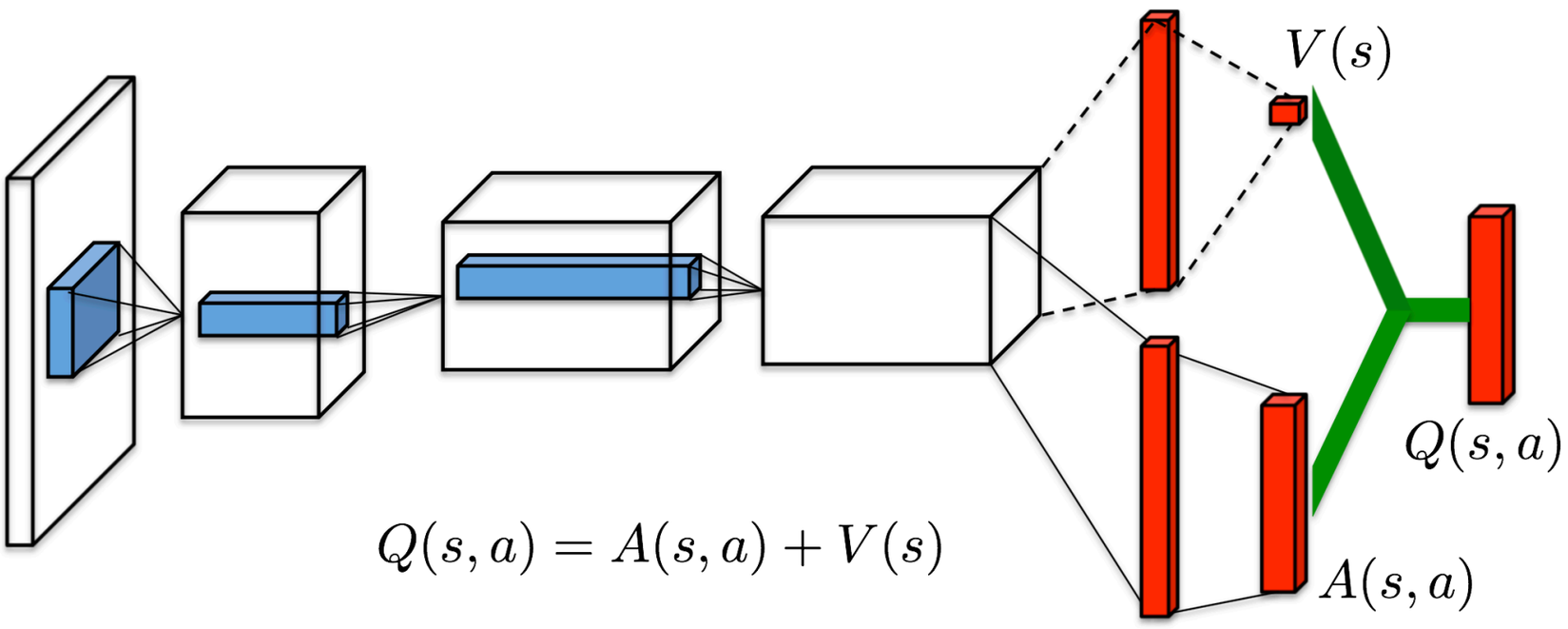
$$\max_{a} A^{*}(s,a) = 0$$

- Advantage function is a relative measure of the importance of each action

Dueling network for DRL

Split Q-network into two channels

- Action-independent value function $V(s)$
- Action dependent advantage function $A(s,a)$



From “Dueling Network Architectures for Deep Reinforcement Learning”, Wang et al., ICML (2016)

Dueling network for DRL

Split Q-network into two channels

- Action-independent value function $V(s)$
- Action dependent advantage function $A(s, a)$

$$Q(s, a) = V(s) + A(s, a)$$

From “Dueling Network Architectures for Deep Reinforcement Learning”, Wang et al., ICML (2016)

Dueling network for DRL

Split Q-network into two channels

- Action-independent value function $V(s)$
- Action dependent advantage function $A(s,a)$

$$Q(s,a) = V(s) + A(s,a)$$

$$Q(s,a) = V(s) + \left(A(s,a) - \max_{a' \in \mathcal{A}} A(s,a') \right)$$

From “Dueling Network Architectures for Deep Reinforcement Learning”, Wang et al., ICML (2016)

Dueling network for DRL

Split Q-network into two channels

- Action-independent value function $V(s)$
- Action dependent advantage function $A(s, a)$

$$Q(s,a) = V(s) + A(s,a)$$

$$Q(s,a) = V(s) + \left(A(s,a) - \max_{a' \in \mathcal{A}} A(s,a') \right)$$

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

From “Dueling Network Architectures for Deep Reinforcement Learning”, Wang et al., ICML (2016)

Policy gradient methods

- Parametric policy approximation

$$\pi(a|s) \approx \pi(a|s;\theta)$$

From “Policy Gradient Methods for Reinforcement Learning with Function Approximation”, Sutton et al., NIPS (2000)

Policy gradient methods

- Parametric policy approximation

$$\pi(a|s) \approx \pi(a|s;\theta)$$

- Optimization problem

$$J(\theta) = \mathbb{E}_{s_0,a_0,\ldots,s_T}\left[\sum_{t=0}^{T}\gamma^t r(s_t,a_t)\right] \to \max_{\theta}$$

From “Policy Gradient Methods for Reinforcement Learning with Function Approximation”, Sutton et al., NIPS (2000)

Policy gradient methods

- Parametric policy approximation

$$\pi(a|s) \approx \pi(a|s;\theta)$$

- Optimization problem

$$J(\theta) = \mathbb{E}_{s_0,a_0,\ldots,s_T}\left[\sum_{t=0}^{T}\gamma^t r(s_t,a_t)\right] \to \max_\theta$$

- Policy gradient theorem

$$\nabla_\theta J(\theta) = \mathbb{E}_{s,a}\left[\nabla_\theta \log \pi(a|s;\theta) Q(s,a)\right]$$

From “Policy Gradient Methods for Reinforcement Learning with Function Approximation”, Sutton et al., NIPS (2000)

Policy gradient methods

- Parametric policy approximation

$$\pi(a|s) \approx \pi(a|s;\theta)$$

- Optimization problem

$$J(\theta) = \mathbb{E}_{s_0,a_0,\ldots,s_T}\left[\sum_{t=0}^{T}\gamma^t r(s_t,a_t)\right] \to \max_\theta$$

- Policy gradient theorem

$$\nabla_\theta J(\theta) = \mathbb{E}_{s,a}\left[\nabla_\theta \log \pi(a|s;\theta) Q(s,a)\right]$$

From “Policy Gradient Methods for Reinforcement Learning with Function Approximation”, Sutton et al., NIPS (2000)

Policy gradient methods

- Parametric policy approximation

$$\pi(a|s) \approx \pi(a|s;\theta)$$

- Optimization problem

$$J(\theta) = \mathbb{E}_{s_0,a_0,\ldots,s_T}\left[\sum_{t=0}^{T}\gamma^t r(s_t,a_t)\right] \to \max_\theta$$

- Policy gradient theorem

$$\nabla_\theta J(\theta) = \mathbb{E}_{s,a}\left[\nabla_\theta \log \pi(a|s;\theta)\,(Q(s,a) - V(s))\right]$$

From “Policy Gradient Methods for Reinforcement Learning with Function Approximation”, Sutton et al., NIPS (2000)

Policy gradient methods

- Parametric policy approximation

$$\pi(a|s) \approx \pi(a|s;\theta)$$

- Optimization problem

$$J(\theta) = \mathbb{E}_{s_0,a_0,\ldots,s_T}\left[\sum_{t=0}^{T}\gamma^t r(s_t,a_t)\right] \to \max_\theta$$

- Policy gradient theorem

$$\nabla_\theta J(\theta) = \mathbb{E}_{s,a}\left[\nabla_\theta \log \pi(a|s;\theta) A(s,a)\right]$$

Advantage function

From “Policy Gradient Methods for Reinforcement Learning with Function Approximation”, Sutton et al., NIPS (2000)

Advantage function estimate

- Advantage function

$$A(s,a) = Q(s,a) - V(s)$$

Advantage function estimate

- Advantage function

$$A(s,a) = Q(s,a) - V(s)$$

$$A(s_t,a_t) = r_t + \gamma \max Q(s_{t+1},a) - V(s_t)$$

$$a$$

Advantage function estimate

- Advantage function

$$A(s,a) = Q(s,a) - V(s)$$

$$A(s_t,a_t) = r_t + \gamma \max_a Q(s_{t+1},a) - V(s_t)$$

$$V(s) = \max_a Q(s,a)$$

Advantage function estimate

- Advantage function

$$A(s,a) = Q(s,a) - V(s)$$

$$A(s_t,a_t) = r_t + \gamma \max_a Q(s_{t+1},a) - V(s_t)$$

$$V(s) = \max_a Q(s,a)$$

- One step sample-based estimate

$$A^1(s_t,a_t) = r_t + \gamma V(s_{t+1}) - V(s_t)$$

Advantage function estimate

- Advantage function

$$A(s,a) = Q(s,a) - V(s)$$

$$A(s_t,a_t) = r_t + \gamma \max_a Q(s_{t+1},a) - V(s_t)$$

$$V(s) = \max_a Q(s,a)$$

- One step sample-based estimate

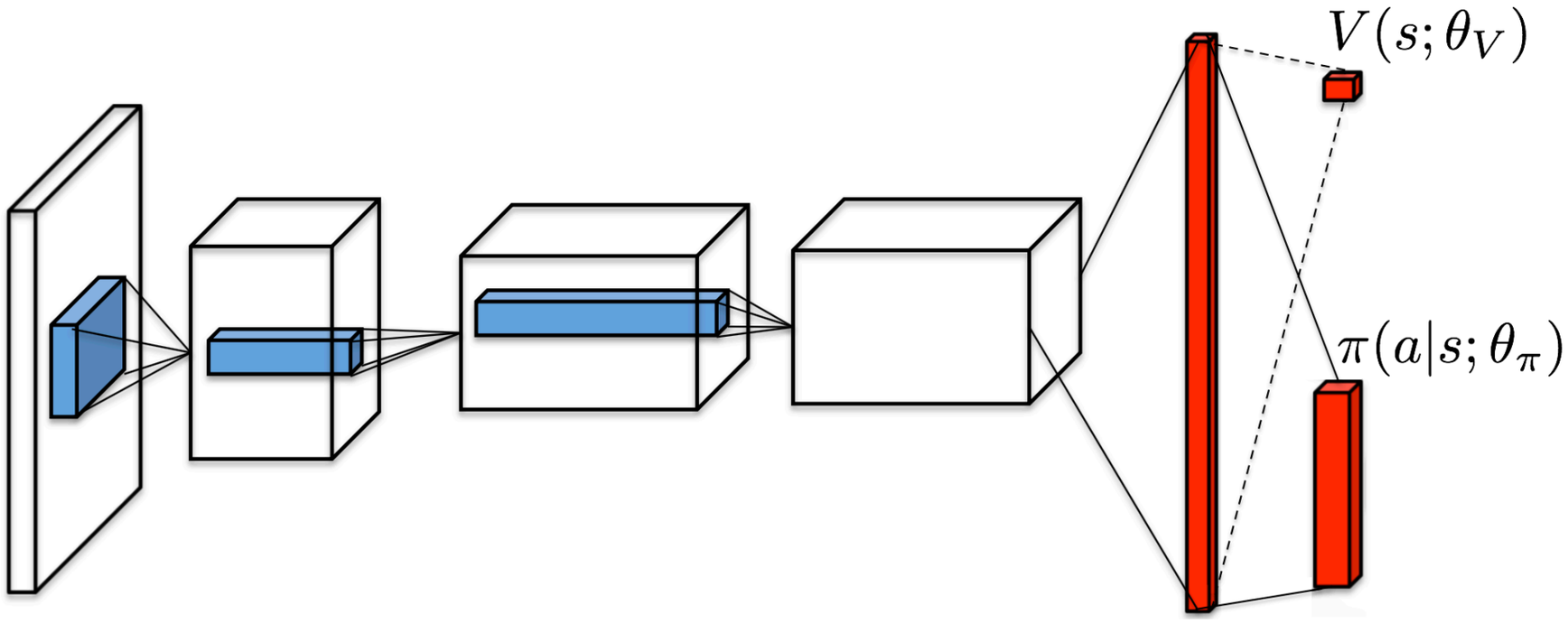
$$A^1(s_t,a_t) = r_t + \gamma V(s_{t+1}) - V(s_t)$$

- n-step sample-based estimate

$$A^n(s_t,a_t) = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n V(s_{t+n}) - V(s_t)$$

Deep policy gradient network

- Input: environment state $s \in \mathcal{S}$ (image)
- Output: policy (softmax distribution over all possible actions) and value function



Model weights update

$$\theta_\pi \leftarrow \theta_\pi + \alpha \nabla_{\theta_\pi} J(\theta_\pi) + \beta \nabla_{\theta_\pi} H(\pi)$$

$$\theta_V \leftarrow \theta_V - \eta \nabla_{\theta_V} J(\theta_V)$$

Model weights update

$$\theta_\pi \leftarrow \theta_\pi + \alpha \nabla_{\theta_\pi} J(\theta_\pi) + \beta \nabla_{\theta_\pi} H(\pi)$$

$$\theta_V \leftarrow \theta_V - \eta \nabla_{\theta_V} J(\theta_V)$$

- Policy gradient

$$\nabla_{\theta_\pi} J(\theta_\pi) = (r_t + \gamma V(s_{t+1}) - V(s_t)) \nabla_{\theta_\pi} \log \pi(a_t | s_t; \theta_\pi)$$

Model weights update

$$\theta_\pi \leftarrow \theta_\pi + \alpha \nabla_{\theta_\pi} J(\theta_\pi) + \beta \nabla_{\theta_\pi} H(\pi)$$

$$\theta_V \leftarrow \theta_V - \eta \nabla_{\theta_V} J(\theta_V)$$

- Policy gradient

$$\nabla_{\theta_\pi} J(\theta_\pi) = (r_t + \gamma V(s_{t+1}) - V(s_t)) \nabla_{\theta_\pi} \log \pi(a_t | s_t; \theta_\pi)$$

- Entropy regularization gradient

$$\nabla_{\theta_\pi} H(\pi) = \nabla_{\theta_\pi} \sum_{a \in \mathcal{A}} \pi(a | s; \theta_\pi) \log \pi(a | s; \theta_\pi)$$

Model weights update

$$\theta_\pi \leftarrow \theta_\pi + \alpha \nabla_{\theta_\pi} J(\theta_\pi) + \beta \nabla_{\theta_\pi} H(\pi)$$

$$\theta_V \leftarrow \theta_V - \eta \nabla_{\theta_V} J(\theta_V)$$

- Policy gradient

$$\nabla_{\theta_\pi} J(\theta_\pi) = (r_t + \gamma V(s_{t+1}) - V(s_t)) \nabla_{\theta_\pi} \log \pi(a_t | s_t; \theta_\pi)$$

- Entropy regularization gradient

$$\nabla_{\theta_\pi} H(\pi) = \nabla_{\theta_\pi} \sum_{a \in \mathcal{A}} \pi(a|s;\theta_\pi) \log \pi(a|s;\theta_\pi)$$

- Value estimation gradient

$$\nabla_{\theta_V} J(\theta_V) = \nabla_{\theta_V} \left[r_t + \gamma V(s_{t+1};\theta_V) - V(s_t;\theta_V) \right]^2$$

Model weights update

$$\theta_\pi \leftarrow \theta_\pi + \alpha \nabla_{\theta_\pi} J(\theta_\pi) + \beta \nabla_{\theta_\pi} H(\pi)$$

$$\theta_V \leftarrow \theta_V - \eta \nabla_{\theta_V} J(\theta_V)$$

- Policy gradient

$$\nabla_{\theta_\pi} J(\theta_\pi) = (r_t + \gamma V(s_{t+1}) - V(s_t)) \nabla_{\theta_\pi} \log \pi(a_t | s_t; \theta_\pi)$$

- Entropy regularization gradient

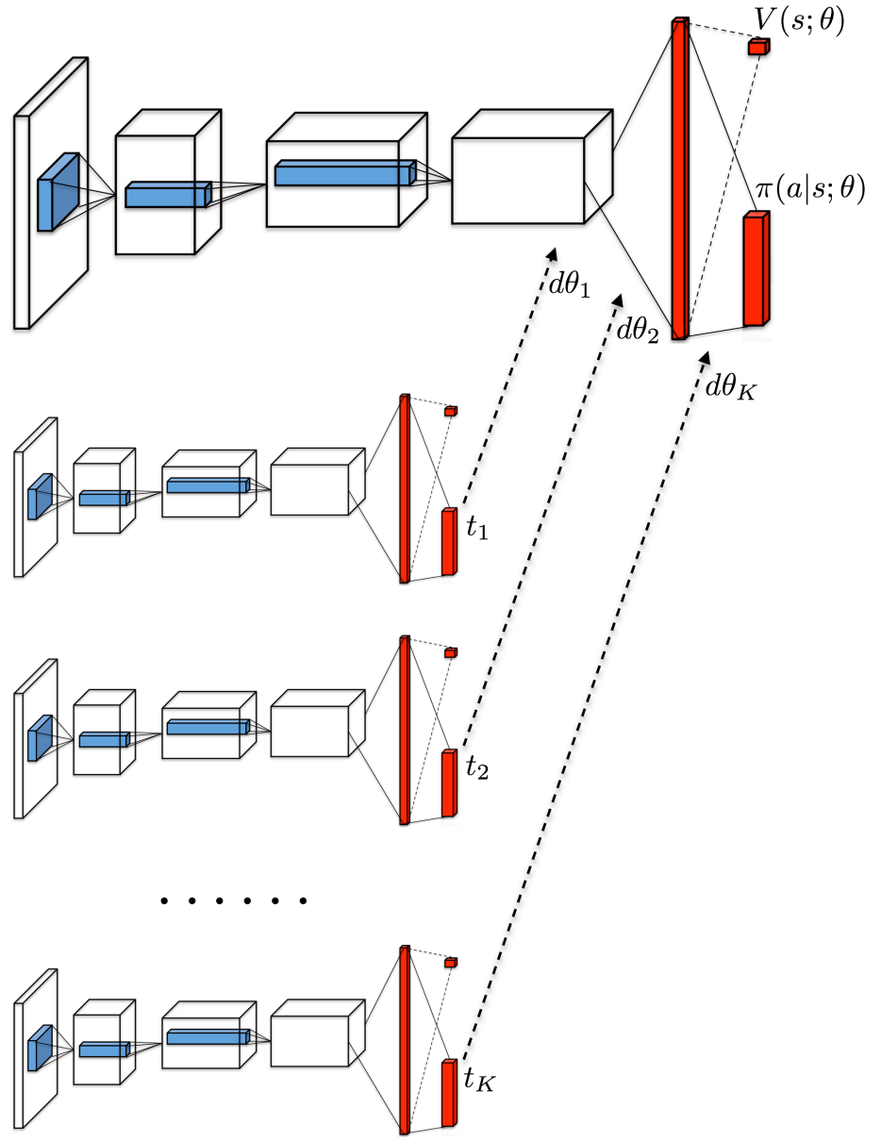
$$\nabla_{\theta_\pi} H(\pi) = \nabla_{\theta_\pi} \sum_{a \in \mathcal{A}} \pi(a|s;\theta_\pi) \log \pi(a|s;\theta_\pi)$$

- Value estimation gradient

$$\nabla_{\theta_V} J(\theta_V) = \nabla_{\theta_V} \left[r_t + \gamma V(s_{t+1};\theta_V) - V(s_t;\theta_V) \right]^2$$

Asynchronous methods (A3C)

- Use several agents to explore the environment simultaneously

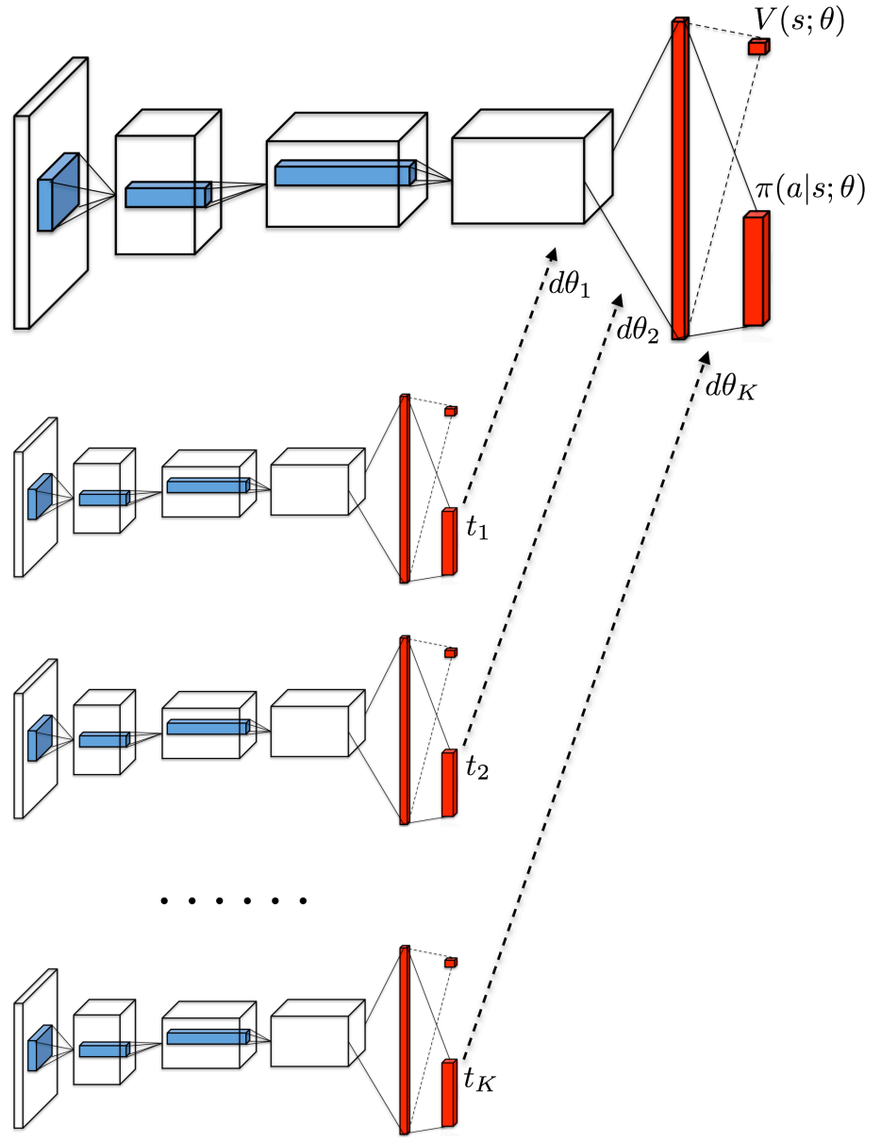


From “Asynchronous Methods for Deep Reinforcement Learning”, Mnih et al., ICML (2016)

Asynchronous methods (A3C)

- Use several agents to explore the environment simultaneously

- Update weights of the target network asynchronously



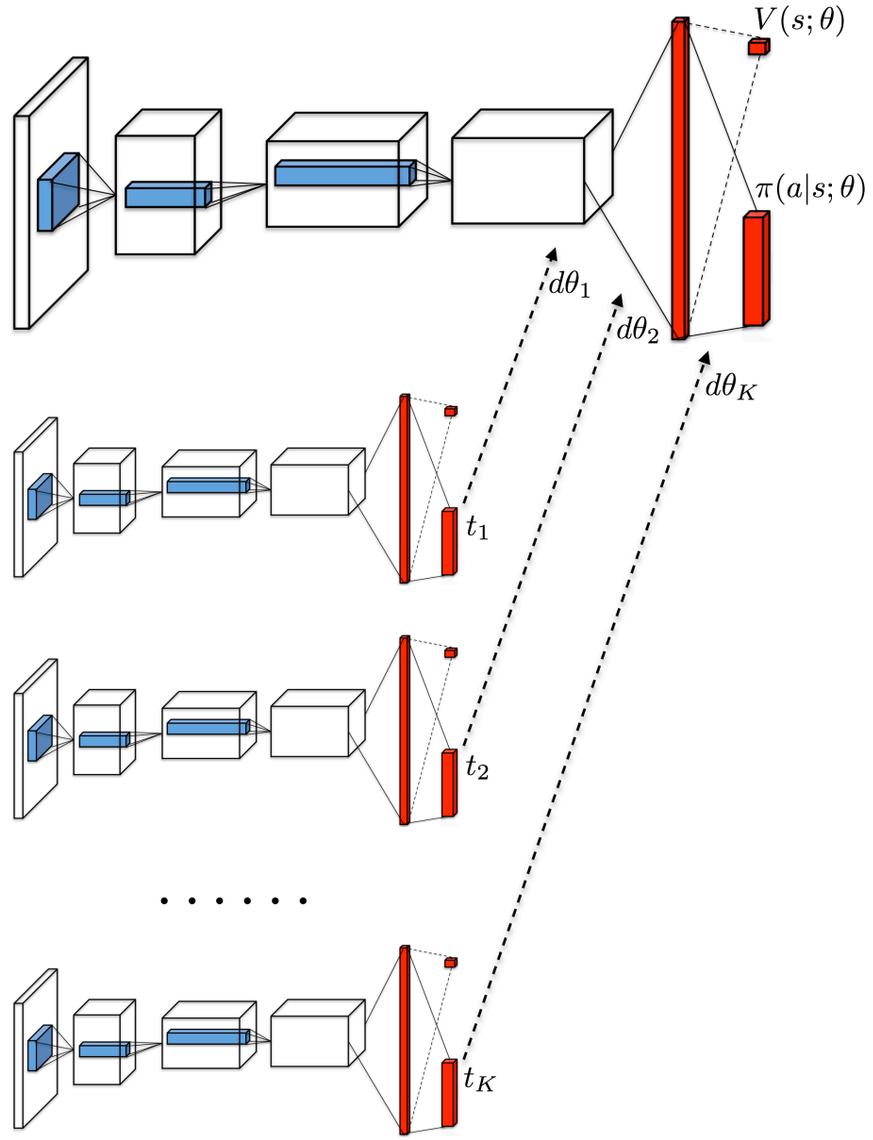
From “Asynchronous Methods for Deep Reinforcement Learning”, Mnih et al., ICML (2016)

Asynchronous methods (A3C)

- Use several agents to explore the environment simultaneously

- Update weights of the target network asynchronously

- Can be trained on a single multi-core CPU instead of GPU



From “Asynchronous Methods for Deep Reinforcement Learning”, Mnih et al., ICML (2016)

Deterministic policy gradient

- Policy in discrete action space

$$\pi(a|s) \approx \pi(a|s;\theta_\pi)$$

$$a = \arg\max_{a \in \mathcal{A}} \pi(a|s;\theta_\pi) = \mu(s)$$

From “Deterministic Policy Gradient Algorithms”, Silver et al., ICML (2014)

Deterministic policy gradient

- Policy in discrete action space

$$\pi(a|s) \approx \pi(a|s;\theta_\pi)$$

$$a = \arg\max_{a \in \mathcal{A}} \pi(a|s;\theta_\pi) = \mu(s)$$

- Policy in continuous action space

$$\pi(a|s) \approx \pi(a|s;\theta_\pi)$$

$$a = \arg\max_{a \in \mathcal{A}} \pi(a|s;\theta_\pi) = \mu(s)$$

From “Deterministic Policy Gradient Algorithms”, Silver et al., ICML (2014)

Deterministic policy gradient

- Policy in discrete action space

$$\pi(a|s) \approx \pi(a|s;\theta_\pi)$$

$$a = \arg\max_{a \in \mathcal{A}} \pi(a|s;\theta_\pi) = \mu(s)$$

- Policy in continuous action space

$$\pi(a|s) \approx \pi(a|s;\theta_\pi)$$

$$a = \arg\max_{a \in \mathcal{A}} \pi(a|s;\theta_\pi) = \mu(s)$$

From “Deterministic Policy Gradient Algorithms”, Silver et al., ICML (2014)

Deterministic policy gradient

- Policy in discrete action space

$$\pi(a|s) \approx \pi(a|s;\theta_\pi)$$

$$a = \arg\max_{a \in \mathcal{A}} \pi(a|s;\theta_\pi) = \mu(s)$$

- Policy in continuous action space

~~$\pi(a|s) \approx \pi(a|s;\theta_\pi)$~~

~~$a = \arg\max_{a \in \mathcal{A}} \pi(a|s;\theta_\pi) = \mu(s)$~~

$$a = \mu(s) \approx \mu(s;\theta_\mu)$$

From “Deterministic Policy Gradient Algorithms”, Silver et al., ICML (2014)

Deterministic policy gradient

- Policy gradient theorem

$$\nabla_{\theta_\pi} J(\theta_\pi) = \mathbb{E}_{s,a} \left[\nabla_{\theta_\pi} \log \pi(a|s;\theta_\pi) Q(s,a) \right]$$

From “Deterministic Policy Gradient Algorithms”, Silver et al., ICML (2014)

Deterministic policy gradient

- Policy gradient theorem

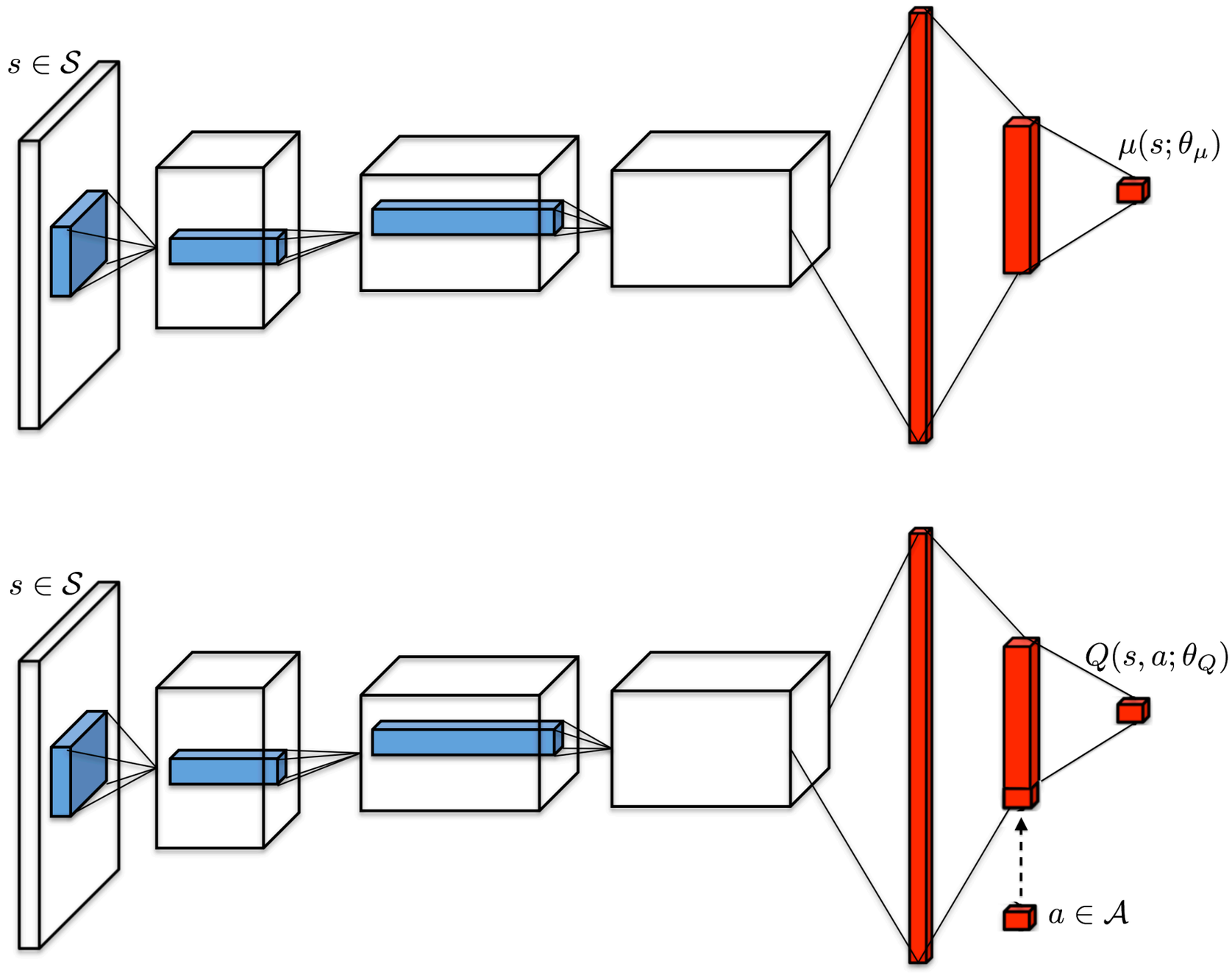
$$\nabla_{\theta_\pi} J(\theta_\pi) = \mathbb{E}_{s,a} \left[\nabla_{\theta_\pi} \log \pi(a|s;\theta_\pi) Q(s,a) \right]$$

- Deterministic policy gradient theorem

$$\nabla_{\theta_\mu} J(\theta_\mu) = \mathbb{E}_s \left[\nabla_a Q(s,a) \nabla_{\theta_\mu} \mu(s;\theta_\mu) \right]$$

From “Deterministic Policy Gradient Algorithms”, Silver et al., ICML (2014)

Deterministic policy gradient



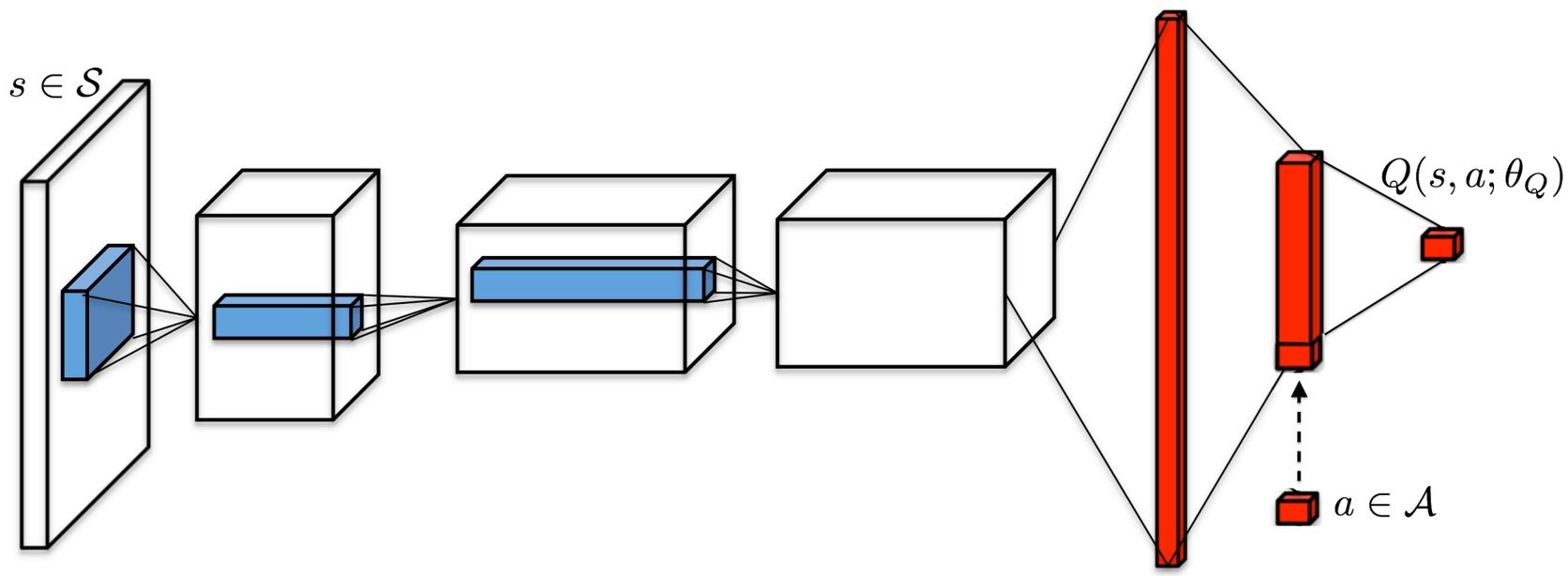
From “Continuous Control With Deep Reinforcement Learning”, Lillicrap, Hunt, et al., ICLR (2016)

Deterministic policy gradient

- Q-network training

$$y^{\mathrm{DDPG}} = r + \gamma Q(s', \mu(s'; \theta_\mu^-); \theta_Q^-)$$

$$\mathcal{L}(\theta_Q) = \frac{1}{N} \sum_{(s,a,r,s')} \left(y^{\mathrm{DDPG}} - Q(s, a; \theta_Q)\right)^2$$

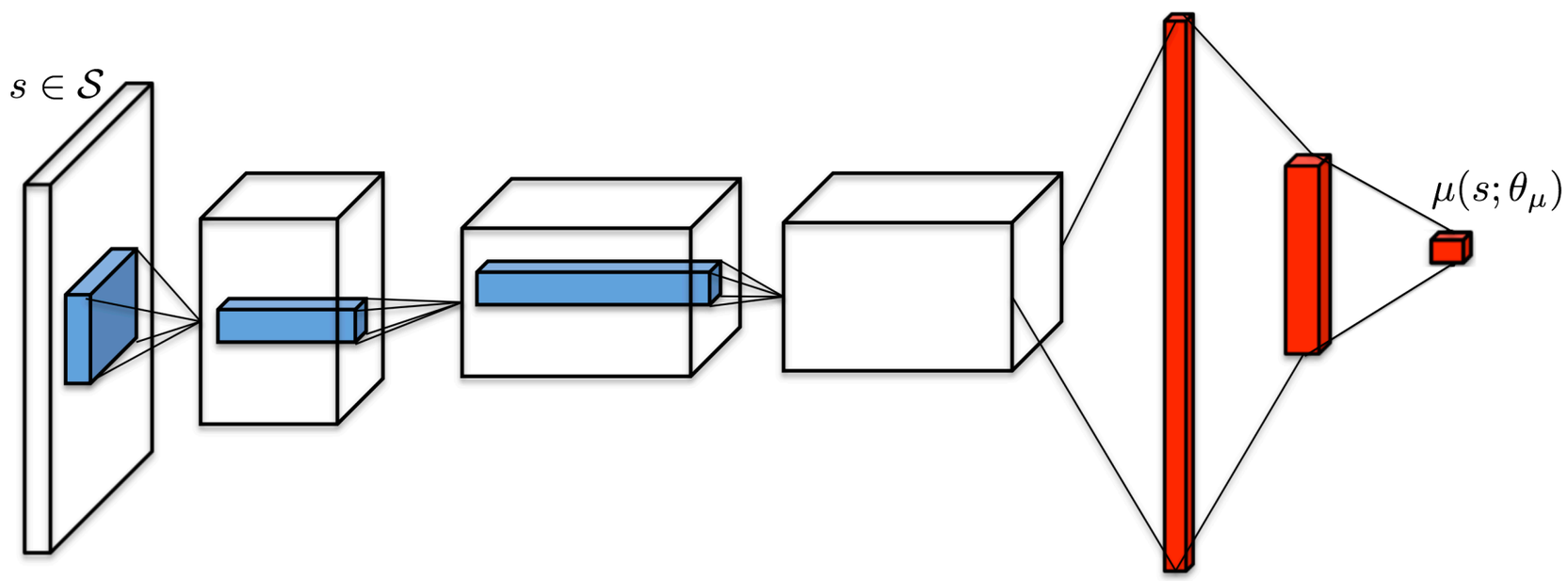


From “Continuous Control With Deep Reinforcement Learning”, Lillicrap, Hunt, et al., ICLR (2016)

Deterministic policy gradient

- Policy network training

$$\nabla\theta_\mu J(\theta_\mu) = \frac{1}{N} \sum_{(s,a,r,s')} \nabla_a Q(s, a; \theta_Q) \nabla_{\theta_\mu} \mu(s; \theta_\mu)$$



From “Continuous Control With Deep Reinforcement Learning”, Lillicrap, Hunt, et al., ICLR (2016)

Our ideas

Linear Q-function approximation

- Linear Q-function approximation

$$Q(s, a; \mathbf{c}) = \sum_{i=1}^{k} c_i \phi_i(s, a)$$

$\mathbf{c} = [c_1, c_2, \ldots, c_k]^T \in \mathbb{R}^k$ — vector of weights

$\phi_i(s, a)$ — i_{th} feature of state-action pair (s, a)

Linear Q-function approximation

- Linear Q-function approximation

$$Q(s, a; \mathbf{c}) = \sum_{i=1}^{k} c_i \phi_i(s, a)$$

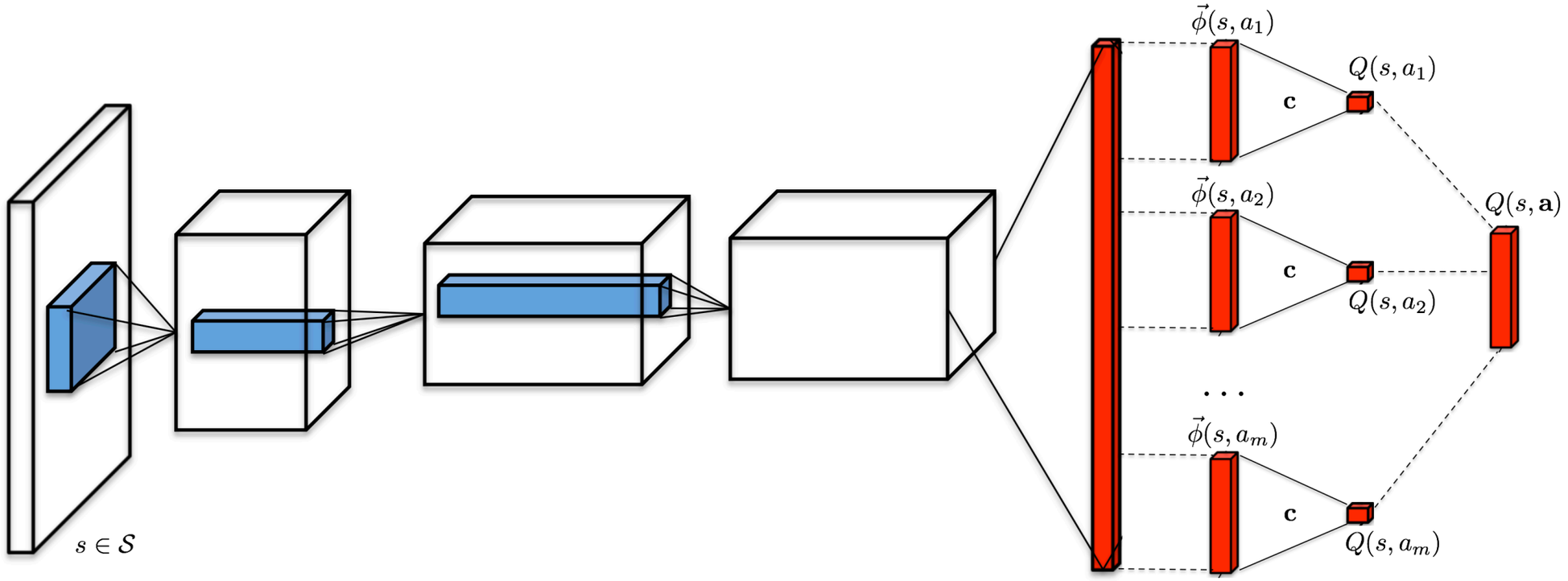
$\mathbf{c} = [c_1, c_2, \ldots, c_k]^T \in \mathbb{R}^k$ — vector of weights

$\phi_i(s, a)$ — i_{th} feature of state-action pair (s, a)

- State-action matrix of n training samples (s, a, r, s')

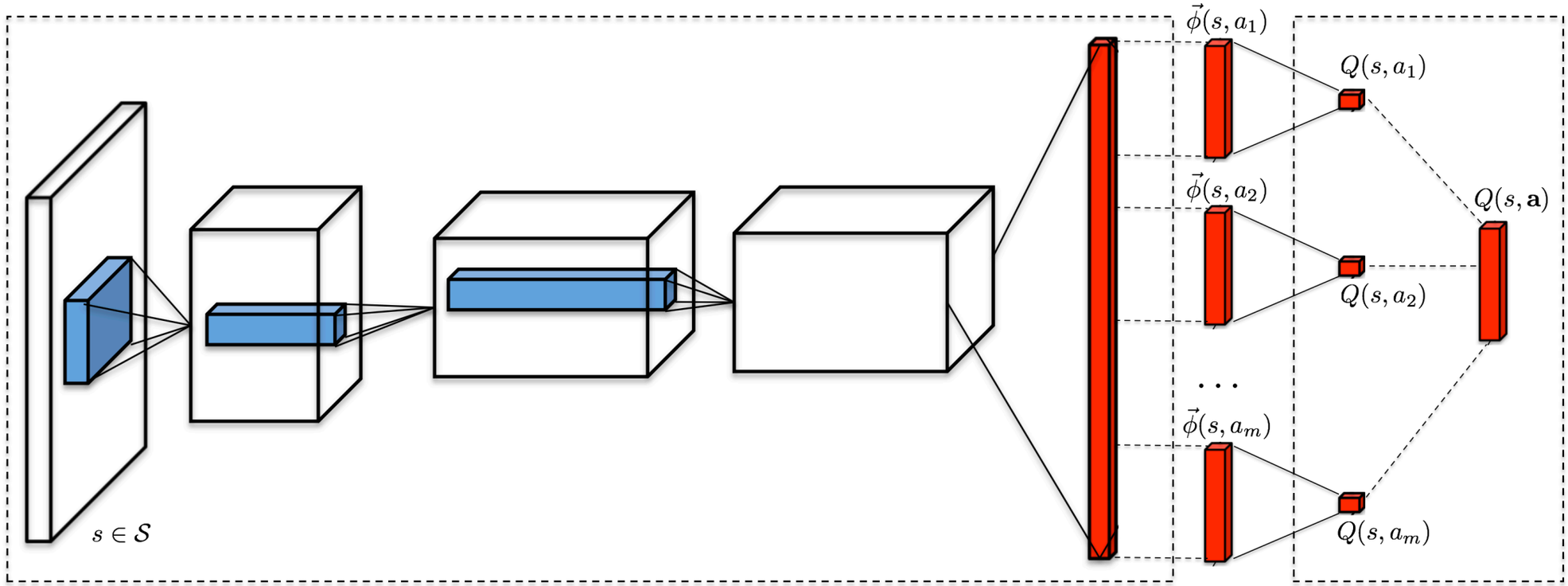
$$\mathbf{\Phi} = \begin{bmatrix} \phi_1(s_1, a_1) & \ldots & \phi_k(s_1, a_1) \\ \ldots & \ldots & \ldots \\ \phi_1(s_n, a_n) & \ldots & \phi_k(s_n, a_n) \end{bmatrix} \in \mathbb{R}^{n \times k}$$

Linear Q-function approximation



$$Q(s,a) = \sum_{i=1}^{k} c_i \phi_i(s,a) = \mathbf{c}^T \vec{\phi}(s,a)$$

Linear Q-function approximation



Representation

$$s \underset{\theta}{\rightarrow} \vec{\phi}(s, a_1), \ldots, \vec{\phi}(s, a_m)$$

Linear Q-function approximation

$$\vec{\phi}(s, a) \underset{\mathbf{c}}{\rightarrow} Q(s, a)$$

Linear Q-function approximation

- Q-function in a matrix form

$$\mathbf{q} = \Phi \mathbf{c}$$

$$\underset{n\times 1}{\mathbf{q}} = \underset{n\times k}{\Phi}\ \underset{k\times 1}{\mathbf{c}}$$

Linear Q-function approximation

- Q-function in a matrix form

$$\mathbf{q} = \Phi\mathbf{c}$$

$$\underset{n\times 1}{\mathbf{q}} = \underset{n\times k}{\Phi}\ \underset{k\times 1}{\mathbf{c}}$$

- Bellman optimality equation

$$\Phi\mathbf{c} = \mathbf{r} + \gamma\Phi'\mathbf{c}$$

$$\underset{n\times k}{\Phi}\ \underset{k\times 1}{\mathbf{c}} = \underset{n\times 1}{\mathbf{r}} + \gamma\ \underset{n\times k}{\Phi'}\ \underset{k\times 1}{\mathbf{c}}$$

Linear Q-function approximation

- Q-function in a matrix form

$$\mathbf{q} = \Phi\mathbf{c}$$

$$\underset{n\times 1}{\mathbf{q}} = \underset{n\times k}{\Phi}\ \underset{k\times 1}{\mathbf{c}}$$

- Bellman optimality equation

$$\underset{s,a}{\Phi}\,\mathbf{c} = \underset{r}{\mathbf{r}} + \gamma\,\underset{s',a'}{\Phi'}\,\mathbf{c}$$

$$\underset{n\times k}{\Phi}\ \underset{k\times 1}{\mathbf{c}} = \underset{n\times 1}{\mathbf{r}} + \gamma\,\underset{n\times k}{\Phi'}\ \underset{k\times 1}{\mathbf{c}}$$

$$a' = \arg\max_{a'} Q(s', a'; \mathbf{c})$$

Linear Q-function approximation

- Q-function in a matrix form

$$\mathbf{q} = \Phi\mathbf{c}$$

$$\underset{n\times 1}{\mathbf{q}} = \underset{n\times k}{\Phi}\ \underset{k\times 1}{\mathbf{c}}$$

- Bellman optimality equation

$$\underset{s,a}{\Phi}\,\mathbf{c} = \underset{r}{\mathbf{r}} + \gamma\,\underset{s',a'}{\Phi'}\,\mathbf{c}$$

$$\underset{n\times k}{\Phi}\ \underset{k\times 1}{\mathbf{c}} = \underset{n\times 1}{\mathbf{r}} + \gamma\,\underset{n\times k}{\Phi'}\ \underset{k\times 1}{\mathbf{c}}$$

$$a' = \arg\max_{a'} Q(s', a'; \mathbf{c})$$

- Exact solution

$$\mathbf{c} = \left(\Phi - \gamma\Phi'\right)^{\dagger}\mathbf{r}$$

Maxvol exploration

- Assume we have observed n transitions (s, a, r, s') and the agent visits state s_t

$$\Phi = \begin{bmatrix} \phi_1(s_1, a_1) & \dots & \phi_k(s_1, a_1) \\ \dots & \dots & \dots \\ \phi_1(s_n, a_n) & \dots & \phi_k(s_n, a_n) \end{bmatrix} \in \mathbb{R}^{n \times k}$$

Maxvol exploration

- Assume we have observed n transitions (s, a, r, s') and the agent visits state s_t

$$\Phi_t = \begin{bmatrix} \phi_1(s_1, a_1) & \dots & \phi_k(s_1, a_1) \\ \dots & \dots & \dots \\ \phi_1(s_n, a_n) & \dots & \phi_k(s_n, a_n) \\ \phi_1(s_t, a_t) & \dots & \phi_k(s_t, a_t) \end{bmatrix} \in \mathbb{R}^{(n+1)\times k}$$

- Which action a_t is best for the agent to choose?

Maxvol exploration

- Assume we have observed n transitions (s, a, r, s') and the agent visits state s_t

$$\Phi_t = \begin{bmatrix} \phi_1(s_1, a_1) & \dots & \phi_k(s_1, a_1) \\ \dots & \dots & \dots \\ \phi_1(s_n, a_n) & \dots & \phi_k(s_n, a_n) \\ \phi_1(s_t, a_t) & \dots & \phi_k(s_t, a_t) \end{bmatrix} \in \mathbb{R}^{(n+1)\times k}$$

- Which action a_t is best for the agent to choose?

$$\Phi(s_t) = \begin{bmatrix} \phi_1(s_t, a^1) & \dots & \phi_k(s_t, a^1) \\ \dots & \dots & \dots \\ \phi_1(s_t, a^m) & \dots & \phi_k(s_t, a^m) \end{bmatrix} \in \mathbb{R}^{m\times k}$$

Maxvol exploration

- Bellman optimality equation

$$\begin{bmatrix} \Phi \\ \vec{\phi} \end{bmatrix} \mathbf{c} = \begin{bmatrix} \mathbf{r} \\ r_t \end{bmatrix} + \gamma \begin{bmatrix} \Phi' \\ \vec{\phi}' \end{bmatrix} \mathbf{c}$$

Maxvol exploration

- Bellman optimality equation

$$\begin{bmatrix} \Phi \\ \vec{\phi} \end{bmatrix} \mathbf{c} = \begin{bmatrix} \mathbf{r} \\ r_t \end{bmatrix} + \gamma \begin{bmatrix} \Phi' \\ \vec{\phi}' \end{bmatrix} \mathbf{c}$$

- RHS can be estimated only after we choose a_t

Maxvol exploration

- Bellman optimality equation

$$\begin{bmatrix} \Phi \\ \vec{\phi} \end{bmatrix} \mathbf{c} = \begin{bmatrix} \mathbf{r} \\ r_t \end{bmatrix} + \gamma \begin{bmatrix} \Phi' \\ \vec{\phi}' \end{bmatrix} \mathbf{c}$$

- RHS can be estimated only after we choose a_t

- Assume that RHS does not depend on $\mathbf{c}$

$$\Phi_t \mathbf{c} = \mathbf{q}^{\text{target}}$$

Maxvol exploration

- Bellman optimality equation

$$\begin{bmatrix} \Phi \\ \vec{\phi} \end{bmatrix} \mathbf{c} = \begin{bmatrix} \mathbf{r} \\ r_t \end{bmatrix} + \gamma \begin{bmatrix} \Phi' \\ \vec{\phi}' \end{bmatrix} \mathbf{c}$$

- RHS can be estimated only after we choose a_t

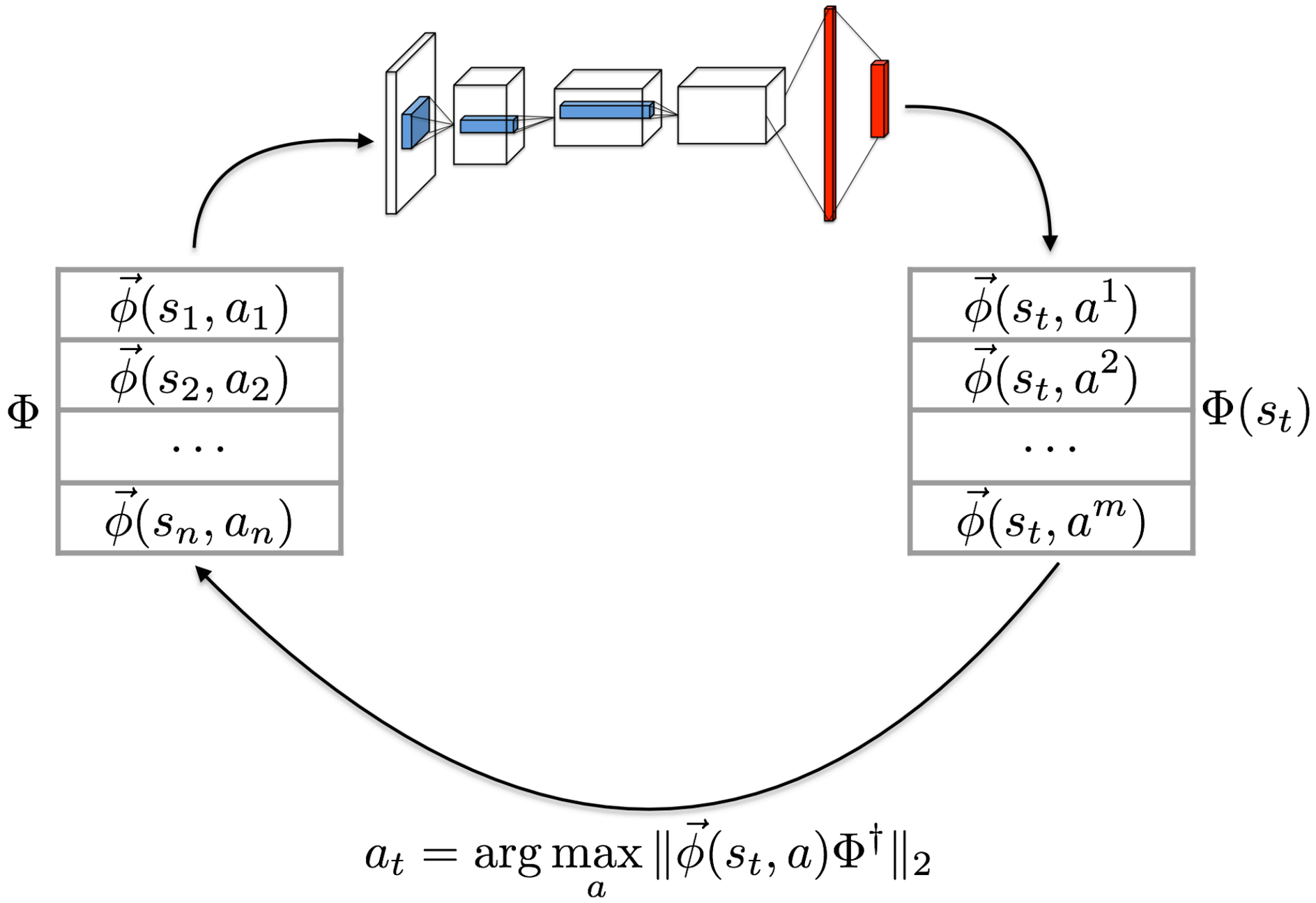
- Assume that RHS does not depend on $\mathbf{c}$

$$\Phi_t \mathbf{c} = \mathbf{q}^{\text{target}}$$

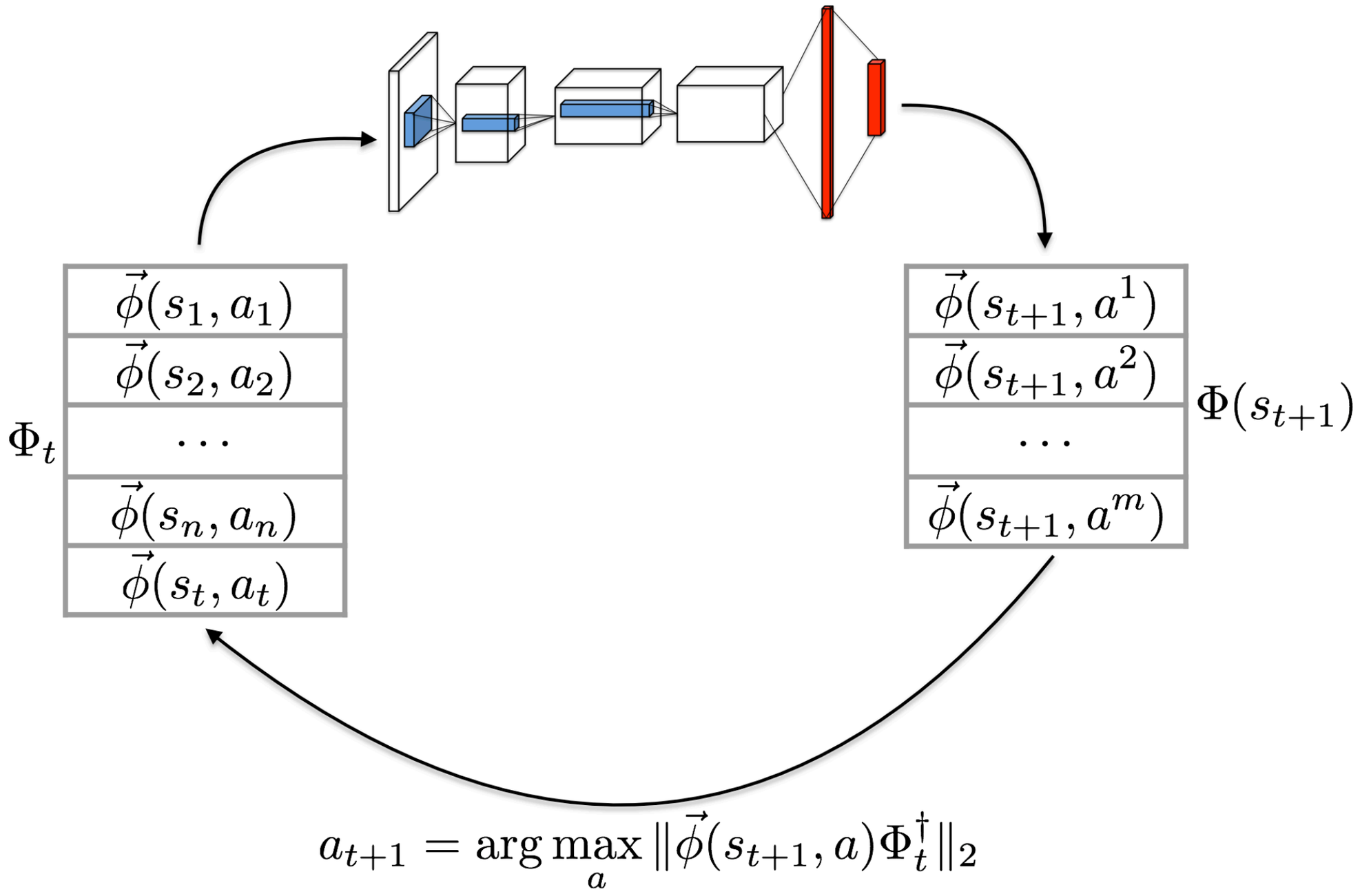
- We need to choose the action a_t such that

$$\|\vec{\phi}(s_t, a_t)\Phi^{\dagger}\|_2 \to \max_{a_t}$$

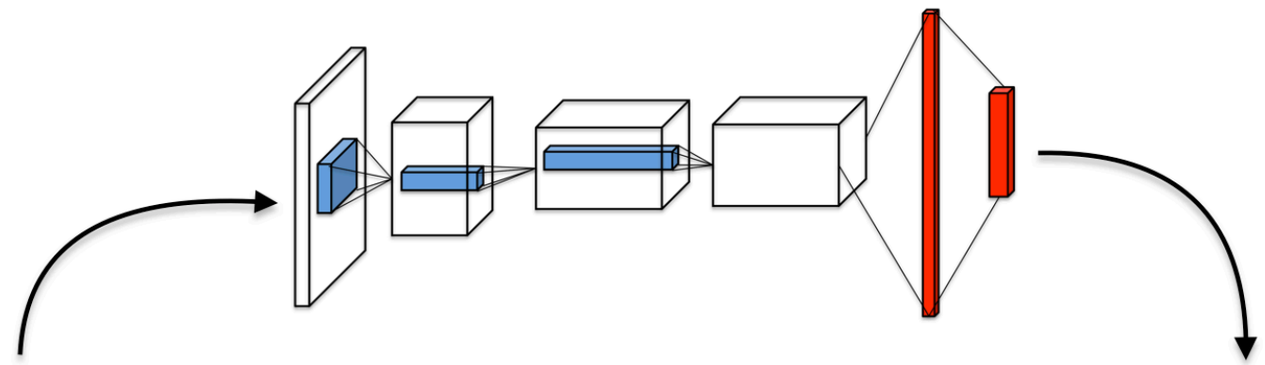
Maxvol exploration



Maxvol exploration



Maxvol exploration



Φ_{t+1}

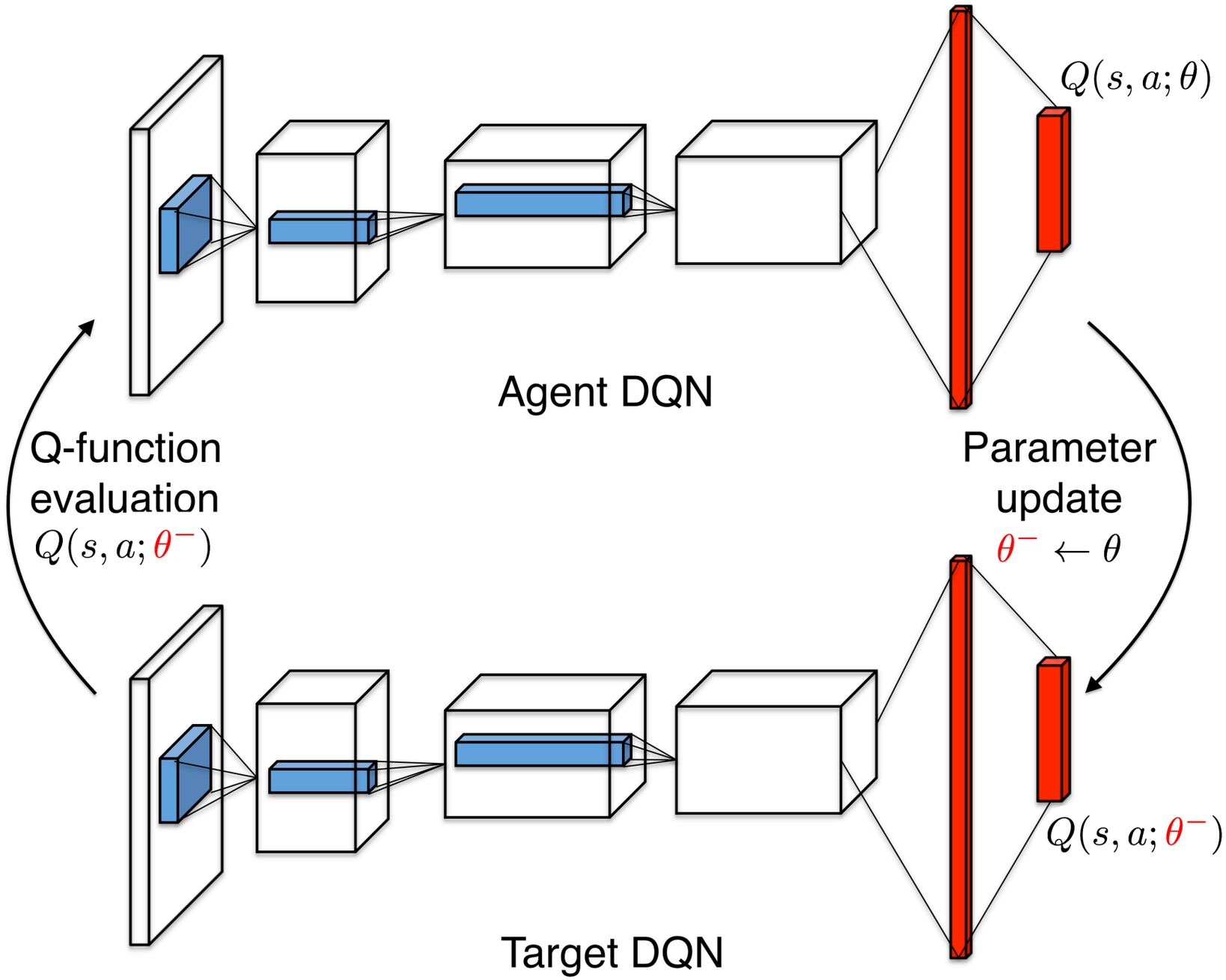
$\vec{\phi}(s_1, a_1)$
$\vec{\phi}(s_2, a_2)$
$\dots$
$\vec{\phi}(s_n, a_n)$
$\vec{\phi}(s_t, a_t)$
$\vec{\phi}(s_{t+1}, a_{t+1})$

$\vec{\phi}(s_{t+2}, a^1)$
$\vec{\phi}(s_{t+2}, a^2)$
$\dots$
$\vec{\phi}(s_{t+2}, a^m)$

$\Phi(s_{t+2})$

$$a_{t+2} = \arg\max_a \| \vec{\phi}(s_{t+2}, a) \Phi_{t+1}^{\dagger} \|_2$$

Deep Q-network



One-shot target update