Off-policy deep reinforcement learning algorithms

Oleksii Hrinchuk

DQN

Online Table Q-learning

1. Initialize Q-function Q (e.g. zeros), set $\pi = \varepsilon\text{-greedy}(Q)$

2. (Q-function update step)
   — Make a step $\{\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{s}_{t+1}\}$ in the environment using π
   — Update Q-values with TD-targets

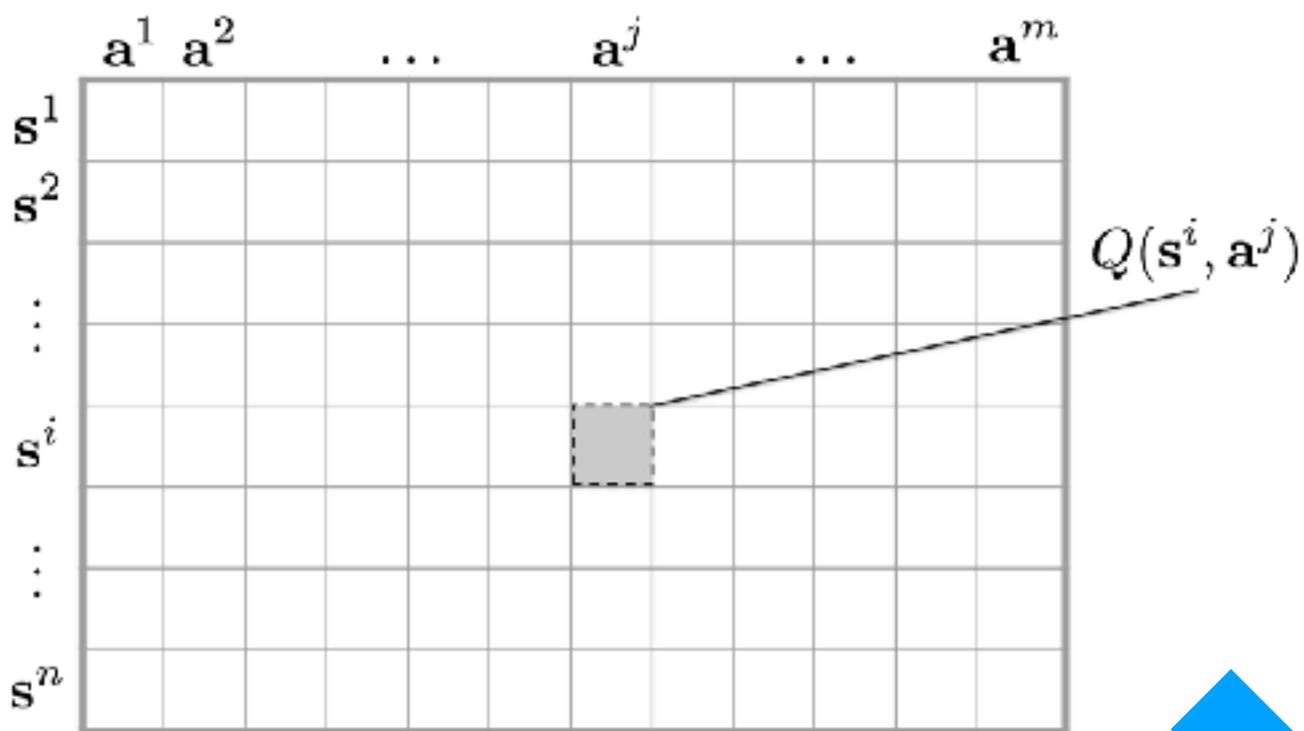
$$y_t = \mathbf{r}_t + \gamma \max_{\mathbf{a}' \in \mathcal{A}} Q(\mathbf{s}_{t+1}, \mathbf{a}')$$

$$Q(\mathbf{s}_t, \mathbf{a}_t) \leftarrow Q(\mathbf{s}_t, \mathbf{a}_t) + \alpha \left[y_t - Q(\mathbf{s}_t, \mathbf{a}_t) \right]$$

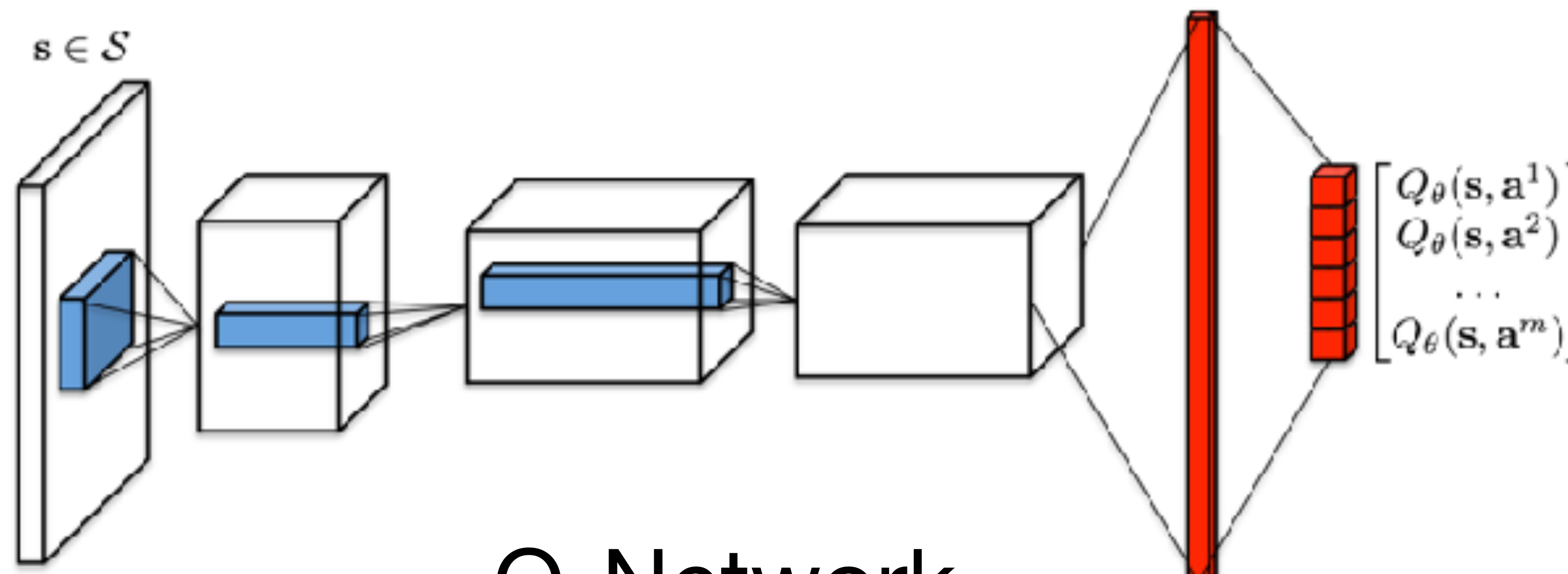
   — Update behavior policy $\pi = \varepsilon\text{-greedy}(Q)$

3. If policy is good, return $\pi^* = \text{greedy}(Q)$, else go to step 2

From table to neural network



Q-table



Q-Network

Online Q-learning with NN

1. Initialize weights θ of Q-Network Q_θ, set $\pi = \varepsilon\text{-greedy}(Q_\theta)$

2. (Q-function update step)
   - Make a step $\{\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{s}_{t+1}\}$ in the environment using π
   - Update Q-Network weights with TD-targets

$$y_t = \begin{cases} \mathbf{r}_t + \gamma \max\limits_{\mathbf{a}' \in \mathcal{A}} Q_\theta(\mathbf{s}_{t+1}, \mathbf{a}'), \ \text{done=False} \\ \mathbf{r}_t, \ \text{done=True} \end{cases}$$

$$\mathcal{L}_\theta(\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{s}_{t+1}) = [Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - y_t]^2$$

$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_\theta(\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{s}_{t+1})$$

But, it does not work!

   - Update behavior policy $\pi = \varepsilon\text{-greedy}(Q_\theta)$

3. If policy is good, return $\pi^* = \text{greedy}(Q_\theta)$, else go to step 2

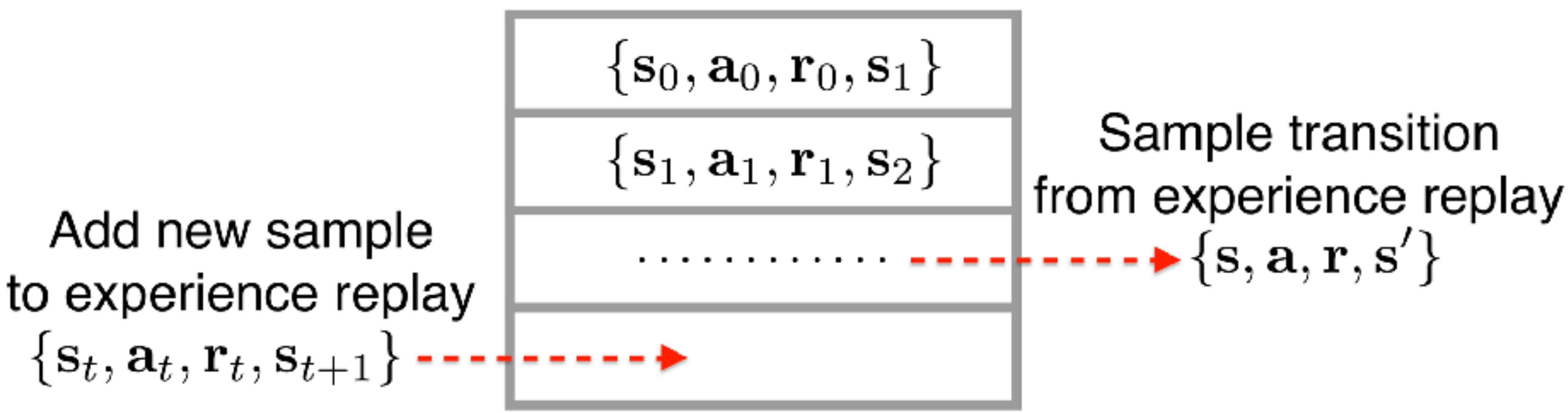
Why it does not work?

- The correlations present in the sequence of observations Experience replay
- The correlation between the Q-values and the TD-targets in the update Freezing target network
- The fact that small updates to Q may significantly change the policy and therefore change the data distribution Stable mini-batch training

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

Experience replay

- Store most recent agent-environment interactions in replay buffer
- Sample transitions uniformly when updating network weights



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

Freezing target network

- Increase of $Q_\theta(\mathbf{s}_t, \mathbf{a}_t)$ results in increase of $Q_\theta(\mathbf{s}_{t+1}, \mathbf{a}')$

$$\mathcal{L}_\theta(\mathbf{d}_t) = \left[Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - \left(\mathbf{r}_t + \gamma \max_{\mathbf{a}' \in \mathcal{A}} Q_\theta(\mathbf{s}_{t+1}, \mathbf{a}') \right) \right]^2$$

- Assume that TD-target y_t does not depend on θ

$$\nabla_\theta \mathcal{L}_\theta(\mathbf{d}_t) = \nabla_\theta Q_\theta(\mathbf{s}_t, \mathbf{a}_t) \left[Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - y_t \right]$$

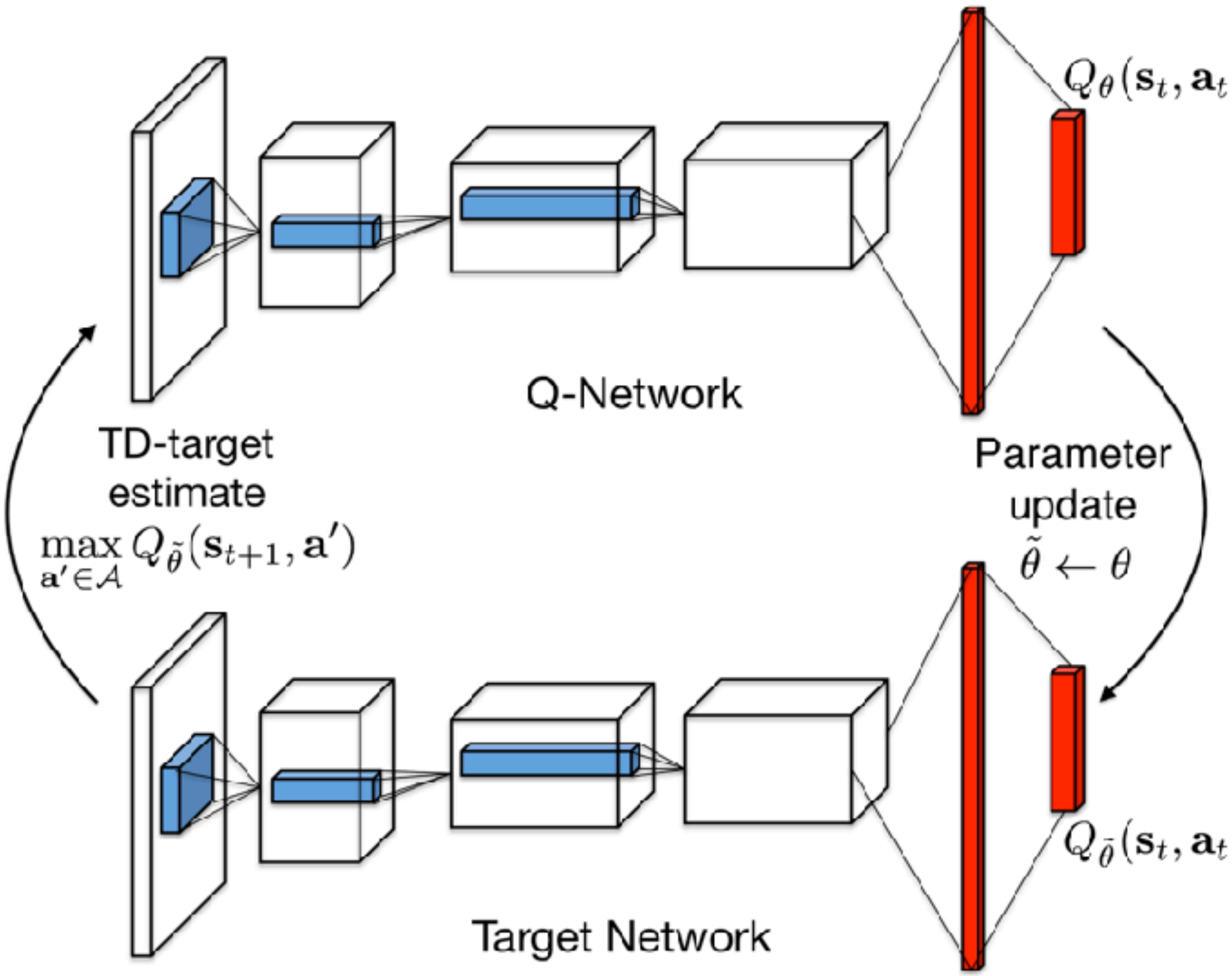
- Use separate network for TD-target evaluation

$$y_t = \mathbf{r}_t + \gamma \max_{\mathbf{a}' \in \mathcal{A}} Q_{\tilde{\theta}}(\mathbf{s}_{t+1}, \mathbf{a}')$$

- Synchronize target network once in a while: after every N iterations do $\tilde{\theta} \leftarrow \theta$

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

Freezing target network



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

Mini-batch training

- After a number of environment steps, sample a mini-batch of transitions (e.g. 32 transitions) from experience buffer and update the network weights

$$\{\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_B\} \sim \mathcal{D}$$

$$\mathcal{L}_\theta = \frac{1}{B} \sum_{i=1}^{B} \mathcal{L}_\theta(\mathbf{d}_i)$$

- Good off-policy training through data re-use

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

Deep Q-Network (DQN)

1. Initialize Q-Network Q_θ , set initial exploration rate $\varepsilon = 1$. Run epsilon-greedy policy w.r.t. Q_θ for a number of steps to initialize $\mathcal{D}$.

2. (Train loop)
   — Make a step in the environment with policy $\pi = \varepsilon\text{-greedy}(Q_\theta)$
   — Put transition $\mathbf{d}_t = \{\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{s}_{t+1}\}$ into replay buffer $\mathcal{D}$
   — If step % q_update_frequency == 0:
     Sample mini-batch of transitions $\{\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_B\} \sim \mathcal{D}$
     Update Q-Network weights $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_\theta$
   — If step % target_update_frequency == 0:
     Update target network weights $\tilde{\theta} \leftarrow \theta$
   — Decay exploration rate $\varepsilon \leftarrow \varepsilon - \Delta\varepsilon$

3. Return greedy policy w.r.t. the learned Q-function Q_θ

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

Hyperparameters

Hyperparameter	Value	Description
minibatch size	32	Number of training cases over which each stochastic gradient descent (SGD) update is computed.
replay memory size	1000000	SGD updates are sampled from this number of most recent frames.
agent history length	4	The number of most recent frames experienced by the agent that are given as input to the Q network.
target network update frequency	10000	The frequency (measured in the number of parameter updates) with which the target network is updated (this corresponds to the parameter *C* from Algorithm 1).
discount factor	0.99	Discount factor gamma used in the Q-learning update.
action repeat	4	Repeat each action selected by the agent this many times. Using a value of 4 results in the agent seeing only every 4th input frame.
update frequency	4	The number of actions selected by the agent between successive SGD updates. Using a value of 4 results in the agent selecting 4 actions between each pair of successive updates.
learning rate	0.00025	The learning rate used by RMSProp.
gradient momentum	0.95	Gradient momentum used by RMSProp.
squared gradient momentum	0.95	Squared gradient (denominator) momentum used by RMSProp.
min squared gradient	0.01	Constant added to the squared gradient in the denominator of the RMSProp update.
initial exploration	1	Initial value of ε in ε-greedy exploration.
final exploration	0.1	Final value of ε in ε-greedy exploration.
final exploration frame	1000000	The number of frames over which the initial value of ε is linearly annealed to its final value.
replay start size	50000	A uniform random policy is run for this number of frames before learning starts and the resulting experience is used to populate the replay memory.

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

DDPG

Continuous actions

- Greedy policy in the case of continuous actions

$$\mu(\mathbf{s}) = \arg\max_{\mathbf{a}\in\mathcal{A}} Q_\theta(\mathbf{s}, \mathbf{a})$$

- Finding maximum is a hard optimization problem we don't know how to solve.

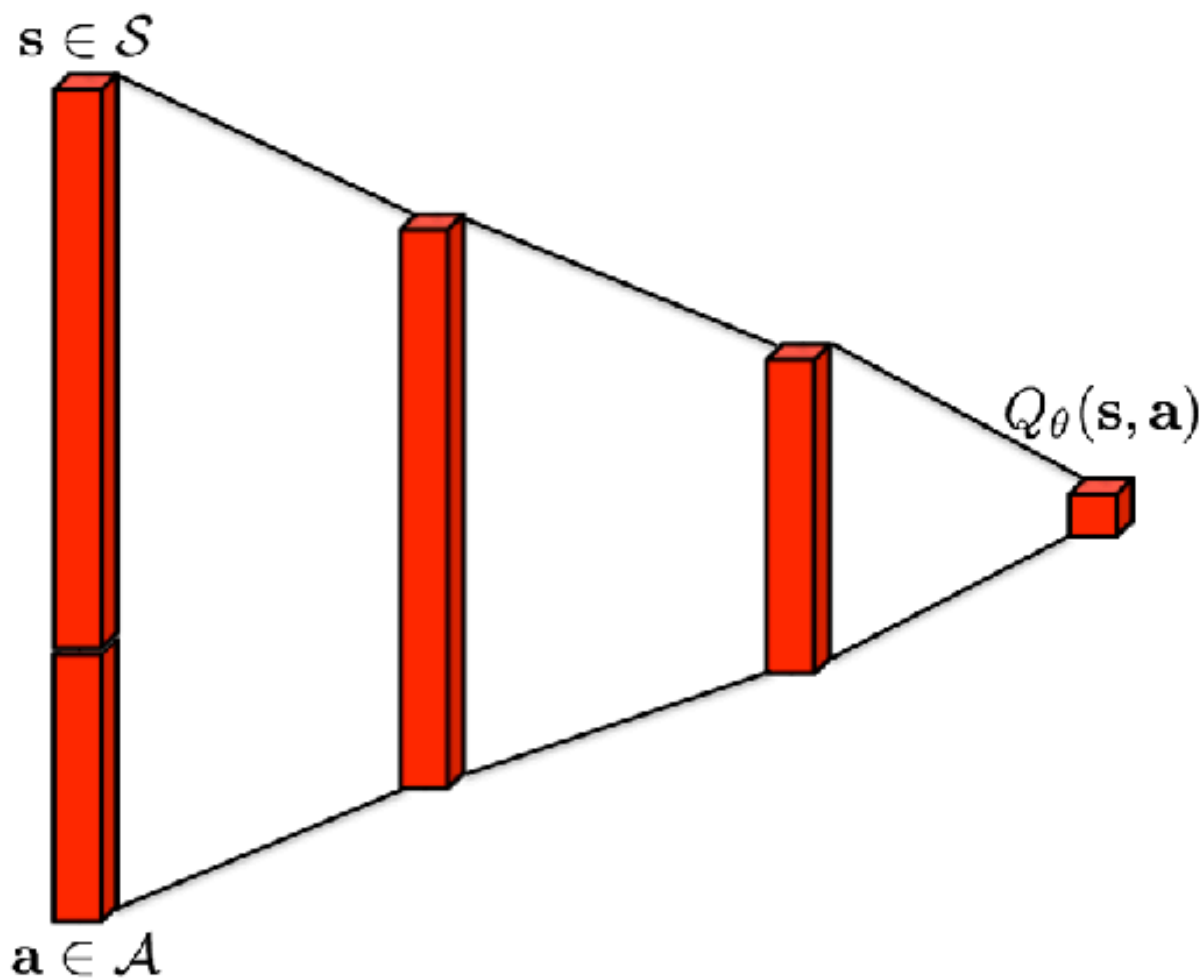
- Idea: train another network which tries to solve this problem for us!

$$\mu_\phi(\mathbf{s}) \approx \arg\max_{\mathbf{a}\in\mathcal{A}} Q_\theta(\mathbf{s}, \mathbf{a}), \quad Q_\theta(\mathbf{s}, \mu_\phi(\mathbf{s})) \to \max_\phi$$

$$\phi \leftarrow \phi + \alpha \nabla_\phi Q_\theta(\mathbf{s}, \mu_\phi(\mathbf{s}))$$

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

Critic network (Q-function)

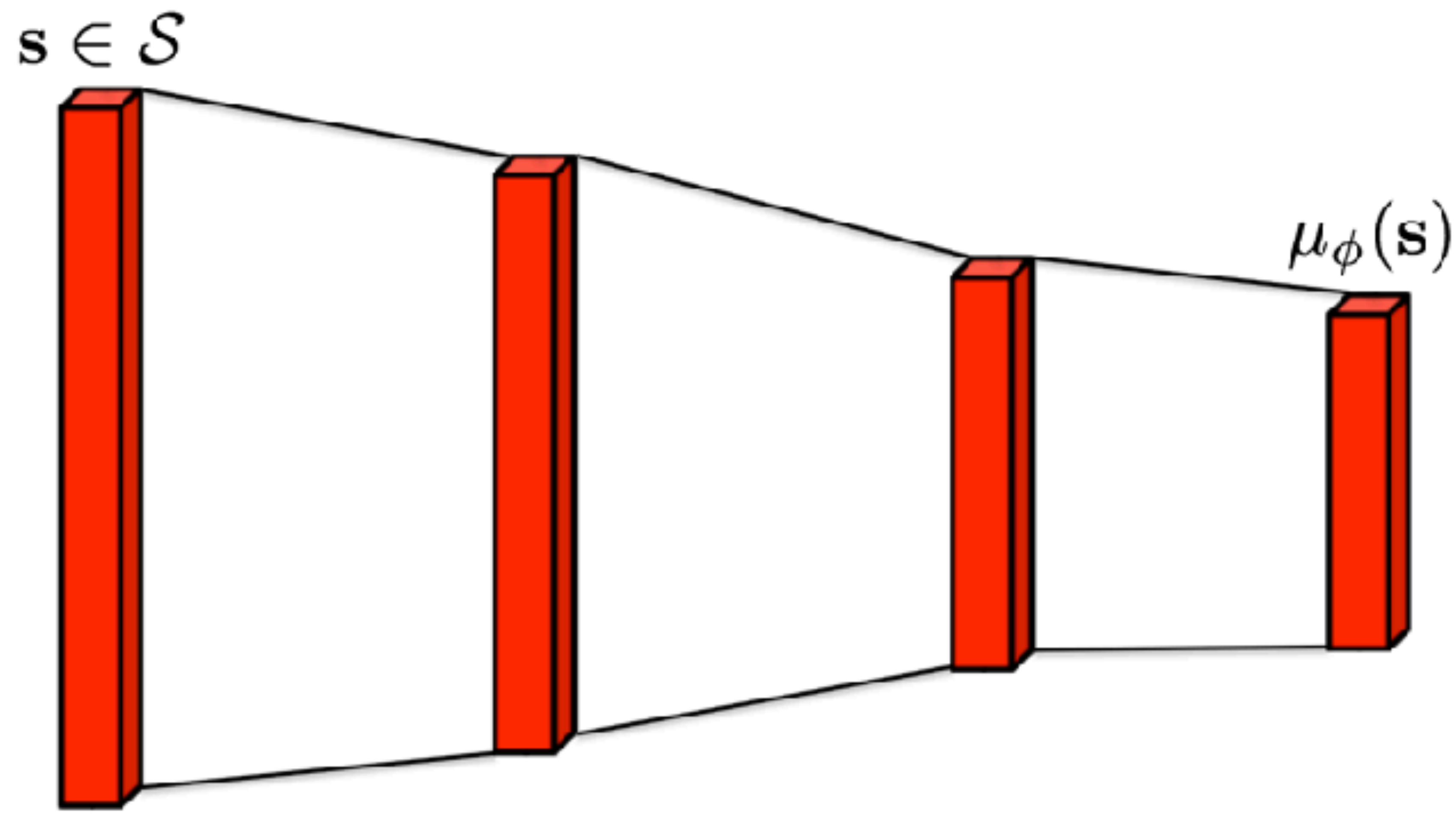


$$\mathcal{L}_\theta^{\text{critic}}(\mathbf{d}_t) = \left[Q_\theta(\mathbf{s}_t, \mathbf{a}_t) - \left(\mathbf{r}_t + \gamma Q_{\tilde{\theta}}(\mathbf{s}_{t+1}, \mu_{\tilde{\phi}}(\mathbf{s}_{t+1})\right)\right]^2$$

Loss function

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

Actor network (policy)



$$\mathcal{L}_\phi^{\text{actor}}(\mathbf{d}_t) = -Q_\theta(\mathbf{s}_t, \mu_\phi(\mathbf{s}_t))$$

Pseudo-loss function

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

Two more distinctions from DQN

- Instead of fully updating target networks once after each N training transitions, update them frequently but slowly

$$\tilde{\theta} \leftarrow (1-\tau)\tilde{\theta} + \tau\theta, \quad \tilde{\phi} \leftarrow (1-\tau)\tilde{\phi} + \tau\phi$$

- Add gaussian noise to actions for exploration

$$\mathbf{a}_t = \mu_\phi(\mathbf{s}_t) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

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

Deep Deterministic Policy Gradient (DDPG)

1. Initialize actor μ_ϕ and critic Q_θ networks, initialize replay buffer $\mathcal{D}$

2. (Train loop)
   — Make a step in the environment with policy $\mu_\phi(\mathbf{s}_t) + \mathcal{N}(0, \sigma^2)$
   — Put transition $\mathbf{d}_t = \{\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{s}_{t+1}\}$ into replay buffer $\mathcal{D}$
   — Sample mini-batch of transitions $\{\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_B\} \sim \mathcal{D}$
   — Update critic weights $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_\theta^{\mathrm{critic}}$
   — Update actor weights $\phi \leftarrow \phi - \alpha \nabla_\phi \mathcal{L}_\phi^{\mathrm{actor}}$
   — Update target networks $\tilde{\theta} \leftarrow (1-\tau)\tilde{\theta} + \tau\theta, \quad \tilde{\phi} \leftarrow (1-\tau)\tilde{\phi} + \tau\phi$

3. Return policy of the learned actor network

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

Improvements to DQN

Double DQN (DDQN)

- Max operator in TD-targets leads to overoptimistic estimates of Q-function.

$$
\begin{aligned}
y_t &= \mathbf{r}_t + \gamma \max_{\mathbf{a}' \in \mathcal{A}} Q_{\tilde{\theta}}(\mathbf{s}_{t+1}, \mathbf{a}') \\
&= \mathbf{r}_t + \gamma Q_{\tilde{\theta}}(\mathbf{s}_{t+1}, \arg\max_{\mathbf{a}' \in \mathcal{A}} Q_{\tilde{\theta}}(\mathbf{s}_{t+1}, \mathbf{a}'))
\end{aligned}
$$

- Q-Network is used to select actions, target network is used to evaluate actions

$$
y_t = \mathbf{r}_t + \gamma Q_{\tilde{\theta}}(\mathbf{s}_{t+1}, \arg\max_{\mathbf{a}' \in \mathcal{A}} Q_{\theta}(\mathbf{s}_{t+1}, \mathbf{a}'))
$$

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

Advantage function

- Advantage function for a given policy π

$$A^{\pi}(\mathbf{s}, \mathbf{a}) = Q^{\pi}(\mathbf{s}, \mathbf{a}) - V^{\pi}(\mathbf{s}), \quad \mathbb{E}_{\mathbf{a}} A^{\pi}(\mathbf{s}, \mathbf{a}) = 0$$

- Optimal advantage function

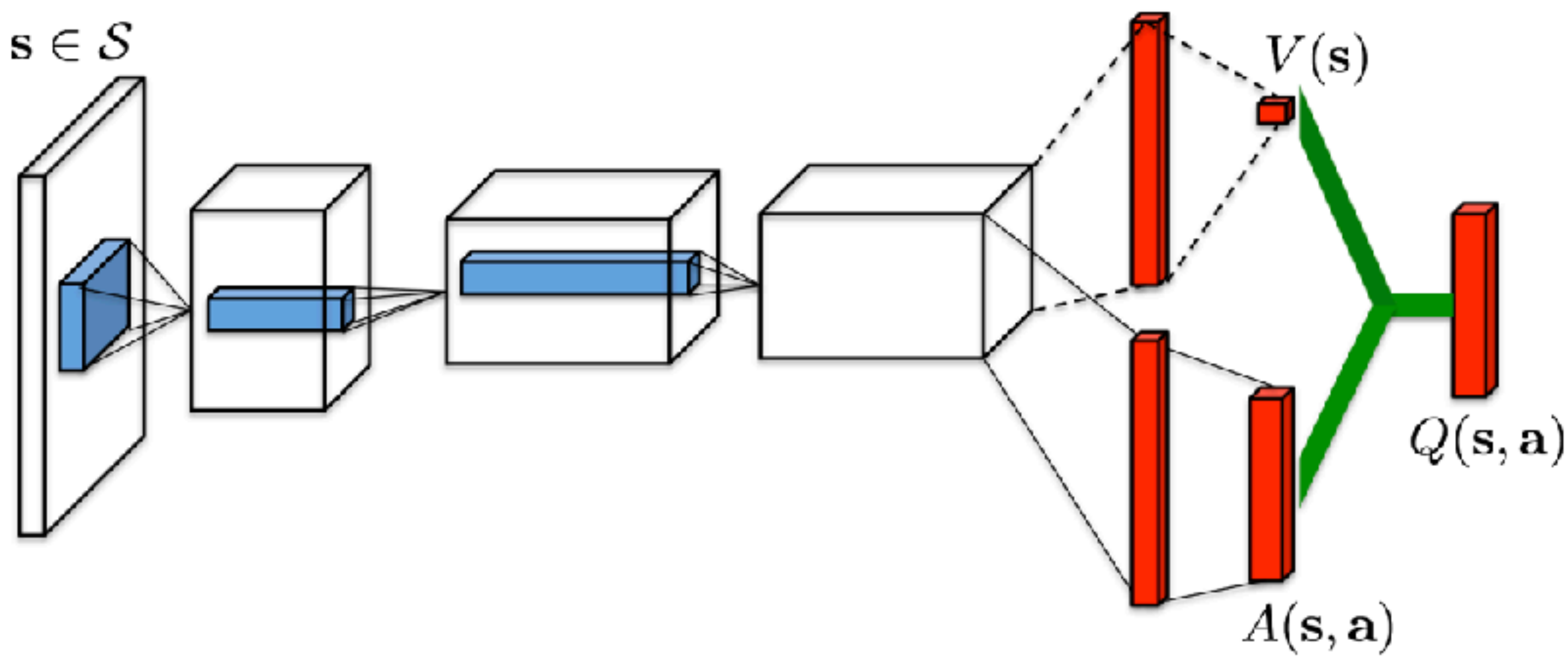
$$A^{*}(\mathbf{s}, \mathbf{a}) = Q^{*}(\mathbf{s}, \mathbf{a}) - V^{*}(\mathbf{s}), \quad \max_{\mathbf{a} \in \mathcal{A}} A^{\pi}(\mathbf{s}, \mathbf{a}) = 0$$

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

$$Q(\mathbf{s}, \mathbf{a}) = V(\mathbf{s}) + A(\mathbf{s}, \mathbf{a})$$

Dueling DDQN

- Split Q-function into two channels
  - Action independent value function $V(\mathbf{s})$
  - Action dependent advantage function $A(\mathbf{s}, \mathbf{a})$



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

Dueling DDQN

- Calculating Q-function as $Q(\mathbf{s}, \mathbf{a}) = V(\mathbf{s}) + A(\mathbf{s}, \mathbf{a})$ results in ambiguity, as

$$Q(\mathbf{s}, \mathbf{a}) = (V(\mathbf{s}) + c) + (A(\mathbf{s}, \mathbf{a}) - c)\,, \quad \forall c$$

- The ambiguity is resolved after adding additional constraint $A(\mathbf{s}, \arg\max_{\mathbf{a} \in \mathcal{A}} A(\mathbf{s}, \mathbf{a})) = 0$

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

- In practice

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

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

Prioritized experience replay

- Transitions with high TD-error are sampled more often

$$\mathbf{d}_t = \{\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{s}_{t+1}\}$$

$$\delta_t = \mathbf{r}_t + \gamma \max_{\mathbf{a}' \in \mathcal{A}} Q(\mathbf{s}_{t+1}, \mathbf{a}') - Q(\mathbf{s}_t, \mathbf{a}_t)$$

- Define prioritization
  - proportional $p_t = |\delta_t| + \epsilon$
  - rank-based $p_t = 1/\mathrm{rank}(t)$

- Sample transitions according to probability

$$P(\mathbf{d}_t) = \frac{p_t^\alpha}{\sum_{t'=1}^{|\mathcal{D}|} p_{t'}^\alpha}$$

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