Deep Reinforcement Learning

Shusen Wang
Stevens Institute of Technology

December 9, 2019

Abstract

This lecture note briefly summarizes three kinds of deep reinforcement learning approaches: value-based methods, policy-based methods, and actor-critic methods. This note is structured as following. First, reinforcement learning terminologies are defined. Second, we study Deep Q Network (DQN), a family of value-based methods, and train DQN using temporal difference (TD) learning. Third, we study policy-based learning and derive policy gradient algorithms. Last, we study standard (random) actor-critic method and deterministic actor-critic method.

1 Notation

Throughout, we use uppercase letters, e.g., X, to denote random variables and lowercase letters, e.g., x, to denote their observations. Let $\mathbb{P}(X = x)$ be the probability of the event "$X = x$". Let $\mathbb{P}(Y = y | X = x)$ be the probability of the event "$Y = y$" under the condition "$X = x$".

Agent: A system that is embedded in an environment and takes actions to change the state of the environment. Examples include robots, industrial controllers, and Mario in the game Super Mario.

State (S): State can be viewed as a summary of the history of the system that determines its future evolution. State space $\mathcal{S}$ is the set that contains all the possible states. At time step t, the past states are observed and we thus know their values: $s_1, \cdots, s_t$; however, the future states $S_{t+1}, S_{t+2}, \cdots$ are unobserved random variables.

Action (A): The agent's decision based on the state and other considerations. Action space $\mathcal{A}$ is the set that contains all the actions. Action space can be a discrete set such as $\{$"left", "right", "up"$\}$ or a continuous set such as $[0, 1] \times [-90, 90]$. At time step t, the past actions are observed: $a_1, \cdots, a_t$, but the future actions $A_{t+1}, A_{t+2}, \cdots$ are unobserved random variables.

Reward (R): Reward is a value received by the agent from the environment as a direct response to the agent's actions. At time step t, all the past rewards are observed: $r_1, r_2, \cdots, r_t$. However, the future reward R_i (for $i > t$) is unobserved, and it depends on the random variables, S_{t+1} and A_{t+1}. Thus, at time step t, the future rewards $R_{t+1}, R_{t+2}, \cdots$ are random variables.

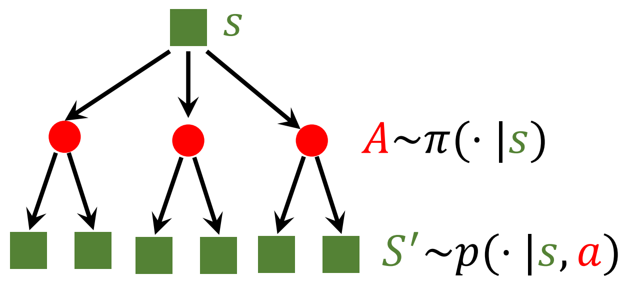

Figure 1: Illustration of the randomness. The action A is randomly sampled according to the policy function. The new state S' is randomly sampled according to the state-transition function.

Policy function (π): The decision-making function of the agent. Policy is the probability density function (PDF): $\pi(a|s) = \mathbb{P}(A = a|S = s)$. The policy function maps the observed state $S = s$ to a probability distribution over all the actions in set $\mathcal{A}$. Since π is a PDF, $\sum_{a\in\mathcal{A}} \pi(a|s) = 1$. The agent will perform action a with probability $\pi(a|s)$, for all $a \in \mathcal{A}$. See the illustration in Figure 1.

State transition (p): Given the current state $S = s$, the agent's action $A = a$ will lead to the new state S' given by the environment. State-transition function is the probability density function (PDF) $p(s'|s, a) = \mathbb{P}(S' = s'|S = s, A = a)$. The environment makes s' the new state with probability $p(s'|s, a)$, for all $s' \in \mathcal{S}$.

Trajectory: The agent's interaction with the environment results in a sequence of (state, action, reward) triplets: $s_1, a_1, r_1, s_2, a_2, r_2, s_3, a_3, r_3, \cdots$

Return (U): Return (aka cumulative future reward) is defined as

$$U_t = R_t + R_{t+1} + R_{t+2} + R_{t+3} + \cdots$$

Discounted return (aka cumulative discounted future reward) is defined as

$$U_t = R_t + \gamma \cdot R_{t+1} + \gamma^2 \cdot R_{t+2} + \gamma^3 \cdot R_{t+3} + \cdots$$

Here, $\gamma \in (0, 1)$ is the discount rate. The return U_t is random because the future rewards $R_t, R_{t+1}, R_{t+2}, \cdots$ are unobserved random variables. Recall that the randomness in the R_i ($i \geq t$) comes from the future states S_i and action A_i.

Action-value function (Q_π): Action-value function $Q_\pi(s_t, a_t)$ measures given state s_t and policy π, how good the action a_t is. Formally speaking,

$$Q_\pi(s_t, a_t) \;=\; \mathbb{E}\big[U_t \,\big|\, S_t = s_t, A_t = a_t\big].$$

The expectation is taken w.r.t. the future actions $A_{t+1}, A_{t+2}, \cdots$ and future states $S_{t+1}, S_{t+2}, \cdots$ which are random variables. Note that $Q_\pi(s_t, a_t)$ depends on the policy function π and the state-transition function p.

Optimal action-value function ($Q^\star$): The optimal action-value function $Q^\star(s_t, a_t)$ measures how good the action a_t is at state s_t. Formally speaking,

$$Q^\star(s,a) \;=\; \max_\pi Q_\pi(s,a).$$

Note that $Q^\star(s,a)$ is independent of the the policy function π.

State-value function (V_π): State-value function $V_\pi(s_t)$ measures given π, how good the current situation s_t is. Formally speaking,

$$V_\pi(s_t) \;=\; \mathbb{E}_{A\sim\pi(\cdot|s_t)}\big[Q_\pi(s_t,A)\big] \;=\; \int_{\mathcal{A}} \pi(a|s_t)\cdot Q_\pi(s_t,a)\,d\,a.$$

Here, the action A is treated as a random variable and integrated out.

Optimal state-value function ($V^\star$): The optimal state-value function $V^\star(s_t)$ measures how good the current situation s_t is. Formally speaking,

$$V_\pi(s) \;=\; \max_\pi V_\pi(s).$$

Note that $V^\star(s)$ is independent of the the policy function π.

2 Value-Based Deep Reinforcement Learning

The optimal action-value function $Q^\star(s,a)$ can be used to control the agent: observing state s_t, the agent performs

$$a_t \;=\; \operatorname*{argmax}_{a\in\mathcal{A}} Q^\star(s_t,a).$$

The optimal action-value function can be approximated by the neural network $Q(s,a;\mathbf{w})$ where $\mathbf{w}$ captures the model parameters. The neural network is called **Deep Q Network (DQN)**.

There are different designs of network architecture. Here, we consider the game Super Mario, in which the the action space is discrete: $\mathcal{A} = \{$"left", "right", "up"$\}$. DQN takes state s_t (which can be a screenshot or several most recent screenshots) as input. The architecture can be

$$\texttt{State} \Rightarrow \texttt{Conv} \Rightarrow \texttt{Flatten} \Rightarrow \texttt{Dense} \Rightarrow \texttt{Values}.$$

In the Super Mario example, DQN outputs a 3-dimensional vector, e.g., $[200, 100, 250]$, whose entries corresponds to the three actions. Then the action should be

$$a_t \;=\; \operatorname*{argmax}_{a} Q(s_t,a;\mathbf{w}).$$

Since $Q(s_t, \text{"up"}; \mathbf{w}) = 250$ is the biggest value among the three, $a_t =$"up" will be the selected action.

DQN is typically trained using **temporal different (TD) learning** [4, 5] which allows for updating the model parameters every time a reward $R_t = r_t$ is observed. By definition, $U_t = \sum_{i=1}\gamma^{i-t}\cdot R_i$. Thus

$$U_t \;=\; R_t + \gamma\cdot U_{t+1}.$$

TD learning makes use of the fact:

$$Q_\pi(s_t, a_t) \;=\; \mathbb{E}\big[U_t \,\big|\, s_t, a_t\big] \;=\; \mathbb{E}\big[R_t + \gamma \cdot U_{t+1} \,\big|\, s_t, a_t\big] \;=\; \mathbb{E}\big[R_t + \gamma \cdot Q_\pi(S_{t+1}, A_{t+1}) \,\big|\, s_t, a_t\big].$$

Since $Q(s_t, a_t; \mathbf{w}) \approx \max_\pi \mathbb{E}[U_t | s_t, a_t]$, we have

$$Q(s_t, a_t; \mathbf{w}) \;\approx\; r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}; \mathbf{w}).$$

Before observing R_t, the expected return was

$$q_t \;=\; Q(s_t, a_t; \mathbf{w})$$

After observing $R_t = r_t$, the expected return is updated to

$$y_t \;=\; r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}; \mathbf{w}),$$

which is called **TD target**. The **TD error** is $\delta_t = q_t - y_t$. We seek to encourage a small TD error and thus define the loss:

$$L_t \;=\; \frac{1}{2}\delta_t^2 \;=\; \frac{1}{2}\big[Q(s_t, a_t; \mathbf{w}) - y_t\big]^2.$$

Pretend y_t is not a function of $\mathbf{w}$. Then the gradient is

$$\mathbf{g}_t \;\triangleq\; \frac{\partial\, L_t}{\partial\, \mathbf{w}}\Big|_{\mathbf{w}=\mathbf{w}_t} \;=\; \delta_t \cdot \frac{\partial\, Q(s_t, a_t; \mathbf{w})}{\partial\, \mathbf{w}}\Big|_{\mathbf{w}=\mathbf{w}_t}.$$

The DQN can be updated by performing a gradient descent: $\mathbf{w}_{k+1} \longleftarrow \mathbf{w}_k - \alpha \cdot \mathbf{g}_t$ where α is the learning rate.

3 Policy-Based Deep Reinforcement Learning

The policy function $\pi(a|s)$ can be used to control the agent: observing the state $S_t = s_t$, the agent randomly samples an action:

$$a_t \;\sim\; \pi(\cdot | s_t).$$

The policy function can be approximated by the neural network $\pi(a|s; \boldsymbol{\theta})$ where $\boldsymbol{\theta}$ captures the model parameters. The neural network is called **policy network**.

There are different designs of network architecture. Here, we also consider the game Super Mario, in which the the action space is discrete: $\mathcal{A} = \{$"left", "right", "up"$\}$. The policy network takes observed state s (which can be a screenshot) as input. The architecture can be

$$\texttt{State} \Rightarrow \texttt{Conv} \Rightarrow \texttt{Flatten} \Rightarrow \texttt{Dense} \Rightarrow \texttt{Softmax} \Rightarrow \texttt{Probabilities}.$$

In the Super Mario example, DQN outputs a 3-dimensional vector, e.g., $\mathbf{p} = [0.2, 0.1, 0.7]$, whose entries corresponds to the three actions. Then the action will be randomly sampled:

$$\mathbb{P}\big(A = \text{“left”}\big) = 0.2, \qquad \mathbb{P}\big(A = \text{“right”}\big) = 0.1, \qquad \mathbb{P}\big(A = \text{“up”}\big) = 0.7.$$

All of the three actions may be selected. If the random sampling is independently repeated 1000 times, then around 200 observations of A are "left", around 100 are "right", and around 700 are "up".

The policy network can be learned using **policy gradient** algorithms. If the actions are discrete, then the state-value function can be written as:

$$V_\pi(s) \;=\; \sum_{a\in\mathcal{A}} \pi(a|s)\cdot Q_\pi(s,a). \tag{3.1}$$

Policy-based learning uses the policy network $\pi(a|s;\boldsymbol{\theta})$ to approximate the policy function $\pi(a|s)$. With the approximation of policy function, $V_\pi(s)$ is approximated by

$$V(s;\boldsymbol{\theta}) \;=\; \sum_{a\in\mathcal{A}} \pi(a|s;\boldsymbol{\theta})\cdot Q_\pi(s,a).$$

Policy gradient is the derivative of $V(s;\boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$ [3]:

$$\begin{aligned}
\frac{\partial\, V(s;\boldsymbol{\theta})}{\partial\,\boldsymbol{\theta}} \;&=\; \frac{\partial \sum_{a\in\mathcal{A}} \pi(a|s;\boldsymbol{\theta})\cdot Q_\pi(s,a)}{\partial\,\boldsymbol{\theta}} \\
&=\; \sum_{a\in\mathcal{A}} \frac{\partial\, \pi(a|s;\boldsymbol{\theta})\cdot Q_\pi(s,a)}{\partial\,\boldsymbol{\theta}} \\
&=\; \sum_{a\in\mathcal{A}} Q_\pi(s,a)\cdot \frac{\partial\, \pi(a|s;\boldsymbol{\theta})}{\partial\,\boldsymbol{\theta}} \\
&=\; \sum_{a\in\mathcal{A}} Q_\pi(s,a)\cdot \pi(a|s;\boldsymbol{\theta})\cdot \frac{\partial\, \log \pi(a|s;\boldsymbol{\theta})}{\partial\,\boldsymbol{\theta}}.
\end{aligned}$$

Here, the third identity follows from that $Q_\pi(s,a)$ does not depend on $\boldsymbol{\theta}$;[1] the last identity follows from that $\frac{\partial \log f(x)}{\partial x} = \frac{1}{f(x)}\cdot\frac{\partial f(x)}{\partial x}$. The above equation can be equivalently written as

$$\frac{\partial\, V(s;\boldsymbol{\theta})}{\partial\,\boldsymbol{\theta}} \;=\; \mathbb{E}_{A\sim\pi(\cdot|s,\theta)}\left[Q_\pi(s,a)\cdot \frac{\partial\, \log \pi(A|s;\boldsymbol{\theta})}{\partial\,\boldsymbol{\theta}} \right]. \tag{3.2}$$

Recall that the approximate state-value function $V(s;\boldsymbol{\theta})$ indicates how good the situation s is if policy $\pi(a|s;\boldsymbol{\theta})$ is used. We thereby have the motivation to update $\boldsymbol{\theta}$ so that $V(s;\boldsymbol{\theta})$ will increase (which means the situation is better.) Thus, the policy network can be updated by policy gradient ascent:

$$\boldsymbol{\theta}_{t+1} \;\longleftarrow\; \boldsymbol{\theta}_t + \beta\cdot \left.\frac{\partial\, V(s;\boldsymbol{\theta})}{\partial\,\boldsymbol{\theta}}\right|_{\theta=\theta_t},$$

where β is the learning rate.

Remark 3.1. *The derivation of policy gradient written in the above is not rigorous! It is a simplified version to make the policy gradient easy to understand. To be rigorous, we must take into account that Q_π depends on the policy π and is thereby a function of $\boldsymbol{\theta}$. However, even is Q_π's dependence on $\boldsymbol{\theta}$ is taken into account, the resulting policy gradient is the same to* (3.2).

To this end, we defined the policy network and derived the policy gradient in (3.2). However, there are two unsolved problems. First, the expectation in (3.2) maybe intractable; this is typically

[1] This assumption is too strong. Since Q_π depends on the policy function π, Q_π can depend on $\boldsymbol{\theta}$. Here, the assumption is used to simplify the derivation.

the case when the action space $\mathcal{A}$ is continuous, e.g., $\mathcal{A} = [0, 1]$. Second, the action-value $Q_\pi(s, a)$ is unknown. We answer the two questions one by one.

What if the expectation in (3.2) is intractable? If the action space $\mathcal{A}$ is continuous, then the expectation (which is an integration) is typically intractable. Given state $S_t = s_t$, if the action $A_t = a_t$ is randomly sampled according to the PDF $\pi(\cdot|s_t; \boldsymbol{\theta})$, then

$$\tilde{\mathbf{g}}_t \;=\; Q_\pi(s_t, a_t) \cdot \frac{\partial \, \log \pi(a_t|s_t; \boldsymbol{\theta})}{\partial \, \boldsymbol{\theta}}$$

is an unbiased estimate of $\frac{\partial \, V(s_t;\theta)}{\partial \, \theta}$. We can think of $\mathbf{g}_\theta(\boldsymbol{\theta})$ as a stochastic gradient and update $\boldsymbol{\theta}$ using stochastic gradient ascent.

How do we know the action-value $Q_\pi(s, a)$? There can be two solutions: first, use the observed return r_t instead of $Q_\pi(s, a)$; second, approximate $Q_\pi(s, a)$ using a neural network. The two solutions are described in the following:

- Play a game to the end, obtain all the rewards $r_1, r_2, \cdots, r_T$, and compute the returns $u_1, u_2, \cdots, u_T$ using the equation $u_t = \sum_{i=t}^{T} \gamma^{i-t} \cdot r_i$. Since $Q_\pi(s_t, a_t) = \mathbb{E}[U_t | s_t, a_t, \pi]$, we can use u_t to replace $Q_\pi(s_t, a_t)$. In this way, the policy gradient (3.2) at time step t becomes

$$\frac{\partial \, V(s_t; \boldsymbol{\theta})}{\partial \, \boldsymbol{\theta}} \;=\; \mathbb{E}_{A \sim \pi(\cdot|s_t,\theta)} \left[u_t \cdot \frac{\partial \, \log \pi(A|s_t; \boldsymbol{\theta})}{\partial \, \boldsymbol{\theta}} \right].$$

  AlphaGo [2] uses this approach.

- Use a value network to approximate $Q_\pi(s, a)$. The value network provides supervision to the policy network. The value network can be learned by temporal difference (TD). This leads to the actor-critic method which is elaborated on in Section 4.1.

4 Actor-Critic Methods

Section 4.1 follows Section 3 and derive the standard (random) actor-critic method. This approach is suitable for problems with discrete action space.[2] Section 4.2 studies deterministic actor-critic method and learn it using deterministic policy gradient algorithm. This method is very useful when the actions are continuous.[3]

4.1 Random Actor-Critic Method

The actor-critic method has two neural networks. Policy network $\pi(a|s; \boldsymbol{\theta})$, which is called actor, approximates the policy function $\pi(a|s)$. Value network $q(s, a; \mathbf{w})$, which is called critic, approximates the action-value function $Q_\pi(a, s)$. In this way, the state-value function $V_\pi(s)$ is approximated by

$$V(s; \mathbf{w}, \boldsymbol{\theta}) \;=\; \mathbb{E}_{A \sim \pi(\cdot|s;\theta)} \big[q(s, A; \mathbf{w}) \big] \;=\; \sum_{a \in \mathcal{A}} \pi(a|s; \boldsymbol{\theta}) \cdot q(s, a; \mathbf{w}).$$

[2]For example, Super Mario's action space {"left", "right", "up"} is a discrete set.

[3]For example, a self-driving car's action can be two-dimensional vectors. The first dimension is the steering angle, and the second dimension is acceleration/deceleration. The action space is obviously continuous.

It is not hard to show the policy gradient is

$$\frac{\partial V(s;\mathbf{w},\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \mathbb{E}_{A\sim\pi(\cdot|s,\theta)}\left[q(s,A;\mathbf{w})\cdot\frac{\partial \log \pi(A|s;\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right].$$

The policy network will be updated using (stochastic) policy gradient ascent. The value network can be updated using temporal different (TD) learning. The following summarizes one iteration of the algorithm.

1. Observe state s_t, and then randomly sample action $a_t \sim \pi(\cdot|s_t;\boldsymbol{\theta}_t)$.
2. Agent performs action a_t and observe reward r_t and new state s_{t+1}.
3. Randomly sample action $a_{t+1} \sim \pi(\cdot|s_{t+1};\boldsymbol{\theta}_t)$. (Agent does not perform action a_{t+1}.)
4. Evaluate the value network and get $q_t = q(s_t,a_t;\mathbf{w}_t)$ and $q_{t+1} = q(s_{t+1},a_{t+1};\mathbf{w}_t)$.
5. Compute the TD error: $\delta_t = q_t - (r_t + \gamma\cdot q_{t+1})$.
6. Update the value network: $\mathbf{w}_{t+1} \longleftarrow \mathbf{w}_t - \alpha\cdot\delta_t\cdot\frac{\partial Q(s_t,a_t;\mathbf{w})}{\partial \mathbf{w}}\big|_{\mathbf{w}=\mathbf{w}_t}$.
7. Update the policy network: $\boldsymbol{\theta}_{t+1} \longleftarrow \boldsymbol{\theta}_t + \beta\cdot q_t\cdot\frac{\partial \log \pi(a_t|s_t;\theta)}{\partial \theta}\big|_{\theta=\theta_t}$.[4]

When learning the policy network (actor), the supervision is not from the rewards; instead, the supervision is from the critic's output $q_t = q(s_t,a_t;\mathbf{w}_t)$. The actor uses the critic's judgments to improve her performance. When training the critic, the supervision is from the rewards. The critic uses ground truth from the environment to make his judgment more accurate.

4.2 Deterministic Actor-Critic Method

Throughout, the policy function is defined as the probability density function $\pi(a|s)$, and the action is randomly sampled according to π. **Deterministic policy** is a function that maps state to actions: $\pi : \mathcal{S} \mapsto \mathcal{A}$, where $\mathcal{S}$ is the state space and $\mathcal{A}$ is the action space. Given the state $\mathbf{s}$, the policy function deterministically outputs action $a = \pi(s)$. Deterministic policy is very useful when the actions are continuous.

Deterministic actor-critic method [1] has two networks: policy network $\pi(s;\boldsymbol{\theta})$ and value network $q(s,a;\mathbf{w})$; see Figure 2. The agent is controlled by the policy network which deterministically maps state s to action a. The value network is used for providing the policy network with supervision. The two networks can be trained in the following way.

The value network can be trained by temporal different (TD) learning. Let $q_t = q(s_t,a_t;\mathbf{w}_t)$ be the prediction and $y_t = r_t + \gamma\cdot q(s_{t+1},a_{t+1};\mathbf{w}_t)$ be the TD target. The TD error is $\delta_t = q_t - y_t$. The model parameters $\mathbf{w}$ can be updated by $\mathbf{w}_{t+1} \longleftarrow \mathbf{w}_t - \alpha\cdot\delta_t\cdot\frac{\partial q(s_t,a_t;\mathbf{w})}{\partial \mathbf{w}}\big|_{\mathbf{w}=\mathbf{w}_t}$.

[4] In most papers and books, the update of the policy network is $\boldsymbol{\theta}_{t+1} \longleftarrow \boldsymbol{\theta}_t + \beta\cdot\delta_t\cdot\frac{\partial \log \pi(a_t|s_t;\theta)}{\partial \theta}\big|_{\theta=\theta_t}$. The difference is that q_t is replaced by δ_t. Both approaches are correct. The use of δ_t is the result of using a baseline which can reduce variance.

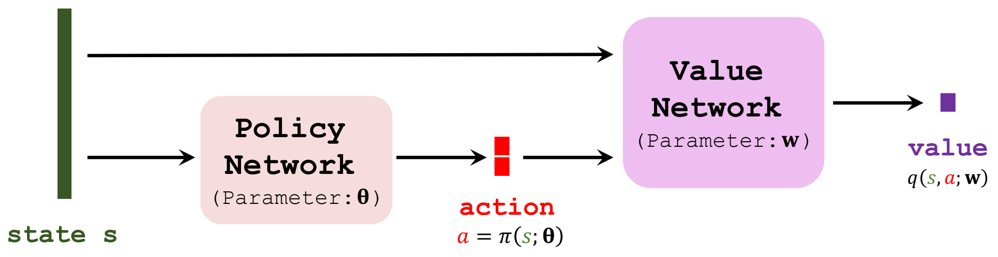
Value Network
(Parameter: $\mathbf{w}$)
Policy Network
(Parameter: $\boldsymbol{\theta}$)
value
$q(s, a; \mathbf{w})$
action
$a = \pi(s; \boldsymbol{\theta})$
state s

Figure 2: Deterministic actor-critic method. The deterministic policy network maps state $s \in \mathcal{S}$ to action $a \in \mathcal{A} \subset \mathbb{R}^2$. The two dimensions of a are, for example, the steering angle and acceleration of a self-driving car. The value network maps the pair (s, a) to a scalar.

Train the policy network by deterministic policy gradient (DPG) which is totally different from the policy gradient we studied previously. Note that the value network $q(s_t, a_t; \mathbf{w})$ evaluates how good it is for the agent to perform action a_t at state s_t. The policy network has motivation to update its parameters $\boldsymbol{\theta}$ so that the action $a_t = \pi(s_t; \boldsymbol{\theta})$ will get a higher evaluation. Intuitively speaking, the policy network (actor) wants to change herself so that the evaluation given by the value network (critic) will increase. The derivative of the objective, i.e., $q(s_t, a_t; \mathbf{w})$, w.r.t. the policy network's parameters $\boldsymbol{\theta}$ is

$$\mathbf{g}(\boldsymbol{\theta}) \;=\; \frac{\partial\, q(s_t, \pi(s_t; \theta); \mathbf{w})}{\partial\, \theta} \;=\; \frac{\partial\, \pi(s_t; \theta)}{\partial\, \theta} \cdot \frac{\partial\, q(s_t, a; \mathbf{w})}{\partial\, a}\bigg|_{a=\pi(s_t;\theta)},$$

where the second identity follows from the chain rule. The policy network is updated by performing gradient ascent: $\boldsymbol{\theta}_{t+1} \longleftarrow \boldsymbol{\theta}_t + \beta \cdot \mathbf{g}(\boldsymbol{\theta}_t)$.

References

[1] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning (ICML)*, pages 387–395, 2014.

[2] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, and Marc Lanctot. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587): 484, 2016.

[3] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1057–1063, 2000.

[4] Richard S Sutton, Csaba Szepesvári, and Hamid Reza Maei. A convergent o (n) algorithm for off-policy temporal-difference learning with linear function approximation. *Advances in Neural Information Processing Systems (NIPS)*, 21(21):1609–1616, 2008.

[5] Richard S Sutton, Hamid Reza Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, and Eric Wiewiora. Fast gradient-descent methods for temporal-difference learning

with linear function approximation. In *International Conference on Machine Learning (ICML)*, pages 993–1000, 2009.