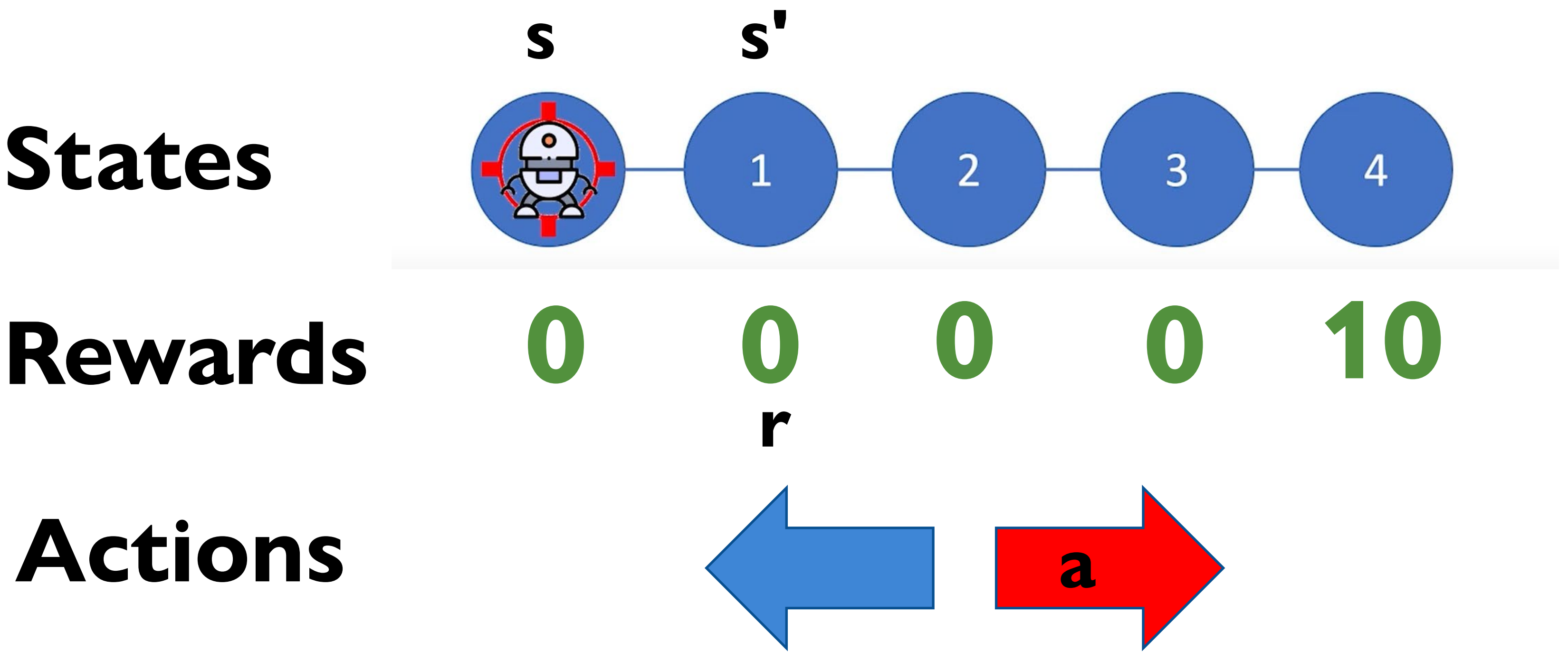
DEEP Q-LEARNING

TAKE A STEP

- In simulator you are in state s
- Take action a
- Earn reward r
- End up in state s’

$$(s, a, r, s')$$



$$(0, \text{right}, 0, 1)$$

ONE STEP LOOK-AHEAD Q-VALUE

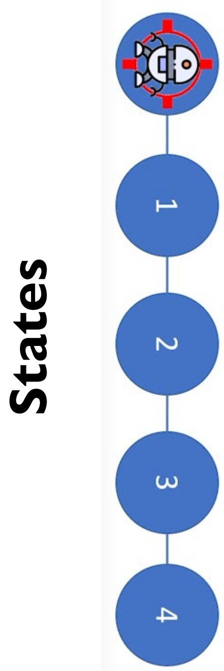
- We can get Q value by looking one step ahead

$$(s, a, r, s')$$

$$\Downarrow$$

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

Actions



←	→
5.9	6.7
6.7	7.3
7.3	8.1
8.1	9
10	10

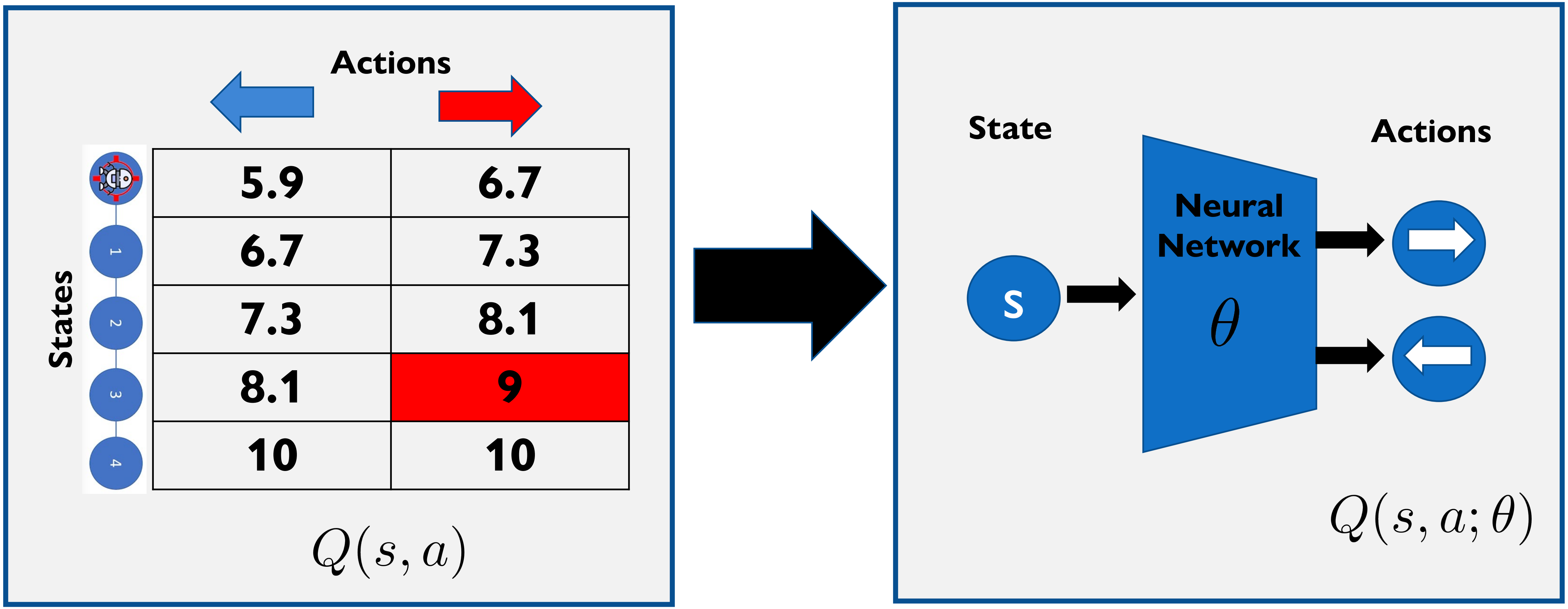
Q-LEARNING

- Q-learning has us update the Q-value as the weighted average of the current value and the one-step look ahead value

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a'))$$

DEEP Q-LEARNING

- In deep Q-learning we replace the Q-table with a neural network



5.9	6.7
6.7	7.3
7.3	8.1
8.1	9
10	10

UPDATING NEURAL NETWORK

- Neural network has a bunch of parameters θ
- We update the parameters so that the one step Q-value equals the current Q-value

UPDATING NEURAL NETWORK

- Neural network has a bunch of parameters θ
- We update the parameters so that the one step Q-value equals the current Q-value
- Define error function:

$$E(\theta) = (r + \gamma \max_{a'} Q(s', a'; \theta) - Q(s, a; \theta))^2$$

UPDATING NEURAL NETWORK

- Neural network has a bunch of parameters θ
- We update the parameters so that the one step Q-value equals the current Q-value
- Define error function:

$$E(\theta) = (r + \gamma \max_{a'} Q(s', a'; \theta) - Q(s, a; \theta))^2$$

- We update the parameters so that the error decreases

$$\theta = \theta - \alpha \frac{dE(\theta)}{d\theta}$$

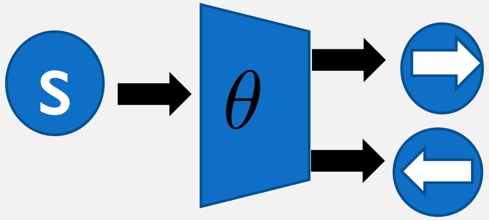
ISSUES WITH DEEP Q-LEARNING

- In Q-learning, we update one spot in the Q-table
- In deep Q-learning, we update the entire “table” because we update all the neural network parameters
  - This causes instabilities in the learning process

Q-Learning

5.9	6.7
6.7	7.3
7.3	8.1
8.1	9
10	10

Deep Q-Learning



5.9	6.7
6.7	7.3
7.3	8.1
8.1	9
10	10

FIXING ISSUES: TARGET NETWORK

FIXING ISSUES: TARGET NETWORK

- One fix is we don't update the target network in the error function every step

$$E(\theta) = (r + \gamma \max_{a'} \underbrace{Q_{target}}_{\textbf{Target network}}(s', a'; \theta)) - \underbrace{Q}_{\textbf{Local network}}(s, a; \theta))^2$$

FIXING ISSUES: TARGET NETWORK

- One fix is we don't update the target network in the error function every step

$$E(\theta) = (r + \gamma \max_{a'} \underbrace{Q_{target}}_{\textbf{Target network}}(s', a'; \theta)) - \underbrace{Q}_{\textbf{Local network}}(s, a; \theta))^2$$

- Every few episodes we update the target network

$$Q_{target} = Q$$

FIXING ISSUES: REPLAY BUFFER

- Another fix is we don't update each step in simulator
- We save steps (s,a,r,s') in an **experience replay buffer**

Experience Replay Buffer

State	Action	Reward	New State
s_1	a_1	r_1	s'_1
s_2	a_2	r_2	s'_2
s_3	a_3	r_3	s'_3
s_4	a_4	r_4	s'_4
s_5	a_5	r_5	s'_5

FIXING ISSUES: REPLAY BUFFER

- To update Q network we sample a batch of steps from the buffer

State	Action	Reward	New State
s_1	a_1	r_1	s'_1
s_2	a_2	r_2	s'_2
s_3	a_3	r_3	s'_3
s_4	a_4	r_4	s'_4
s_5	a_5	r_5	s'_5

FIXING ISSUES: REPLAY BUFFER

- To update Q network we sample a batch of steps from the buffer

State	Action	Reward	New State
s_1	a_1	r_1	s'_1
s_2	a_2	r_2	s'_2
s_3	a_3	r_3	s'_3
s_4	a_4	r_4	s'_4
s_5	a_5	r_5	s'_5

Sample

$$(s_2, a_2, r_2, s_2')$$
$$(s_4, a_4, r_4, s_4')$$
$$(s_5, a_5, r_5, s_5')$$

FIXING ISSUES: REPLAY BUFFER

- To update Q network we sample a batch of steps from the buffer
- Then we apply the update step using that batch of samples

Update

$$(s_2, a_2, r_2, s'_2)$$
$$(s_4, a_4, r_4, s'_4)$$
$$(s_5, a_5, r_5, s'_5)$$

$$\begin{aligned} E(\theta) = &(r_2 + \gamma \max_{a'} Q(s'_2, a'; \theta)) - Q(s_2, a_2; \theta))^2 \\ &+ (r_4 + \gamma \max_{a'} Q(s'_4, a'; \theta)) - Q(s_4, a_4; \theta))^2 \\ &+ (r_5 + \gamma \max_{a'} Q(s'_5, a'; \theta)) - Q(s_5, a_5; \theta))^2 \end{aligned}$$

$$\theta = \theta - \alpha \frac{dE(\theta)}{d\theta}$$

DEEP Q-LEARNING PAPER

- Deep Q-learning was invented at Google in 2015 to play Atari games

Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih **Koray Kavukcuoglu** **David Silver** **Alex Graves** **Ioannis Antonoglou**

Daan Wierstra **Martin Riedmiller**

DeepMind Technologies

{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

Abstract

We present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. We apply our method to seven Atari 2600 games from the Arcade Learning Environment, with no adjustment of the architecture or learning algorithm. We find that it outperforms all previous approaches on six of the games and surpasses a human expert on three of them.

DEEP Q-LEARNING ATARI SCORES