Introduction to Reinforcement Learning
Week 5

Dr. Apurva Narayan
http://anarayan.com

Q-Learning

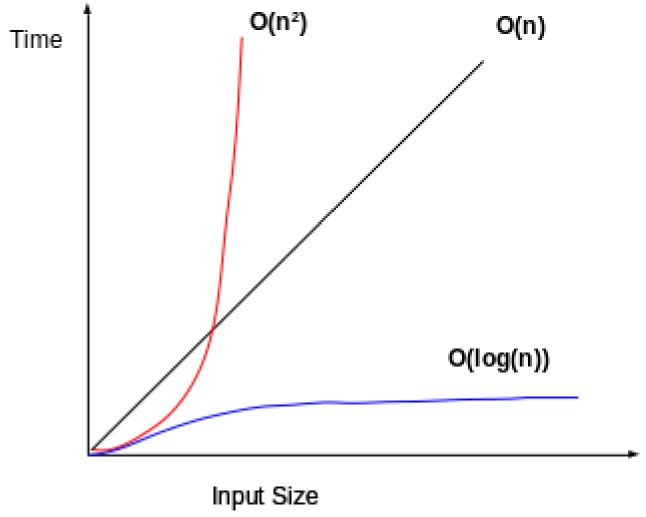
$$Q(s,a) = r(s,a) + \gamma \max_a Q(s',a)$$

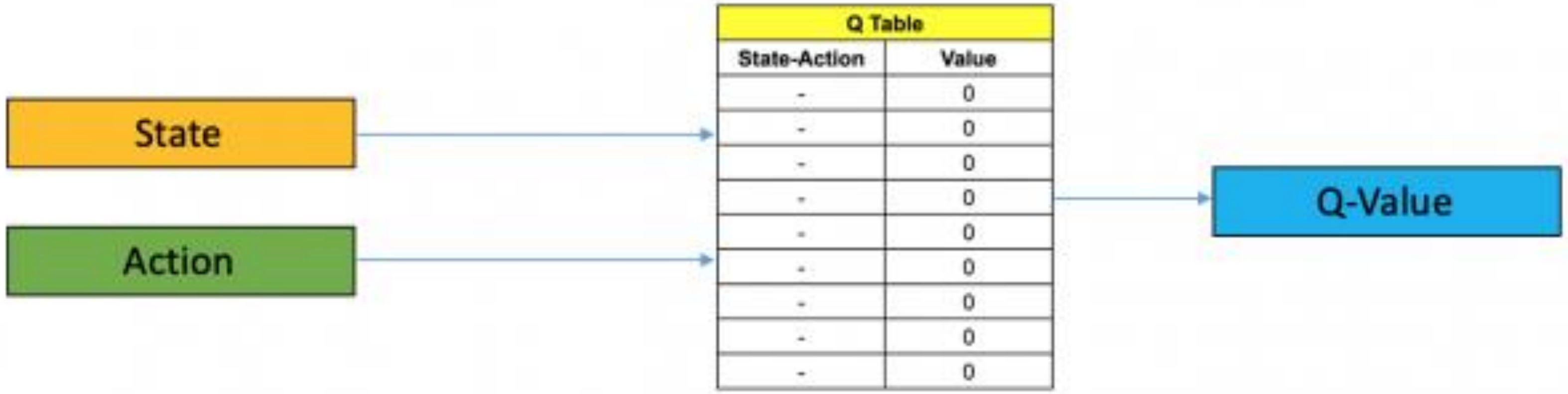
$$Q(s,a) \rightarrow \gamma \mathrm{Q}(\mathrm{s}',\mathrm{a}) + \gamma^2 Q(s'',a) \ldots \ldots \ldots \gamma^n Q(s''^{\ldots n},a)$$

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)\Big]$$

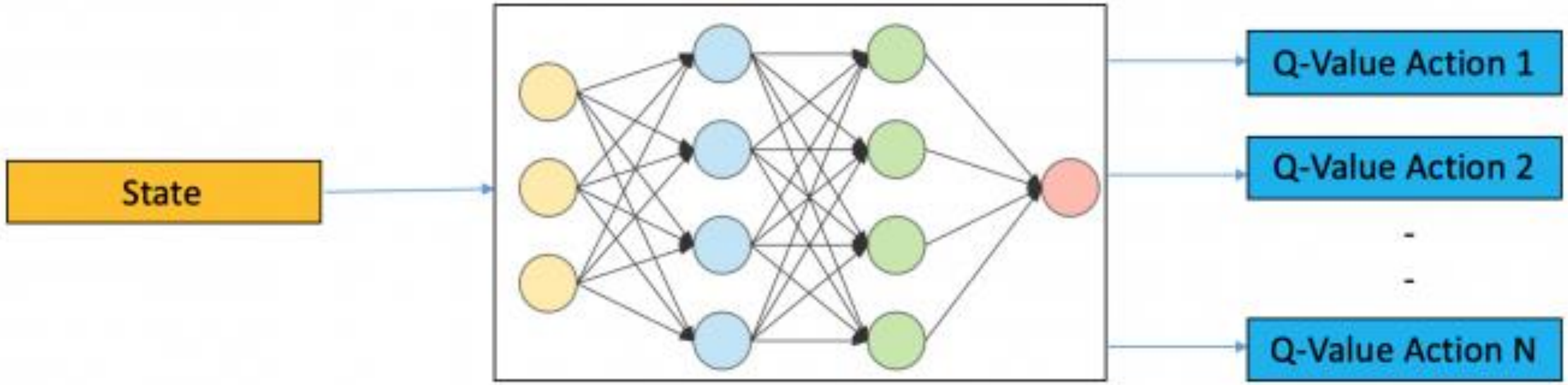
Why “Deep Q-Learning” ?

- The amount of memory required to save and update that table would increase as the number of states increases
- The amount of time required to explore each state to create the required Q-table would be unrealistic



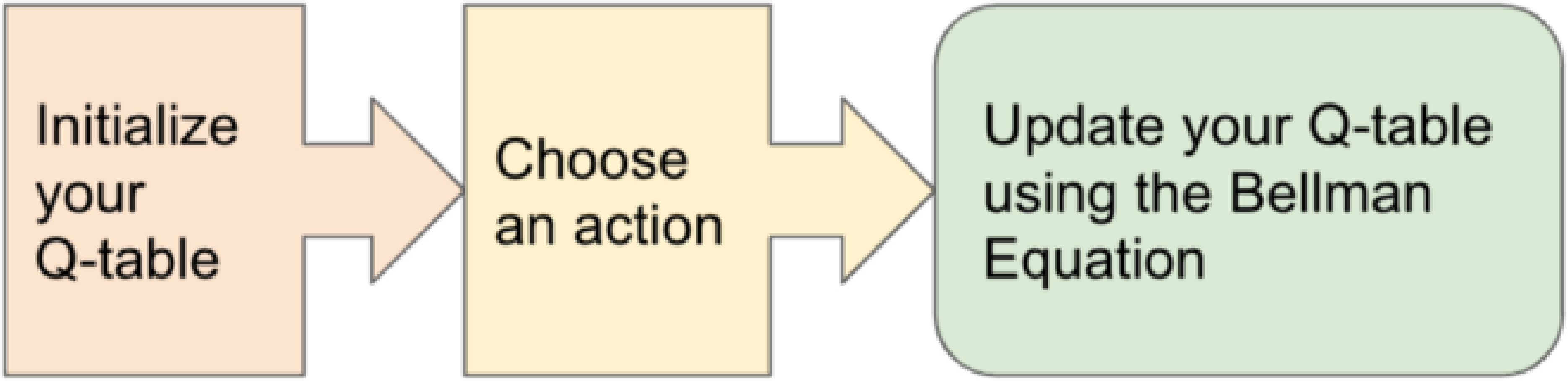


Q Learning



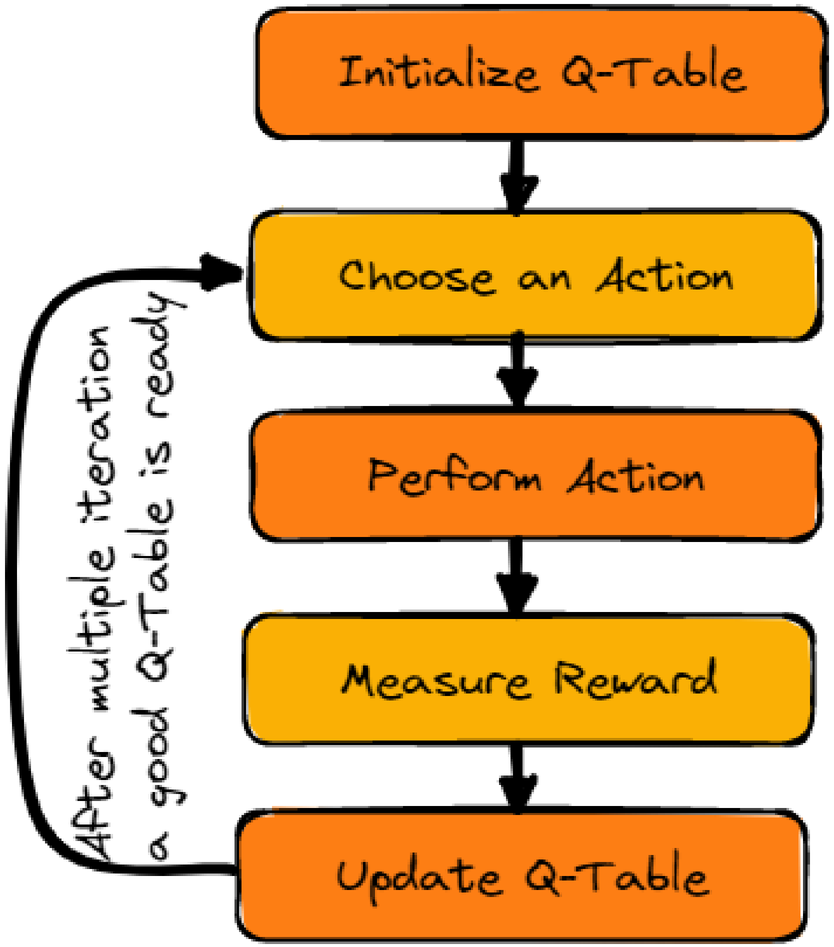
Deep Q Learning

Vanilla Q-Learning



1. Initialize your Q-table
2. Choose an action using the Epsilon-Greedy Exploration Strategy
3. Update the Q-table using the Bellman Equation

Q-learning algorithm



$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

New Q-value estimation

Former Q-value estimation

Learning Rate

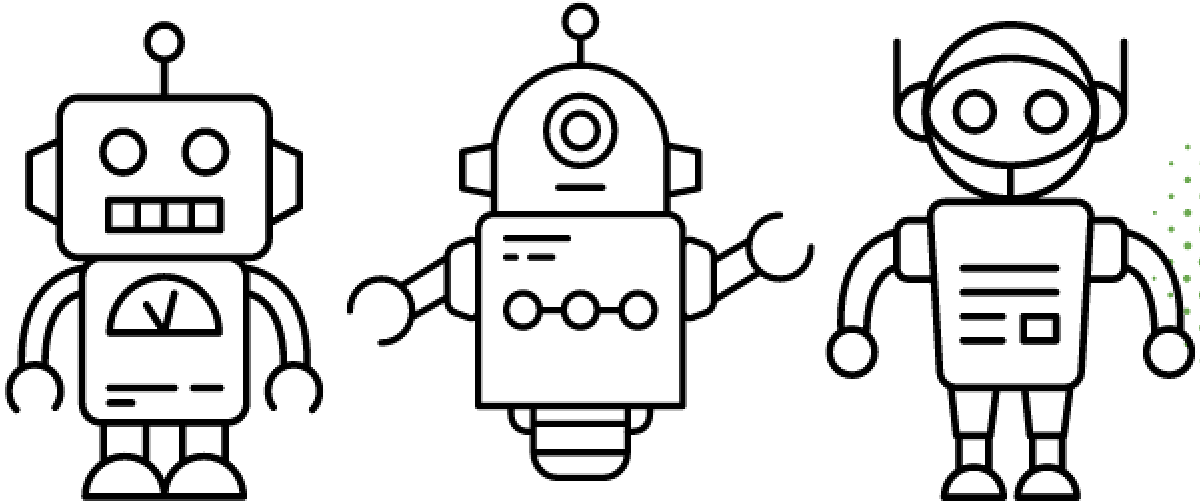
Immediate Reward

Discounted Estimate optimal Q-value of next state

Former Q-value estimation

TD Target

TD Error



Steps to use Deep Q-Learning

1. All the past experience is stored by the user in memory
2. The next action is determined by the maximum output of the Q-network
3. The loss function here is mean squared error of the predicted Q-value and the target Q-value – Q*.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[\boxed{R_{t+1} + \gamma \max_a Q(S_{t+1}, a)} - Q(S_t, A_t)\right]$$

- Value Function Approximation
  - Linear approximation
  - Neural network approximation
    - Deep Q-network

Q-function Approximation

- Let $s = (x_1, x_2, \dots, x_n)^T$

- Linear

  $Q(s, a) \approx \sum_i w_{ai} x_i$

- Non-linear (e.g., neural network)

  $Q(s, a) \approx g(\boldsymbol{x}; \boldsymbol{w})$

Gradient Q-learning

- Minimize squared error between Q-value estimate and target
  - Q-value estimate: $Q_{\boldsymbol{w}}(s,a)$
  - Target: $r + \gamma \max_{a'} Q_{\overline{\boldsymbol{w}}}(s',a')$

$\overline{\boldsymbol{w}}$ fixed

- Squared error:

$$Err(\boldsymbol{w}) = \frac{1}{2}\left[Q_{\boldsymbol{w}}(s,a) - r - \gamma \max_{a'} Q_{\overline{\boldsymbol{w}}}(s',a')\right]^2$$

- Gradient

$$\frac{\partial Err}{\partial \boldsymbol{w}} = \left[Q_{\boldsymbol{w}}(s,a) - r - \gamma \max_{a'} Q_{\overline{\boldsymbol{w}}}(s',a')\right] \frac{\partial Q_{\boldsymbol{w}}(s,a)}{\partial \boldsymbol{w}}$$

Gradient Q-learning

Initialize weights $\boldsymbol{w}$ uniformly at random in $[-1,1]$

Observe current state s

Loop

- Select action a and execute it
- Receive immediate reward r
- Observe new state s'
- Gradient: $\frac{\partial Err}{\partial \boldsymbol{w}} = \left[Q_{\boldsymbol{w}}(s,a) - r - \gamma \max_{a'} Q_{\boldsymbol{w}}(s',a')\right] \frac{\partial Q_{\boldsymbol{w}}(s,a)}{\partial \boldsymbol{w}}$
- Update weights: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \frac{\partial Err}{\partial \boldsymbol{w}}$
- Update state: $s \leftarrow s'$

Recap: Convergence of Tabular Q-learning

- Tabular Q-Learning converges to optimal Q-function under the following conditions:

  $\sum_{n=0}^{\infty} \alpha_n = \infty$ and $\sum_{n=0}^{\infty} \alpha_n^2 < \infty$

- Let $\alpha_n(s,a) = 1/n(s,a)$
  - Where $n(s,a)$ is # of times that (s,a) is visited

- Q-learning

$$Q(s,a) \leftarrow Q(s,a) + \alpha_n(s,a)[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

Convergence of Linear Gradient Q-Learning

- Linear Q-Learning converges under the same conditions:

$$\sum_{n=0}^{\infty} \alpha_n = \infty \text{ and } \sum_{n=0}^{\infty} \alpha_n^2 < \infty$$

- Let $\alpha_n = 1/n$
- Let $Q_{\boldsymbol{w}}(s,a) = \sum_i w_i x_i$

- Q-learning

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha_n \left[Q_{\boldsymbol{w}}(s,a) - r - \gamma \max_{a'} Q_{\boldsymbol{w}}(s',a') \right] \frac{\partial Q_{\boldsymbol{w}}(s,a)}{\partial \boldsymbol{w}}$$

Divergence of Non-linear Gradient Q-learning

- Even when the following conditions hold

  $$\sum_{n=0}^{\infty} \alpha_n = \infty \text{ and } \sum_{n=0}^{\infty} \alpha_n^2 < \infty$$

  non-linear Q-learning may diverge

- Intuition:
  - Adjusting $\boldsymbol{w}$ to increase Q at (s, a) might introduce errors at nearby state-action pairs.

Mitigating divergence

- Two tricks are often used in practice:

1. Experience replay
2. Use two networks:
   - Q-network
   - Target network

Target Network

- Idea: Use a separate target network that is updated only periodically

repeat for each (s, a, s', r) in mini-batch:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha_t \left[\underbrace{Q_{\boldsymbol{w}}(s,a)}_{\text{update}} - r - \gamma \max_{a'} \underbrace{Q_{\overline{\boldsymbol{w}}}(s',a')}_{\text{target}} \right] \frac{\partial Q_{\boldsymbol{w}}(s,a)}{\partial \boldsymbol{w}}$$

$$\overline{\boldsymbol{w}} \leftarrow \boldsymbol{w}$$

- Advantage: mitigate divergence

Target Network

- Similar to value iteration:

repeat for all s

$$\underbrace{V(s)}_{\text{update}} \leftarrow \max_a R(s) + \gamma \sum_{s'} \Pr(s'|s,a)\, \underbrace{\bar{V}(s')}_{\text{target}} \quad \forall s$$

$\bar{V} \leftarrow V$

repeat for each (s, a, s', r) in mini-batch:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha_t \left[\underbrace{Q_{\boldsymbol{w}}(s,a)}_{\text{update}} - r - \gamma \max_{a'} \underbrace{Q_{\bar{\boldsymbol{w}}}(s',a')}_{\text{target}} \right] \frac{\partial Q_{\boldsymbol{w}}(s,a)}{\partial \boldsymbol{w}}$$

$\bar{\boldsymbol{w}} \leftarrow \boldsymbol{w}$

Deep Q-network

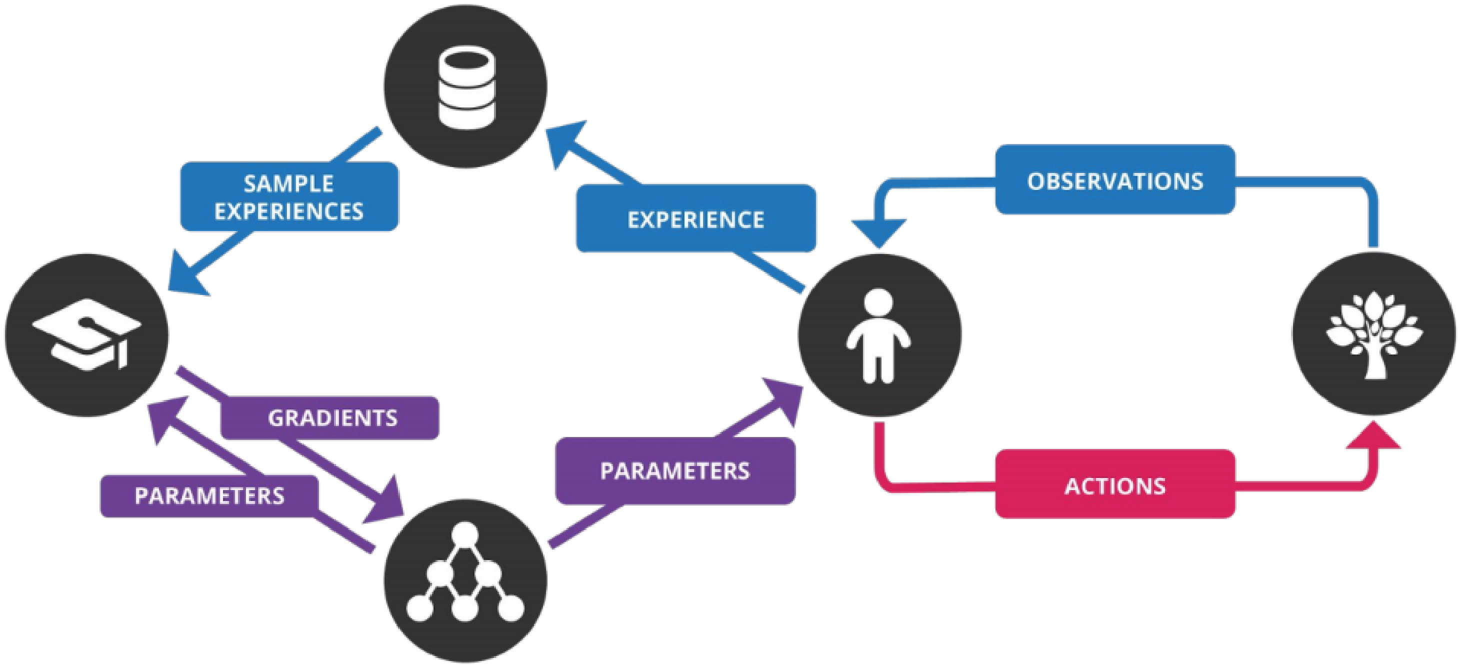
- Google Deep Mind:

- Deep Q-network: Gradient Q-learning with
  - Deep neural networks
  - Experience replay
  - Target network

- Breakthrough: human-level play in many Atari video games

Experience Replay

> To perform experience replay, we store the agent's experiences – $e_t = (s_t, a_t, r_t, s_{t+1})$



Experience Replay

- Idea: store previous experiences (s, a, s', r) into a buffer and sample a mini-batch of previous experiences at each step to learn by Q-learning

- Advantages
  - Break correlations between successive updates (more stable learning)
  - Fewer interactions with environment needed to converge (greater data efficiency)

Steps Involved in a Deep-Q Network

1.Preprocess and feed the game screen (state s) to our DQN, which will return the Q-values of all possible actions in the state

2.Select an action using the epsilon-greedy policy. With the probability epsilon, we select a random action *a* and with probability 1-epsilon, we select an action that has a maximum Q-value, such as a = argmax(Q(s,a,w))

3.Perform this action in a state *s* and move to a new state *s'* to receive a reward. This state s' is the preprocessed image of the next game screen. We store this transition in our replay buffer as <s,a,r,s'>

4.Next, sample some random batches of transitions from the replay buffer and calculate the loss

5.It is known that: which is just the squared difference between target Q and predicted Q

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

6.Perform gradient descent with respect to our actual network parameters in order to minimize this loss

7.After every C iterations, copy our actual network weights to the target network weights

8.Repeat these steps for *M* number of episodes

Deep Q-network

Initialize weights $\boldsymbol{w}$ and $\overline{\boldsymbol{w}}$ at random in $[-1,1]$

Observe current state s

Loop

Select action a and execute it

Receive immediate reward r

Observe new state s'

Add (s, a, s', r) to experience buffer

Sample mini-batch of experiences from buffer

For each experience $(\hat{s}, \hat{a}, \hat{s}', \hat{r})$ in mini-batch

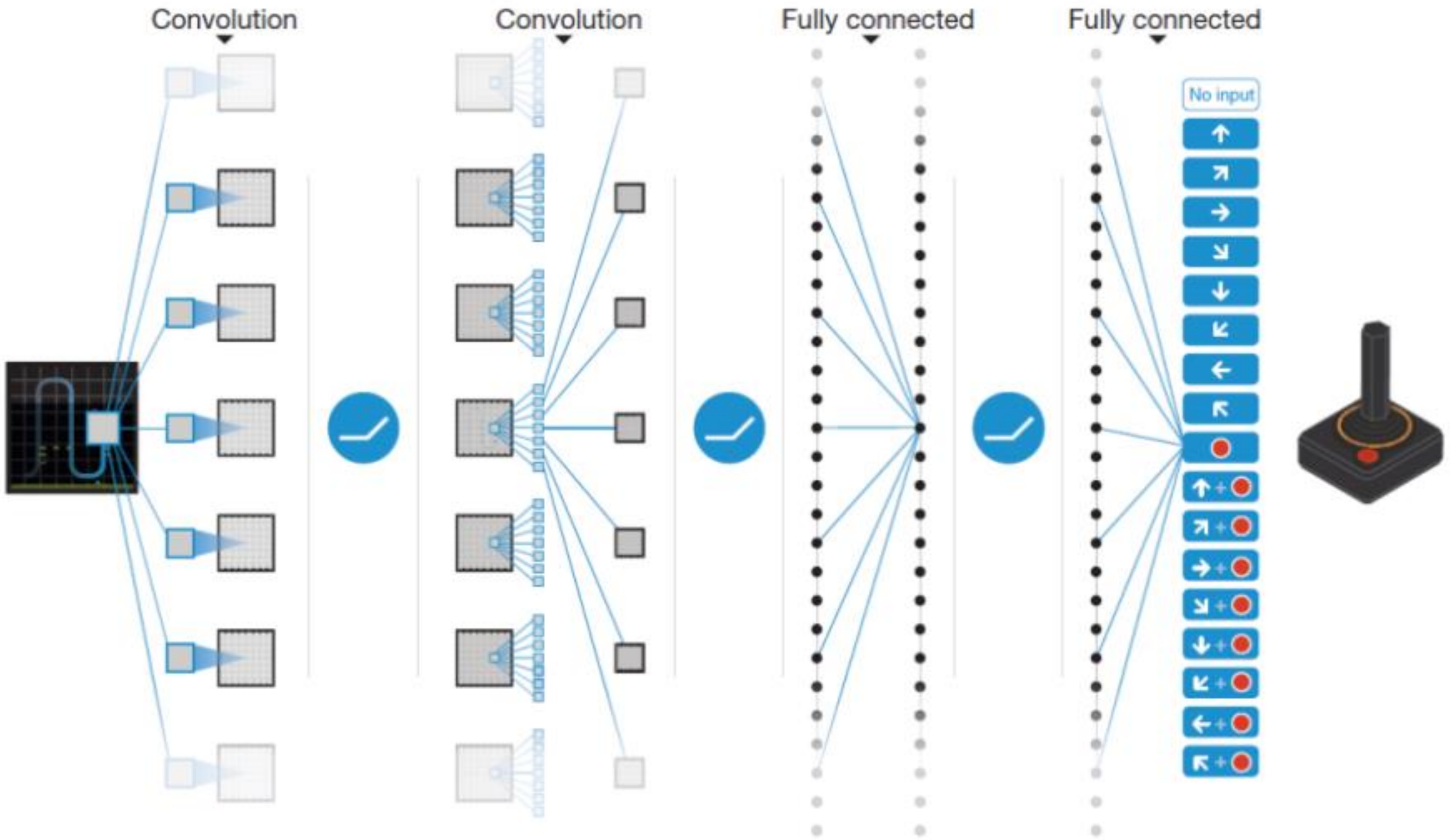
Gradient: $\frac{\partial Err}{\partial \boldsymbol{w}} = \left[Q_{\boldsymbol{w}}(\hat{s}, \hat{a}) - \hat{r} - \gamma \max_{\hat{a}'} \mathrm{Q}_{\overline{\boldsymbol{w}}}(\hat{s}', \hat{a}')\right] \frac{\partial Q_{\boldsymbol{w}}(\hat{s},\hat{a})}{\partial \boldsymbol{w}}$

Update weights: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \frac{\partial Err}{\partial \boldsymbol{w}}$

Update state: $s \leftarrow s'$

Every c steps, update target: $\overline{\boldsymbol{w}} \leftarrow \boldsymbol{w}$

Deep Q-Network for Atari



DQN versus Linear approx.