



CS 760: Machine Learning **Reinforcement Learning III**

Josiah Hanna

University of Wisconsin-Madison

December 5, 2023

Announcements

- Homework 7 due December 12 at 9:30 am.
- Final exam: December 18 from 2:45 - 4:45 pm in the Social Sciences building.
- Course evaluations available until 12/13.
 - Currently at 35% participation. > 50% to receive a negative topic list for final.

Lecture Goals

At the end of today's lecture, you will be able to:

1. Implement fundamental dynamic programming approaches to reinforcement learning.
2. Implement the q-learning algorithm.
3. Explain the key techniques necessary for using neural networks in q-learning.

Quiz

You are a video game company and you are developing a car racing game. You want to develop an AI bot that can beat human drivers. You decide to use reinforcement learning to train this bot. Describe how you might define the task Markov decision process.

Possible

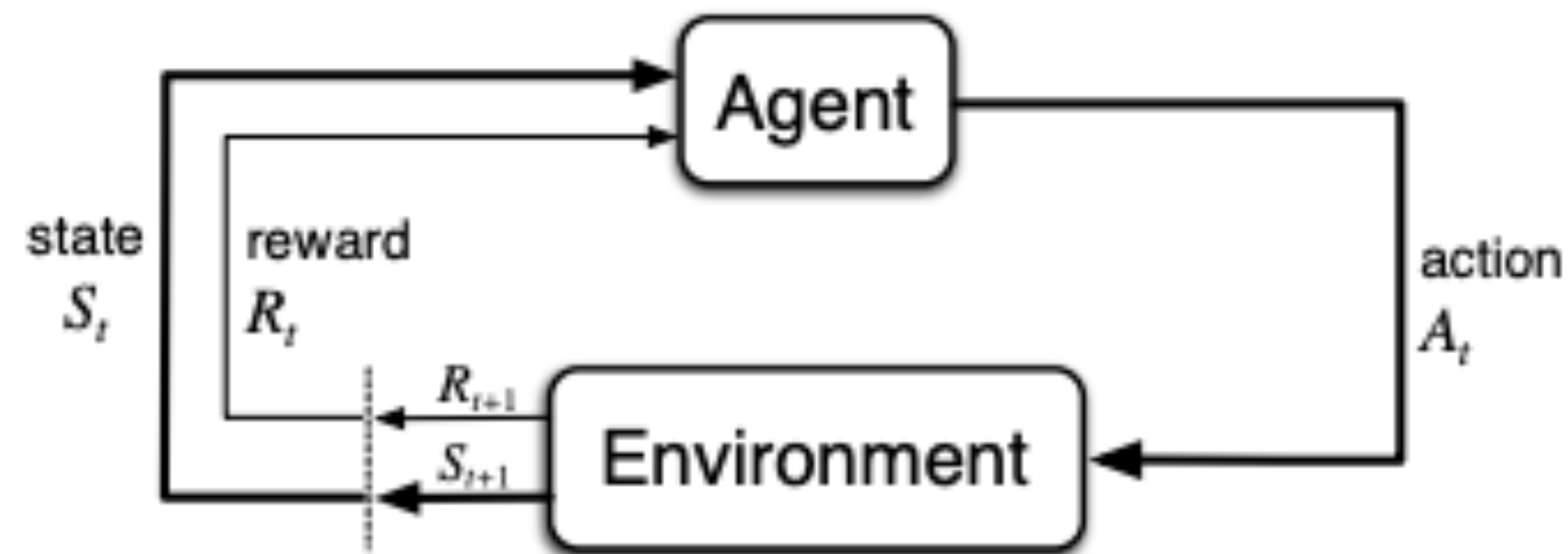
States: Position and velocity for all agents on the track. Key requirement is that state is a sufficient summary of the past for predicting the future.

Actions: driving controls

Reward: +1 for winning, -1 for losing, other rewards for exhibiting desired behaviors.

Data in Reinforcement Learning

Agent learns from the sequence of data seen while acting in task Markov decision process:

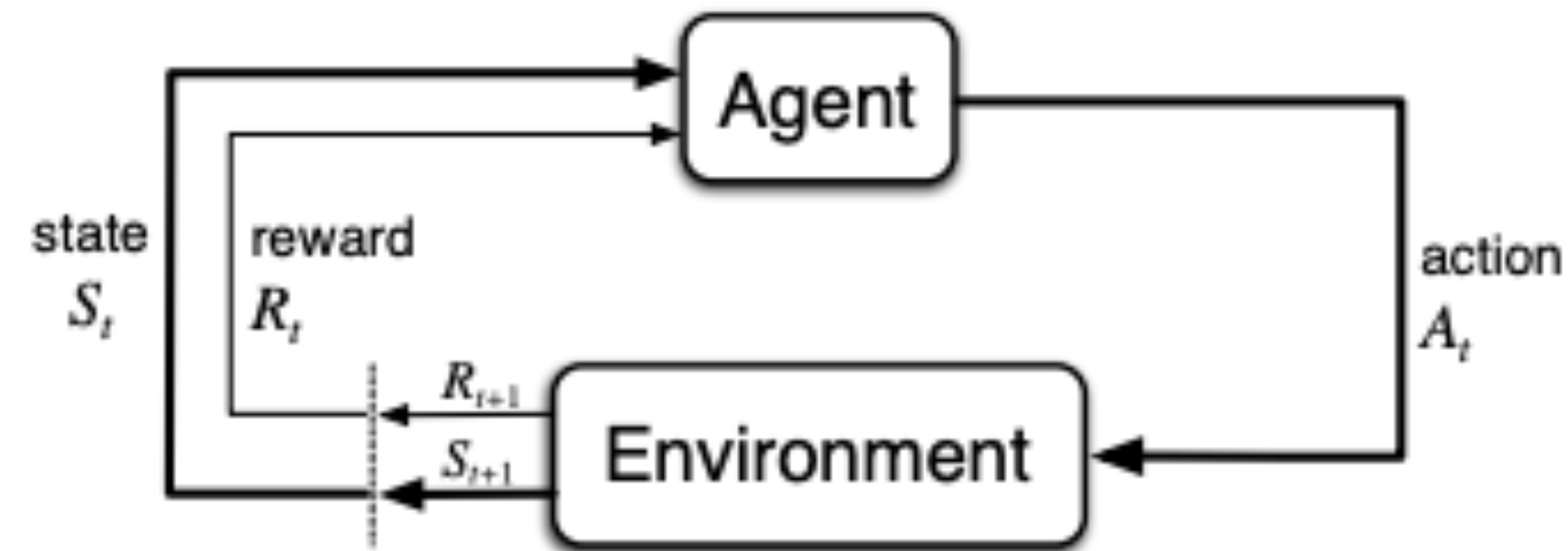


$\dots S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, \dots$

$$S_{t+1}, R_{t+1} \sim p(\cdot | S_t, A_t)$$

$$A_{t+1} \leftarrow \pi(S_{t+1})$$

Reinforcement Learning



Agent's objective is to find policy, π , so as to maximize the **expected cumulative discounted** reward from each state:

For compactness, using p for next states and rewards.

$$v_{\pi}(s) = \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \mid S_0 = s, A_t \leftarrow \pi(S_t), S_{t+1}, R_{t+1} \sim p(\cdot \mid S_t, A_t)\right]$$

For brevity, \mathbf{E}_{π} will be used for $\mathbf{E}[\dots \mid A_t \leftarrow \pi(S_t), S_{t+1}, R_{t+1} \sim p(\cdot \mid S_t, A_t)]$

Bellman Equation

- Bellman equation expresses state-value, $v_{\pi}(s)$, in terms of expected reward and state-values at next time-step.

$$v_{\pi}(s) = \mathbf{E}_{\pi}[R_{t+1} + \gamma v_{\pi}(S_{t+1}) | S_t = s]$$

- Or to be more explicit, using the definition of expectation:

$$v_{\pi}(s) = \sum_a \pi(a | s) \sum_{s'} \sum_r p(s', r | s, a) [r + \gamma v_{\pi}(s')]$$

Note: stochastic policy; gives probability for action a in state s .

Optimality

- Agent's objective: find policy that maximizes $v_{\pi}(s)$ for all s .
- Possibly multiple but always at least one deterministic optimal policy in a finite MDP.
- $\pi^{\star}(s) = \arg \max_a q_{\star}(s, a)$ $q_{\star}(s, a) = \mathbf{E}[R_{t+1} + \gamma v_{\star}(S_{t+1}) \mid S_t = s, A_t = a]$

Value of taking action a and then acting optimally for all future time-steps.

Optimal Value Functions

- Like all policies, the optimal policy has a state-value and action-value function:
 - $v_{\star}(s) = \mathbb{E}[R_{t+1} + \gamma v_{\star}(S_{t+1}) | S_t = s]$
 - $q_{\star}(s, a) = \mathbb{E}[R_{t+1} + \gamma v_{\star}(S_{t+1}) | S_t = s, A_t = a]$
- The optimal policy is greedy with respect to the optimal action-values, i.e.,
$$\pi^{\star}(s) = \arg \max_a q_{\star}(s, a)$$

Bellman Optimality Equation

$$v_{\star}(s) = \mathbf{E}_{\pi^{\star}}[q_{\star}(s, a)]$$

State-value is expected action-value.

$$= \sum_a \pi^{\star}(a | s) q_{\star}(s, a)$$

Definition of expectation.

$$= \max_a q_{\star}(s, a)$$

Optimal policy is greedy w.r.t q_{\star}

$$= \max_a \mathbf{E}_{\pi^{\star}}[G_t | S_t = s, A_t = a]$$


Definition of action-value .

$$= \max_a \mathbf{E}_{\pi^{\star}}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a]$$

Recursive definition of return.

$$= \max_a \mathbf{E}_{\pi^{\star}}[R_{t+1} + \gamma v_{\star}(S_{t+1}) | S_t = s, A_t = a]$$

Definition of state-value.


$$v_{\star}(s) = \max_a \sum_{s', r} p(s', r | s, a) [r + \gamma v_{\star}(s')]$$

Definition of expectation.

Value Iteration

- Turn Bellman optimality equation into a value function update rule.
- Maintain a look-up table of values, one for each state.
- Set all values to zero initially, $v_0(s) \leftarrow 0$ for all states s .
- Then loop over all states and update until convergence to $v_\star(s)$:

$$v_{k+1}(s) \leftarrow \max_a \sum_{s',r} p(s', r | s, a) [r + \gamma v_k(s')]$$

- Optimal policy can be determined as:

$$\pi^\star(s) \leftarrow \arg \max_a q_\star(s, a) = \arg \max_a \sum_{s',r} P(s', r | s, a) [r + \gamma v_\star(s')]$$

Value Iteration Demo

https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_dp.html

Q-learning

- Value iteration is not a learning method.
 - Requires knowledge of transitions and rewards to compute updates and determine optimal policy.
- Instead, want to learn from **transitions (s,a,s',r) as the agent experiences them**: Agent is in state s and takes action a and then receives reward r and transitions to state s' .
- Q-learning: initialize $q(s, a) = 0$ for all states and actions and then, for the transition at time t , update: **Learning rate**

$$q_{t+1}(s_t, a_t) \leftarrow (1 - \alpha)q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a'} q(s_{t+1}, a'))$$

Old Estimate **Learning Target**

Why is Q-learning reasonable?

- Consider a modified version of value iteration that learns action-values:

$$q_{k+1}(s, a) \leftarrow \sum_{s', r} p(s', r | s, a) [r + \gamma \max_{a'} q_k(s', a')]$$

- Without p we cannot compute right hand side **but** can approximate it after experiencing a *single* sample of the reward and resulting next state.

$$q_{k+1}(s, a) \approx r + \gamma \max_{a'} q_k(s', a')$$

- With only a single reward and next state the update is noisy.
 - Moves q towards q_\star in expectation but any single update has error.
- Use a step-size parameter, $\alpha \in (0, 1)$, to mitigate noise:

$$q_{k+1}(s, a) \leftarrow (1 - \alpha)q_k(s, a) + \alpha(r + \gamma \max_{a'} q_k(s', a'))$$

Q-learning Pseudocode

- Parameters: step-size $\alpha \in (0,1)$, discount $\gamma \in [0,1)$.
 - Initialize $q(s, a)$ arbitrarily for all states and actions except terminal states have $q(\text{terminal}, a) = 0$ for all a .
 - Loop for each episode:
 - Initialize initial state s
 - Loop for each step of episode until s is terminal:
 - Choose a from s using an exploration policy (next slide).
 - Take action a and observe r and s'
 - $q(s, a) \leftarrow q(s, a) + \alpha(r + \gamma \max_{a'} q(s', a') - q(s, a))$ Equivalent to update on previous slide
 - $s \leftarrow s'$
- Converges to $q_{\star}(s, a)$!**
***assuming each (s,a) visited an infinite number of times**

Q-learning Exploration

- Must execute some policy to produce data during learning. What policy should we follow?
- **Greedy policy:** in state s take action $\arg \max_a q(s, a)$.
- Consider an MDP with some state s and two actions that both (deterministically) lead to the same state s' . Action 1 gives a reward of +1 and action 2 gives a reward of +100. We initialize action-values to be zero and happen to take action 1 on our first visit to s . What will the greedy policy do on the next visit to s ?
 - **Exploitation only leads to sub-optimal convergence.**
- **ϵ -greedy:** With probability ϵ take a random action, with probability $1 - \epsilon$ select the greedy action.
- How to set ϵ ?

Q-learning Demo

https://cs.stanford.edu/people/karpathy/reinforcejs/gridworld_td.html

Q-learning with Function Approximation

- What is difficult about representing action-value function with a look-up table?
 - Intractable with large numbers of states and actions.
- Could instead use a class of functions that generalizes to unseen states and actions. Examples: linear functions or neural networks.
 - How to train while learning from transitions (s, a, s', r) ?
- Imagine we had an oracle that told us $q_*(s, a)$.
 - Could create supervised learning instances $((s, a), r + \gamma \max_{a'} q_*(s', a'))$
 - Then we could use supervised learning to train $q(s, a, \theta)$ via regression.
- We don't have such an oracle so instead we will use $q(s, a, \theta)$ to provide the training target.

Semi-gradient Learning in RL

- Instead of using a table, represent action-values as a function with learnable parameters, $q(s, a, \theta)$.
- Semi-Gradient Q-learning:
 - $$\theta_{t+1} \leftarrow \theta_t + \alpha \underbrace{(r_{t+1} + \gamma \max_{a'} q(s_{t+1}, a', \theta_t) - q(s_t, a_t, \theta_t))}_{\text{Temporal difference error}} \nabla q(s_t, a_t, \theta_t)$$
 - Example: parameter vector, θ_t , could be all weights and biases of a neural network.
 - Use back propagation to compute gradient of $q(s, a, \theta)$ for any (s, a) .
 - Adjust each weight in proportion to gradient of output times **temporal difference error**.

Stability with Neural Networks

- Neural networks are typically trained with i.i.d. data and fixed targets.
 - $x_i, y_i \sim D$ and we are learning some underlying function such that $f(x_i) = y_i$.
- Using neural networks with Q-learning breaks both assumptions.
 - Training may be unstable and diverge; lacks theoretical guarantees.
- Deep Q-Network (DQN) uses two key methods to stabilize training:
 - Experience replay: keep around old data to update network.
 - Target networks: Use an older copy of network parameters to compute the target for updates; update this older copy at a slower rate than main parameters used.

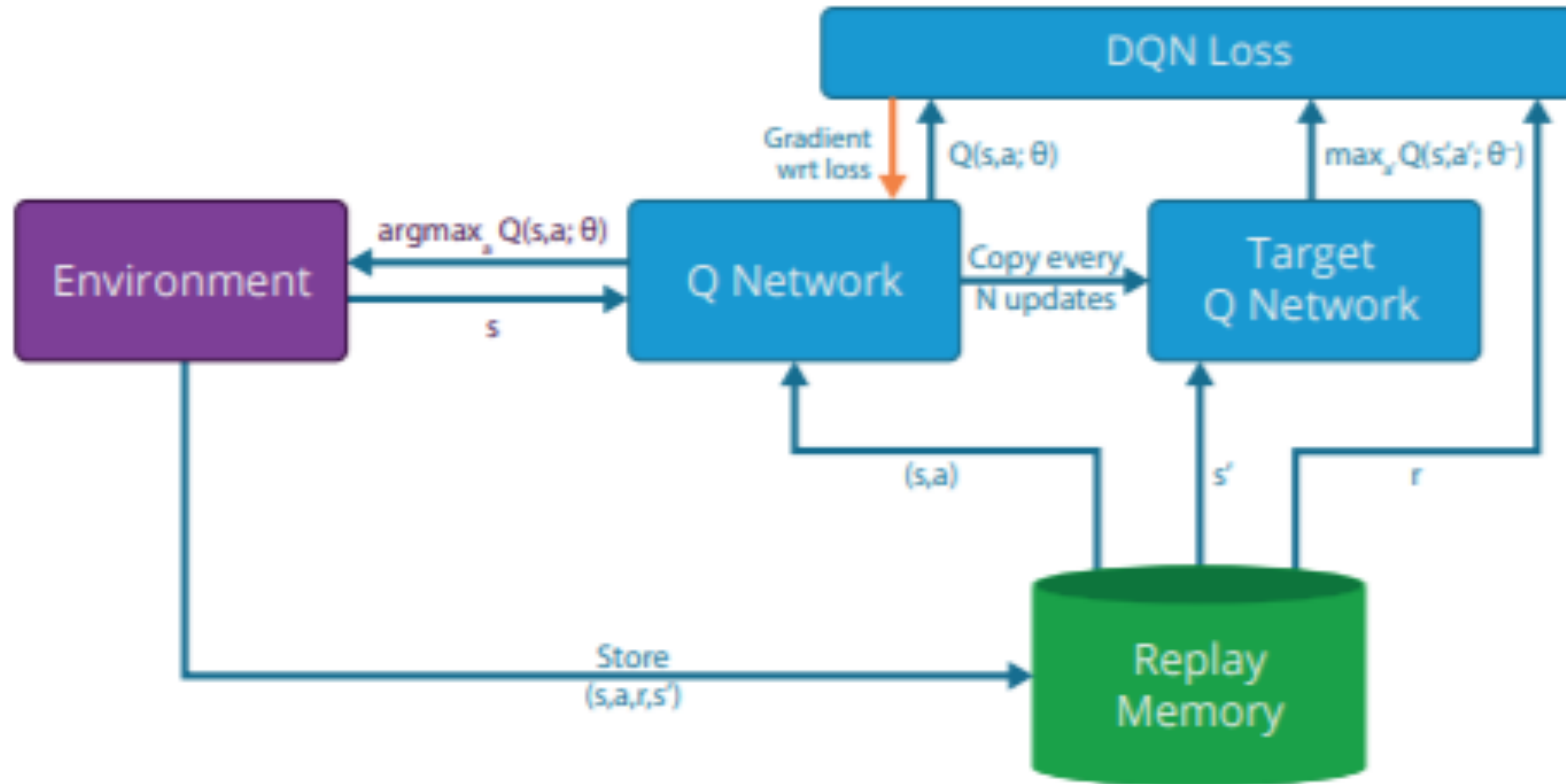
Experience Replay

- The basic semi-gradient Q-learning algorithm processes (s, a, s', r) transitions as they are experienced and then discards them.
- **Experience replay:** save the most recent transitions (in DQN, the past 1 million) and use a random subset to update the action-value function.
 - Re-uses data; reduces correlation between samples.
 - Learning becomes more like supervised neural network training where we train from a static data set.
- Other choices besides random subset can improve performance [1].

Target Networks

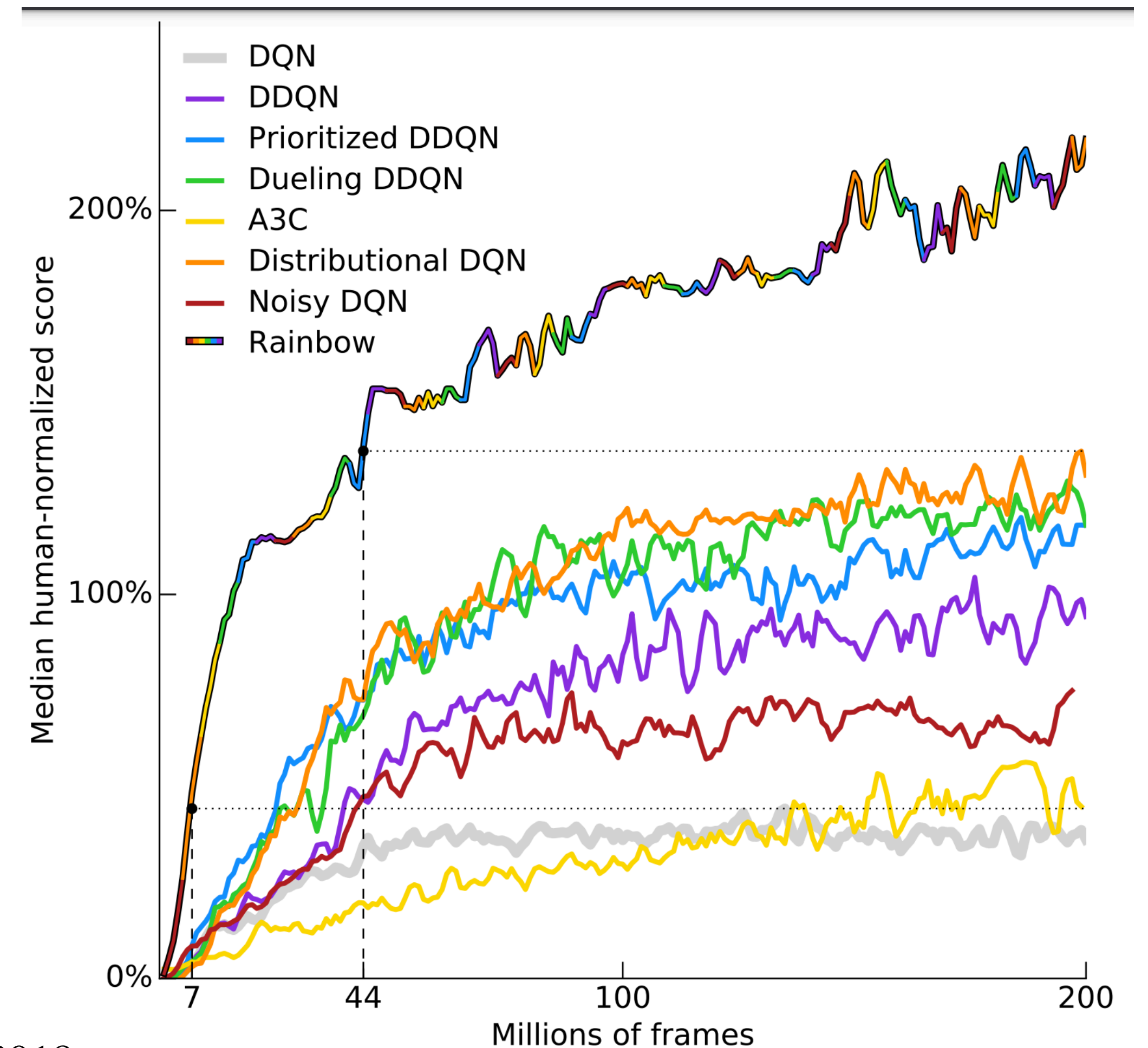
- The basic Q-learning algorithm always uses the most recent action-values to form the training target $r_{t+1} + \gamma \max_{a'} q(s_{t+1}, a', \theta)$
- DQN uses a separate **target network** to compute $\gamma \max_{a'} q(s_{t+1}, a', \tilde{\theta})$.
- The target network is infrequently updated by setting the target network parameters to be the same as the main network's parameters, i.e., $\tilde{\theta} \leftarrow \theta$.
- Makes the learning target more stable as in supervised learning.

DQN Architecture



Looking Forward

- DQN (arguably) launched a surge of interest in deep reinforcement learning that has led to many exciting new applications and RL developments.
- DQN is widely used in practice though many improvements have been made.

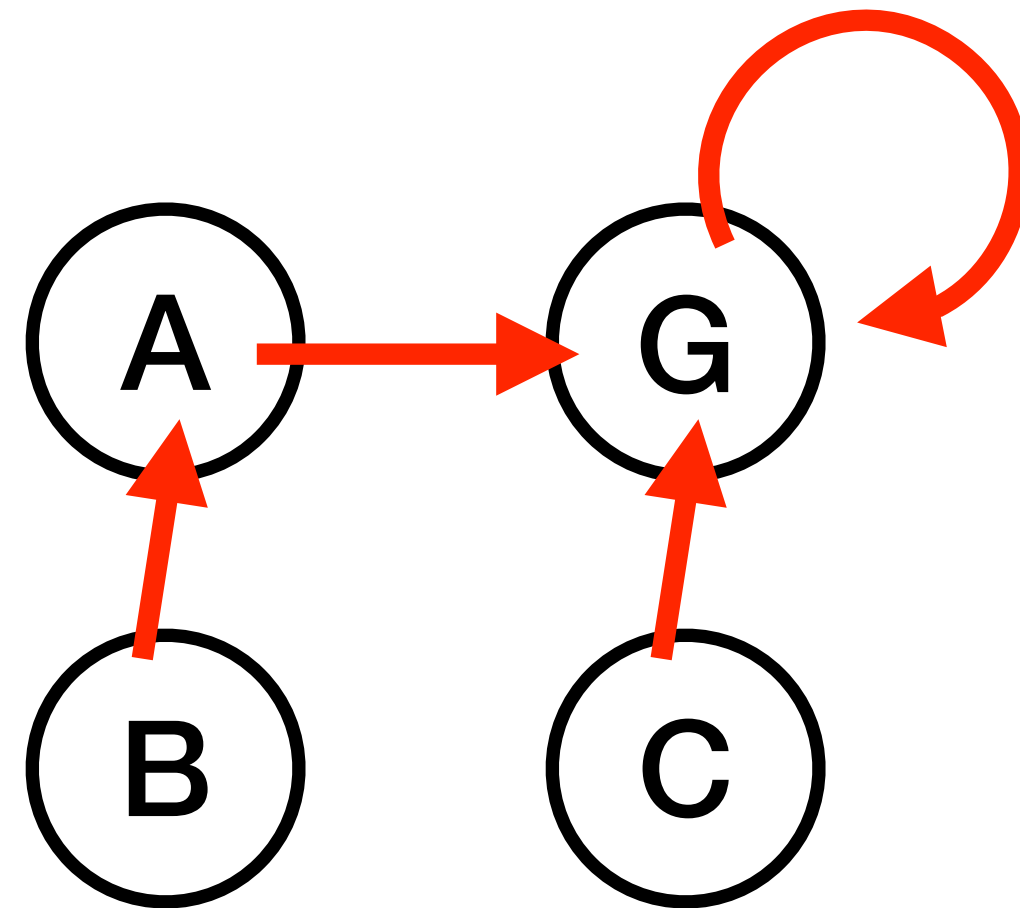


Rainbow: Combining Improvements in Deep Reinforcement Learning. Hessel et al. 2018.

<https://www.deepmind.com/blog/agent57-outperforming-the-human-atari-benchmark>

Quiz

Consider the following MDP which has deterministic transitions and $\gamma = 0.8$. In each state, the agent can either stay in the current state or take an action that takes it to the state given by the red arrows. We run q-learning with all action-values initially set to 0.



$$r(B) = 20; r(A) = 10; r(C) = 20; r(G) = 100$$

Summary

- Value Iteration: directly compute optimal value function given knowledge of task transition and reward function.
- Q-learning: a learning method that approximates value iteration.
- Deep Q-learning: approximates Q-learning with deep neural networks.



Thanks Everyone!

Slides adapted from Advanced Topics in RL and based on Chapter 4 of Reinforcement Learning: An Introduction.