CS 760: Machine Learning
Reinforcement Learning III

Josiah Hanna

University of Wisconsin-Madison

December 7, 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 52% participation. > 75% to receive 2 points extra credit on final.

Lecture Goals

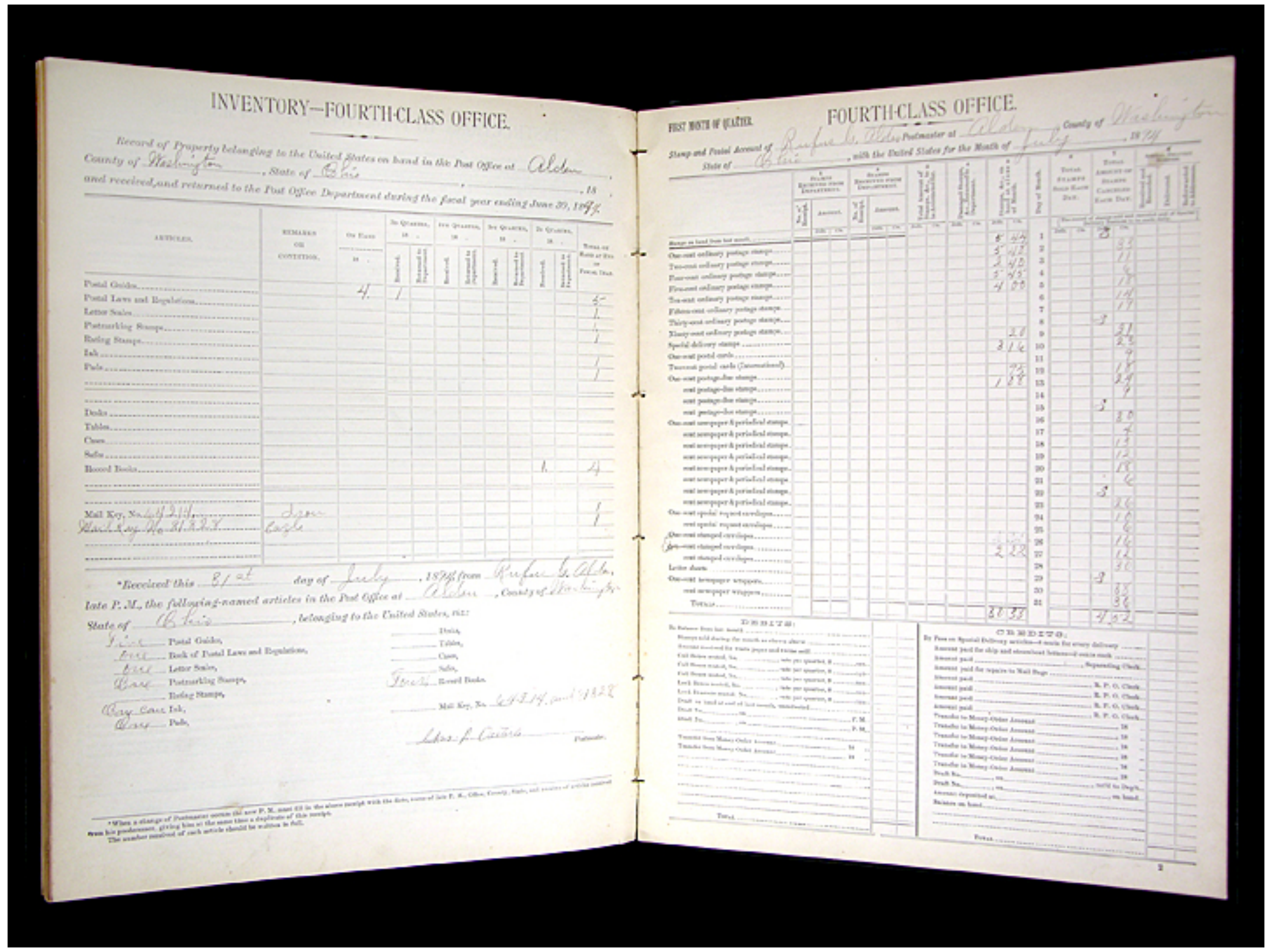
At the end of today's lecture, you will be able to:

1. Explain the key techniques necessary for using neural networks in q-learning.
2. Explain the policy gradient objective and optimization approach.
3. Explain how to combine value-based and policy-based reinforcement learning to obtain actor-critic methods.

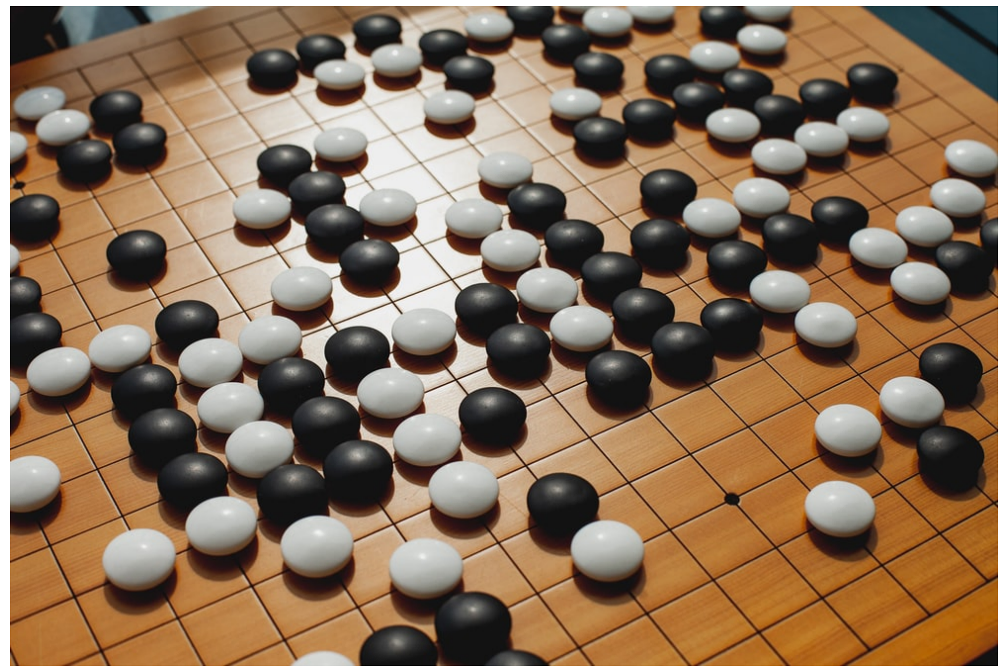
Beyond Tables

So far:

- Represent everything with a table
  - Value function **V**: table size $|S| \times 1$
  - **Q** function: table size $|S| \times |A|$



- Too big to store in memory for many tasks
  - Backgammon: 10^{20} states. Go: 3^{361} states
  - Need some other approach



Semi-gradient Q-Learning

- Instead of using a table, represent action-values as a function with learnable parameters, $q(s, a, \theta)$.

- Semi-Gradient Q-learning:

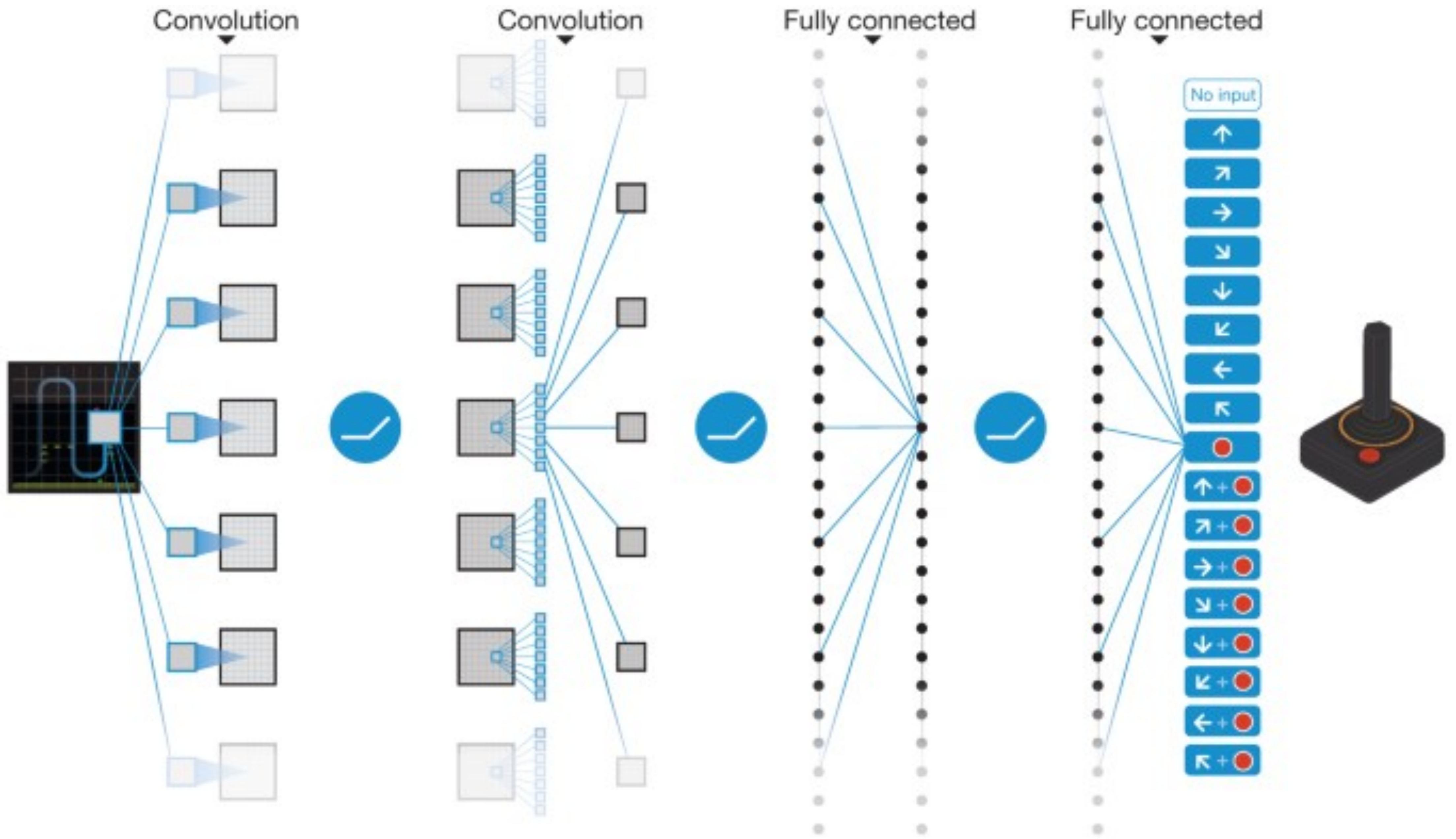
  - $\theta_{t+1} \leftarrow \theta_t + \alpha \underbrace{(r_{t+1} + \gamma \max_{a'} q(s_{t+1}, a', \theta_t) - q(s_t, a_t, \theta_t))}_{\text{Temporal difference error}} \nabla q(s_t, a_t, \theta_t)$

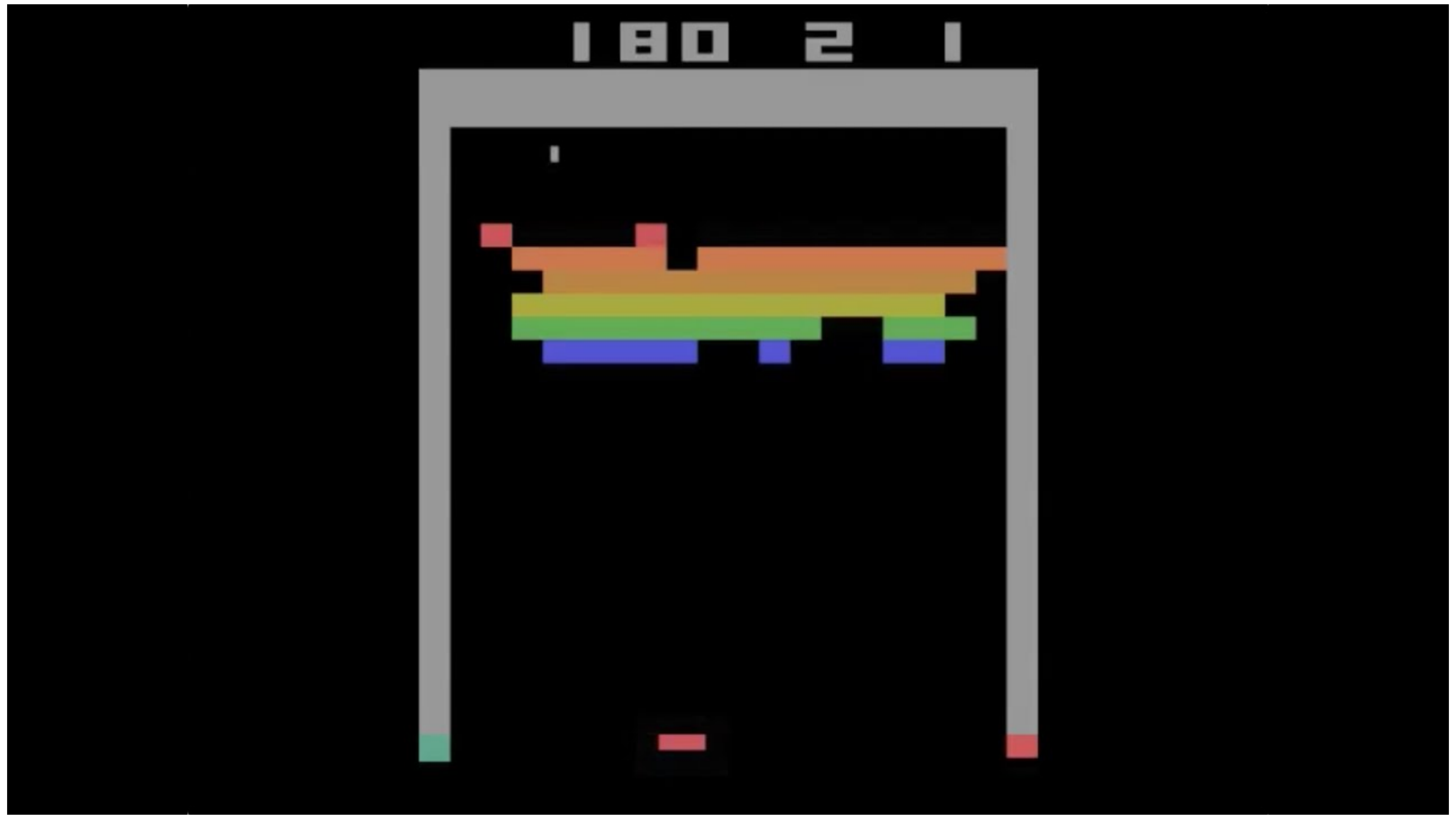
- Example: parameter vector, θ_t, could be all weights and biases of a neural network.

- 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.

Deep Q-learning





Mnih et al, "Human-level control through deep reinforcement learning"

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 and 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].

[1] Prioritized Experience Replay. Schaul et al. 2015.

Target Networks

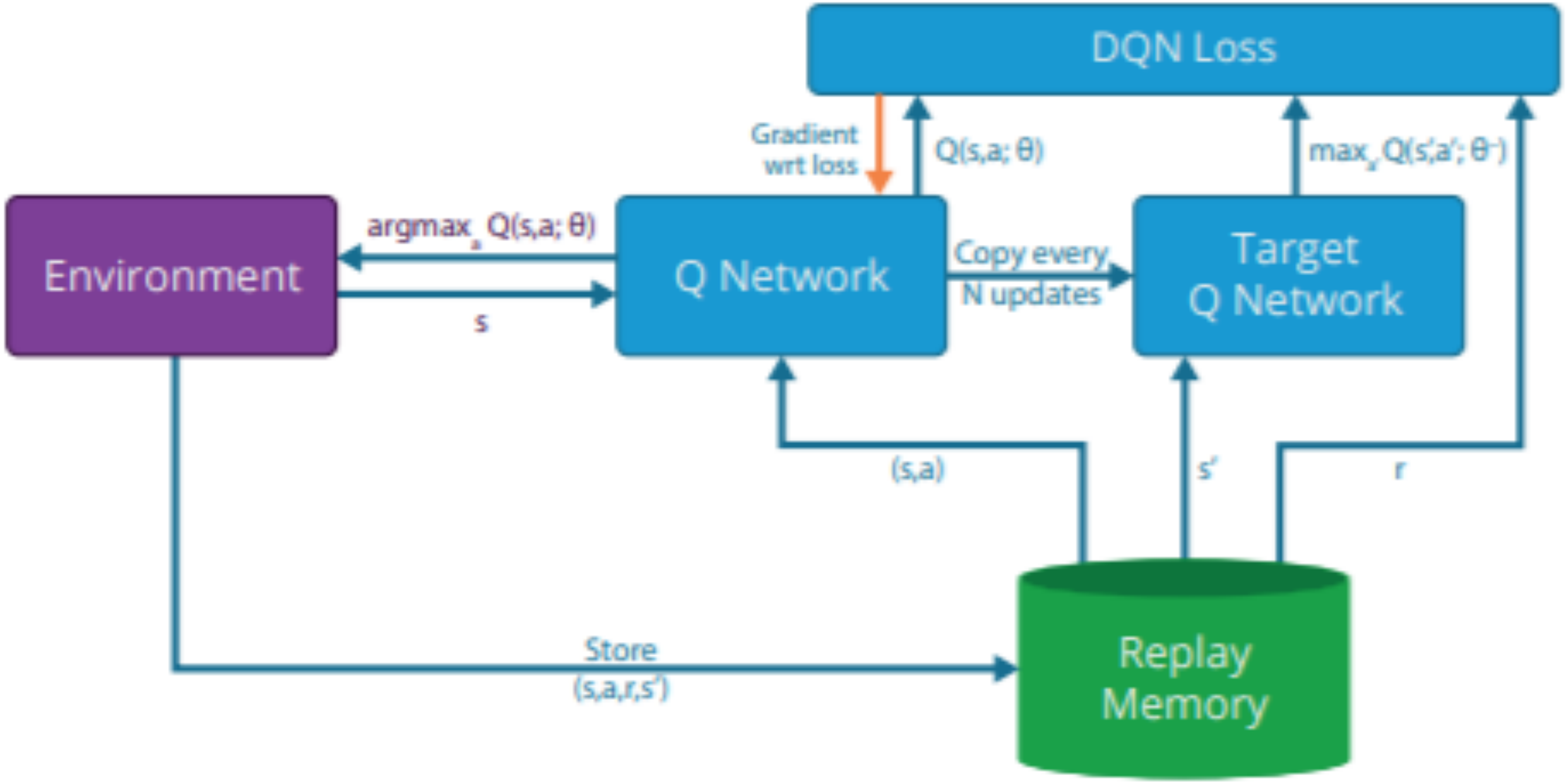
- The basic Q-learning algorithm always uses the most recent action-values to form the training target $r_{t+1} + \gamma \max_{a'} q(s_{t+1}, a', \theta)$

- 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., $\tilde{\theta} \leftarrow \theta$.

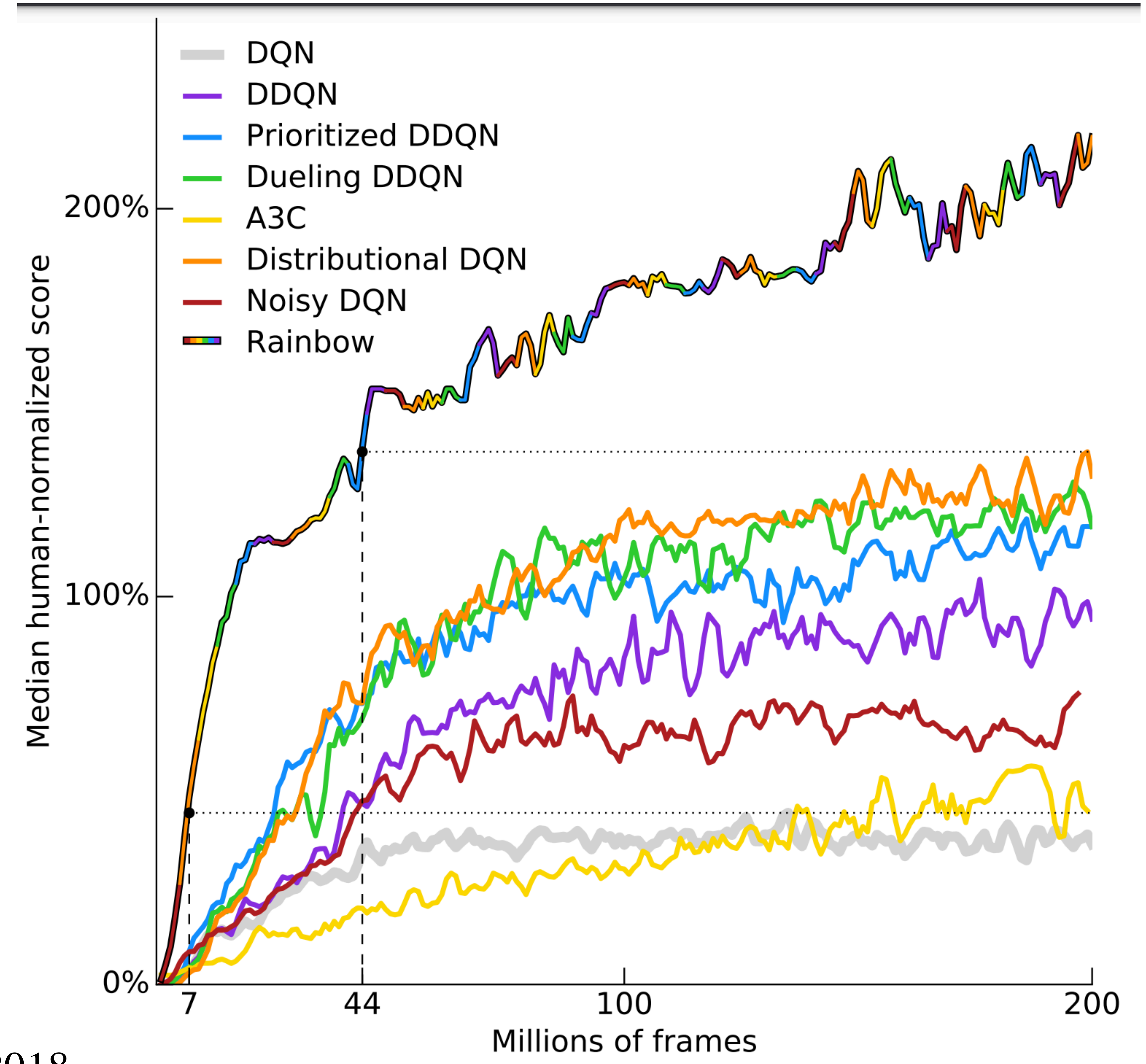
  - Makes the learning target more stable as in supervised learning.

DQN Architecture



Looking Forward

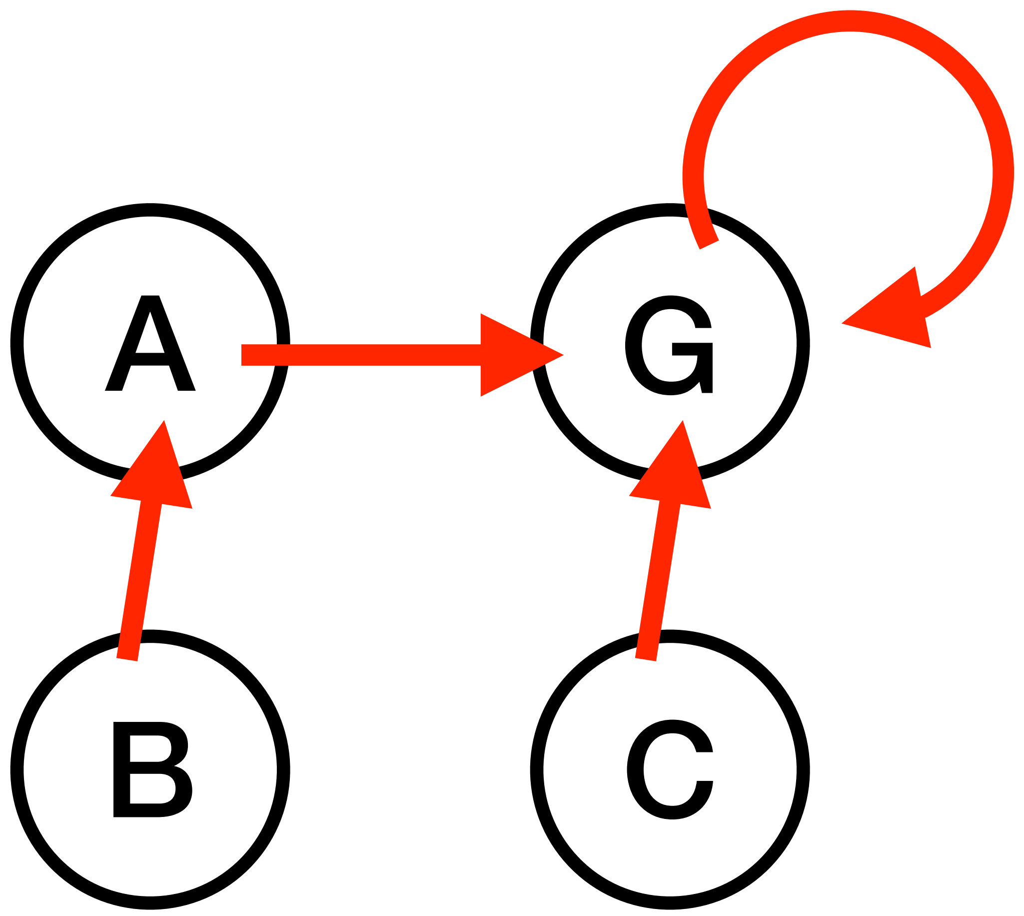
- DQN (arguably) launched a surge of interest in deep reinforcement learning that has led to many exciting new applications and RL developments.
- DQN is widely used in practice though many improvements have been made.



Rainbow: Combining Improvements in Deep Reinforcement Learning. Hessel et al. 2018.
https://www.deepmind.com/blog/agent57-outperforming-the-human-atari-benchmark

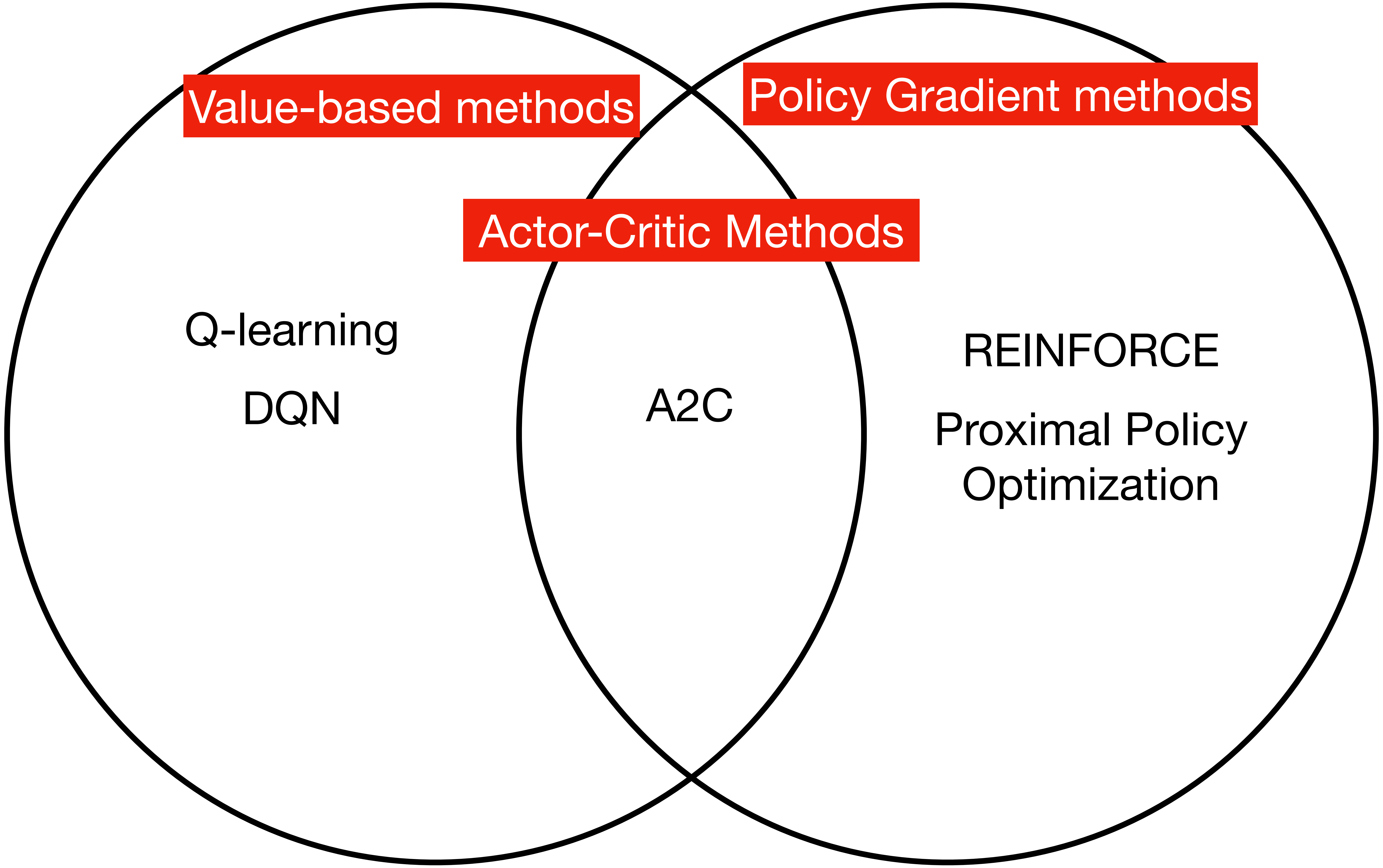
Quiz

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. Write the action-value table after observing the state sequence, B, A, G, G.



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

Model-Free Reinforcement Learning



Policy-based RL

- So far the policy is implicit. Q-learning learns a value function and then acts greedily w.r.t. values.
- Policy-based methods instead explicitly learn the policy.
- $\pi_\theta(a \mid s) = \Pr(A_t = a \mid S_t = s; \theta)$
- Objective is to find policy that maximizes expected future reward:

$$J(\theta) := \sum_s \mu_\theta(s) v_{\pi_\theta}(s)$$

Expected sum of discounted future rewards from s under π_θ

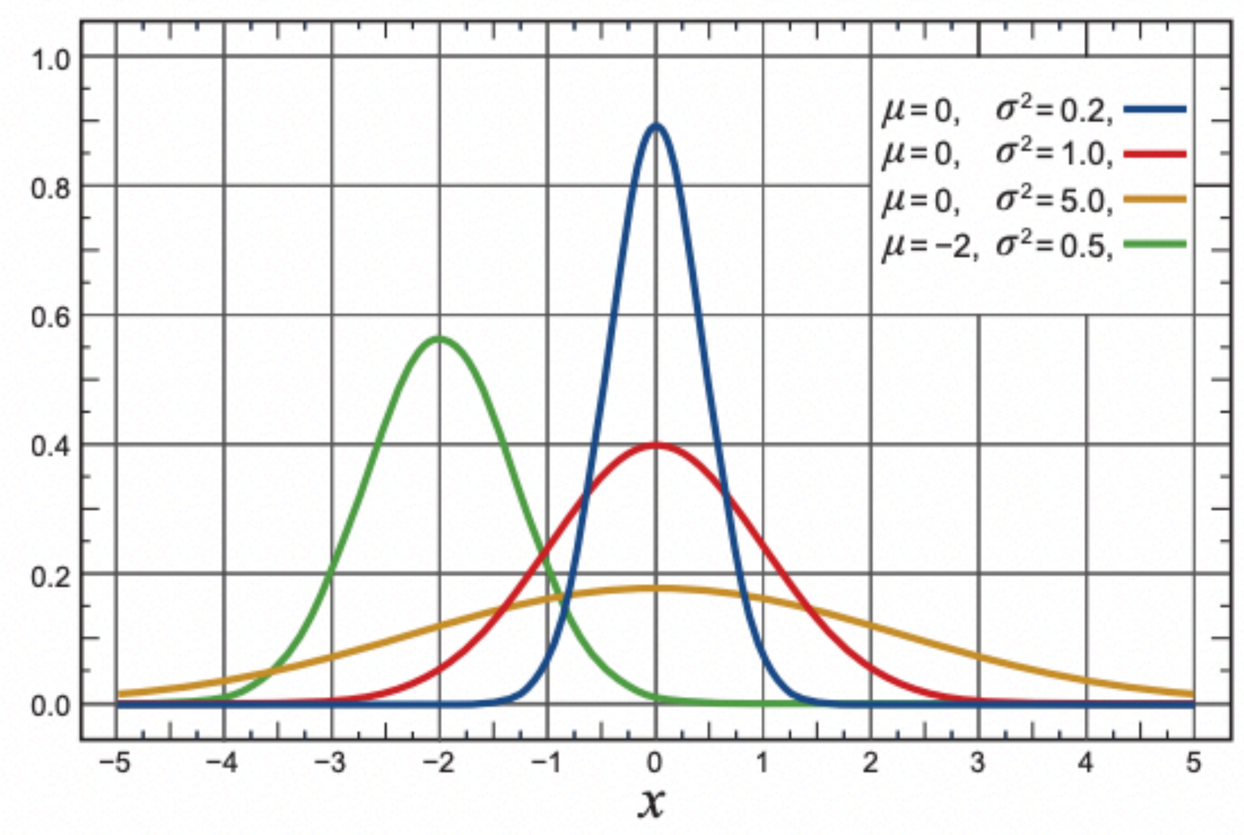
Frequency of visitations to s under π_θ

Why Policy-based?

- Advantages to policy-based methods?
  - More easily handle continuous actions.
  - Policy gradient theorem provides stronger convergence guarantees under function approximation.
  - Useful for partial observability.
  - Policy may be simpler to approximate.
- Disadvantages?
  - May be easier to approximate action-values.
  - Policy is a simple function of the action-values.

Policy Parameterizations

- Policy can be **any** parameterized and differentiable distribution.
- Need $\pi_\theta(A_t = a \mid s)$ and $\nabla_\theta \pi_\theta(A_t = a \mid s)$ exists.
- For discrete action RL tasks, typically use a softmax distribution with logits given by a neural network.
  - Same model that we use for multi-class classification.
- For continuous action RL tasks, typically use a Gaussian distribution with mean and variance each given by a neural network.
  - Same model used for multiple regression.



Policy Gradient Theorem

- $J(\theta) := \sum_s \mu_\theta(s) v_{\pi_\theta}(s)$
- $$\nabla_\theta J(\theta) \propto \sum_s \sum_a \mu_\theta(s) \pi_\theta(a \mid s) q_{\pi_\theta}(s, a) \nabla_\theta \log \pi_\theta(a \mid s)$$
$$= \mathbf{E}[q_{\pi_\theta}(s, a) \nabla_\theta \log \pi_\theta(a \mid s) \mid s \sim \mu_\theta, a \sim \pi_\theta(\cdot \mid s)]$$
- The direction in which an infinitesimally small change to θ produces the maximum increase in $J(\theta)$.
- Nice property: $\nabla_\theta J(\theta)$ does not depend on any gradients of p or $\mu_\theta(s)$.
    - Only have to differentiate policy which is known by the learner.

REINFORCE

- $\nabla_\theta J(\theta) = \mathbf{E}[q_{\pi_\theta}(s,a)\nabla_\theta \log \pi_\theta(a \mid s) \mid s \sim \mu_\theta, a \sim \pi_\theta(\cdot \mid s)]$ cannot be directly computed.
  - But it can be estimated using data obtained by running π_θ.
- Approximate $q_{\pi_\theta}(s_t, a_t)$ with the sum of discounted rewards following s_t and a_t, i.e., G_t.
- $\theta_{t+1} \leftarrow \theta_t + \alpha G_t \nabla_\theta \log \pi(a_t \mid s_t)$

REINFORCE

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

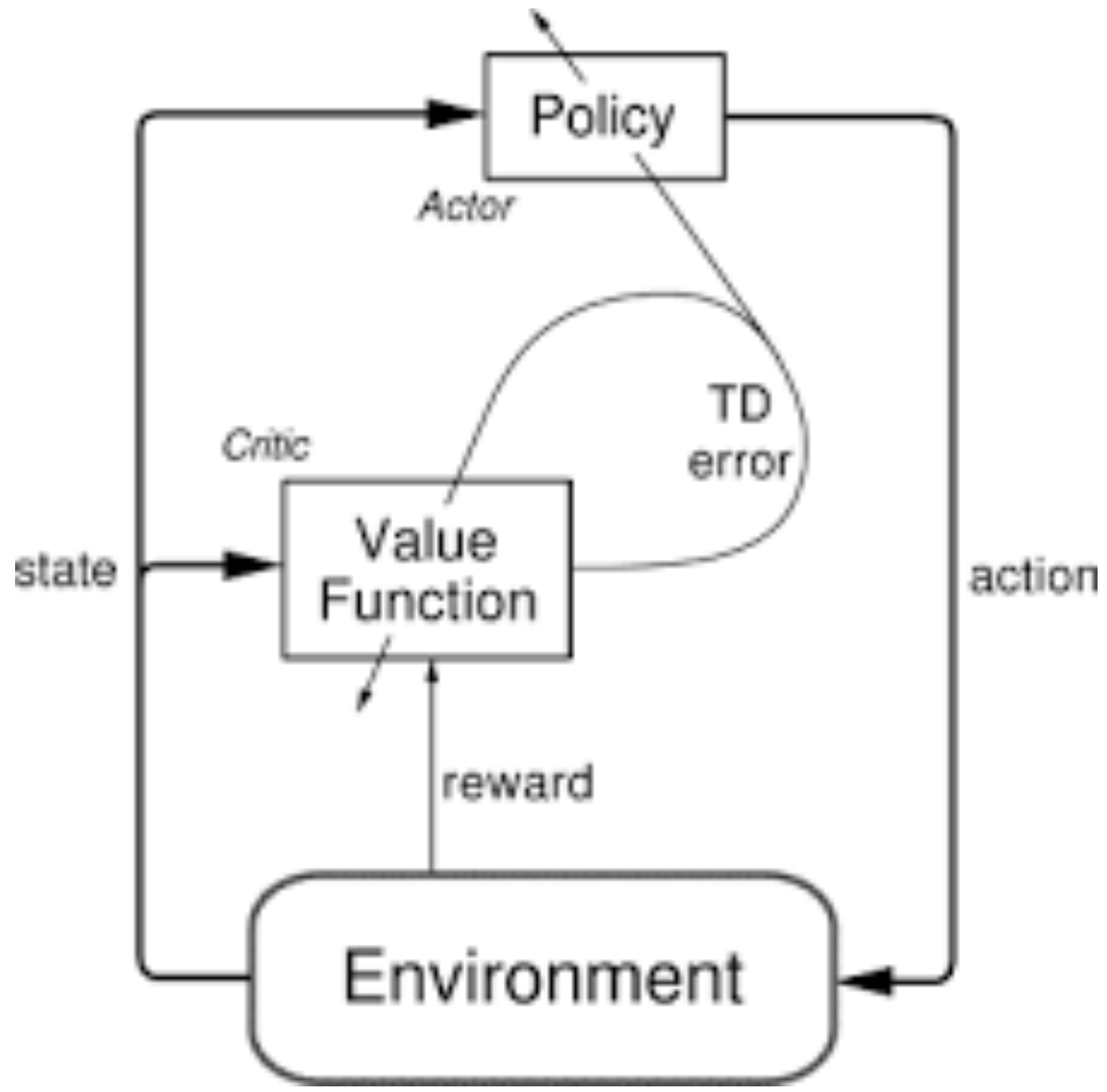
Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \boxed{\gamma^t} G \nabla \ln \pi(A_t|S_t, \boldsymbol{\theta})$

Usually dropped in practice

Is the policy gradient a gradient? Nota and Thomas. 2020.
Bias in Natural Actor-Critic Algorithms. Thomas. 2014.

Actor-Critic Methods

- Basic REINFORCE does not use a value function.
  - Learns directly from the sum of discounted rewards following an action.
- Actor-critic methods use learned value functions to drive policy changes.
- New update:
  - $\delta_t \leftarrow R_{t+1} + \gamma \hat{v}(S_{t+1}) - \hat{v}(S_t)$
  - $\theta_{t+1} \leftarrow \theta_t + \alpha \delta_t \nabla_\theta \ln \pi(A_t | S_t)$



Actor-Critic Methods

One-step Actor–Critic (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S, \boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$ (if S' is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S, \boldsymbol{\theta})$
    $\cancel{I \leftarrow \gamma I}$
    $S \leftarrow S'$

Comparing REINFORCE and Basic Actor-Critic

- For comparison, assume that actor-critic only updates at end of episodes.

- REINFORCE update:

$$\theta \leftarrow \theta + \alpha \sum_{t=0}^{T} \tilde{G}_t \nabla_\theta \log \pi_\theta(a_t \mid s_t) \qquad \tilde{G}_t = (\sum_{t'=t}^{T} \gamma^{t'} R_{t'+1}) - v_{\pi_\theta}(s_t)$$

- Actor-Critic update:

$$\theta \leftarrow \theta + \alpha \sum_{t=0}^{T} \delta_t \nabla_\theta \log \pi_\theta(a_t \mid s_t) \qquad \delta_t = R_{t+1} + \gamma v_{\pi_\theta}(S_{t+1}) - v_{\pi_\theta}(s_t)$$

- Can generalize learning signal:

$$\delta_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n v_{\pi_\theta}(S_{t+n}) - v_{\pi_\theta}(s_t)$$

If $t + n$ is greater than termination step T then R_{t+n} is taken to be zero.

Advantage Actor-Critic (A2C)

- Basic multi-step actor-critic method that works well with deep networks.
  - Policy and value function are represented as neural networks with parameters θ and ϕ respectively.
- A2C alternates collecting n steps of experience in task environment and then updating a state-value function and a policy with the learning signal $\delta_t^{(n)}$.

$$\delta_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n v_\phi(S_{t+n}) - v_\phi(s_t)$$

- Training losses:

$$L(\phi) = \sum_{t=0}^{n} (\delta_t^{(n)})^2 \qquad L(\theta) = \sum_{t=0}^{n} \delta_t^{(n)} \log \pi_\theta(a_t \mid s_t)$$

*Must stop gradient through value estimate at times greater than t; be careful to not update critic when optimizing actor.

Summary

- Deep Q-learning: approximates Q-learning with deep neural networks.
- Policy-based methods: directly learn policy with gradient ascent.
- Actor-critic methods: learn value functions (critic) that provide a learning signal for improving the policy (actor).

Thanks Everyone!

Slides adapted from Advanced Topics in RL and based on Chapter 13 of Reinforcement Learning: An Introduction.