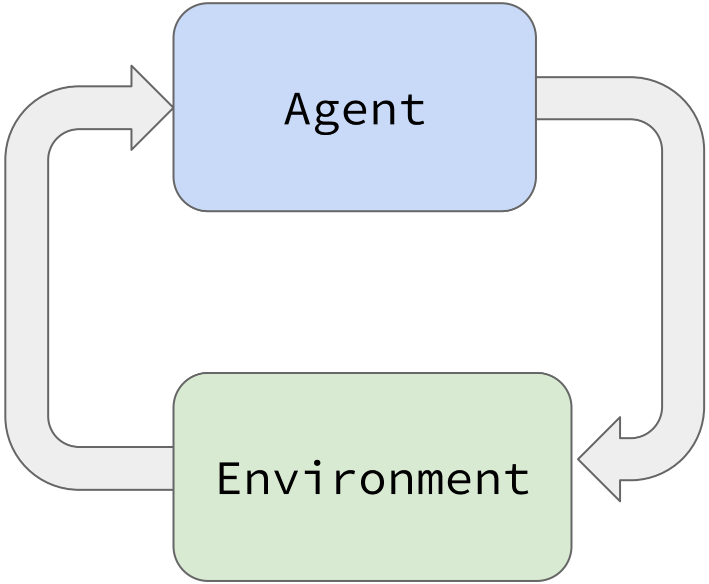
Reinforcement learning

A brief overview through the A3C paper

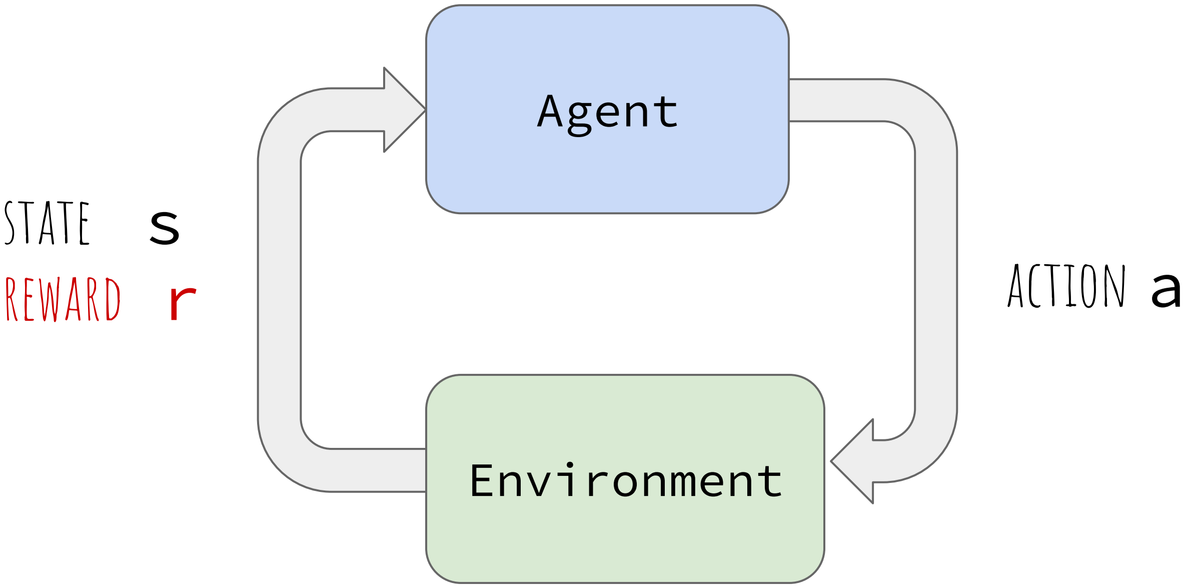
BACK TO BASICS

1

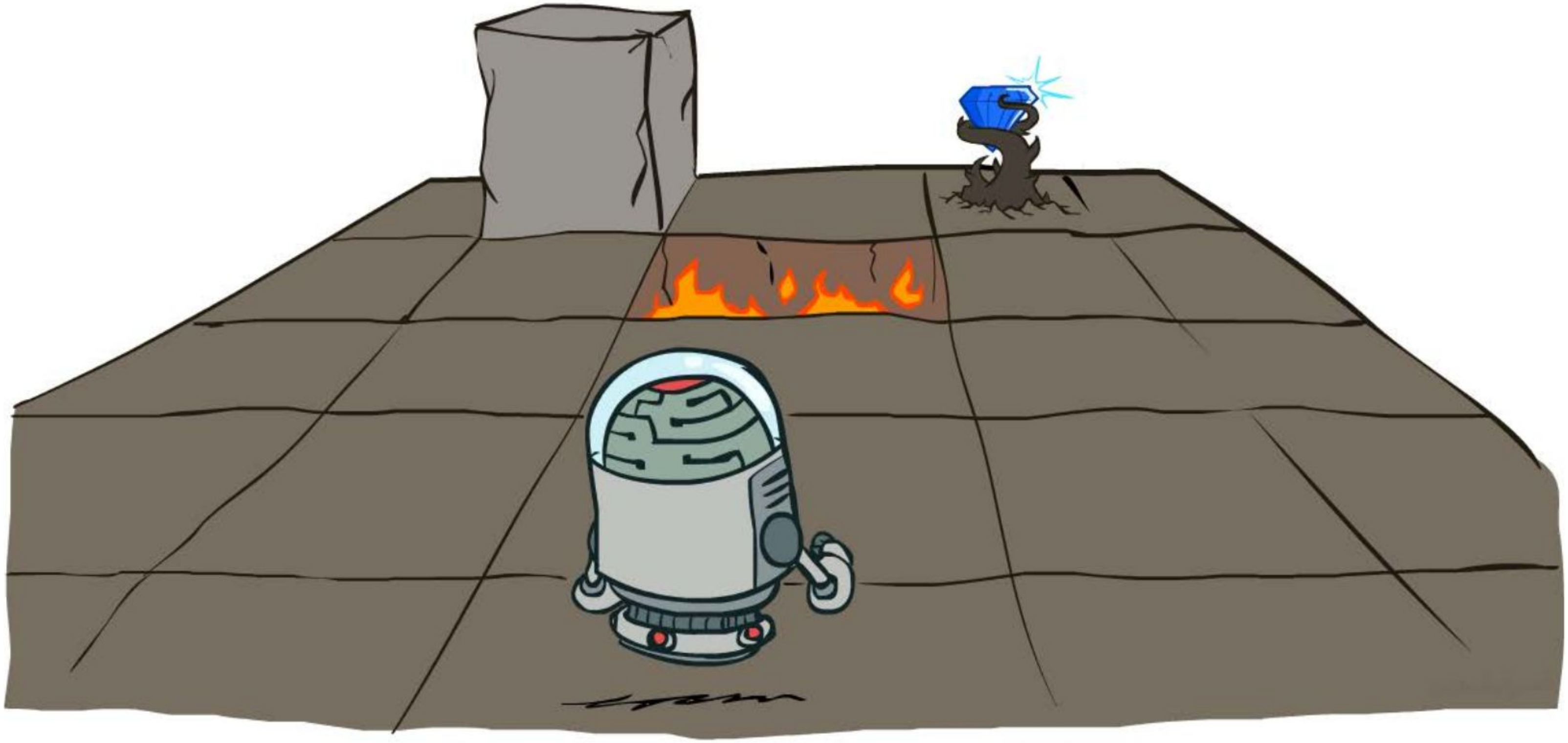
BACK TO BASICS



BACK TO BASICS



Markov Decision Processes : Example



Markov Decision Processes

Markov Decision Processes

set of actions $a \in A$

set of states $s \in S$

transition function $T(s, a, s')$

reward function $R(s, a, s')$

start / terminal state

Markov Decision Processes

Andrey Markov
1856 - 1922

Markov Decision Processes

The stochastic process is memoryless

Markov Decision Processes

The stochastic process is memoryless

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t, \ldots S_0 = s_0)$$

$$P(S_{t+1} = s' | S_t = s_t, A_t = a_t)$$

RL Framework

RL Framework

At each timestep t:

RL Framework

At each timestep t:

- The agent receives a **state** s_t from S

RL Framework

At each timestep t:

- The agent receives a **state** s_t from S
- The agent chooses **action** a_t from A, according to π

RL Framework

At each timestep t:

- The agent receives a **state** s_t from S
- The agent chooses **action** a_t from A, according to π
- The agent receives s_{t+1} and a **reward** r_t

RL Framework

At each timestep t:

- The agent receives a **state** s_t from S
- The agent chooses **action** a_t from A, according to π
- The agent receives s_{t+1} and a **reward** r_t

The process continues until the agent reaches a **terminal state**

RL Framework

At each timestep t:

- The agent receives a **state** s_t from S
- The agent chooses **action** a_t from A, according to π
- The agent receives s_{t+1} and a **reward** r_t

The process continues until the agent reaches a **terminal state**

The **return** is $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$

RL Framework

At each timestep t:

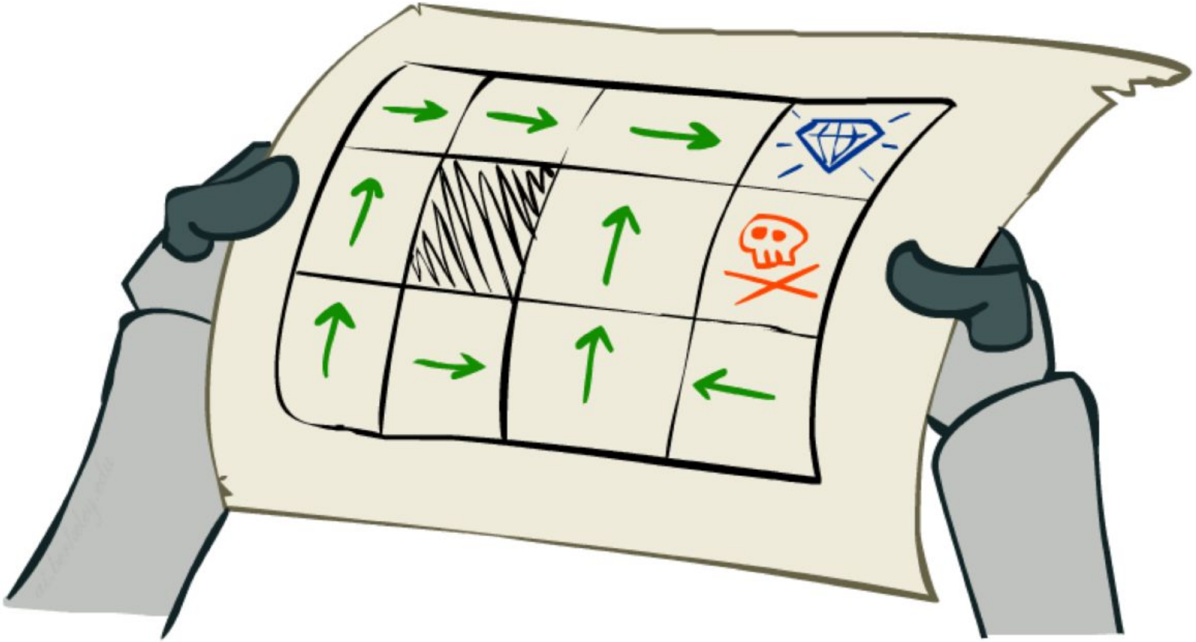
- The agent receives a **state** s_t from S
- The agent chooses **action** a_t from A, according to π
- The agent receives s_{t+1} and a **reward** r_t

The process continues until the agent reaches a **terminal state**

The **return** is $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$

The goal of the agent is to maximize the **expected return** from each state s_t

Goal



Value Methods

Value Methods

State value

$$V^{\pi}(\boldsymbol{s}) = \mathbb{E}[R_t | \boldsymbol{s}_t = \boldsymbol{s}]$$

Value Methods

State value $V^\pi(s) = \mathbb{E}[R_t | s_t = s]$

Action value $Q^\pi(s, a) = \mathbb{E}[R_t | s_t = s, a_t = a]$

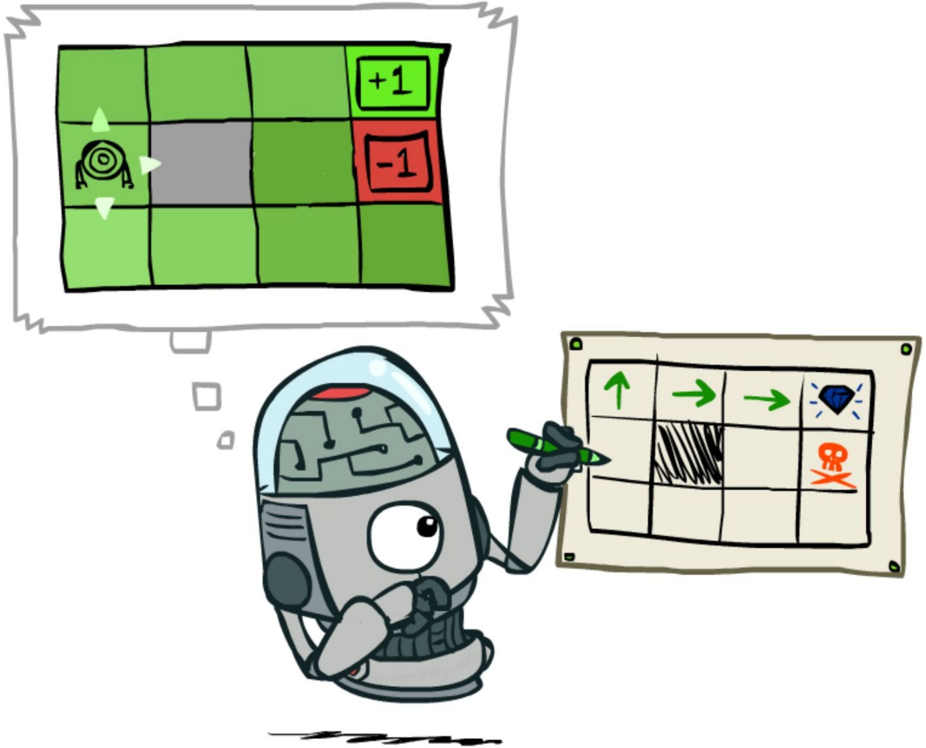
Value Methods

State value $V^\pi(s) = \mathbb{E}[R_t | s_t = s]$

Action value $Q^\pi(s, a) = \mathbb{E}[R_t | s_t = s, a_t = a]$

Optimal value function $Q^*(s, a) = \max_\pi Q^\pi(s, a)$

Value Methods



Q-Learning

Q-Learning

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha)Q^{old}(s_t, a_t) + \alpha(r_t + \gamma \max_{a'} Q^{old}(s_{t+1}, a'))$$

Approximate setting

Approximate setting

$$Q(s, a; \theta)$$

Approximate setting

$$Q(s, a; \theta)$$

$$Q^*(s, a) \approx Q(s, a; \theta)$$

Approximate setting

$$Q(s, a; \theta)$$

$$Q^*(s, a) \approx Q(s, a; \theta)$$

$$L_i(\theta_i) = \mathbb{E}[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)]^2$$

Approximate setting

$$Q(s, a; \theta)$$

$$Q^*(s, a) \approx Q(s, a; \theta)$$

$$L_i(\theta_i) = \mathbb{E}[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)]^2$$

r directly affects one Q(s,a)

N-step Q-Learning

N-step Q-Learning

Use n-step return instead!

$$R^n = r_t + \gamma r_{t+1} + \ldots + \gamma^{n-1} r_{t+n-1} + \max_a \gamma^n Q(s_{t+n}, a)$$

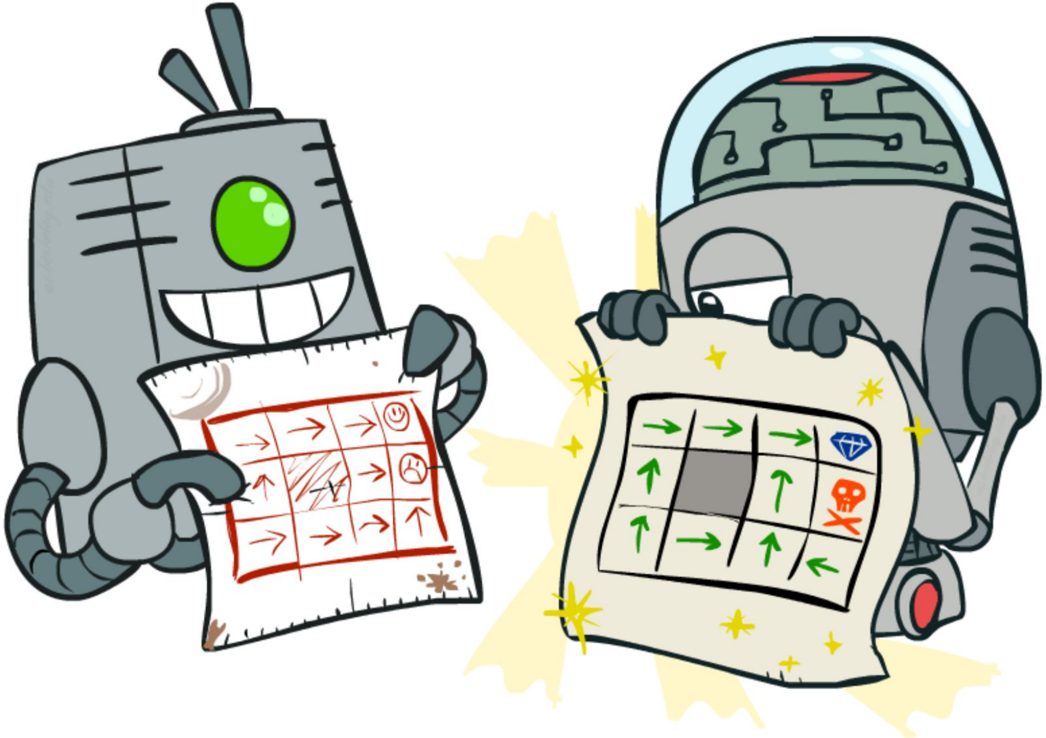
N-step Q-Learning

Use n-step return instead!

$$R^n = r_t + \gamma r_{t+1} + \ldots + \gamma^{n-1} r_{t+n-1} + \max_a \gamma^n Q(s_{t+n}, a)$$

r affects n Q(s,a)

Policy Methods



Policy Methods

Value methods: implicit policy

$$\pi(a|s)$$

Policy Methods

Value methods: implicit policy $\pi(a|s)$

Policy methods: explicit policy $\pi(a|s;\theta)$

Policy Methods

Value methods: implicit policy $\pi(a|s)$

Policy methods: explicit policy $\pi(a|s;\theta)$

Gradient ascent on the return $\mathbb{E}[R_t]$

Policy Methods

Value methods: implicit policy $\pi(a|s)$

Policy methods: explicit policy $\pi(a|s;\theta)$

Gradient ascent on the return $\mathbb{E}[R_t]$

REINFORCE update $\nabla_\theta \log \pi(a_t|s_t;\theta)R_t$

Williams, 1992

Policy Methods

REINFORCE update

$$\nabla_\theta \log \pi(a_t | s_t ; \theta) R_t$$

Policy Methods

REINFORCE update

$$\nabla_\theta \log \pi(a_t | s_t; \theta) R_t$$

Variance problem

Policy Methods

REINFORCE update $\nabla_\theta \log \pi(a_t | s_t ; \theta) R_t$

Variance problem $\nabla_\theta \log \pi(a_t | s_t ; \theta)(R_t - b_t(s_t))$

Policy Methods

REINFORCE update $\nabla_\theta \log \pi(a_t | s_t; \theta) R_t$

Variance problem $\nabla_\theta \log \pi(a_t | s_t; \theta)(R_t - b_t(s_t))$

Common baseline $b_t(s_t) \approx V^\pi(s_t)$

Policy Methods

REINFORCE update

$$\nabla_\theta \log \pi(a_t | s_t ; \theta) R_t$$

Variance problem

$$\nabla_\theta \log \pi(a_t | s_t ; \theta)(R_t - b_t(s_t))$$

Common baseline

$$b_t(s_t) \approx V^\pi(s_t)$$

Advantage

$$A(a_t, s_t) = Q(a_t, s_t) - V(s_t)$$

Policy Methods

REINFORCE update

$$\nabla_\theta \log \pi(a_t | s_t ; \theta) R_t$$

Variance problem

$$\nabla_\theta \log \pi(a_t | s_t ; \theta)(R_t - b_t(s_t))$$

Common baseline

$$b_t(s_t) \approx V^\pi(s_t)$$

Advantage

$$A(a_t, s_t) = Q(a_t, s_t) - V(s_t)$$

Sutton & Barto, 1998; Degris *et al*. 2012

ASYNCHRONOUS RL

2

Mnih et al, 2016

Asynchronous Methods for Deep Reinforcement Learning

Volodymyr Mnih[1] VMNIH@GOOGLE.COM
Adrià Puigdomènech Badia[1] ADRIAP@GOOGLE.COM
Mehdi Mirza[1,2] MIRZAMOM@IRO.UMONTREAL.CA
Alex Graves[1] GRAVESA@GOOGLE.COM
Tim Harley[1] THARLEY@GOOGLE.COM
Timothy P. Lillicrap[1] COUNTZERO@GOOGLE.COM
David Silver[1] DAVIDSILVER@GOOGLE.COM
Koray Kavukcuoglu [1] KORAYK@GOOGLE.COM

[1] Google DeepMind
[2] Montreal Institute for Learning Algorithms (MILA), University of Montreal

Abstract

We propose a conceptually simple and lightweight framework for deep reinforcement learning that uses asynchronous gradient descent for optimization of deep neural network controllers. We present asynchronous variants of

line RL updates are strongly correlated. By storing the agent's data in an experience replay memory, the data can be batched (Riedmiller, 2005; Schulman et al., 2015a) or randomly sampled (Mnih et al., 2013; 2015; Van Hasselt et al., 2015) from different time-steps. Aggregating over memory in this way reduces non-stationarity and decorrelates updates, but at the same time limits the methods to

MOTIVATION

MOTIVATION

RL + function approximation = ?

MOTIVATION

RL + function approximation = ?

Strongly correlated online RL updates

MOTIVATION

RL + function approximation = ?

Strongly correlated online RL updates

Experience replay

MOTIVATION

RL + function approximation = ?

Strongly correlated online RL updates

Experience replay

MOTIVATION

RL + function approximation = ?

Strongly correlated online RL updates

Experience replay

Riedmiller, 2005; Schulman *et al*. 2015a

MOTIVATION

RL + function approximation = ?

Strongly correlated online RL updates

Experience replay

Riedmiller, 2005; Schulman *et al*. 2015a

Mnih *et al*., 2013; 2015; Van Hasselt *et al*. 2015

MOTIVATION

RL + function approximation = ?

Strongly correlated online RL updates

Experience replay

Off-policy only

Riedmiller, 2005; Schulman *et al*. 2015a

Mnih *et al*., 2013; 2015; Van Hasselt *et al*. 2015

Asynchronous RL

Asynchronous RL

Multiple agents in parallel

Asynchronous RL

Multiple agents in parallel

Multiple instances of the environment

Asynchronous RL

Multiple agents in parallel

Multiple instances of the environment

On-policy: Sarsa, n-step methods, actor-critic methods

Asynchronous RL

Multiple agents in parallel

Multiple instances of the environment

On-policy: Sarsa, n-step methods, actor-critic methods

GPU -> CPU

Asynchronous RL framework

Asynchronous RL framework

1 asynchronous actor-learner = 1 thread

Asynchronous RL framework

1 asynchronous actor-learner = 1 thread

1 asynchronous actor-learner = 1 different exploration policy

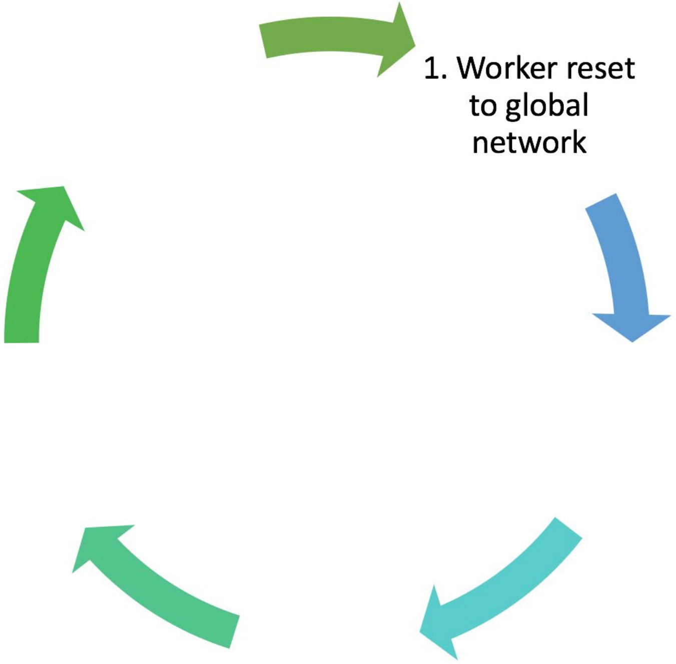
Asynchronous RL framework

1 asynchronous actor-learner = 1 thread

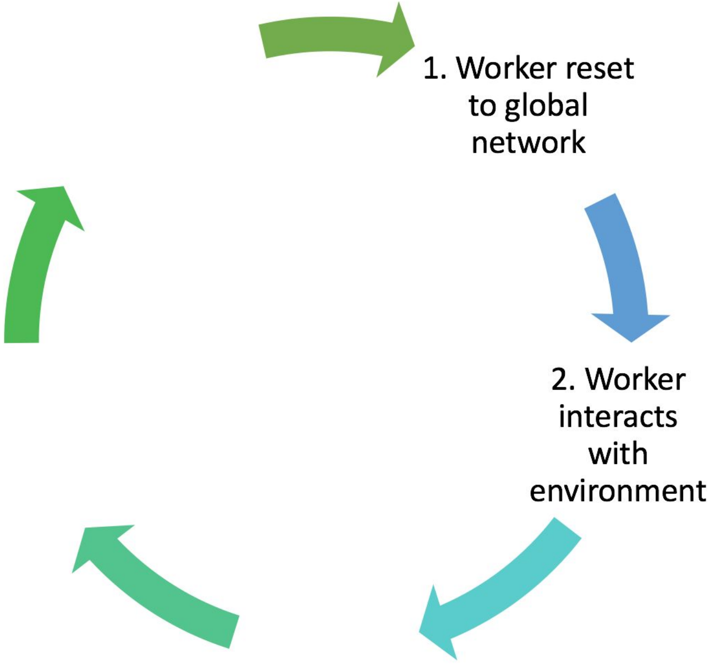
1 asynchronous actor-learner = 1 different exploration policy

No replay memory

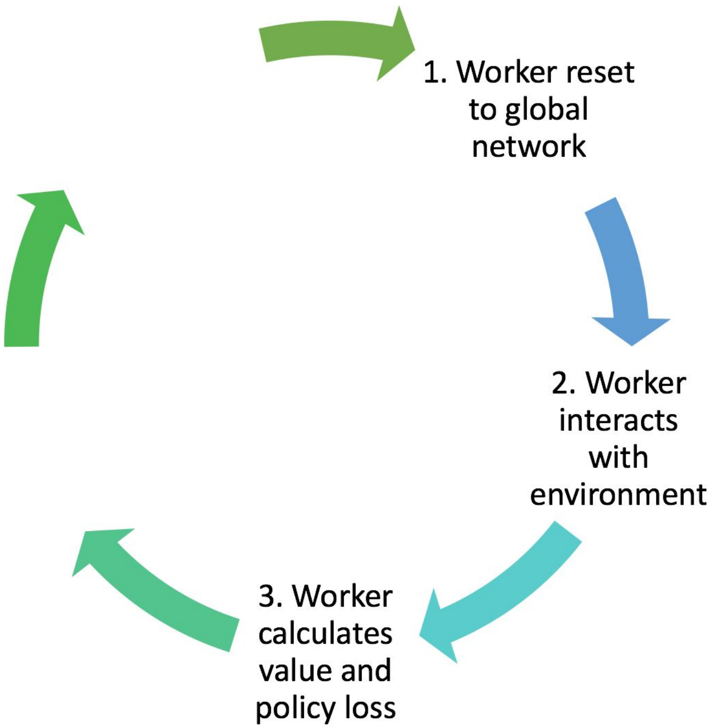
Asynchronous RL framework



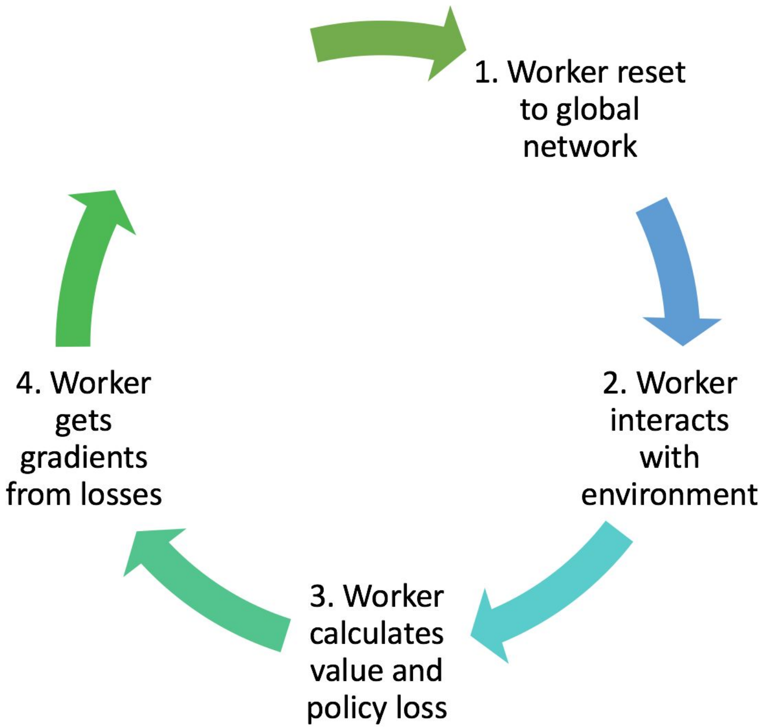
Asynchronous RL framework



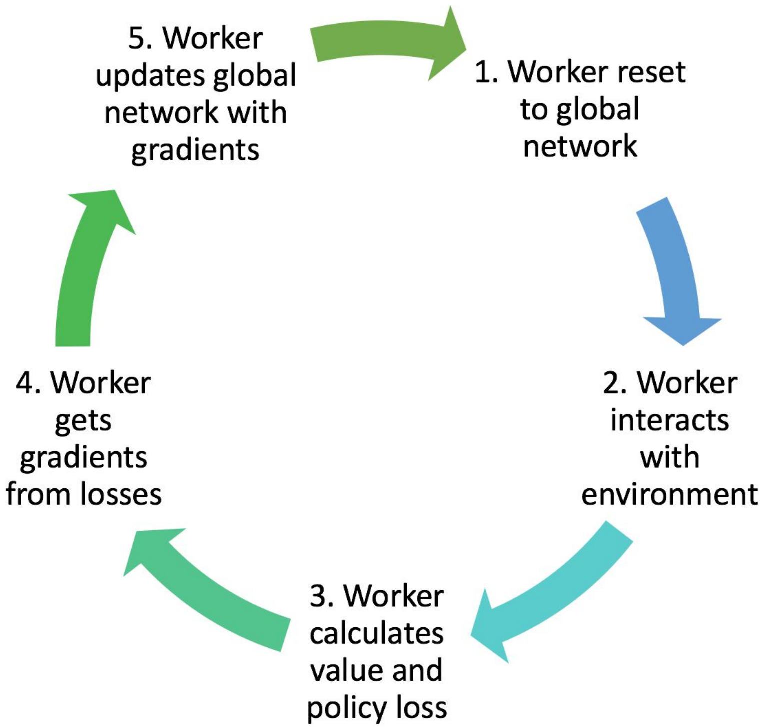
Asynchronous RL framework



Asynchronous RL framework



Asynchronous RL framework



Asynchronous one-step Q-learning

Asynchronous one-step Q-learning

1 thread = 1 copy of the environment

Asynchronous one-step Q-learning

1 thread = 1 copy of the environment

At each step computes the gradient of the Q-learning loss

Asynchronous one-step Q-learning

1 thread = 1 copy of the environment

At each step computes the gradient of the Q-learning loss

$$L_i(\theta_i) = \mathbb{E}[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)]^2$$

Asynchronous one-step Q-learning

1 thread = 1 copy of the environment

At each step computes the gradient of the Q-learning loss

Accumulate gradients over multiple time steps

Algorithm 1 Asynchronous one-step Q-learning - pseudocode for each actor-learner thread.

// Assume global shared θ, θ^-, and counter $T = 0$.
Initialize thread step counter $t \leftarrow 0$
Initialize target network weights $\theta^- \leftarrow \theta$
Initialize network gradients $d\theta \leftarrow 0$
Get initial state s
repeat
    Take action a with ϵ-greedy policy based on $Q(s, a; \theta)$
    Receive new state s' and reward r
    $y = \begin{cases} r & \text{for terminal } s' \\ r + \gamma \max_{a'} Q(s', a'; \theta^-) & \text{for non-terminal } s' \end{cases}$
    Accumulate gradients wrt θ: $d\theta \leftarrow d\theta + \frac{\partial (y - Q(s,a;\theta))^2}{\partial \theta}$
    $s = s'$
    $T \leftarrow T + 1$ and $t \leftarrow t + 1$
    if $T \mod I_{target} == 0$ **then**
        Update the target network $\theta^- \leftarrow \theta$
    end if
    if $t \mod I_{AsyncUpdate} == 0$ or s is terminal **then**
        Perform asynchronous update of θ using $d\theta$.
        Clear gradients $d\theta \leftarrow 0$.
    end if
until $T > T_{max}$

Algorithm 1 Asynchronous one-step Q-learning - pseudocode for each actor-learner thread.

// Assume global shared θ, θ^-, and counter $T = 0$.
Initialize thread step counter $t \leftarrow 0$
Initialize target network weights $\theta^- \leftarrow \theta$
Initialize network gradients $d\theta \leftarrow 0$
Get initial state s
repeat
    Take action a with ϵ-greedy policy based on $Q(s, a; \theta)$
    Receive new state s' and reward r

$$y = \begin{cases} r & \text{for terminal } s' \\ r + \gamma \cancel{\max}_{a'} Q(s', a'; \theta^-) & \text{for non-terminal } s' \end{cases}$$

Sarsa

    Accumulate gradients wrt θ: $d\theta \leftarrow d\theta + \frac{\partial (y - Q(s,a;\theta))^2}{\partial \theta}$
    $s = s'$
    $T \leftarrow T + 1$ and $t \leftarrow t + 1$
    if $T \mod I_{target} == 0$ **then**
        Update the target network $\theta^- \leftarrow \theta$
    end if
    if $t \mod I_{AsyncUpdate} == 0$ or s is terminal **then**
        Perform asynchronous update of θ using $d\theta$.
        Clear gradients $d\theta \leftarrow 0$.
    end if
until $T > T_{max}$

Asynchronous n-step Q-learning

Asynchronous n-step Q-learning

Use forward view instead of backward view

Asynchronous n-step Q-learning

Use forward view instead of backward view

Easier for training NNs with momentum

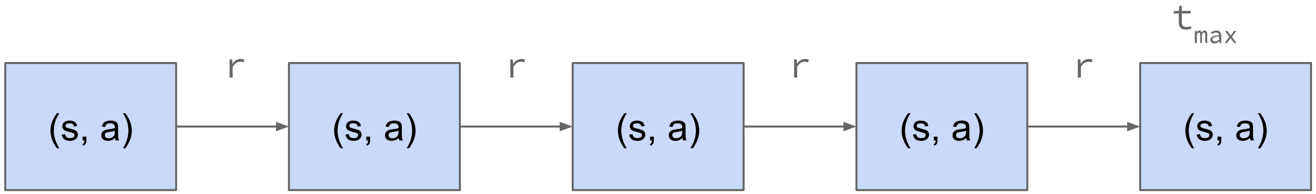
Asynchronous n-step Q-learning

Use forward view instead of backward view

Easier for training NNs with momentum

First select actions up to t_{max} or until terminal state

Asynchronous n-step Q-learning



Asynchronous Advantage Actor Critic (A3C)

Asynchronous Advantage Actor Critic (A3C)

Maintains $\pi(a_t, s_t; \theta)$ and $V(s_t; \theta_v)$

Asynchronous Advantage Actor Critic (A3C)

Maintains $\pi(a_t, s_t; \theta)$ and $V(s_t; \theta_v)$

Also operates in forward view (update after every t_{max})

Asynchronous Advantage Actor Critic (A3C)

Maintains $\pi(a_t, s_t; \theta)$ and $V(s_t; \theta_v)$

Also operates in forward view (update after every t_{max})

Update $\nabla_{\theta'} \log \pi(a_t | s_t; \theta') A(s_t, a_t; \theta, \theta_v)$

Asynchronous Advantage Actor Critic (A3C)

Maintains $\pi(a_t, s_t; \theta)$ and $V(s_t; \theta_v)$

Also operates in forward view (update after every t_{max})

Update $\nabla_{\theta'} \log \pi(a_t | s_t; \theta') A(s_t, a_t; \theta, \theta_v)$

Advantage (k up to t_{max}) $\sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}; \theta_v) - V(s_t; \theta_v)$

A3C details

A3C details

θ and θ_v come from the same network with two heads

A3C details

θ and θ_v come from the same network with two heads

Add entropy regularization term

Results

3

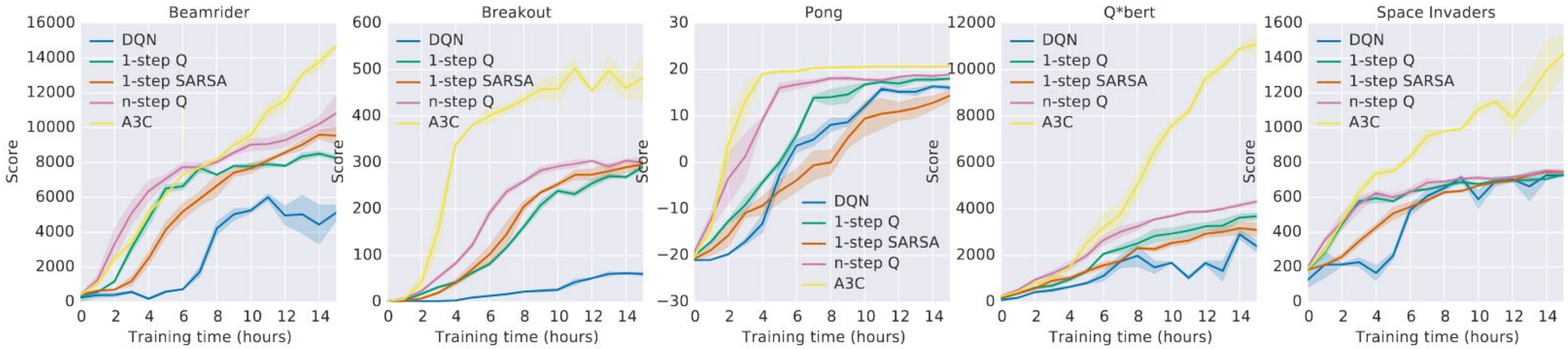
Benchmarks

Arcade Learning Environment (Bellemare et al., 2012)

TORCS 3D Racing simulator (Wyman et al., 2013)

A3C only: MuJoCo (Todorov, 2015)

ATARI 2600



DQN: Nvidia K40 GPU

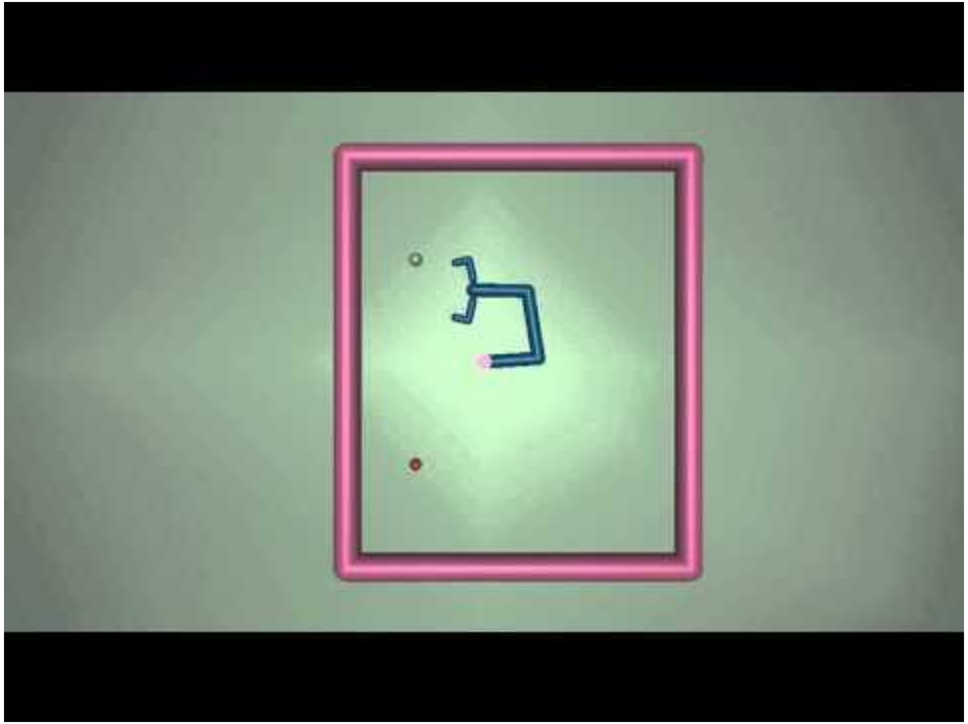
Asynchronous: CPU (16 cores)

ATARI 2600

Method	Training Time	Mean	Median
DQN	8 days on GPU	121.9%	47.5%
Gorila	4 days, 100 machines	215.2%	71.3%
D-DQN	8 days on GPU	332.9%	110.9%
Dueling D-DQN	8 days on GPU	343.8%	117.1%
Prioritized DQN	8 days on GPU	463.6%	127.6%
A3C, FF	1 day on CPU	344.1%	68.2%
A3C, FF	4 days on CPU	496.8%	116.6%
A3C, LSTM	4 days on CPU	623.0%	112.6%

Table 1. Mean and median human-normalized scores on 57 Atari games using the human starts evaluation metric. Supplementary Table SS3 shows the raw scores for all games.

MuJoCo



REFERENCES AND CREDITS

Illustrations from Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley http://ai.berkeley.edu

REFERENCES AND CREDITS

Bellemare, Marc G, Naddaf, Yavar, Veness, Joel, and Bowling, Michael. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 2012.

Degris, Thomas, Pilarski, Patrick M, and Sutton, Richard S. Model-free reinforcement learning with continuous action in practice. In *American Control Conference (ACC), 2012*, pp. 2177–2182. IEEE, 2012.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*. 2013.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A., Veness, Joel, Bellemare, Marc G., Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K., Ostrovski, Georg, Petersen, Stig, Beattie, Charles, Sadik, Amir, Antonoglou, Ioannis, King, Helen, Kumaran, Dharshan, Wierstra, Daan, Legg, Shane, and Hassabis, Demis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015. URL http://dx.doi.org/10.1038/nature14236.

REFERENCES AND CREDITS

Riedmiller, Martin. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *Machine Learning: ECML 2005*, pp. 317–328. Springer Berlin Heidelberg, 2005.

Schulman, John, Levine, Sergey, Moritz, Philipp, Jordan, Michael I, and Abbeel, Pieter. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, 2015a.

Sutton, R. and Barto, A. *Reinforcement Learning: an Introduction*. MIT Press, 1998.

Todorov, E. *MuJoCo: Modeling, Simulation and Visualization of Multi-Joint Dynamics with Contact (ed 1.0)*. Roboti Publishing, 2015.

Williams, R.J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.

QUESTIONS

4