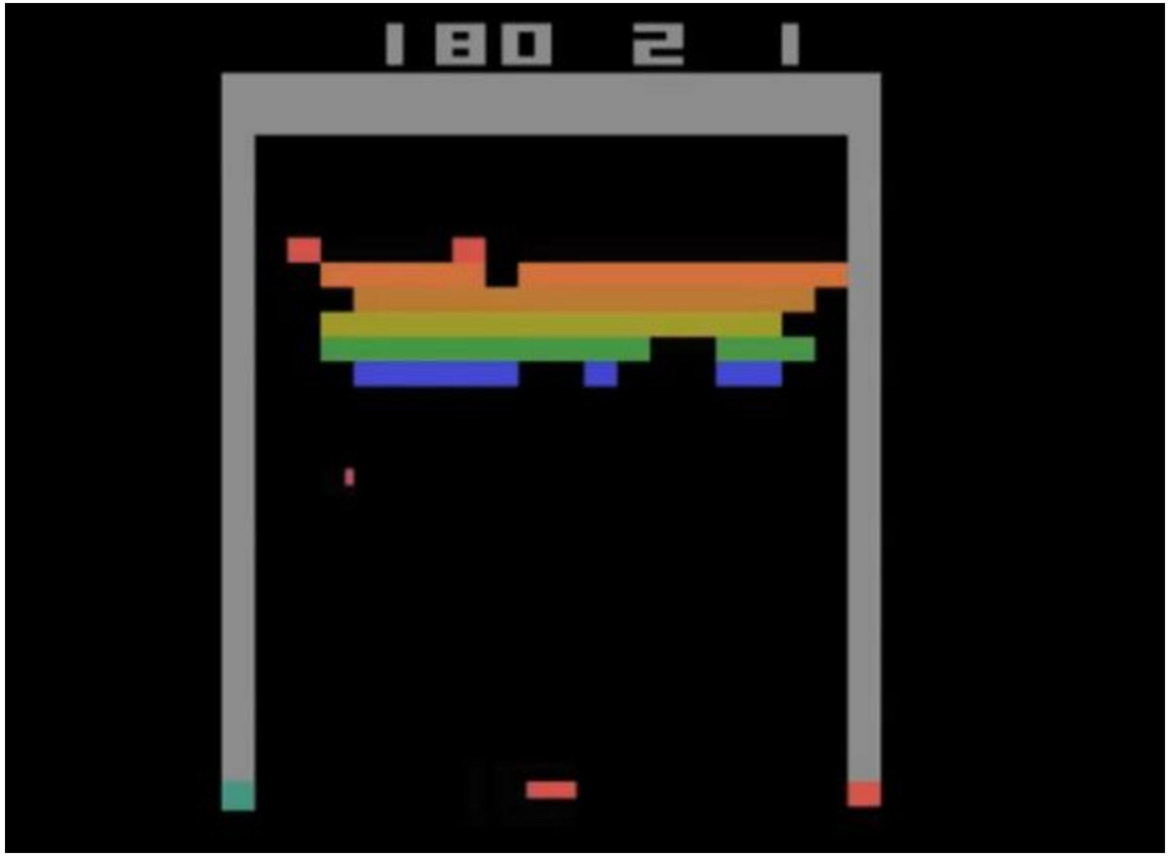
Asynchronous Methods for Deep Reinforcement

Mnih, Badia, Mirza, Graves, Lillicrap, Harley, Silver, Kavukcuoglu

Topic: Actor Critic methods

Presenter: Adelin Travers

Motivation



Learn from raw pixels, not states

Motivation

- Experience replay
  - Data from previous experiences stored in dedicated memory
- At each step:
  - Can batch data
  - Can sample randomly

=> Augments stability

  - reducing non-stationarity
  - decorelates updates

Problem

- Only off policy learning
  - Data generated from a previous policy.
- High memory usage
- High computational cost per interaction with the environment

Previous approaches based on compute parallelization:

- Specialized hardware such as GPU
- Massively distributed architectures

Outline

- Contributions
- Background
- Algorithms
- Experimental results
- Discussion
- Limitations and open issues

Contributions

- Investigate alternatives to replay memory
- Previous work parallelized agents and shared replay memory
- Propose to parallelize the learning experience
- Duplicate both the agents and environments
- Learning is shared among the agents but experience is not
  - Obtain a more stationary process and speed up exploration
- Demonstrate deep RL for value-, policy-based methods both On- and off-policy
- Divide by 2 the state of the art training time while on a single server's 16 CPUs

Outline

- Contributions
- Background
- Algorithms
- Experimental results
- Discussion
- Limitations and open issues

Background

One-step Q-learning

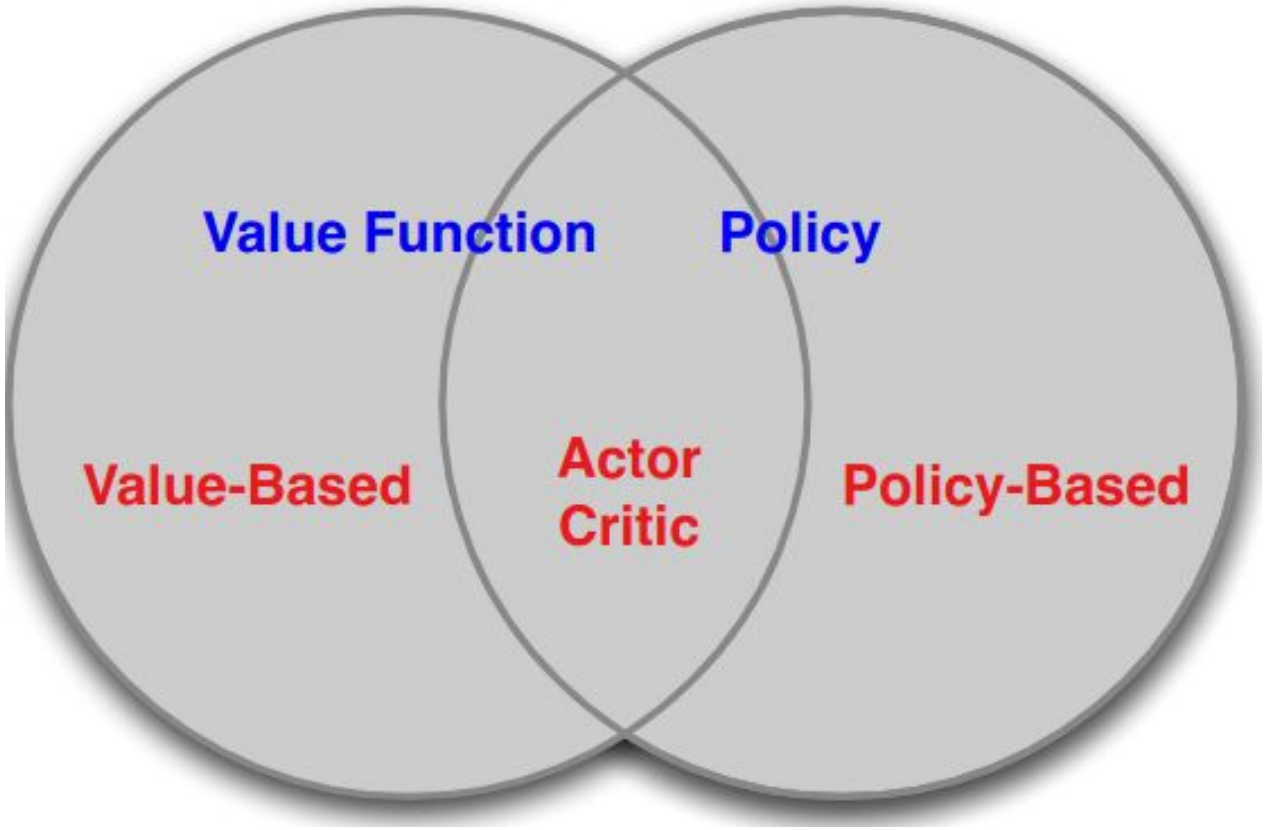
$$L_i(\theta_i) = \mathbb{E}\left(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\right)^2$$

N-step Q-learning

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

Background

- Actor-critic
- Reduce Monte-carlo policy gradients variance
- Combine Value based methods and policy gradients



[David Silver, RL Lectures]

Background

- Parameterize the Q-value function

$$Q_w(s, a) \approx Q^{\pi_\theta}(s, a)$$

- *Approximate policy gradient*

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(s, a) \; Q_w(s, a)\right]$$
$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s, a) \; Q_w(s, a)$$

[David Silver, RL Lectures]

Background

- Critic can be a baseline
- Can take the value function

- Policy gradient on the advantage function

$$A^{\pi_\theta}(s, a) = Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s)$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s, a)\ A^{\pi_\theta}(s, a)\right]$$

[David Silver, RL Lectures]

Outline

- Contributions
- Background
- Algorithms
- Experimental results
- Discussion
- Limitations and open issues

Algorithm: one-step Q-learning

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: one-step Q-learning

repeat
  Take action a with $\boxed{\epsilon\text{-greedy policy based on } Q(s,a;\theta)}$
  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: one-step Q-learning

repeat

  Take action a with ϵ-greedy policy based on $Q(s, a; \theta)$

  Receive new state s' and reward r

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

  $\boxed{\text{Accumulate gradients wrt } \theta\text{: } d\theta \leftarrow d\theta + \frac{\partial (y - Q(s,a;\theta))^2}{\partial \theta}}$

  $s = s'$

  $T \leftarrow T + 1$ 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: one-step Q-learning

repeat

Take action a with ϵ-greedy policy based on $Q(s, a; \theta)$

Receive new state s' and reward r

$$\boxed{y = \begin{cases} r \\ r + \gamma \max_{a'} Q(s', a'; \theta^-) \end{cases}} \quad \begin{matrix} \text{for terminal } s' \\ \text{for non-terminal } s' \end{matrix}$$

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: A3C

repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
  $t_{start} = t$
  Get state s_t
  repeat
    Perform a_t according to policy $\pi(a_t|s_t;\theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until terminal s_t **or** $t - t_{start} == t_{max}$

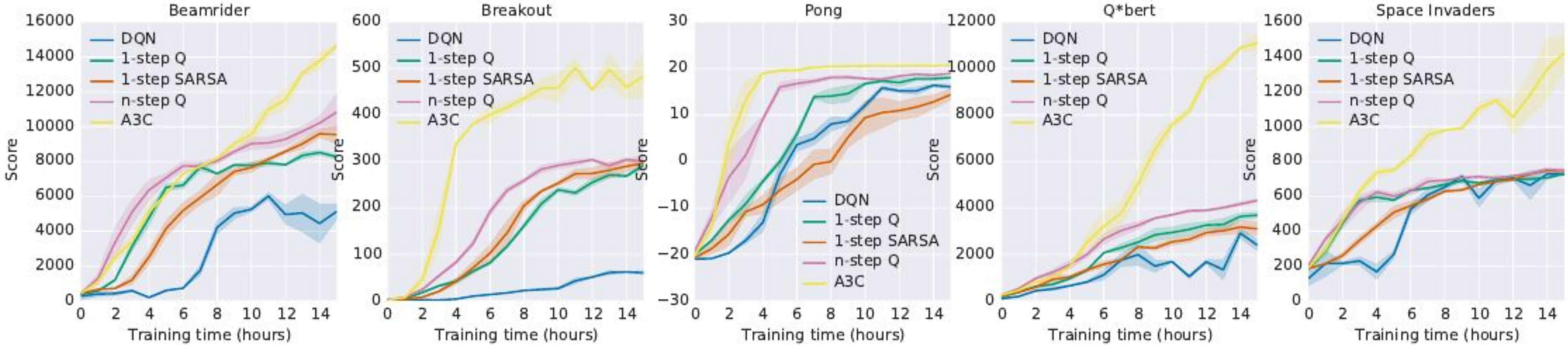
$$R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$$

  for $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
    Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial\left(R - V(s_i;\theta'_v)\right)^2 / \partial\theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

Outline

- Contributions
- Background
- Algorithms
- Experimental results
- Discussion
- Limitations and open issues

Experimental Results



All variants outperform DQN in training speed and performance

Experimental Results

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%

2x speedup on CPU

Outline

- Contributions
- Background
- Algorithms
- Experimental results
- Discussion
- Limitations and open issues

Discussion

	Number of threads				
Method	1	2	4	8	16
1-step Q	1.0	**3.0**	**6.3**	**13.3**	**24.1**
1-step SARSA	1.0	**2.8**	**5.9**	**13.1**	**22.1**
n-step Q	1.0	**2.7**	**5.9**	**10.7**	**17.2**
A3C	1.0	2.1	3.7	6.9	12.5

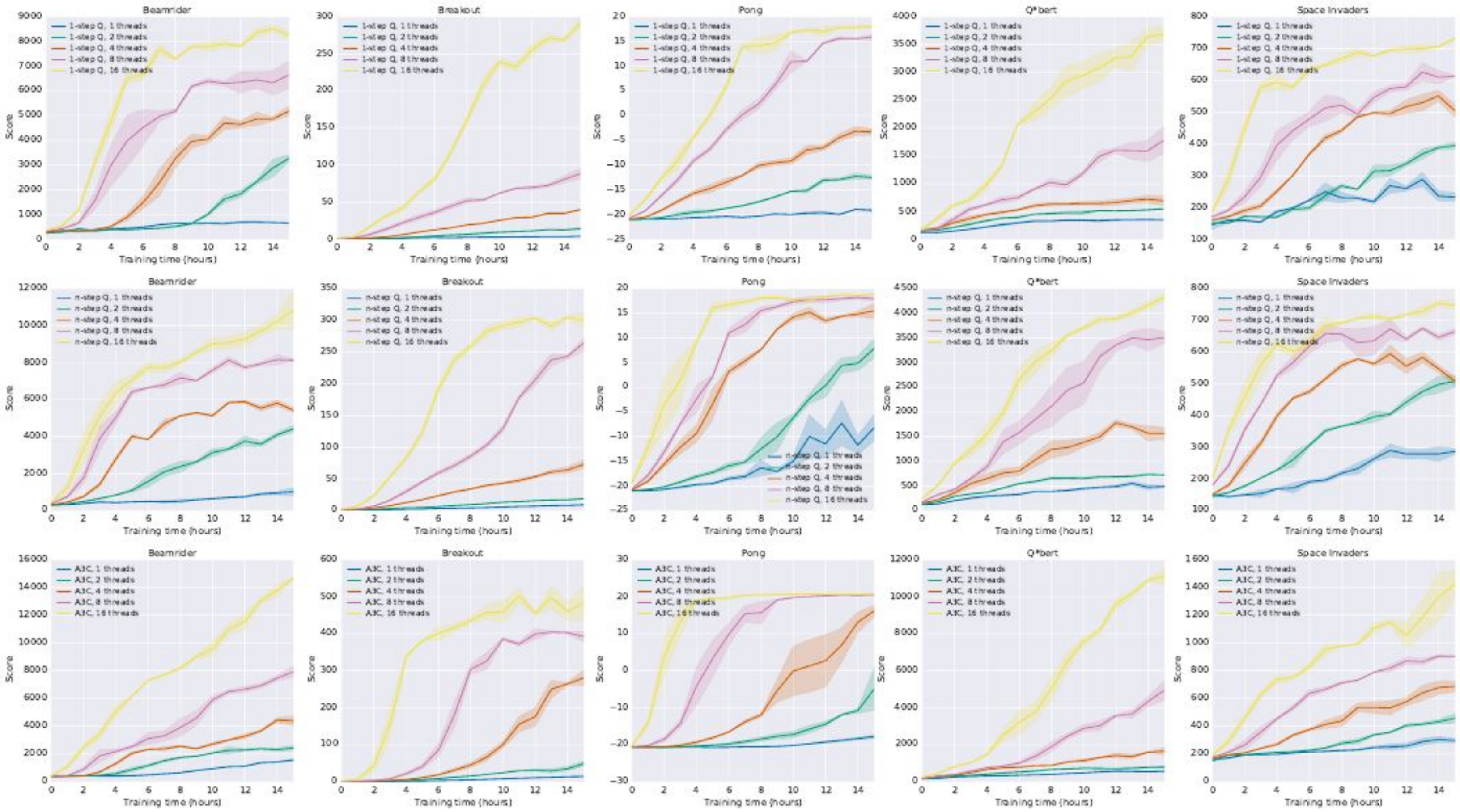
Superlinear mean thread improvement for all methods but A3C

Discussion

1-step Q

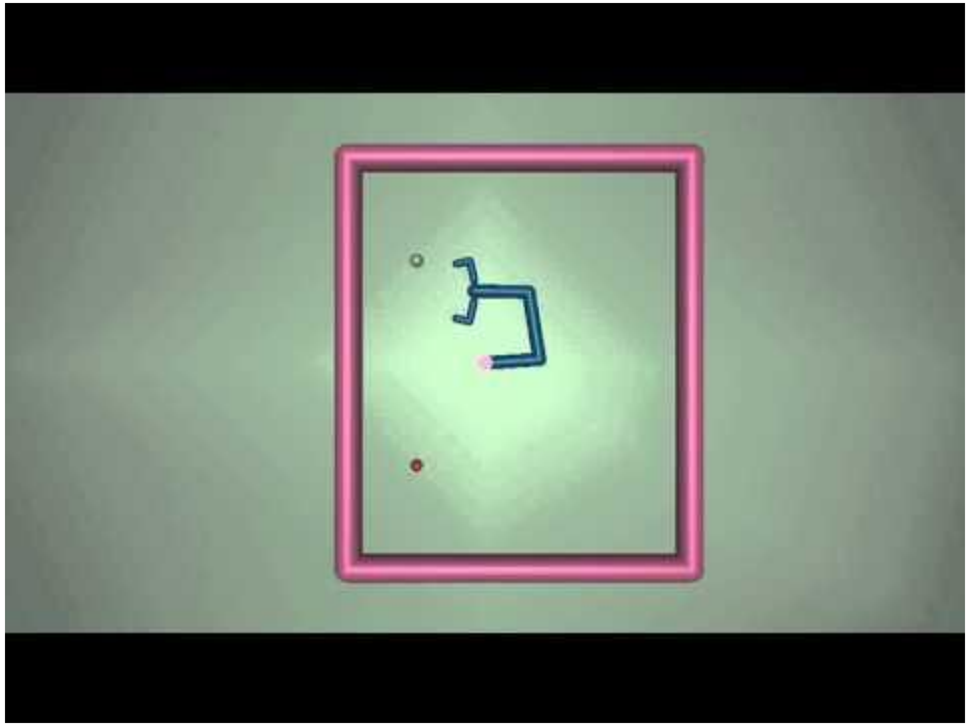
N-step Q

A3C



Thread speedup is dependent on the games

Discussion



Capable of handling discrete and continuous state spaces

Outline

- Contributions
- Background
- Algorithms
- Experimental results
- Discussion
- Limitations and open issues

Limitations and Open Issues

- Performance very dependent on the game
- If interactions with the environment are expensive, limited success
  - Combine with experience replay?
- Forward view only
  - Backward view is more common in RL
- Better ways to estimate the advantage function
  - Generalized advantage estimation

Contributions (recap)

- Alternatives to replay memory
- Previous work parallelized replay memory/computation
- Parallelize the learning experience
- Duplicate both the agents and environments
- Learning is shared among the agents but experience is not
  - Obtain a more stationary process and speed up exploration
- Demonstrate deep RL for value-, policy-based methods both On- and off-policy
- Divide by 2 the state of the art training time while on a single server’s 16 CPUs