Asynchronous Methods for Deep Reinforcement Learning（深度强化学习的异步方法）

作者：Volodymyr Mnih，Adrià Puigdomènech Badia，Mehdi Mirza，Alex Graves，Tim Harley，Timothy P. Lillicrap，David Silver，Koray Kavukcuoglu

单位：Google DeepMind，Montreal Institute for Learning Algorithms （MILA）, University of Montreal

会议：International Conference on Machine Learning（ICML 2016）

论文地址：https://arxiv.org/abs/1602.01783

论文贡献：提出一种异步梯度优化框架，在各类型强化学习算法上都有较好效果，其中效果最好的就是异步优势演员-评论员（Asynchronous advantage actor-critic，A3C）。

- **Motivation(Why)**：神经网络与强化学习算法相结合可以让算法更有效，但稳定性较差。通常的做法是使用经验回放来去除数据的相关性，让训练更加稳定。但这个方法有两个缺点：一是需要较多的内存和算力，二是只适用于异策略（off-policy）强化学习算法。
- **Main Idea(What)**：本文提出使用异步并行的方法来代替经验回放技术，这种方法不仅可以去除数据相关性，所需资源较少，并且可以应用于多种类型强化学习方法。

Asynchronous RL Framework（异步强化学习架构）

本文将异步架构用于四种强化学习算法上：一步Sarsa，一步Q学习，n 步Q学习以及优势演员-评论员（advantage actor-critic，A2C）。这里可以看到此架构对于同/异策略、基于价值/策略的强化学习算法均适用。

异步框架有两大特点：

1. 只使用一台带有多核CPU的机器，不需要GPU。这种方式可以减少机器间的通信成本，并且可以用Hogwild! 的方式进行训练。
2. 多个行动者（actor/worker）并行可以采取不同的策略与环境互动让数据关联性减弱从而不需要经验回放技术，这使得框架可以用于同策略方法，并且训练时间与行动者数量近似线性关系。

先来聊聊Hogwild!，这是框架的核心，不清楚的可参考知乎讲解，下图来源此文：

（2）Hogwild（Lock free）

Hogwild方法[21]可以看作异步SGD的一个变形。The master是共享内存系统，对异步SGD来说，如果在the master与第i个worker交互期间，第j个worker也传来了子梯度，那么在W<—W - ηΔWi完成之前W<—W – ηΔWj将被阻塞（i，j∈{1，2，……，p}）。这意味着为了避免在the master机器共享内存系统内的权值更新冲突，这里使用了锁机制。锁机制确保master中同一时间只有一个子梯度进行权值更新。而Hogwild方法移除了锁机制，允许master同一时间处理多个子梯度。文章[21]证明了Hogwild的lock free方法的收敛性。

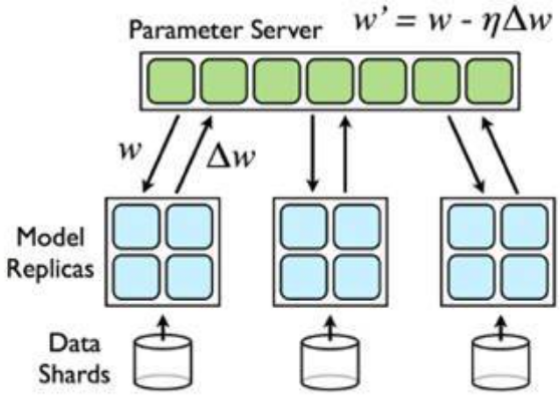

Figure 5: This figure [5] is an illustration of Parameter Server. Data is partitioned to workers. Each worker computes a gradient and sends it to server for updating the weight. The updated model is copied back to wokers

这里举个栗子：假设网络有100个参数，worker i传来的梯度更新了40个参数后worker j传来的就开始更新了，当i更新完后前面的一些参数被j被覆盖掉了，但不要紧，i更新完了照样把当前更新后参数同步给i，这就是异步的意思。

接下来介绍四种异步算法：

Asynchronous one-step Q-learning（异步一步Q学习）

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 $T > T_{max}$

- θ 和 θ^- 是共享的，任何线程可以修改
- 梯度累积是一个增加batch size的技巧，并且可以减少覆盖其他线程更新的机会（更新频率降低了）
- I_{target}是同步两网络参数的间隔
- $I_{AsyncUpdate}$是线程更新共享参数θ的间隔

Asynchronous one-step Sarsa（异步一步Sarsa）

与上面的异步一步Q学习相比，只有目标价值（target value）变为了$r + \gamma Q(s', a'; \theta')$，其余部分相同。

Asynchronous n-step Q-learning（异步n步Q学习）

Algorithm S2 Asynchronous n-step Q-learning - pseudocode for each actor-learner thread.

// Assume global shared parameter vector θ.
// Assume global shared target parameter vector θ^-.
// Assume global shared counter $T = 0$.
Initialize thread step counter $t \leftarrow 1$
Initialize target network parameters $\theta^- \leftarrow \theta$
Initialize thread-specific parameters $\theta' = \theta$
Initialize network gradients $d\theta \leftarrow 0$
repeat
  Clear gradients $d\theta \leftarrow 0$
  Synchronize thread-specific parameters $\theta' = \theta$
  $t_{start} = t$
  Get state s_t
  repeat
    Take action a_t according to the ϵ-greedy policy based on $Q(s_t, a; \theta')$
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ \max_a Q(s_t, a; \theta^-) & \text{for non-terminal } s_t \end{cases}$
  for $i \in \{t-1, \ldots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \frac{\partial \left(R - Q(s_i, a_i; \theta')\right)^2}{\partial \theta'}$
  end for
  Perform asynchronous update of θ using $d\theta$.
  if $T \mod I_{target} == 0$ **then**
    $\theta^- \leftarrow \theta$
  end if
until $T > T_{max}$

- 一步法的缺点是 r 只直接影响导致r产生的 $Q(s,a)$ 值，其他的Q值被间接影响。n步法得到的r可以影响前面n步的Q值，这使得通过r更新Q值更加有效。后文实验有关于两者的比较。
- 相较与一步法多了个线程专属参数θ'，这个参数在选取动作以及计算梯度的时候会使用，不受其他线程的影响，可以让训练更稳定。

Asynchronous advantage actor-critic（A3C)

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$
// Assume thread-specific parameter vectors θ' and θ'_v
Initialize thread step counter $t \leftarrow 1$
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 $T > T_{max}$

- 共享了演员，评论家两网络参数，每个线程也有自己专属的，其他的与n步Q学习基本一样

实验

Atari游戏

在5个Atari游戏上比较算法训练速度：

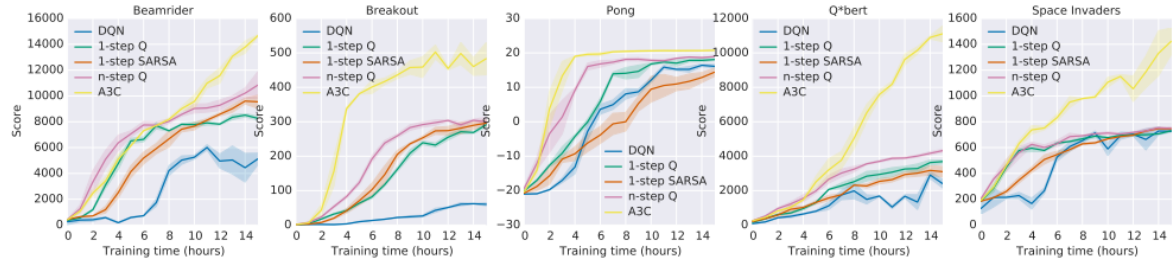

Figure 1. Learning speed comparison for DQN and the new asynchronous algorithms on five Atari 2600 games. DQN was trained on a single Nvidia K40 GPU while the asynchronous methods were trained using 16 CPU cores. The plots are averaged over 5 runs. In the case of DQN the runs were for different seeds with fixed hyperparameters. For asynchronous methods we average over the best 5 models from 50 experiments with learning rates sampled from $LogUniform(10^{-4}, 10^{-2})$ and all other hyperparameters fixed.

- DQN用K40GPU，异步方法用16核的CPU，可以看到异步优于DQN，并且n步方法是优于一步方法的，A3C明显优于其他算法。

在57个Atari游戏上比较算法性能：

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.

- 把A3C与当时在Atari游戏上最好的一些算法进行比较，可以看到A3C训练4天的性能强于其他算法训练8天的水平。

TORCS Car Racing Simulator（TORCS赛车模拟器）

在比Atari游戏更难的赛车环境里检验异步算法性能：

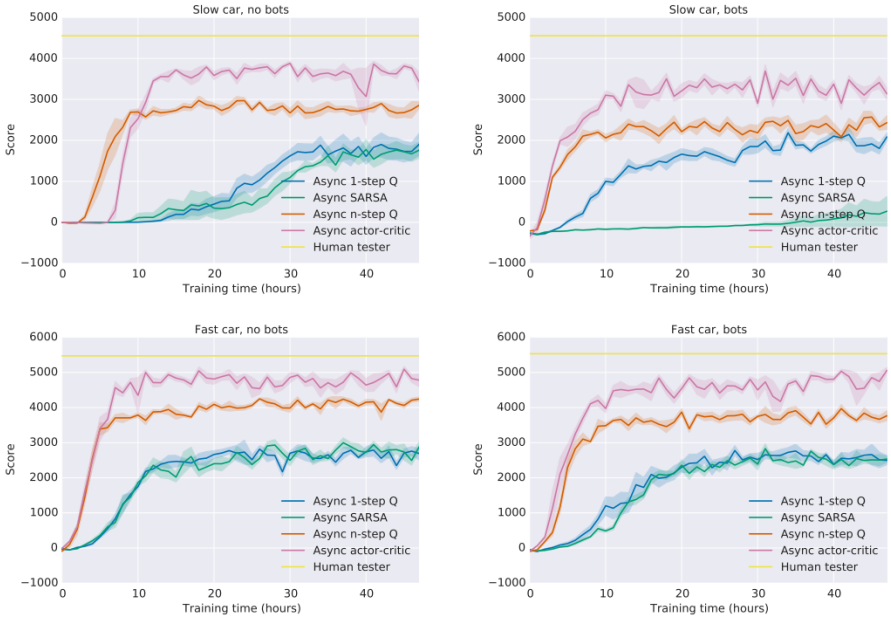

Figure S6. Comparison of algorithms on the TORCS car racing simulator. Four different configurations of car speed and opponent presence or absence are shown. In each plot, all four algorithms (one-step Q, one-step Sarsa, n-step Q and Advantage Actor-Critic) are compared on score vs training time in wall clock hours. Multi-step algorithms achieve better policies much faster than one-step algorithms on all four levels. The curves show averages over the 5 best runs from 50 experiments with learning rates sampled from $LogUniform(10^{-4}, 10^{-2})$ and all other hyperparameters fixed.

- 可以看到A3C也是明显优于其他算法。

Continuous Action Control Using the MuJoCo Physics Simulator（使用MuJoCo物理模拟器进行连续动作控制）

在Mujoco环境里检验A3C对于连续型动作状态任务的性能：

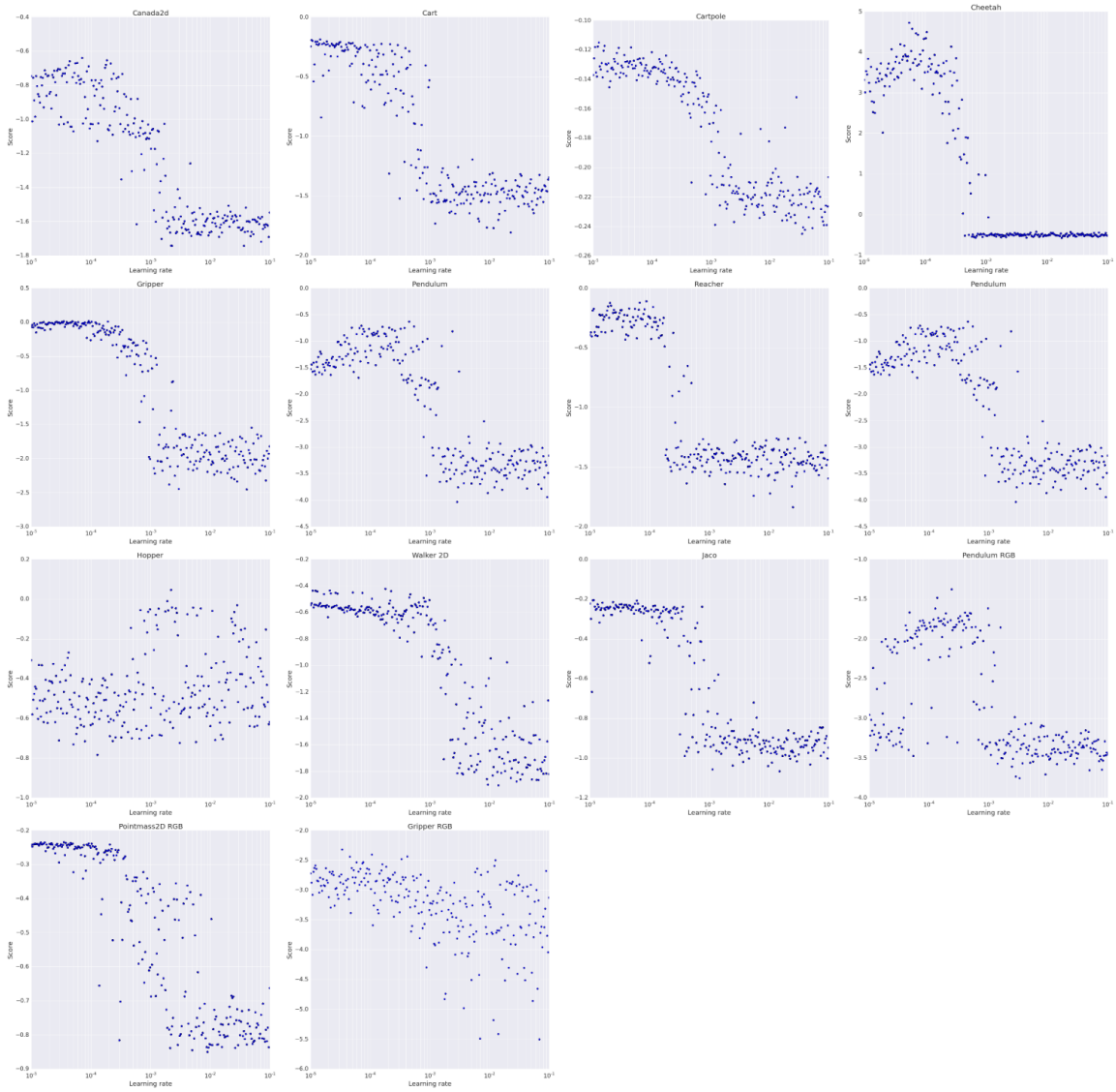

Figure S7. Performance for the Mujoco continuous action domains. Scatter plot of the best score obtained against learning rates sampled from $LogUniform(10^{-5}, 10^{-1})$. For nearly all of the tasks there is a wide range of learning rates that lead to good performance on the task.

- 在绝大多数任务上A3C在学习率（learning rate）在大范围变动下都有很好的表现。

Labyrinth（迷宫）

这是一个3D迷宫环境，每一轮（episode）都会随机生成一个迷宫，迷宫里有苹果和门，智能体找到一个苹果得1分，找到门得10分。找到门后智能体会被随机放到迷宫任意一个地方，并且之前被找到的苹果会重新生成。一轮持续60秒，结束后新的一轮重新开始。

这个环境需要智能体找到一种通用的策略，因为每一轮迷宫都不一样，这对智能体挑战较大，但作者训练A3C后发现此算法能达到很好的效果。

Scalability and Data Efficiency（可扩展性和数据效率）

在Atari游戏上检验算法的训练时间和数据效率随着线程数变化会如何改变：

	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

Table 2. The average training speedup for each method and number of threads averaged over seven Atari games. To compute the training speed-up on a single game we measured the time to required reach a fixed reference score using each method and number of threads. The speedup from using n threads on a game was defined as the time required to reach a fixed reference score using one thread divided the time required to reach the reference score using n threads. The table shows the speedups averaged over seven Atari games (Beamrider, Breakout, Enduro, Pong, Q*bert, Seaquest, and Space Invaders).

- 作者提出用一个线程达到固定分数的时间除以n个线程达到分数的时间，来衡量算法通过增加线程的提速程度。结果表明训练速度会随着线程数的增加而提升，这证明异步框架可以很好地利用训练资源。

Robustness and Stability（稳健性和稳定性）

在5个Atari游戏中检验异步算法的稳健性和稳定性：

A3C:

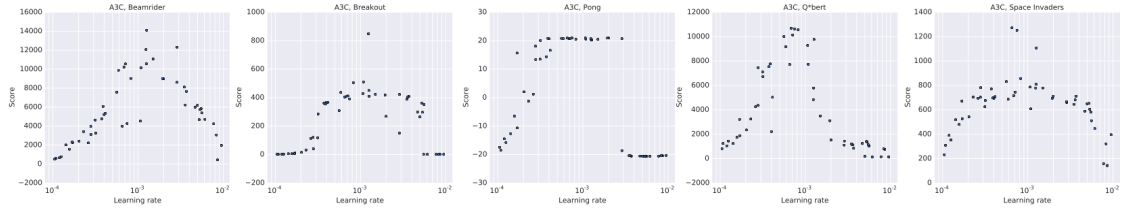

Figure 2. Scatter plots of scores obtained by asynchronous advantage actor-critic on five games (Beamrider, Breakout, Pong, Q*bert, Space Invaders) for 50 different learning rates and random initializations. On each game, there is a wide range of learning rates for which all random initializations acheive good scores. This shows that A3C is quite robust to learning rates and initial random weights.

其他的三种异步算法：

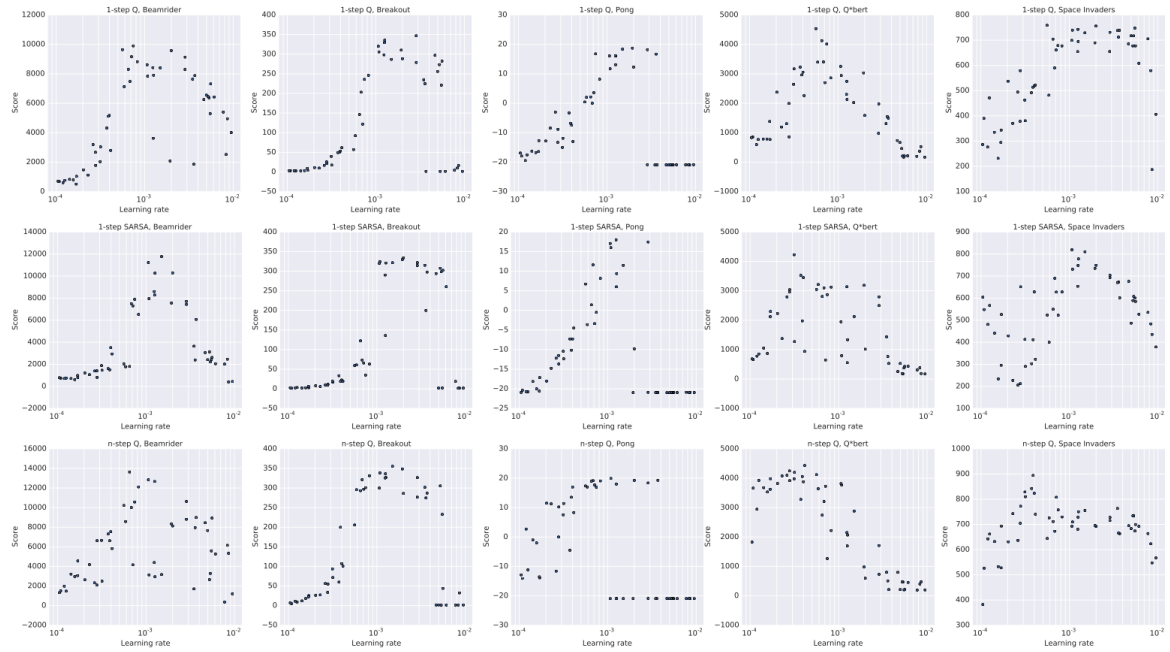

Figure S11. Scatter plots of scores obtained by one-step Q, one-step Sarsa, and n-step Q on five games (Beamrider, Breakout, Pong, Q*bert, Space Invaders) for 50 different learning rates and random initializations. All algorithms exhibit some level of robustness to the choice of learning rate.

- 结果表明所有的算法在学习率在大范围变化时都有较好的稳健性和稳定性。

总结

异步框架有几大优点：

- 效果好的同时所需资源较少
- 可以应用于同策略、异策略等多种强化学习算法
- 数据利用率高，稳定性较好
- 可以广泛结合其他算法的优点提升性能

参考资料

- Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent
- 知乎讲解
- 知乎讲解

作者：汪聪

单位：武汉工程大学

研究方向：机器博弈，强化学习

联系方式：xyfcw@qq.com