Asynchronous Methods for Deep Reinforcement Learning (A3C)

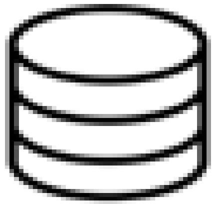
Smart Systems Lab.
YONSEI UNIVERSITY, SEOUL, KOREA

Jiwi Chong

Previous algorithms and A3C

❖ **Previous RL variants achieved promising results in complex problems, such as Atari games**
- DQN, DDQN, TRPO

❖ **However:**
- Several of these algorithms were online-based
- Use of experience replay (experience memory)
  - High computation cost



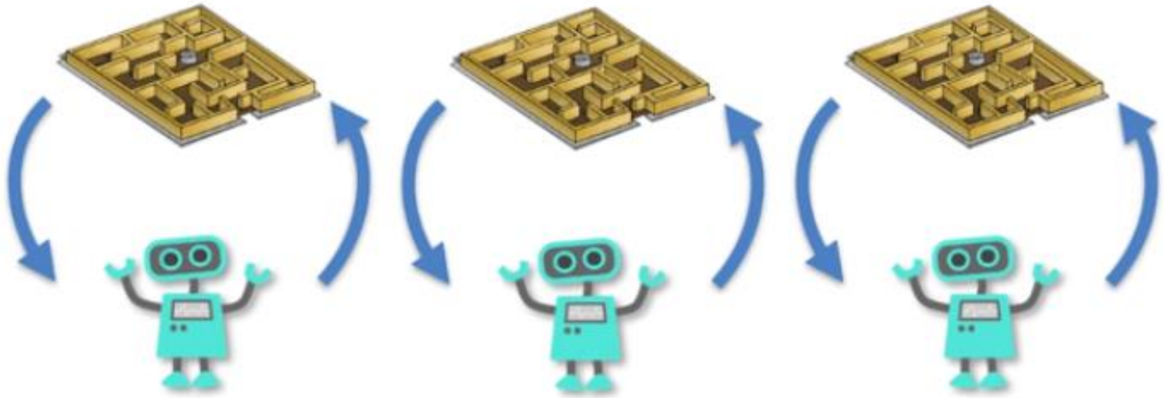
❖ **Motivation for creation of A3C:**
- Algorithm that learns effectively and efficiently without the necessity of large dataset of experiences
- Learning method without correlation of agent's data
- Allows usage of several other RL methods with deep neural networks
  - On-policy methods: SARSA, Actor-Critic
  - Off-policy methods: Q-learning

Characteristics

- Use of multiple, parallel learners

- Each learner:
  - Has its own copy of environment
  - Learns independently without waiting to see what other agents learn
  - Learns with its own variety of samples to learn the policy
  - Learns with minimal number of transition samples, reducing computation cost

- Other learners can provide information in case one learner starts to learn sub-optimally

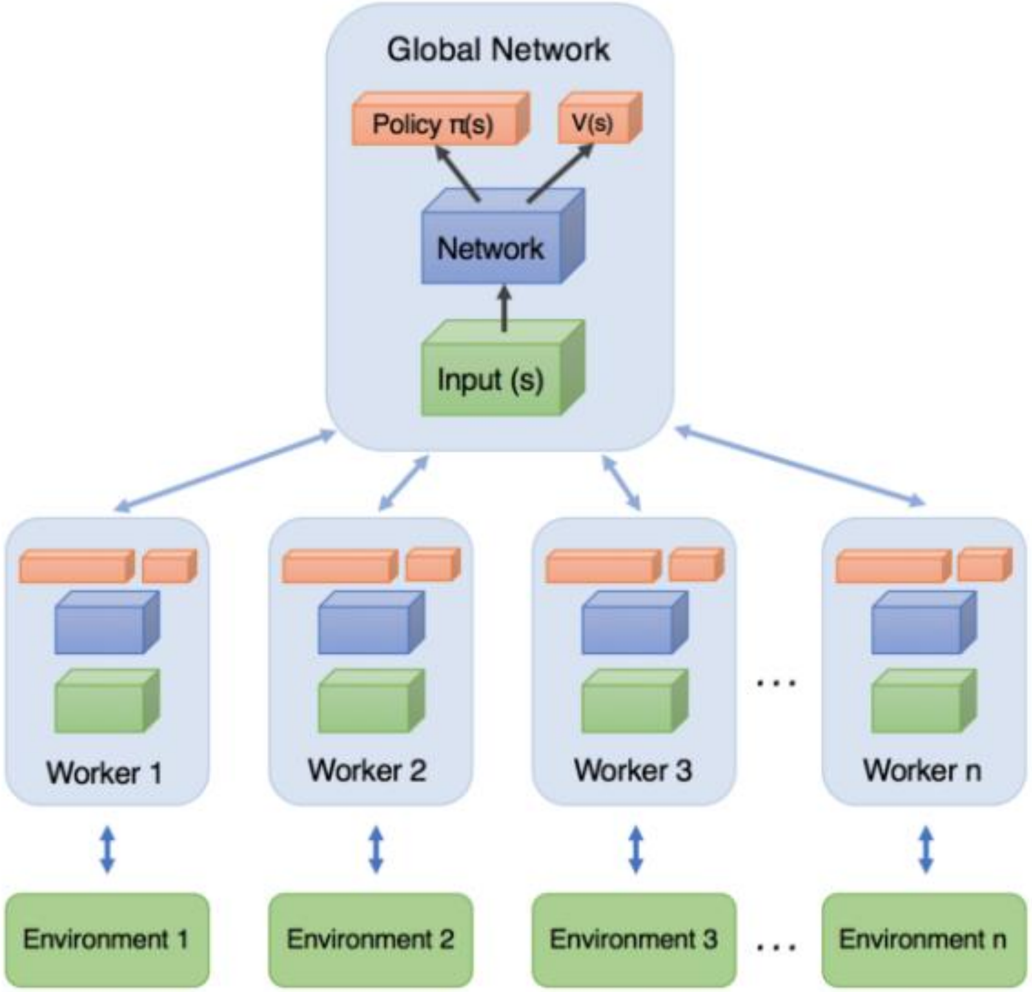


A3C

Allows the global network to have its parameters updated

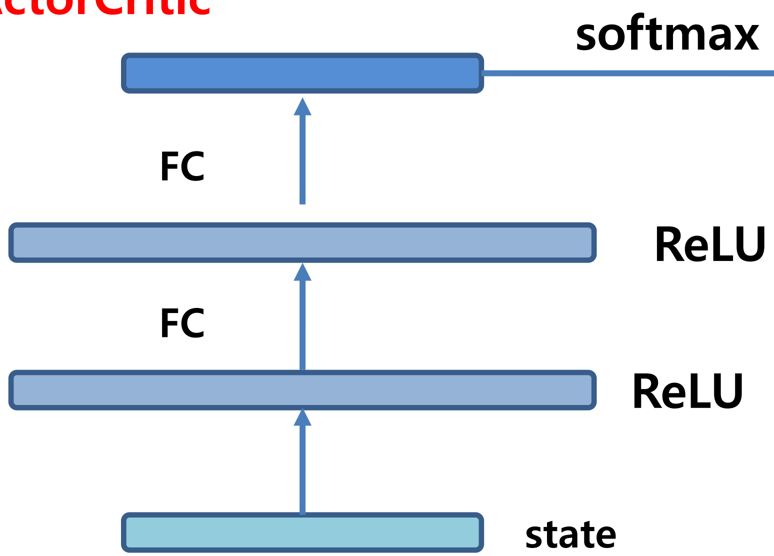
Enlarged variety of experiences from parallel workers

- Experiences coming from different parts in the environment's transitions
- Less correlation in transitions
- Different exploration policies can be tested in each worker to maximize such diversity



ActorCritic model

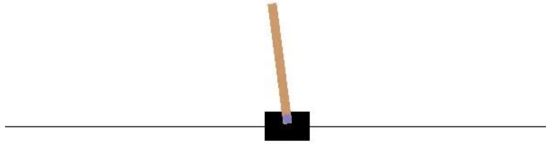
ActorCritic

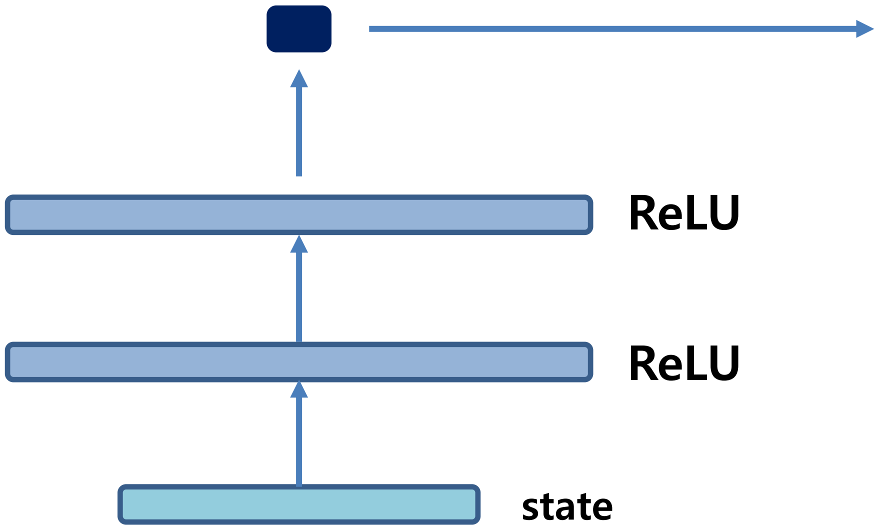


Actions as probability distribution

Action sampled from categorical distribution if it is discrete

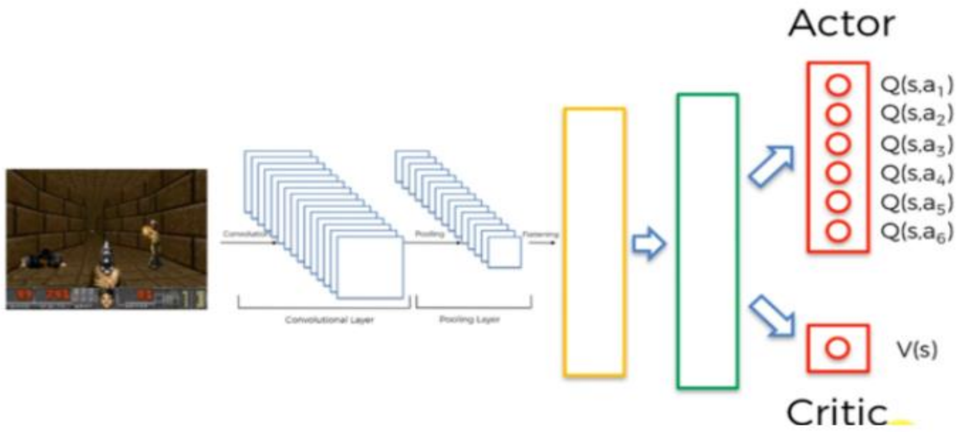
❖ CartPole Example here!





State value ($V(s_t)$)

❖ A Convolutional neural network (CNN) is typically used with one softmax output and one linear output for value function



A3C learning process in a nutshell

A3C:

- Follows n-step returns to update both the policy and value function

e.g. Q-learning

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

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

- The agent learns after every t-max actions or when a terminal state is reached $$\nabla_{\theta'} \log \pi(a_t|s_t;\theta')\underbrace{A(s_t, a_t; \theta, \theta_v)}$$

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

A3C actor-critic pseudocode

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 $\pi(a_t|s_t;\theta')$ → From probability distribution of Actor
    Receive reward r_t and new state s_{t+1}
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until terminal s_t **or** $t - t_{start} == t_{max}$
  $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$ → Take into account last state's value only
  for $i \in \{t-1, \dots, t_{start}\}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')\boxed{(R - V(s_i;\theta'_v))}$ → Actor loss
    Advantage
    Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial\left(R - V(s_i;\theta'_v)\right)^2 / \partial\theta'_v$ → Critic loss
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

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 $T > T_{max}$

The parameters of the global Network are updated with those of local networks

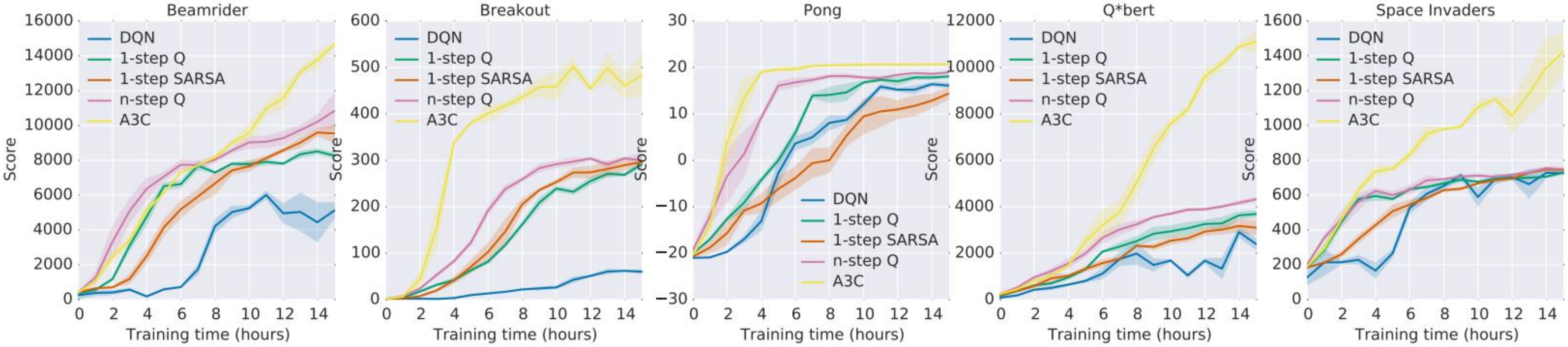
Considerable loss function component

❖ Entropy

- Can be considered to encourage exploration
- Prevent convergence to early suboptimal deterministic policies
- Particularly helpful for tasks that require hierarchical behavior

$$\nabla_{\theta'} \log \pi(a_t|s_t;\theta')(R_t - V(s_t;\theta_v)) + \boxed{\beta \nabla_{\theta'} H(\pi(s_t;\theta'))}$$

❖ Experiments on some of the Atari games



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.

Conclusion

❖ A3C:

- Has stabilizing effect on the learning process
- For problems that involve expensive interaction with environment (e.g. TORC)
  - Experience replay could be used in A3C
- A3C could be further improved by using different methods of estimating advantage functions