dongguk UNIVERSITY

심층 강화 학습

Final Project Report

2023126703 박준서

<목차>

I. Paper Review

본 프로젝트에서 탐구할 reinforcement learning paper는 "Asynchronous Methods for Deep Reinforcement Learning"이다. 2016년, International conference on machine learning (ICML)에 발표된 논문이다. 2023년 12월 5일 기준으로 9949회의 인용이 존재한다.

1. Summarization

본 논문이 제시하고자 하는 것은 "Asynchronous Algorithm Applicable to All Reinforcement Learning"으로 볼 수 있다. 즉, 어떤 RL algorithm에도 일반적으로 잘 작동하면서 효율적인 비동기적 병렬 학습 방식을 제시하는 논문이다.

2. Introduction

1) Deep neural network는 RL이 효과적으로 수행되도록 풍부한 표현을 제공한다. 하지만, 근본적으로 online agent가 관찰한 data sequence는 non-stationary이며 time-correlation이 강하기 때문에, deep neural network를 사용한 online RL algorithm은 unstable하다.
2) 이 문제를 해결하고자 agent의 data를 experience replay buffer에 저장하여, non-stationarity를 줄이고 time-correlation을 끊었다. 이런 방식은 data를 mini-batch 단위로 묶을 수 있고 randomly sampling을 할 수 있지만 동시에 off-policy RL algorithm으로 제한시키는 단점이 있다.
3) 또한, experience replay buffer의 다음의 추가적인 단점이 존재한다.
   (a) 실제 상호 작용에 비해, 더 많은 메모리와 계산을 요구한다.
   (b) Old policy에서 생성된 data (target)로 update해야 한다.

4) 본 논문에서는 non-stationarity를 줄이고 time-correlation을 끊기 위해, experience replay buffer 대신에 병렬 처리 기반의 CPU 연산을 통한 비동기적 학습을 제시한다.

(a) 병렬성은 agents의 data를 decorrelate 할 수 있다. 즉 stationary process.

(b) On-policy (e.g., Sarsa, n-step method, actor-critic) 뿐만 아니라, Off-policy (e.g., Q-learning) 에도 견고하게 적용된다.

3. Related Work

1) The General Reinforcement Learning Architecture (Gorila), 2015

- 분산 환경에서 agent를 비동기적으로 훈련했다. Replay memory, learner를 각각의 agent마다 복제해서 학습했다. Gradient값을 asynchronous하게 중앙 서버로 보내서 모델을 업데이트하고 이를 주기적으로 하위 learner에 보내주는 방식이다.
- Gorila에서 사용한 별도의 기계 및 매개 변수 서버 대신, 단일 기계의 여러 CPU thread를 사용하는 것이 본 논문과 다른 점으로 볼 수 있다.

2) In Recent Advances in Reinforcement Learning, 2011

- 선형 함수를 근사를 사용하여 batch reinforcement learning을 병렬화했다.
- Collection of experience 나 학습 안정화를 위해서 병렬화를 사용하지는 않았다. 이 점이 본 논문과 다른 점이다.

3) Etc. (~2008)

4. Method

4-1. Asynchronous RL Framework

1) Asynchronous update를 사용하여 개념적으로 간단하고 가벼운 framework를 제안한다.
2) 서로 다른 actor-learner가 환경의 서로 다른 부분을 탐색할 가능성이 높기 때문에, 다양성이 극대화된다.
3) 병렬 actor-learner를 사용하는 것은 다음의 장점을 얻는다.
   (a) 병렬 actor-learner 수에 비례해 선형인 훈련 시간 단축을 얻을 수 있다.
   (b) Replay memory에 의존하지 않으므로 Sarsa & actor-critic 과 같은 on-policy algorithm에도 안정적으로 사용할 수 있다.

4-2. Pseudocode

본 section에선, asynchronous method가 적용된 algorithm pseudocode를 분석한다.

1) Asynchronous one-step Q-learning

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 $T > T_{max}$

Figure 1. Asynchronous one-step Q-learning pseudocode

✓ Notation

① θ: $global\ network$

② θ^-: $fixed\ target\ network$

③ $t\ mod\ I_{AsyncUpdate} == 0$: 각 worker 가 쌓은 경험으로 global network update

④ $T\ mod\ I_{target} == 0$: fixed target network parameter를 global network parameter 로 update

✓ Local network는 없고 global network만 존재한다.

✓ DQN에서 사용했던 fixed target network를 그대로 적용한다.

2) Asynchronous advantage 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 $\pi(a_t|s_t;\theta')$
    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: $d\theta_v \leftarrow d\theta_v + \partial\,(R - V(s_i;\theta'_v))^2/\partial\theta'_v$
  end for
  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

Figure 2. Asynchronous A3C pseudocode

✓ Notation

① θ, θ_v: $global\ \ network$

② θ', θ'_v: $local\ \ network$

✓ Target value를 구할 때, 1-step ~ 5-step 까지의 cumulative reward를 사용한다.

3) Asynchronous one-step Sarsa

Asynchronous one-step Sarsa: The asynchronous one-step Sarsa algorithm is the same as asynchronous one-step Q-learning as given in Algorithm 1 except that it uses a different target value for $Q(s,a)$. The target value used by one-step Sarsa is $r + \gamma Q(s', a'; \theta^-)$ where a' is the action taken in state s' (Rummery & Niranjan, 1994; Sutton & Barto, 1998). We again use a target network and updates accumulated over multiple timesteps to stabilize learning.

Figure 3. Difference between Q-learning and Sarsa

✓ Asynchronous one-step Q-learning에서 target value만 $r + \gamma Q(s', a'; \theta^-)$ 로 바꿔주면 된다.

✓ 마찬가지로 DQN에서의 fixed target network를 적용시킨다.

5. Experimental result

1) Atari 2600 games

- Atari 2600 game 중 5개의 게임에 대해서 실험한 결과이다.
- 해당 실험결과에서 사용한 algorithm은 asynchronous method를 적용시킨 1-step Q, 1-step Sarsa, n-step Q, A3C 와 기존의 DQN이다. 즉, DQN이 비교대상이다.
- DQN을 제외한 나머지 method는 16 CPU core를 (no GPU) 사용했다.
- 모든 실험에서 4가지 비동기 방법들이 DQN 보다 학습이 빠르게 진행되며, 성공적으로 훈련시킨 것을 확인할 수 있다. 그 중에서도 A3C가 학습속도도 빠르고 성능이 가장 좋았다.

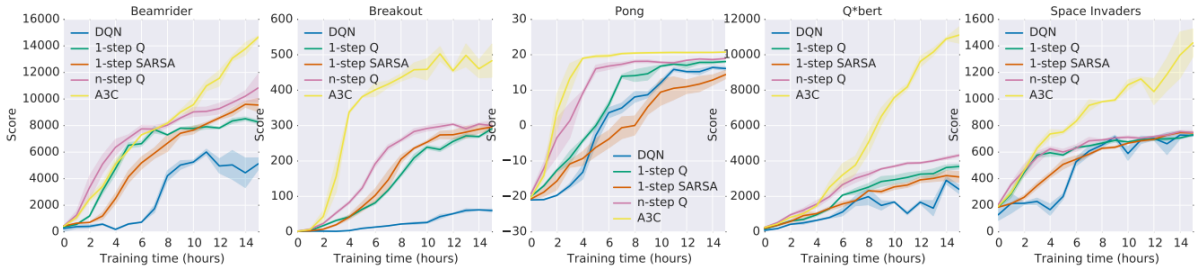

Figure 4. Experiment on 5 games out of Atari 2600 games

2) Training speed according to the number of actor-learner

- Asynchronous actor-learner 수를 증가시킴에 따라 달성된 훈련 속도 향상을 보여준다.
- $number\ of\ thread = 16$ 까지는 super linear인 것을 알 수 있다.

	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

Figure 5. Training speed according to the number of actor-learner

3) Robustness and stability

- 해당 실험에서는 비동기 알고리즘의 안정성과 견고성을 확인할 수 있다.
- 네 가지 알고리즘(a3c, 1-step Q, 1-step Sarsa, n-step Q) 각각에 대해서 50개의 다른 학습률과 무작위 초기화를 사용하여 5개의 게임에서 모델을 훈련한 결과이다.
- 각 방법과 게임 조합에 대해 좋은 점수를 얻는 학습률 범위가 일반적으로 있으며, 좋은 학습률이 있는 영역에 0점이 거의 없다. 이 결과로부터, 모든 방법이 학습률과 무작위 초기화의 선택에 대해 상당히 견고하다는 걸 알 수 있다.

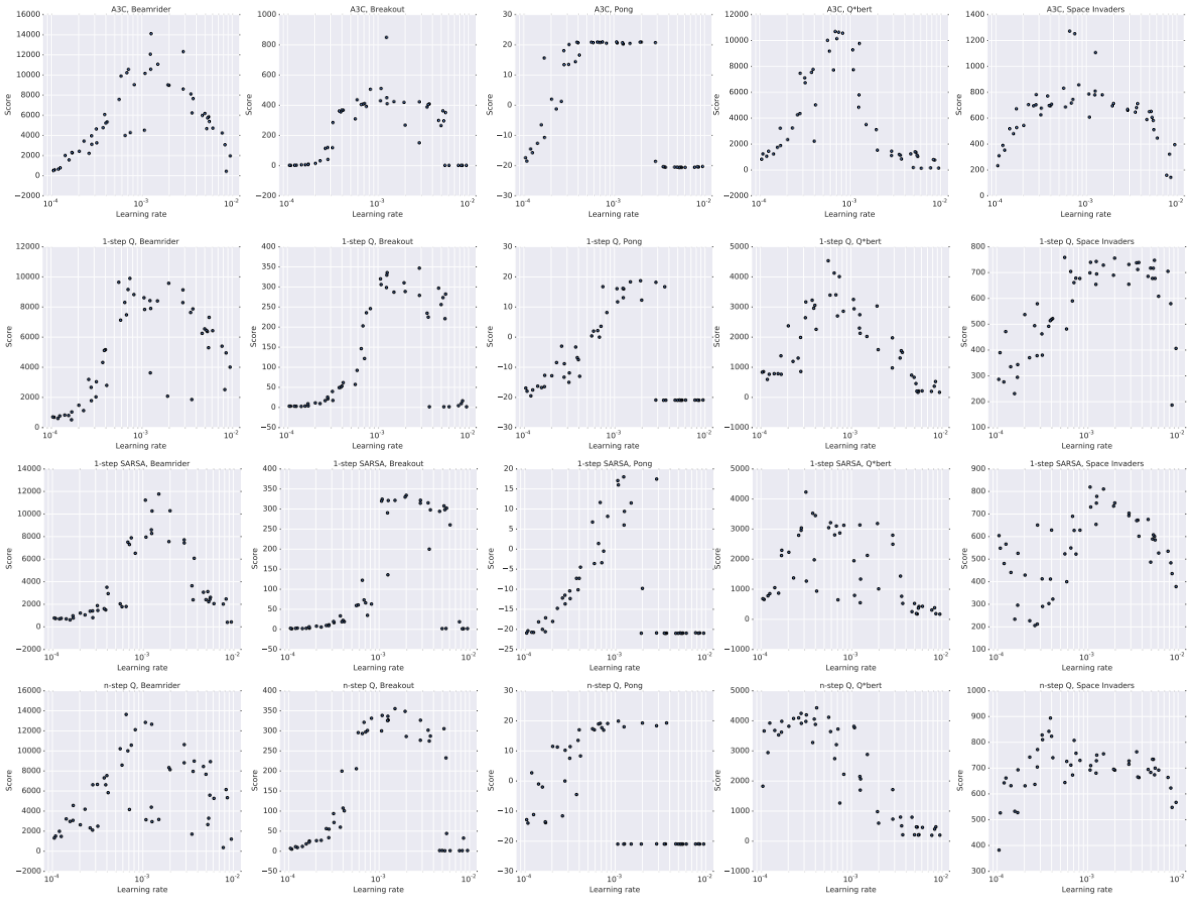

Figure 6. Experiments on the stability and robustness of asynchronous algorithms

6. Conclusion

1) RL algorithm의 scale-up method를 제시했다.

   (a) On/Off policy & value/policy based 모두 stable하다.

   (b) Experience replay 대신 병렬적 actor가 decorrelation을 가능하게 해준다.

   (c) GPU 대신 CPU thread를 사용했다.

   (d) Actor-learner 수에 따라 학습 속도는 super linear하다.

2) 이전의 deep RL은 주로 GPU와 같은 특수 하드웨어 또는 대규모 분산 architecture에 의존했다. 그러나 본 논문에서는 표준 멀티코어 CPU가 있는 단일 기기에서 실행되고, 그럼에도 GPU 기반 알고리즘보다 훨씬 빠른 시간에 더 나은 결과를 달성했다.

II. 구현 및 실험

1. 개요

논문에서 실험한 네 가지 알고리즘(a3c, 1-step Q, 1-step Sarsa, n-step Q) 중, 세 가지 알고리즘(a3c, 1-step Q, 1-step Sarsa)을 구현한다. 또한, 논문에서의 비교 대상인 DQN을 구현하여 비교한다.

2. 환경

- Reinforcement learning algorithm을 비교할 수 있는 toolkit인 OpenAI gym을 사용한다.
- *OpenAI GYM==0.22.0*
- *CartPole-v1*에 대해서 실험을 진행했다. Agent 가 취할 수 있는 action은 총 2가지이다. 최종적으로 얻을 수 있는 score는 500 점으로, 500 점을 달성하는 것을 목표로 삼았다.
-

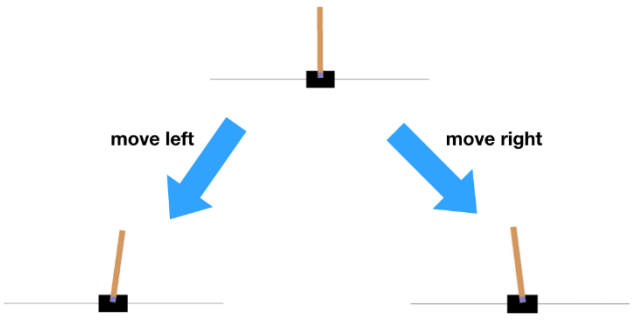

Figure 7. CartPole-v1

3. 실험

- 논문에서 소개한 네 가지 알고리즘(a3c, 1-step Q, 1-step Sarsa, n-step Q) 중, 세 가지 알고리즘(a3c, 1-step Q, 1-step Sarsa)을 구현했다. 해당 알고리즘들과 DQN 알고리즘과 비교를 진행했다.

- 각 코드에서 지정한 hyper-parameter 들은 algorithm마다 성능이 다를 수 있기에 항상 같지 않다. 최대한 다양한 시도를 통해 적절한 hyper-parameter를 자체적으로 정하여 실험을 진행했다.

- 각 algorithm들의 network는 간단한 fully connected layer로써 구성을 했다. Optimizer는 AdamW로 모두 동일하다.

1) DQN algorithm

(1) Hyper-parameter

Learning rate	0.0005
Gamma	0.98
Buffer limit	50000
Batch size	32
Epsilon	Max(0.002, 0.08 – 0.01*(epi/200))

(2) Performance

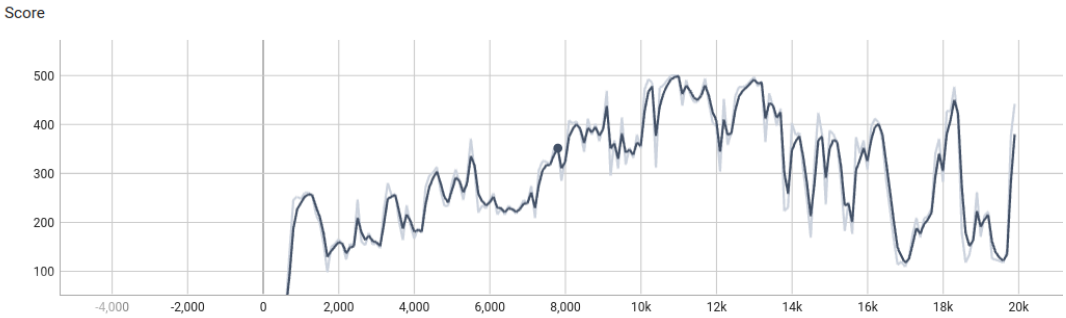

Figure 8. DQN performance

2) Asynchronous one-step Q-learning algorithm

(1) Hyper-parameter

16개의 cpu를 사용해서 multi-processing을 진행했다. Agent마다 Exploration과 exploitation을 다양하게 할 수 있도록 epsilon값을 random하게 설정했다. 5번마다의 경험을 바탕으로 network를 update한다. Target network는 100 epi마다 update한다.

Learning rate	0.0002
Gamma	0.99
Multi-process	16
Update interval	5
Target update interval	100
Epsilon	Exploration = random.uniform(0.05, 0.15) Epsilon = max(0.002, exploration – 0.01*(epi/3000))

(2) Performance

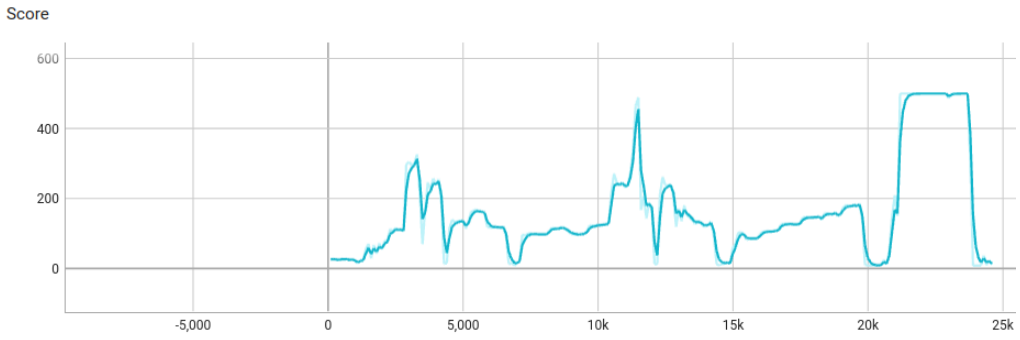

Figure 9. Q-learning performance

3) Asynchronous one-step Sarsa algorithm

(1) Hyper-parameter

Asynchronous one-step Q-learning algorithm 과 동일하다.

Learning rate	0.0002
Gamma	0.99
Multi-process	16
Update interval	5
Target update interval	100
Epsilon	Exploration = random.uniform(0.05, 0.15) Epsilon = max(0.002, exploration – 0.01*(epi/3000))

(2) Performance

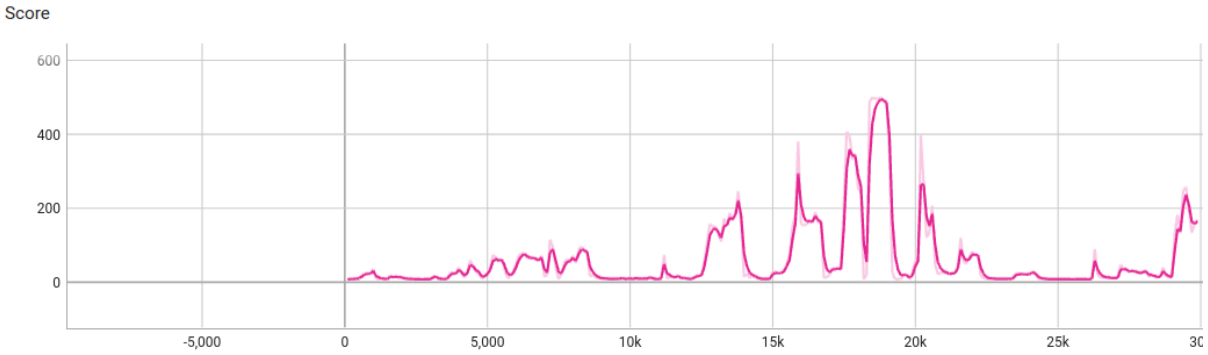

Figure 10. Sarsa performance

4) A3C algorithm

(1) Hyper-parameter

Learning rate	0.0002, 0.0005, 0.0001
Gamma	0.99
Multi-process	8, 16
Update interval	5, 8, 15
Target update interval	100

(2) Performance

A3C algorithm의 경우, 성능이 대체로 나오지 않았다. Hyper-parameter를 바꿔가면서 해도 일정 episode가 지나면 학습이 전혀 진행되지 않는 것을 확인할 수 있다.

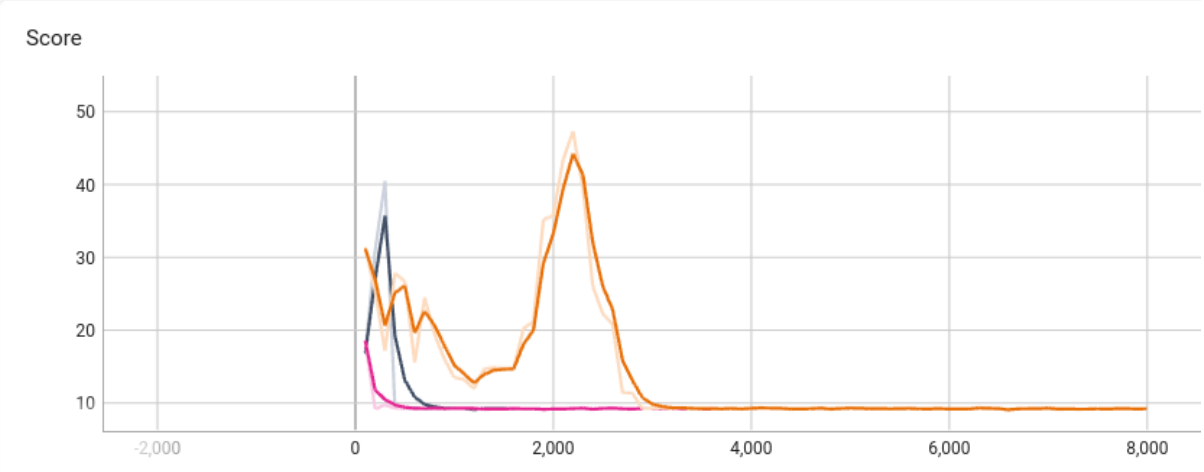

Figure 11. A3C performance

4. 개선점

- 본 section 에서는 성능이 현저히 떨어졌던 A3C algorithm 에 대해서 개선하고자 한다. 성능이 떨어진 이유를 다음과 같이 분석해봤다.

(1) 풀고자 하는 문제가 너무 간단하기 때문에 비슷한 경험이 쌓일 확률이 높고, action 의 수가 2 가지로 너무 적다. 따라서, process 수가 너무 많다면 오히려 agent 의 경험들이 비슷한 경우가 많아 time-correlation 을 끊을 수 없다.

(2) 또한, network 가 총 2 가지의 task(value function 학습/policy 학습)를 해결하기 위해 학습된다. Unstable 할 수 있다.

- 이러한 문제를 가정하고 해결하고자 (1) process 수를 16 개에서 5 개로 줄였다. (2) gradient clipping 을 활용하여 너무 큰 gradient 크기를 가지지 않게 제한했다.

4-1. 실험

(1) Hyper-parameter

Learning rate	0.0002
Gamma	0.99
Multi-process	5
Update interval	8
Target update interval	100
Clip grad norm	4

(2) Performance

짧은 epi 만으로도 최고 점수인 500 점을 달성했다.

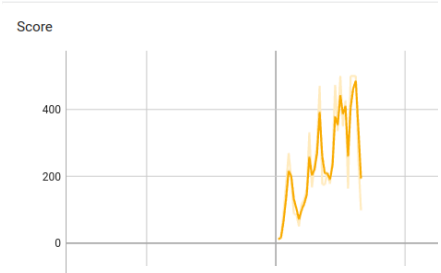

Figure 12. A3C(+clipping)

III. 결과 분석

최종적으로 실험 결과들을 분석하고자 한다.

1. Number of actor-learner in A3C

Gradient clipping 을 이용해 stable 하게 학습을 하되, 4. 개선점 – (1)에서 얘기한 문제점을 다시 한 번 확인하고자 했다. 모든 hyper-parameter 는 동일하게 설정하고 process 수만 16 으로 늘린 뒤 실험했다. Clipping 으로 인해 stable 하게 학습을 진행하긴 하지만, 성능이 증가하진 않았다.

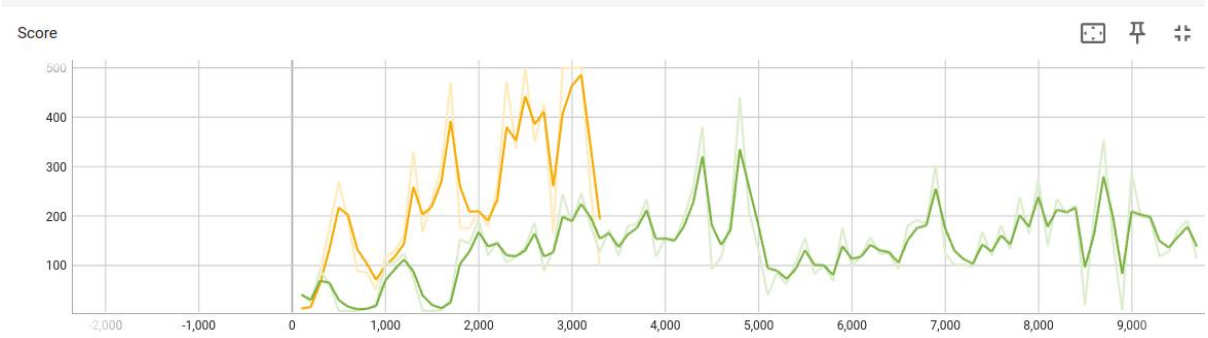

Figure 13. yellow: process-5; green: process-16

그럼에도, 기존의 A3C 결과와 비교했을 때에는 성능이 좋다. 즉, gradient clipping의 효과가 있다는 걸 알 수 있다.

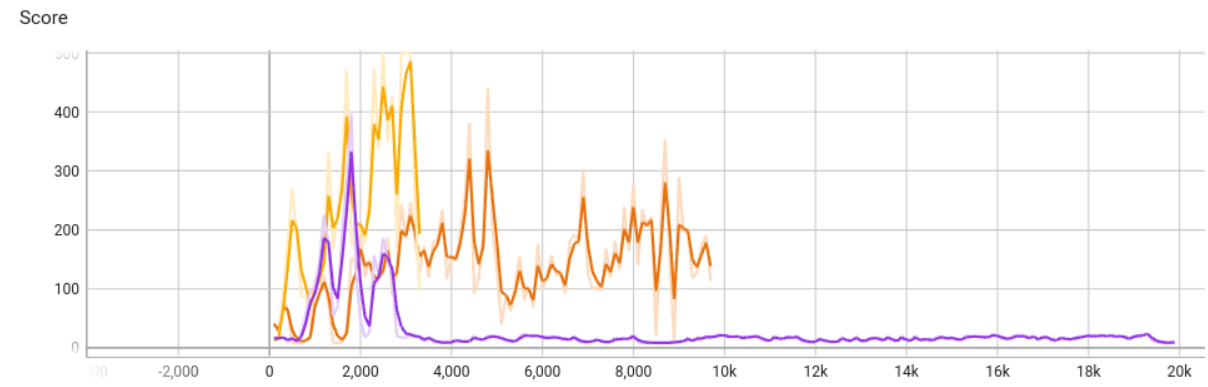

Figure 14. yellow: clipping, process-5; orange: clipping, process-16; others

2. Super linear

논문에서 actor-learner 수에 비례해 학습 시간이 단축됐던 실험 결과와 달리 제대로 측정할 수 없었기 때문에 검증하지는 못했다. CartPole이 단순한 문제이기에 발생하는 문제로 예상된다.

3. 최고 점수(500)를 달성하는데 걸린 시간

개선된 A3C algorithm의 성능이 가장 좋았다.

Algorithm	Episode(epi)	Elapsed time
DQN	11,000	9.773min
One-step Q-learning	18,800	2.506hr
One-step Sarsa	11,500	2.308hr
A3C(+gradient clipping)	**3100**	**3.778min**