REPRODUCIBILITY CHALLENGE: ANTICIPATORY ASYNCHRONOUS ADVANTAGE ACTOR-CRITIC (A4C): THE POWER OF ANTICIPATION IN DEEP REINFORCEMENT LEARNING

Abhiram Mullapudi, Zhonghao Zhou, Satat Ojha
University of Michigan
Ann Arbor, MI-48105, USA
abhiramm, zhongzz, ojhasata @umich.edu

ABSTRACT

This work evaluates the the hypothesis proposed in the A4C paper (Anonymous (2017)): extending the action space to include sequences of actions as a part of their policy; which leads to better exploration and convergence. To test this, we reproduced the proposed Anticipatory Deep-Q Network (ADQN) and Anticipatory Asynchronous Advantage Actor-Critic (A4C) algorithms. We compared the results of our implementation of ADQN on the classic *Cartpole* environment with DQN to reproduce the paper's result. Then to evaluate the efficiency of the proposed A4C algorithm, our implementation of A4C variants was compared with the state of the art A3C (GA3C) for one (*Qbert*) of the four Atari game results presented in the A4C paper.

1 INTRODUCTION

In the recent past, Deep reinforcement learning (RL) has had several breakthroughs; from DQN (Mnih et al. (2015)) emerging to AlphaGo (Silver et al. (2016)) attaining dazzling results. However, these algorithms are computationally intensive and tend to converge slowly. To address this problem, more efficient, light weight deep RL algorithms that parallelize the work load, such as Asynchronous Advantage Action-Critic (A3C) (Mnih et al. (2016)) and GA3C (Babaeizadeh et al. (2017)) were proposed. Still, these algorithms may take a long time to produce good results, often due to slow exploration. Furthermore, deep RL algorithms might take an extensive amount of time to identify good sequences of actions (Anonymous (2017)). To address these issues, authors of the A4C paper propose an idea: learn to anticipate a sequence of actions rather than basic singular ones. Anticipation would extend the current action space to include sequences of basic actions. So instead of trying to learn the best basic action for a state, the agent is forced to explore the best action sequence.

2 EXPERIMENTS

Although the paper provides detailed algorithms for the proposed ADQN and A4C variants, descriptions of the network architectures and most hyper-parameters for the results presented were not indicated. Thus, we had to create our own implementations; code and trained weights for the experiments in this work are provided at `https://github.com/abhiramm7/icrl_eecs_498`. Neural Networks were implemented using Keras [Tensorflow] (Chollet et al. (2015)) with the environments from openAI-gym (Brockman et al. (2016)). The hyper-parameters we used are in the appendix.

2.1 DQN AND ADQN

For the classic cartpole environment, while DQN picks an action from $\{L, R\}$, ADQN chooses an action from $\{L, R\}$ or a meta-action, a sequence of basic actions $\{LL, LR, RL, RR\}$; both,

by observing a particular state. To do so, we altered our DQN implementation to include meta-actions, allowing the agent to simultaneously implement a sequence of basic actions and regard the corresponding state and reward transitions.

Furthermore, for a state, ADQN preforms twice the number of updates to the action-value function compared to DQN. As such, we extended DQN's replay buffer to $\{s_t, s_{t+1}, s_{t+2}, a_t, a_{t+1}, r_t, r_{t+1}, terminal_t, terminal_{t+1}\}$. During a training step, we sample an experience batch from the replay buffer and update the values of $Q(s_t, a_t)$ and $Q(s_t, a^+)$ where ${a^+}_t$ is the meta-action corresponding to the sequence a_t, a_{t+1}.

We then noticed that the A4Cs' authors did not use a target network for ADQN. In our experiments with multiple architectures, we observed that this approach resulted in a consistently unstable neural network. Hence we used a target network to provide a moving target for the action-value function. The results of ADQN and DQN were then compared, with DQN as a baseline. The rewards for both algorithms were clipped at 200.

2.2 A4C AND GA3C

To test the efficiency of A4C, we compared its performance to the state of the art GA3C on *Qbert*. GA3C implementation was made available by it's authors. A4C builds upon A3C by adding an extending the action space to include a sequence of actions. The A4C paper also describes three variants to efficiently update gradients over the extended actions space: independent, dependent, and switching.

In independent-updating, each meta-action is viewed as a separate action the agent can choose, and updated as such. This is done by extending the action space for a environment to include a combination of basic actions. For example, in *Qbert*, an agent can pick six actions, which are extended to 42 $(6 \times 6 + 6)$ actions. We achieved this by modifying the environment to include the 42 actions. The rewards signal for a meta action is the sum of the reward achieved for each basic action in the meta-action. Then we trained GA3C on this modified environment to evaluate the performance of independent-updating A4C.

In dependent-updating, for a particular state in the training process, we accumulate twice the amount of gradients when compared to independent-updating. During an episode of dependent-updating, if an agent picks a meta-action, it is recorded as a sequence of basic actions. In the training process for a particular state, the gradient is accumulated twice: once for the basic action chosen at that state, and another for the meta-action corresponding to the basic action in that state and the basic action in the next state. To do this, we extend the action space to include 42 actions by modifying GA3C to compute the extended action space from the basic set of actions provided by the environment. Then, we altered the GA3C code-base to record a meta-action as a sequence of basic actions. Lastly, based on this set of basic actions, during the training process we compute the gradient for the basic action, as well as the gradient for the meta-action that corresponds to that basic action and the basic action that follows it. The policy is updated based on these accumulated gradients.

Switching-updating is a combination of dependent and independent updating. We ran the dependent-updating A4C first for approximately four hours, then recorded the neural network weights generated. As both independent and dependent updating A4C variants share the same architecture, we use the neural network weights from depended-updating A4C to continue training on independent-updating A4C.

3 RESULTS

3.1 ADQN

DQN worked well, was easy to reproduce, and achieved similar results to the A4C paper's. However, ADQN without a target network across multiple architectures and hyper-parameters did not preform as well as DQN. Furthermore, it was hard to reproduce as not only were hyper-parameters not provided, but we weren't sure if the update functions were right. The original updates lacked a target network, which resulted in an extremely unstable network, so we added our own target network. Nonetheless, ADQN with a target network, although stabler, did not do as well as DQN across

multiple architectures and hyper-parameters. However, it is notable that ADQN does reach the optimal solution, it just does not stay there. We believe that this is because *Cartpole* does not benefit much from having a sequence of actions, as the optimal one is an alternating sequence of left and right, and explores too much. Furthermore, DQN can learn two actions much faster than ADQN can learn six. To further explore the benefits of ADQN we might compare it to DQN on something that could take advantages of sequences better, such as *Qbert*.

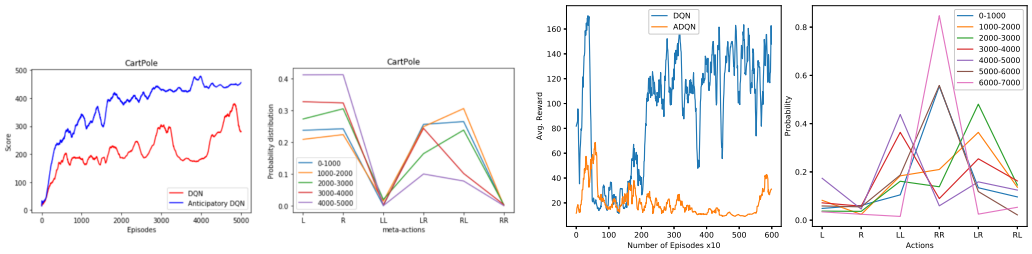

Figure 1: ADQN/DQN **Left**: A4C Paper's Rewards of ADQN and DQN on *Cartpole* **Right**: Our Smoothen Rewards With a Moving Window of 10 of ADQN with target network and DQN *Cartpole*

3.2 A4C

Reproducing the gradient update variations of A4C were well explained, but it took a long time to understand the GA3C implementations and modify them. DU-A4C's logic was complicated, and took a while, but the rest were relatively easier, especially SW-A4C after IU-A4C and DU-A4C were done. Overall, A4C appeared to preform better than our baseline, GA3C; although we could not get the exact values, likely due to the difference in hyper-parameters. In our graphs, we smoothed the results using a moving window of length 200. As there was a lot of noise, the average went down, but we consistently got scores of 2-3 thousand for DU-A4C and IU-A4C while SW-A4C scored around 3.5-4 thousand. However we note that *Qbert* is a simple Atari game with a small action space, so if we have more time and computing power, we would like to test it on an environment with a much larger action space and compare A4C and GA3C.

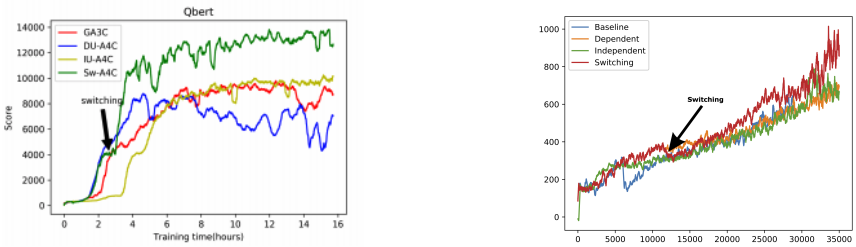

Figure 2: A4C/GA3C **Left**: A4C Paper's Performance of A4C and GA3C on *Qbert* **Right**: Our Performance of A4C and GA3C on *Qbert* after 22 hours

4 Conclusion

We found that A4C preformed better than GA3C on *Qbert*, but don't think it means much because it is just a particular environment. The extended action space explodes in large actions spaces, and even more so if we were to look at a longer sequence of actions. Even in *Cartpole*, it seemed that ADQN got stuck exploring than exploiting. Therefore we feel that anticipation is not a scalable deep RL method.

REFERENCES

Anonymous. Anticipatory asynchronous advantage actor-critic (a4c): The power of anticipation in deep reinforcement learning. *openreview*, Nov 2017. URL https://openreview.net/forum?id=rkKkSzb0b.

Mohammad Babaeizadeh, Iuri Frosio, Stephen Tyree, Jason Clemsons, and Jan Kautz. Ga3c, Apr 2017. URL https://github.com/NVlabs/GA3C.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

François Chollet et al. Keras, 2015.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pp. 1928–1937, 2016.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, and et al. Mastering the game of go with deep neural networks and tree search. *Nature News*, Jan 2016. URL https://www.nature.com/articles/nature16961.

APPENDIX

ADQN AND DQN PARAMETERS

DQN and ADQN were both trained on Google Cloud Compute Instances using Tesla K80 GPU. The hyper-parameters used in our DQN and ADQN implementations are as follows:

	DQN	ADQN
Neural Network Architecutre	32Neurons(1 Hidden layer)	32Neurons(2 Hidden layer)
Replay Buffer Size	50000	50000
Target Update	1000	1000
Adam Learning Rate	0.003	0.003
Batch Size	32	32
$\epsilon_{max}, \epsilon_{min}$	1.0, 0.1	1.0, 0.1

Table 1: ADQN/DQN Hyper-Parameters

A4C PARAMETERS

GA3C and A4C variations were all trained for 22 hours on Google Cloud Compute Instances using Tesla K80 GPU. The hyper-parameters were set to be the default parameters provided in GA3C. The neural network architecture for GA3C was maintained through all A4C variations.