A4C: ANTICIPATORY ASYNCHRONOUS ADVANTAGE ACTOR-CRITIC

Tharun Medini∗
Electrical and Computer Engineering
Rice University
Houston, TX 77054, USA
{tharun.medini}@rice.edu

Xun Luan∗ & Anshumali Shrivastava
Department of Computer Science
Rice University
Houston, TX 77054, USA
{xun.luan, anshumali}@rice.edu

ABSTRACT

We propose to extend existing deep reinforcement learning (Deep RL) algorithms by allowing them to additionally choose sequences of actions as a part of their policy. This modification forces the network to anticipate the reward of action sequences, which, as we show, improves the exploration leading to better convergence. We propose a method that squeezes more gradients from the same number of episodes and thereby achieves higher scores and converges faster. Our proposal is simple, flexible, and can be easily incorporated into any Deep RL framework. We show the power of our scheme by consistently outperforming the state-of-the-art GA3C algorithm on popular Atari Games.

1 INTRODUCTION

There is still a fundamental barrier in current Deep Reinforcement Learning (RL) algorithms, which is slow progress due to poor exploration. During the early phases, when the network is just initialized, the policy is nearly random. Thus, the initial experiences are primarily several random sequences of actions with very low rewards. Finding a good sequence via network exploration can take a significantly long time, especially when there are only very rare sequences of actions which gives high rewards, while most others give low or zero rewards. The exploration can take a significantly long time to witness those rare combinations of good moves.

The current state-of-the-art algorithm for RL is Asynchronous Advantage Actor-Critic (A3C) (Mnih et al., 2016; 2015). A3C has a central network that is shared by multiple agents playing the game in parallel. Each agent periodically sends gradient updates to the central networks and the networks accumulates gradients from all agents to update at once. The updated network is then used by the agents to continue playing. In a remarkable follow up to this simple parallel gradient descent based algorithm, a GPU version of it GA3C (Babaeizadeh et al., 2016) was developed and extensively tested by researchers at NVIDIA. It is the most optimized algorithm for Reinforcement Learning particularly on Atari games, tailored for HPC platforms such as GPUs. Outperforming GA3C in running time is hard as it will require both algorithmic and systems advancement.

We present a simple opportunity of improving the convergence of deep RL and beat GA3C in time. In particular, we show that instead of learning to map the reward over a basic action space \( \mathcal{A} \) for each state, we should force the network to anticipate the rewards over an enlarged action space \( \mathcal{A}^+ = \bigcup_{k=1}^{K} \mathcal{A}_k \) which contains sequential actions like \( (a_1, a_2, ..., a_k) \). While using the same episode information like A3C, we extract more gradient updates by training with action tuples apart from the basic actions.

2 OUR PROPOSAL: A4C

At a high level, our proposal extends the basic action set \( \mathcal{A} \) to an enlarged action space \( \mathcal{A}^+ = \bigcup_{k=1}^{K} \mathcal{A}_k \), which also includes sequences of actions up to length \( K \). As an illustration, let us say

∗Equal Contribution of authors
$A = \{L, R\}$ and we allow 2-step anticipation, therefore our new action space is $A^+ = A \cup A^2 = \{L, R, LL, LR, RL, RR\}$. Each element $a^+$ belonging to $A^+$ is called a meta-action, which could be a single basic action or a sequence of actions. Typical deep reinforcement learning algorithms have a DNN to output the estimated Q values (expected cumulative future reward) or policy distributions according to basic action set $A$. In our algorithm, we instead let the DNN output values for each meta-action in the enlarged action set $A^+$. Overall, we are forcing the network to anticipate the “goodness” of meta-actions a little further, and have a better vision of the possibilities earlier in the exploration phase.

Figure 1 shows our idea applied to Cartpole game learned by DQN [Mnih et al., 2013]. DQN is a value-based algorithm whose network approximates the maximum expected future reward values for each action. If we see each gradient update as a training sample sent to the network, DQN generates 1 training sample for each action-reward frame. We believe one frame could provide more information than that. For an experience sequence $\ldots, s_i, a_i, r_i, s_{i+1}, a_{i+1}, r_{i+1}, s_{i+2}, \ldots$, we will get two updates for state $s_i$:

$$L_{i,1}(\theta_i) = (r_i + \gamma \max_{a' \in A^+} Q(s_{i+1}, a'|\theta_i) - Q(s_i, a_i|\theta_i))^2$$

$$L_{i,2}(\theta_i) = (r_i + \gamma r_{i+1} + \gamma^2 \max_{a' \in A^+} Q(s_{i+2}, a'|\theta_i) - Q(s_i, a_i|\theta_i))^2$$

This update improves the intermediate representation aggressively leading to superior convergence. In practice, we could organize them into one single training vector, as illustrated in the Figure.

![Figure 1: A toy example for ADQN with an enlarged action set $\{L, R, LL, LR, RL, RR\}$. For input $s_0$, we have 2 gradients, one for action $L$ and other for action $LR$.](image)

The same idea can also be applied to the state-of-the-art GA3C (GPU enabled A3C) algorithm [Babaeizadeh et al., 2016]. In A3C, the neural network is used for two parts: prediction and training. In the prediction part, our method lets the neural network output a distribution of actions from $A^+$. For each state, we choose a meta-action $a^+$ according to the output distribution. If $a^+$ contains only one action, this single action will be executed. If $a^+$ corresponds to an action sequence $(a_1, a_2, ..., a_k)$, these actions will be executed one by one in order.

A4C is a strict generalization of A3C and it allows for three kinds of gradient updates for given action-reward frame: Dependent Updating (DU), Independent Updating (IU), and Switching.

### 2.0.1 Dependent Updating (DU)

When we take an action sequence and get rewards, we not only calculate the gradients for this sequence, but also for its corresponding preceding subsequences. For example, in a 2-step multi-action setting, we get an experience queue of $(s_0, a_0, r_0, s_1, a_1, r_1, s_2, \ldots)$. No matter $(a_0)$ was taken as a basic action or $(a_0, a_1)$ was taken as a multi-step action, we will update both of them for state $s_0$. In this case, we get 2 times more gradient updates as A3C for the same amount of episodes, resulting in aggressive updates which lead to accelerated convergence, especially during the initial phases of the learning.
2.0.2 Independent Updating (IU)

In this scheme, the reward of $a^+$ is the sum of rewards obtained by taking all the basic actions in $a^+$ one by one in order. The next state of $a^+$ is the state after taking all the actions in the sequence. While updating, we only use the information of summed reward, and the next state of $a^+$ without regards to the dependencies and relations between meta-actions.

2.0.3 Switching

Our experiments suggest that DU converges faster than GA3C on Atari games for the first few hours of training. Particularly, DU shows a big gap over the speed of original A3C in the Pong game (see Section 3.1). However, after training for a couple of hours, we observe that aggressive updates cause the network to saturate quickly. This phenomenon is analogous to Stochastic Gradient Descent (SGD) where initial updates are aggressive but over time we should decay the learning rate (Bottou, 2010). On the other hand, IU offers less aggressive updates and sustains growth in rewards for longer times. To exploit the benefits of both methods, we start with DU and switch over to IU after sometime in the training. For the best outcome, we should switch when DU starts to saturate. Confirming the saturation without human observation is difficult as the rewards are extremely variant. Hence, we set the switching time for each game separately after manually observing the reward curves for DU. Among the games that we report, switching time was 2 hrs for Pong and SpaceInvaders and 2.5 hrs for Qbert and Beamrider. We observe that switching seems to have robust performance in experiments with regards to different choice of hyperparameters.

3 Evaluations

3.1 Atari Games

![Figure 2: Comparison of three variants of A4C against GA3C. The baseline GA3C is shown in red, Dependent Updates(DU) in blue, Independent Updates(IU) in cyan and the Switching (Sw) in green. The light color fill is one standard deviation away on either side of the mean curve.](image)

We demonstrate our A4C experiments on 4 popular Atari-2600 games namely Pong, Qbert, Beam-Rider, and SpaceInvaders. We use the environments provided by OpenAI Gym for these games. Atari-2600 games are the standard benchmarks for Reinforcement Learning Algorithms. We compare our results against the state-of-the-art GPU based Asynchronous Actor-Critic (GA3C) framework from NVIDIA whose code is publicly available (at [https://github.com/NVlabs/GA3C](https://github.com/NVlabs/GA3C)). We compare our results time-wise on the same games chosen by GA3C paper.

We ran the baseline GA3C code on various games on our machine with a 14 core CPU and a single Tesla K20 GPU. We ran each variant(DU, IU, Switching, GA3C) for 3 times on each game on our
machine with 14 core CPU and a Tesla K20 GPU. The plots show the average scores and standard deviation with respect to time. We use the optimal setting of hyper parameters as suggested by GA3C paper (MinTrainingBatchSize=40, GradientClipping=False, LearningRate=0.003). We use 2-step anticipation in our experiments of A4C. The network architecture of A4C algorithm is the same as GA3C [Babaeizadeh et al., 2016] except the output policy layer. It consists of 2 convolutional layers; first layer with $8 \times 8$ filters (16 of them) and the second layer with $4 \times 4$ filters (32 of them). They are followed by a dense layer with 256 nodes. The last layer is the typical softmax layer with as many nodes as the number of effective actions (basic + multi-step).

Figure 2 shows the comparison of three variants of A4C updates against GA3C for four games. Note that the baseline GA3C plots (in red) are very similar to the ones reported in the original paper. We notice that the Independent Updates (IU) performs much better than GA3C on SpaceInvaders and BeamRider games (with 6 and 9 basic actions respectively). In particular, IU achieves a score of 4800 on BeamRider game which is way better than the best result mentioned in GA3C paper. IU crosses 3000 score in just 6 hrs while it takes 13 hrs for GA3C to achieve the same score. At the same time, we notice that the Dependent Updates (DU) method (in blue) starts to rise faster than GA3C but doesn’t sustain the growth after sometime owing to reasons mentioned in Section 2.0.3. The only case where DU maintains the growth for considerable amount of time is Pong (3 basic actions). It is evident that DU is suited for small action spaces and IU is suited for larger action spaces. The hybrid Switching method performs remarkably well consistently on all the games, achieving higher scores than the best of GA3C. For example, on QBert game, Switching achieves a score of 12000 in just 5 hrs. The best result mentioned in original GA3C paper achieves similar score in 20 hrs. The other re-runs of QBert in GA3C paper stall at a score of 8000. Switching also achieves a score of 700 on SpaceInvaders game where the best result in GA3C paper achieves < 600. In all, we notice that Switching blends the advantages for both DU and IU and is the most robust method for varied scenarios.

4 CONCLUSION AND FUTURE WORK

We propose a simple yet effective technique of adding anticipatory actions to the state-of-the-art GA3C method for reinforcement learning and achieve significant improvements in convergence and overall scores on several popular Atari-2600 games. We propose a strategy that treats each multi-step action as a sequence of basic actions and extracts gradients for the complete action sequence as well as the preceding sub-sequences. We also identify some issues that challenge the sustainability of our approach and propose simple workarounds to leverage most of the information from higher-order action space. However, the action space grows exponentially with the order of anticipation. Addressing large action space, therefore, remains a pressing concern for future work.

REFERENCES


