Review: Asynchronous Methods for Deep Reinforcement Learning[1]

Ashutosh Jha, Parikshit Hegde

February 2017

1 Background

Deep-Q-Networks[2] by DeepMind opened up a whole range of research by showing that it was possible for a single agent(a neural network with the same hyper parameters) to be trained to perform exceptionally well on a whole range of tasks. They stabilized a previously very unstable training process with the use of a Replay Memory and holding the target constant for a number of training steps. This process took about 8 days to learn to play an Atari games even when trained on the best GPU's.

Distributed Frameworks such as DistBelief [3] existed in Supervised and Unsupervised Learning settings before which enabled a much faster training process. In the domain of Reinforcement Learning such a distributed setting was first implemented with the Gorila[4] framework. In Gorila, they used multiple actor-learners training on their own version of the environment and making updates to a master parameter server. They used actually used multiple machines to train which induced a communication cost between the machines. Also, they still used the concept of Replay Memory to train the networks.

2 Asynchronous Deep RL

In the Asynchronous DRL framework proposed in this paper, they improved upon the work done in Gorila. One significant change was that instead of using multiple machines, they used the multiple threads(16 in their case) within the same processor for the actor-learners which significantly improved performance because of the reduction in communication overhead.

One other significant change was that they got rid of the Replay Memory and made the training process Online. Recall that in previous works Replay Memory was used to get rid of the correlation between successive sample(i.e., being from the same part of the state-space). In this framework, since we already have 16 learners exploring different parts of the state space, there is no need for stabilization using Replay Memory. In particular, when a different exploration policy is used for the different actor-learners it was observed that the training process was very well stabilized. This increases the speed in training because sampling from the Replay Memory(especially when it is very large) had a huge overhead. Also, by making the learning process online, we can now use On-Policy methods such as actor-critic to train our network.

They have experimented with these RL algorithms to train the network: Asynchronous One-Step Q Learning Asynchronous One-Step SARSA, Asynchronous n-step Q Learning and Asynchronous Advantage Actor Critic(A3C). The general idea in all these algorithms remain the same as that given in Sutton and Barto[5]. Some changes are that as in DQN the actor-learners use a slowly changing target network to compute the loss, and that the gradients from the various actor learners are collected in batches and then applied to the master network. This is so as to reduce the chances of one actor-learner over-writing the updates of another actor-learner.

3 Asynchronous Advantage Actor Critic

We will elaborate a bit more on the A3C as it was found to be giving the best results among the algorithms that were tried. In this framework, we maintain a policy($\pi(a_t|s_t;\theta)$) and a value function($V(s_t;\theta_v)$). The updates for the value function are the same as that in Q Learning except that we make updates on the state-value function instead of state-action value function. The update for the policy is given by the gradient $\nabla_\theta \log \pi(a_t|s_t;\theta)A(s_t,a_t;\theta,\theta_v)$: where A is an estimate of the advantage function. Here the n-step return was used to compute the advantage function. However, the n-step return had a peculiar implementation: the actor produced 20 steps(in general) and then the 20th step was updated using the 1-step return, 19th step was using 2step return and so on until the first state which used the 20-step return. The reason for this given in the paper was that this was easier to implement. Could it be that this implementation offered a faster convergence because of a bias-variance tradeoff? We know that TD(0)[a one-step update] had a high bias and Monte-Carlo had a high variance. Could it be that the different step-returns in this implementation uses a mix of slightly biased estimates and estimates with slightly higher variance to provide faster convergence?

The optimization method used in all the methods was RMS prop. They found that sharing the statistic across the different actor learners provided for a more robust learning.

They experimented the proposed techniques on several platforms: Atari, TORCS, MuJoCo Physics Simulator and Labyrinth. On previously tested platforms it was that all the proposed techniques beat the classical DQN with A3C dominating in all the tasks. Labyrinth was a new environment which was experimented and produced good results with just 1 day of training.

One conclusion one would make when using multiple learners is to expect a linear increase in speedup because of a linear increase in computation because of multiple parallel learners. However it was observed that there was at least an order of magnitude more speedup. This shows that the multiple learners enabled training to be more stable and hence trained faster. This opens the door for further scaling up the model.

4 Some Comments

- Could a similar framework be implemented with the use of multiple GPUs? The GPUs would perform the Matrix Computations in the forward and backward pass much faster, however the communication overhead between the CPU and the GPU when communicating the gradients would be higher. Which side wins in this trade-off?
- The paper talks about one goal being the use of cheaper resources to achieve the same effect(even better) as with GPUs. Since most processors have a dual/quad core, how does this method work with 2 or 4 parallel learners?

References

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

[2] 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.

[3] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[4] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, Shane Legg, Volodymyr Mnih, Koray Kavukcuoglu, and David Silver. Massively parallel methods for deep reinforcement learning. *CoRR*, abs/1507.04296, 2015.

[5] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction, 2^{nd} edition draft, 2017.