# **Deep Reinforcement Learning on Playing OpenAI Gym Games**

|  |  |
| --- | --- |
| Lingjie Kong1  Stanford University  Stanford Center for Professional Development  ljkong@stanford.edu | Ruixuan Ren1  Stanford University  Stanford Center for Professional Development  ruixuan@stanford.edu |

**Abstract**

*Reinforcement Learning (RL) is an area regarding how agents act to an unknown environment for maximizing its rewards. Unlike Markov Decision Process (MDP) in which agent has full knowledge of its state, rewards, and transitional probability, RL agent utilizes exploration and exploitation to cover model uncertainty. Because the model usually has a large input feature space, a neural network (NN) is often used to summarize the correlation between input feature and output state action value. Our goal is to improve existing algorithms or potentially develop new algorithms, specifically double A3C. We will implement DQN, double DQN, dueling DQN and A3C (Asynchronous Advantage Actor-Critic) to play OpenAI Gym Atari 2600 games to obtain benchmark performance. Then we will propose our implementation on double A3C, an improved version of state-of-the-art A3C algorithm. We will compare its performance, data efficiency and computation efficiency to the other methods.*

# **Introduction**

Behaviorist psychology regarding taking the best actions to optimize agent’s reward at a specific state inspired the development of reinforcement learning. Up to now, reinforcement learning has been studied in many disciplines such as control theory, information theory, statistics, and so on.

Markov Decision Process (MDP) was used to solve classical decision-making problem where agent has full knowledge of the environment including state, reward, and transitional probability. Due to the limitation in knowledge of the environment, Q learning was developed to let agent explore to find potential optimal solution as well as exploit to optimize the current good solution.

Due to large input state space, it is impossible to use a look-up table like in MDP. Neural network (NN) is used instead. Since a single-hidden-layer neural network is a universal function approximator, it can capture the non-linear relationship between input and output. The network will be trained using the gradient of its loss function, carried out by forward and backward propagations in the NN. The fully trained model will be used to infer based on the current state input, what will be the optimal action to take in order to maximize its rewards.

Reinforcement Learning bring new challenges on how to build and train an efficient neural network. Specifically, RL agent must learn from sparse and noisy data collected through its interactions with the environment. These sparse and noisy data might cause instability during training. Moreover, reward can be delayed. Therefore, it requires efficient method to reward early actions that bring good results later in the training. The environment is often assumed to be static in reinforcement learning. However, as the agent interacts with the real environment, it might change the environment. Therefore, good algorithms are desired to capture the changes in environment dynamics as well.

In this article, the performances of deep Q-network (DQN), double DQN, dueling DQN and asynchronous advantage actor-critic (A3C) will be compared, using three Atari games: Pong, Breakout and Ice Hockey. A variant of the A3C – double A3C will be presented. Double A3C utilizes the strengths from double DQN and A3C, and we hope to see better results from it compared to the benchmarks.



Figure 1(left to right) Pong, Breakout, Ice Hockey

# **Related Work**

High-dimensional visual input is challenging because of its large scale of data. Therefore, many successful RL model in the past is based on carefully hand-selected features. However, it is impossible to handpick features for every environment, and a more generic framework is desired.

The breakthrough in computer vision leads to new ideas to extract feature representations from environment more efficiently [1]. Neural network structures such as convolutional neural networks (CNN), multilayer perception and Boltzmann machine graphic model are often used, which can take large size input features with a relatively small amount of trainable variables.

In addition to the challenge of input feature representation, other challenges are also presented in reinforcement learning. Unlike supervised learning which assumes identical and independent distributed (IID) dataset, reinforcement learning must learn from noisy, delayed and highly correlated rewards.

Q-Learning [2] is often used to train reinforcement learning model to reach decent level of performance in simple environment. In Q-Learning, one needs update Q value Q(*s, a*) for state action pair, where each Q(*s, a*) is the expected utility of taking specific action *a* in state *s,* following the optimal policy onwards. However, it is impossible to explicitly store Q value for large input state space. One common solution is to use function approximation, where we extract featuresfrom (*s, a*) and define a functionto approximate Q(*s, a*). Then, instead of optimizing the estimation of Q values, the model is trained to optimize the parameters in.

Deep Reinforcement Learning [3] uses a neural network, specifically Deep Q-Network (DQN), as approximate function. It has been shown that the agents trained by DQN can reach better-than-human performances in playing many Atari 2600 games. Further studies of Double DQN [4] and Dueling DQN [5] improve both the convergence speed and performance compared to vanilla DQN. With accessibility to GPU, DQN can be trained in relatively fast speed.

Recently, asynchronous method has shown the potential to outperform previous algorithms like DQN [6]. In particular, Asynchronous Advantage Actor-Critic (A3C), can be trained two times faster than DQN with only multi-core CPU, and achieves higher performances in most of the Atari 2600 games.

# **Approach**

Convolutional neural network includes convolutional layers, activation functions and fully connected layers. One can also add max pooling and normalization layers to help improve speed of convergence as well as performance. The basic structure is so called AlexNet [1] as shown below.



Figure 2 Convolutional Neural Networks

In the context of discrete action space, DQN will pass fully connected layers output through couple more dense layers to generate output with the same dimension of action space. Action with the highest Q value will be picked as the optimal action in that state.

DQN is often trained with experience replay which helps break the correlation between sampled data and improves data efficiency. Specifically, agent will store certain amount of sampled data in buffer and pick them randomly in training. DQN will be trained using gradients, calculated from the minimization of the loss between current Q value and the updated Q value. The latter is estimated by taking the optimal action under the current Q value model. DQN algorithm [3] is as below.



Figure 3 DQN

However, vanilla DQN has the problem of biased overestimation by using the same network for picking greedy policy as well as training Q value. This problem can be resolved by utilizing two independent networks, known as double Q-learning algorithm. In Double Q-learning, two Q value functions are trained independently, with one to determine the greedy policy and the other to determine its value [7].

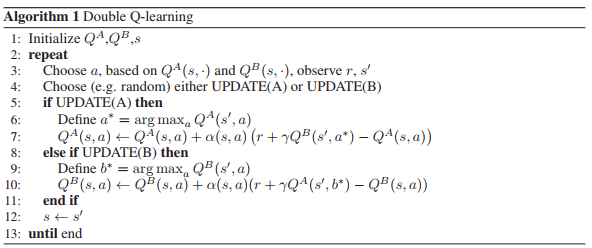


Figure 4 Double Q Learning

Instead of training a second network, double DQN copies over older weights of the target network to the second network, which is used for action evaluation when calculating gradients for target network’s weights. It has been shown that Double DQN finds better policies and obtains better results on the Atari 2600 domain compared to Vanilla DQN [4].

Another breakthrough on DQN is to use Dueling Network Architecture [5]. Instead of a single network for Q value, dueling DQN models state value and state action advantage function separately in the fully-connected layers. Dueling DQN updates estimations on action values and action benefits simultaneously, which gives better action-state values approximation. Assumes weights and are the parameters of fully-connected layers for state value and action advantage. Then we can express the action-state value as:



Figure 5 Single stream Q-network (top) and dueling Q-network (bottom)

Asynchronous advantage actor-critic (A3C) algorithm demonstrates better performance than any aforementioned algorithms. A3C utilizes multi-threaded asynchronous variant of advantage actor-critic algorithm, in which the actor is to improve the current policy and the critic evaluates the current policy. The algorithm of A3C is as below [6].



Figure 6 A3C

Our approach evolves from Double Q learning [7] and state-of-the-art A3C algorithm [6]. The key technique in Double Q learning is to train two Q value functions independently. We believe the such technique can also be applied to A3C algorithm to help improve convergence speed. Specifically, we add a second set of parameters and randomly pick from one set to update both for policy and for value in Figure 6 A3C algorithm. We hope two independent values will break the correlation in sampling and give better speed of convergence. We call this method double A3C as shown in Figure 7(b). There is one more network with less shared convolutional and fully connected layers as shown in Figure 7(c) called less shared double A3C. In Figure 7(d), the network with almost no shared parameters is called no shared double A3C.

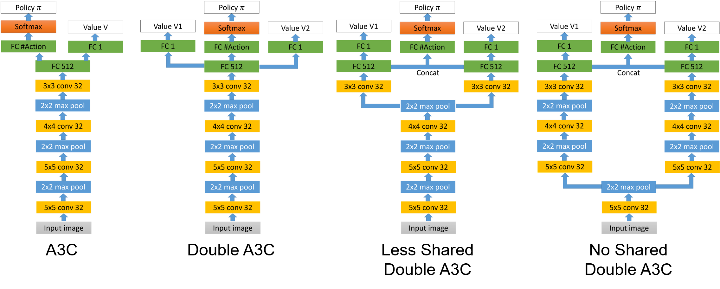


Figure 7 Network architecture of (a) vanilla A3C, (b) double A3C, (c) less shared double A3C (d) no shared double A3C

In the A3C network Figure 6(a), the input is 4 consecutive 84x84x3 RGB image frames. Therefore, the total input size is 84x84x12. Four convolutional layers and three max pooling layers were used to extract the input image information. Then, a fully connected layer will reshape and generate output with dimension of 512 (FC512). The output from this fully connected layer (FC512) will be used to generate a value with dimension of 1. This output value will estimate the input state value. Meanwhile, a different fully connected layer will be applied to generate output with the same number of actions. This will be past into a softmax layer to convert output to policy which is a probability representation between 0 and 1.

In A3C, value parameter and policy parameter share some common parameters from the first convolutional layer up to the first fully connected layer. Only the output layers are different for value and policy. From a deep learning perspective, it is good to share the same extracted features. Then, both value and policy output should be built on top of the same feature extractor for beter performance.

Unlike A3C which only has one single to estimate the input state value, double A3C will have two different value estimations and . Meanwhile, it will have one single policy . The training update for double A3C is similar to classical A3C. However, under the condition that there are two values, one will be sampled randomly for update. If is sampled for update at certain state, R shown in Figure 6 will be initialized by as below

The following update for and will be as below

Similarly, if is selected, we can update with similar method.

In double A3C from Figure 6(b), first value parameters , second value parameters , and policy parameters share some variables as well. Double A3C shares the same parameters from the first convolutional layer up to the first fully connected layer. Then, different fully connected layers are used to generate . , and individually. By using two different value functions, we believe it can help remove the correlation from consecutive state samples. The uncorrelated estimation from one value function will lead another value function for fast convergence and eventually better performance.

Under the condition that double A3C still shares most parameters from first convolutional layers to first fully connected layers, we try to build network with even less shared parameters which we believe will lead to more independent estimation for and . We wonder that whether this can help even break the correlation during update which can lead to better convergence. Therefore, we introduce less shared double A3C in Figure 6(c) and no shared double A3C in Figure 6(d). Less shared double A3C has 3 shared convolutional and max pooling layers while no shared double A3C only has 1 shared convolutional and max pooling layers. To generate the policy , fully connected layers from and will be concatenated and then softmax will be applied on top of that.

# **Experiment**

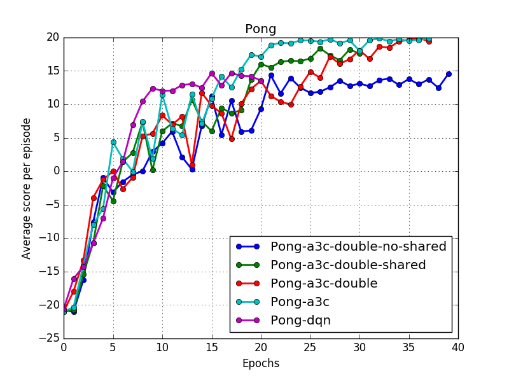
Three Atari games: Breakout, Ice Hockey and Pong were used to evaluate performances of A3C, double A3C and less shared double A3C, no shared double A3C under OpenAI Gym Atari 2600 games. We analyze both the speed of convergence with respect to time as well as training efficiency after each epoch.

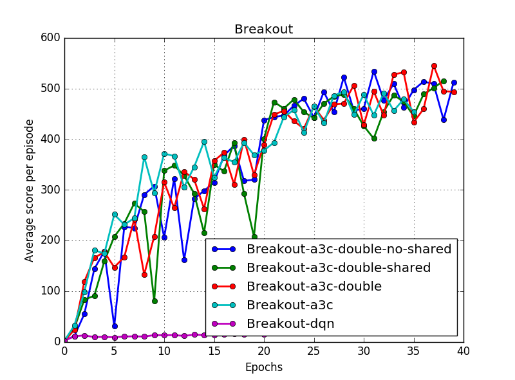
Our A3C model is built based on existing A3C model using tensorpack [8]. Our code is published under our Github repository [9].

Different from DQN which only maintain single agent for training, our A3C model keeps 3 agents in parallel to collect samples for update. As a result, because of A3C maintains multiple thread, it does not require cache state, action tuple for experience reply.

We utilize fast speed Microsoft Azure cloud computing source to train our model. 6 cores E5-2690v3 Intel CPU as well as K80 NVIDIA GPU are used to speed up our training.

The result for pong, breakout, and ice hockey are shown as below. We evaluated average score vs. epochs to see training efficiency. Specifically, by using the same amount of data in each epoch, we want to compare the efficiency of different algorithms. In each epoch, it has 6000 updates with batch size of 128. We also evaluated average score vs. times to ssee how long it will take for different algorithms to converge.





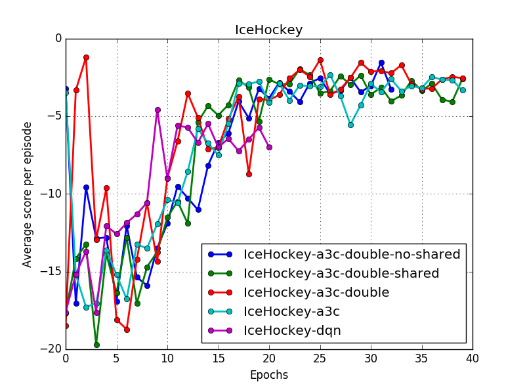
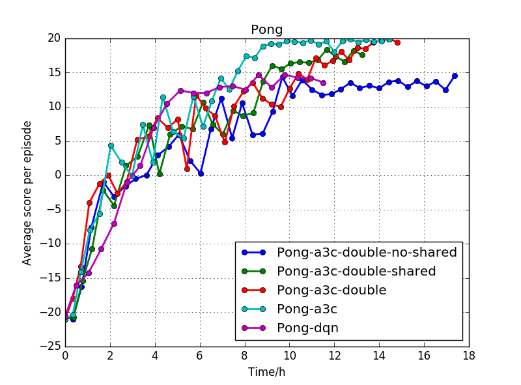
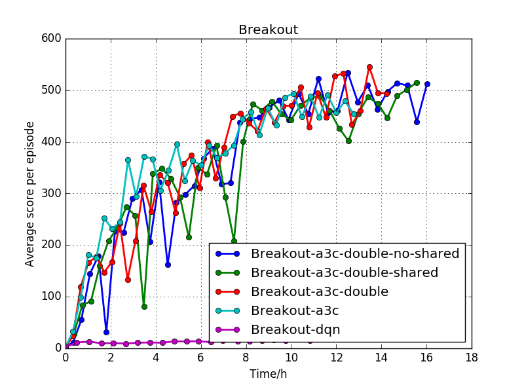


Figure 8 Comparison of data efficiency between all A3C and DQN. X-axis is the total number of training epochs in which each epoch has 6000 update steps. Y-axis is the average score





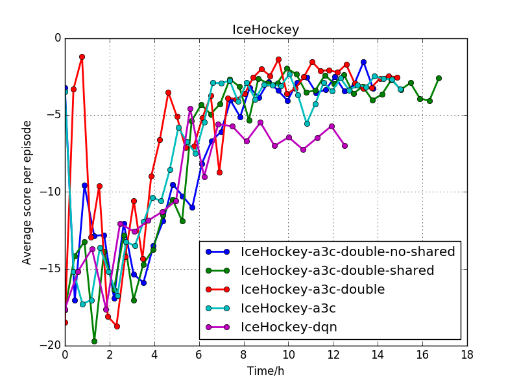


Figure 9 Comparison of training speed between all A3C and DQN. X-axis is the total number of training time. Y-axis is the average score

In Figure 8, it shows the average score at the end of each epoch. Pong is a relative simple game. Therefore, all A3C as well as DQN reach almost converge to the same average score. Breakout is an intermediate level game. All kinds of A3C largely outperform DQN. Ice hockey is a hard game. Even though all kinds of A3C converge to slight better score, it is far from reaching normal human players level. Regarding the comparison between all A3C, it looks like methods of double, less shared double, no shared double A3C which introduces a second value function are at the same level of performance compared to vanilla A3C.

In Figure 9, it shows the speed of convergence. For DQN, it takes about 37 minutes for each epoch update to finish. For no shared double A3C and less shared double A3C, both takes about 25 minutes for each epoch update. double A3C and Vanilla A3C take about 22 minutes.

# **Discussion**

For future work, we like to train even less shared A3C to see whether it will give better convergence. We will also compare data efficiency among three different A3C methods. We potentially want to come up with one more new algorithm to experiment on. If time allows, we would like to expand the benchmark by implementing vanilla DQN and its variants.

# **References**

[1] A. Krizhevsky, I. Sutskever, and H. Geoffrey E., “ImageNet Classification with Deep Convolutional Neural Networks,” *Adv. Neural Inf. Process. Syst. 25*, pp. 1–9, 2012.

[2] C. J. C. H. Watkins and P. Dayan, “Q-learning,” *Mach. Learn.*, vol. 8, no. 3–4, pp. 279–292, 1992.

[3] D. Zoran *et al.*, “Playing Atari with Deep Reinforcement Learning,” *arXiv*, vol. 32, no. Ijcai, pp. 1–9, 2016.

[4] D. S. H. Van Hasselt, A. Guez, “Deep Reinforcement Learning with Double Q-learning,” p. in proceedings of AAAI 2016.

[5] Z. Wang, N. de Freitas, and M. Lanctot, “Dueling Network Architectures for Deep Reinforcement Learning,” *arXiv*, no. 9, pp. 1–16, 2016.

[6] V. Mnih et al., “Asynchronous Methods for Deep Reinforcement Learning,” p. in proceedings of ICML 2016.

[7] H. Van Hasselt, A. C. Group, and C. Wiskunde, “Double Q-learning,” *Nips*, pp. 1–9, 2010.

[8] Y. Wu, “A3C-Gym,” no. GitHub Repository, p. https://github.com/ppwwyyxx/tensorpack

[9] L. Kong and R. Ren, “Deep Reinforcement Learning on Playing OpenAI Gym Games,” no. GitHub Repository, p. https://github.com/lingjiekong/CS234Project.