Double A3C: Deep Reinforcement Learning on OpenAI Gym Games

Yangxin Zhong[1]
Stanford University
Computer Science Department
yangxin@stanford.edu

Jiajie He[1]
Stanford University
Mechanical Engineering Department
jiajie@stanford.edu

Lingjie Kong[1]
Stanford University
Stanford Center for Professional Development
ljkong@stanford.edu

Abstract

Reinforcement Learning (RL) is an area of machine learning figuring out how agents take actions in an unknown environment to maximize its rewards. Unlike classical Markov Decision Process (MDP) in which agent has full knowledge of its state, rewards, and transitional probability, reinforcement learning utilizes exploration and exploitation for the model uncertainty. Under the condition that the model usually has a large state space, a neural network (NN) can be used to correlate its input state to its output actions to maximize the agent's rewards. We will propose and implement Double A3C, an improved version of state-of-the-art Asynchronous Advantage Actor-Critic (A3C) algorithm to play OpenAI Gym Atari 2600 games to beat its benchmarks for our project.

1. Introduction

Reinforcement Learning (RL) is inspired by behaviorist psychology regarding taking the best actions to optimize agent's reward at a specific state. There have been studies in many disciplines such as control theory, information theory, statistics, and so on.

Classical decision-making problem was formed as a Markov Decision Process (MDP) where people need to have full knowledge of the environment and carefully model its state reward, transitional reward, as well as transitional probability. Due to this limitation, reinforcement learning with Q learning was developed to let agent explore to find possible optimal solution and exploit to optimize the good solutions found up to now.

Under the condition that correlating the large input state space to agent action is not accomplished through look up table like MDP, neural network is used to capture the non-linear relationship between input and output. During the training, forward and backward propagation will be used to train the weight at each layer. With fully trained model, it will be used to inference based on the current state input, what will be the optimal action to take in order to maximize its rewards.

2. Related Work

Figuring out how to control agent from high-dimensional inputs like vision input is one of the biggest challenges of reinforcement learning. Most successfully RL model before is based on carefully selected feature with linear combined values. Obviously, the quality of the selected feature representation will largely influence the performance.

With the fast development of computer vision, it leads to some breakthroughs on how to extract the feature representation more efficiently by using more efficient models [1]. All these methods utilize ideas of neural network structures such as Convolutional Neural Networks, (CNN), Recurrent Neural Networks (RNN), Multilayer Perception, Boltzmann Machine Graphic Model, and so on.

Besides the challenge from the input feature representation, reinforcement learning presents other challenges. First, traditional machine learning requires large number of carefully labelled data. However, reinforcement learning algorithm has to learn from scalar rewards which is most of the time noisy and delayed from the current state. Second, unlike most supervised learning algorithm which assume the independence of samples, RL's sample are highly correlated.

Q-Learning algorithm [2] with stochastic gradient descent is often used to train reinforcement learning model. In Q-Learning algorithm, we need to store and update a Q value estimate Q(s, a) for each (s, a) pair, where Q(s, a) is the expected utility or value of taking action a in state s and then following the optimal policy afterwards. However, if we have a large state or action space, it will be expensive to store Q values for all (s, a) pairs. One of the common solutions to this issue is to use function approximation, where we extract features θ from (s, a) and define a function $f(\theta)$ to approximate Q(s, a). Then optimizing the estimation of Q values turns into optimizing the parameters in $f(\theta)$.

Deep Reinforcement Learning [3] uses a deep neural network, which is called Deep Q-Network (DQN), as the approximate function $f(\theta)$ of Q values. This research

[1]All authors contributed equally to this work.

showed that the agents trained by DQN can achieve high performances in playing Atari 2600 games in most cases. Further studies of Double DQN [4] and Dueling DQN [5] proposed methods to improve the convergence speed and final performance of DQN. All the DQN models mentioned above can be trained with GPU at a high speed.

Recently, asynchronous method has been proposed to apply to the Deep Reinforcement Learning [6]. The study showed that their best algorithm, Asynchronous Advantage Actor-Critic (A3C), can be trained 2x faster than DQN even if it uses a multi-core CPU instead of GPU. Moreover, agents trained by A3C can achieve higher performances in most of the Atari 2600 games than DQN models.

3. Approach

Our approach will be based on the Double DQN [4] and state-of-the-art A3C algorithm [6]. The key technique in Double DQN is to use two deep neural networks with the same structure but different parameters as two approximate functions of Q values. This technique can reduce the overestimations of action values under certain conditions and improve the agent performance. We believe the same technique can also be applied to A3C algorithm. To be more concrete, we can use two sets of parameters of the same neural network to approximate the values of states and optimal policy ($V(s_i;\theta_v)$ and $\pi(a_t|s_t;\theta)$ in Algorithm S3 below). We hope to achieve higher or at least the same performance compared to the vanilla A3C algorithm.

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 $T > T_{max}$

4. Experiment

We will train and evaluate our approach using the environment of OpenAI Gym Atari 2600 games. The input will be the screen images in each game and the output of our method will be the optimal policy. We will compare the average performance of agents trained by vanilla A3C algorithm and our version on different Atari games to measure success. Moreover, we will analyze whether our method will benefit or harm the convergence speed of A3C algorithm.

5. Conclusion

6. 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] V. Mnih *et al.*, "Playing Atari with Deep Reinforcement Learning," *arXiv*, vol. 32, no. Ijcai, pp. 1–9, 2016.

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

[5] Z. Wang *et al.*, "Dueling Network Architectures for Deep Reinforcement Learning," in proceedings of ICML 2016.

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