Paper Review

Benjamin Bush, Eric Tang, Steven Hartman*
Washington University in St.Louis

(Dated: Oct 2018)

CONTENTS

* haotang@wustl.edu

I. DUELING DEEP Q-NETWORK

Deep Q-Learning, the new version of Q-Learning with neural network fitting in was first introduced in 2014 and since then, a lot of improvements have been made. Especially, one version of the alternatives, Dueling DQN, leads to better policy evaluation if we are given many similar-valued actions and outperforms the state of the art on the Atari game with 2600 domain[1]. Q-values correspond to how good the agent is to be at the state and taking an action at the state and the Q-value at the given fixed state is denoted by $Q(s, a)$. The main idea of Dueling DQN is that it decomposes $Q(s, a)$ as $V(s)$, the summation of the value of staying at that state and $A(s, a)$, the advantage of taking action a at the state. Intuitively, the advantage function subtracts the value of the state from the Q function to obtain a relative measure of the importance of each single action given that the agent stay at that state. Suppose we are given a fixed policy π, then

$$A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$$

The key insight behind the new architecture is that for some states, it is unnecessary to estimate the value of each action choice; However for other states, it is of paramount importance to know which action to take[1]. In the prosthetic project, each action vector with dimension of 19 is less important to think about if the values of the different actions are very similar. On the dueling architecture, it can learn which states are valuable without having to the learn the effect of any action. The reason Dueling Deep Q-networking is useful in our project is that the agent might not need to take any additional action in the state when the reward is high. No action needs to be taken at the moment when the skeletal muscles are supposed to be relaxed.Therefore, when we design our convolutional neural networks for the prosthetic project, instead of following the convolutional layers with a single sequence of fully connected layers, we use two sequences of fully connected layers. Then the two sequences are aggregated together and make the output Q. If we aggregate the stream using

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha)$$

we will fall into the issue of identifiability[1] that we cannot find $A(s, a)$ and $V(s)$ and it is troublesome for gradient descent. To avoid that, Dueling DQN will use

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + (A(s, a; \theta, \alpha) - \frac{1}{A} \sum_{a'} A(s, a'; \theta, \alpha))$$

which is subtracting the average advantage of all actions in that state so that the Duel DQN will accelerate the training and help use find more reliable Q values for each action. In our implementation, keeping the same convolutional neural networks, we will modify the DQN architecture by only adding new sequences (one is for value and one is for advantage) and the aggregating layer.

II. PROXIMAL POLICY OPTIMIZATION ALGORITHMS

The main purposes for applying Proximal Policy Optimization (PPO) is that it strikes a balance among easy implementation, sample efficiency and easy to tune. Unlike DQN which only learns form stored offline data, proximal policy optimization learns online. For off-policy algorithms like DQN, the agent cannot pick up actions and it will learn via exploration but play without exploration. For proximal policy optimization, it will follow its own policy and pick up its own actions so that it will directly learn from whatever if it encounters. Since policy gradient methods will only use the collect experience once for updating, policy gradient methods are less sample efficient than DQN. The most commonly used general policy optimization methods usually start by defining the policy gradient laws as

$$\hat{g} = \hat{\mathbb{E}}_t[\nabla_{\theta} \log \pi_{\theta}(a_t|s_t)\hat{A}_t]$$

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t[\log \pi_{\theta}(a_t|s_t)\hat{A}_t]$$

π_{θ} is the policy that it takes it takes the observed states from the environment and makes the action to take. $\hat{A}_t$ is basically trying to estimate what the relative value of the selected action. We call it advantage estimate which tells us how much better was the action that we take based on the expectation of what will normally happen in the state. In short, $\hat{A}_t$ tells us whether the action is better than expected or not. $\hat{\mathbb{E}}_t[\log \pi_{\theta}(a_t|s_t)\hat{A}_t]$ will be the final optimization object that is used in policy gradient.Therefore, intuitively if the $\hat{A}_t$ is positive, gradient will be positive and it will increase the likelihood to choose the action. But if we keep running gradient descent, the parameter in the neural network will be updated outside of the range. So we have to make sure that the policy will never be updated too far away from the old policy. So within introducing Trust Region Methods (TRPO)[2], we modify the objective function to

$$\max \hat{\mathbb{E}}_t[\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)}\hat{A}_t]$$

. Moreover TRPO adding a KL constraint which effectively guarantees that the new updated policy is close to the old one. However, the KL constraint just adds overhead to the optimization and will lead to some bad training behaviors. Therefore PPO will overcome the problem above. Let $r_t\theta = \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta old}(a_t|s_t)}$, TRPO will be

$$\max \hat{\mathbb{E}}_t[r_t\theta\hat{A}_t]$$

so the TRPO is more readable. Then we penalize changes to the polcy that move $r_t\theta$ away from 1 then main object function will be

$$\max \hat{\mathbb{E}}_t[\min(r_t\theta\hat{A}_t), clip(r_t\theta, 1-\epsilon, 1+\epsilon)\hat{A}_t]$$

and the epsilon is a hyper-parameter. $r_t\theta\hat{A}_t$ is the default objective for normal TRPO but $clip(r_t\theta, 1-\epsilon, 1+\epsilon)\hat{A}_t$ is a truncated version of the first term so that the clipped term will limit the effect of the gradient update for advantage larger than 0[2]. Basically, PPO does the same as TRPO but it forces the policy updates to be conservative if it moves so far away from the current policy and moreover, PPO always outperforms. In general, PPO method have the stability and reliability of trust-region methods but are much simpler to implement[2]. Based on the continuous experiment given by this paper,PPO outperforms the previous methods on almost all the continuous control environments. Due to the fact that our training agent is take action continuously, we will apply this method in our implementation for training.

[1] Z. Wang,T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, *Dueling Network Architectures for Deep Reinforcement Learning*, (NIPS2016, London, 2016).

[2] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, *Proximal Policy Optimization Algorithms*, (OpenAI, 2017).