Final Report: Direct Policy Gradient-Based Robot Control with Data Selection

Zhengkun Dai, Xiaoyu Bai
Department of ECE
Carnegie Mellon University
Pittsburgh, PA 15213
zhengkud@andrew.cmu.edu, xiaoyub@andrew.cmu.edu

Abstract

As the essence of reinforcement learning, function approximation has been a dominant approach especially with approximating value function to determine optimal policy. While an alternative approach of policy approximation has been proposed to directly use non-linear function such as deep neural network to approximate optimal policy without deducing value function which is sometimes intractable in complex cases such as robot control. In addition, we plan to explore the training data efficiency in our project. With comparison of using raw data and data with human interference, we want to study the effect of data selection on training through policy-based approximation.

1. Introduction

Although value function approximation remains an essential part in reinforcement learning and its empirical successes, it sometimes suffers from a fundamental limitation that the error they seek to minimize does not guarantee good performance for the resulting policy. The major limitation of value function approximation is listed as following. First, it only aims to find the optimal deterministic policy, whereas the optimal policy can sometimes be stochastic under the circumstance of non-MDP such as rock-paper-scissors. Second, some arbitrary small change in the estimated value of action can result in significant policy change, meaning the convergence is sometimes unstable. Third, continuous or high-dimensional action space such as robot control could make the learning quite inefficient. On the contrary, people have developed direct policy-base approximation which directly yields optimal policy [1-4].

In policy approximation method, people directly use non-linear approximators with its own parameter such as neural network or decision tree to directly compute the stochastic policy. Take neural network for instance, the input is still a representation of the state while the output can be action selection probability with softmax function. If we use θ to denote the parameter in the approximator, then the update rule can be represented as:

$$\theta \doteq \theta + \alpha \nabla_{\theta} J(\theta)$$

where α is the step size or learning rate and $J(\theta)$ is the objective function. Once the above can be achieved, then the policy usually can be assured to converge to a locally optimal policy in the performance measure J.

It has been proved that an unbiased estimate of the gradient can be obtained from experience using an approximate value function satisfying certain properties [1]. It is also proved that as far as optimizing the parameters of the MDP is concerned, we can deal with both expected discounted reward and the expected average reward [2]. Also in reference [2], it describes formally the gradient ascent approach to optimizing the performance of a parameterized Markov chain, and gives a closed-form expression for the gradient as a function of the gradient of the transition matrix.

Neural network has been widely used as a non-linear approximator of function approximation and proves itself to perform effectively especially with the technique of experience replay. Aside from experience replay which enables batching and randomly sampling, people have also proposed other techniques such as asynchronous executing with parallel agents [4]. Yet the main stream of data sampling is still experience replay, and we will stick to that in our implementation.

However policy-based approximation also has some drawbacks. For example, Monte-Carlo policy gradient (also named REINFORCE) has the disadvantage of slow convergence compared with value-based approximation as stated in the previous research work [5-6]. And before 2000, the policy-based function approximation draws relatively little attention.

Aiming at accelerating convergence in policy gradient method, we have combined experience replay with the data selection method that we have proposed in this work. In experience replay, many training data have been stored in the memory space and will be randomly sampled by batch. To make this process more efficient, we aim to give some critical data more priority to be selected during sampling. The way we judge whether a particular data is critical or not is dependent on the corresponding reward value. In conventional experience replay, people just sample data with a certain batch size N. By contrast, we sample the data for T times and sort the data by the absolute value of reward. Then use the fist N data to train the neural network. This means in most cases, training data are filtered and those comes with relatively larger reward absolute value have been selected. The larger T, the larger reward absolute values have been selected.

We have implemented this approach in OpenAI Gym Mujoco environment of “HalfCheetah-v1”. Both these environment are basic continuous control tasks running in a fast physics simulator. The episodic reward value indicates how far the agent has moved forward in one step. Total reward is calculated as the distance that the agent has moved in a fixed time span. Negative reward indicates that agent has moved backwards which is undesired.

2. Methods

2.1 Network Structure

For continuous action space, we use actor-critic architecture from reference [7]. In the environment of “HalfCheetah-v1”, the observation dimension is fairly low and this simplifies the network structure.

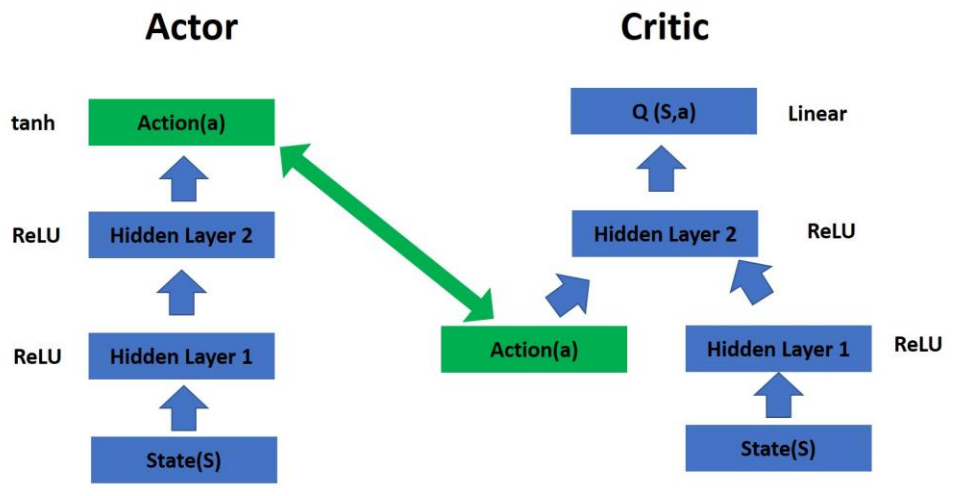


Fig. 1 Illustration of actor-critic structure.

The structure of actor network and critic network is shown as following. The purpose of the actor is choosing action according the state from environment, which is the policy network we need to learn. The purpose of the critic network is learning the Q (S, a) according to the Bellman equation. The network structure and hyperparameters are taken from reference [7]. For actor network, we use two hidden layers to construct the network, the activation function of the hidden layer is ReLU activation function. The activation function of the output layer is tanh, which limits the action value between (-1, 1). The critic network is also constructed with two hidden layers. The critic network takes two input: state(S) and action(a) at different depth. The first hidden layer provides some study of the state first, then the action is incorporated at the second layer to provide the output Q (S, a) at output layer. The actor and critic are connected by the action layer as shown in the graph.

The learning target is provided by the target network. The learning process flow is shown in the following graph. There are two actor networks and two critic networks. One actor network is online target actor network, which computes a from S, the other is target network which computes a' from S'. One critic network is online critic network, which computes Q (S, a), the other is target network which computes Q' (S', a'). The process is divided into two parts. One is indicated by the red arrows, which is the interaction of online actor with environment. Before feeding the action predicted by the online actor to the environment, a Gaussian noise is added to fully explore the possible action space. The other is the training process of the networks. The training process starts from batch sampling from replay memory. S' is fed to the target actor to get a', then S' and a' is fed to target critic network to compute Q' (S', a'). (S, a) is directly fed into online critic network to compute Q (S, a). The target value for online network is computed based in Bellman equation $Y = r + \gamma Q'(S', a')$. And loss is the L2 norm of Y - Q (S, a). Then loss is propagated to the online actor through the connection of action layer. After the training the targets networks are updated softly from the online network.

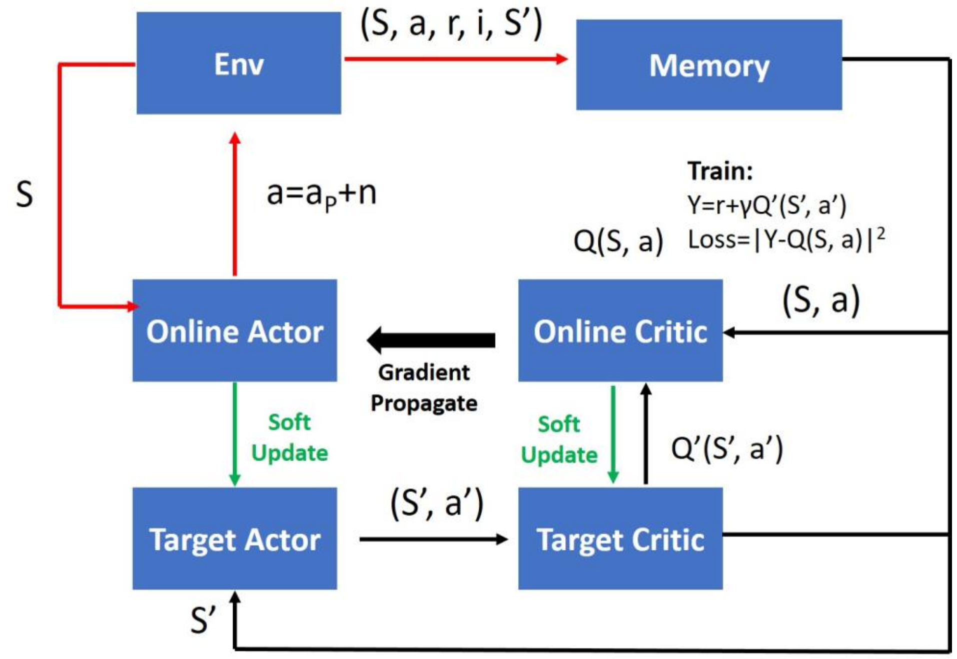


Fig. 2. Illustration of online network and target network interaction.

2.2 Data Selection Method

As mentioned in the introduction, we have also added our own feature which is data selection method in this implementation. Data selection method has been illustrated in Fig. 3. If the batch size is N, we randomly sample in the memory for T times so we can have N × T data. Then we sort this T folds of data according to the absolute value of reward. The top N data are selected for using. Therefore the selected data tends to have large reward values no matter positive and negative. However as training process goes by, the negative reward values will gradually disappear when filling memory pool.

One thing we realize in practice is that the number T cannot be too large. If T is larger than 3, then almost all the rewards are big values and they are much repeated. We want to select critical data which trains the network faster. However if repeated data are put into the network, then the training process will in turn becomes even slower. This is especially important in relatively simpler system when observation space and action space are not that big.

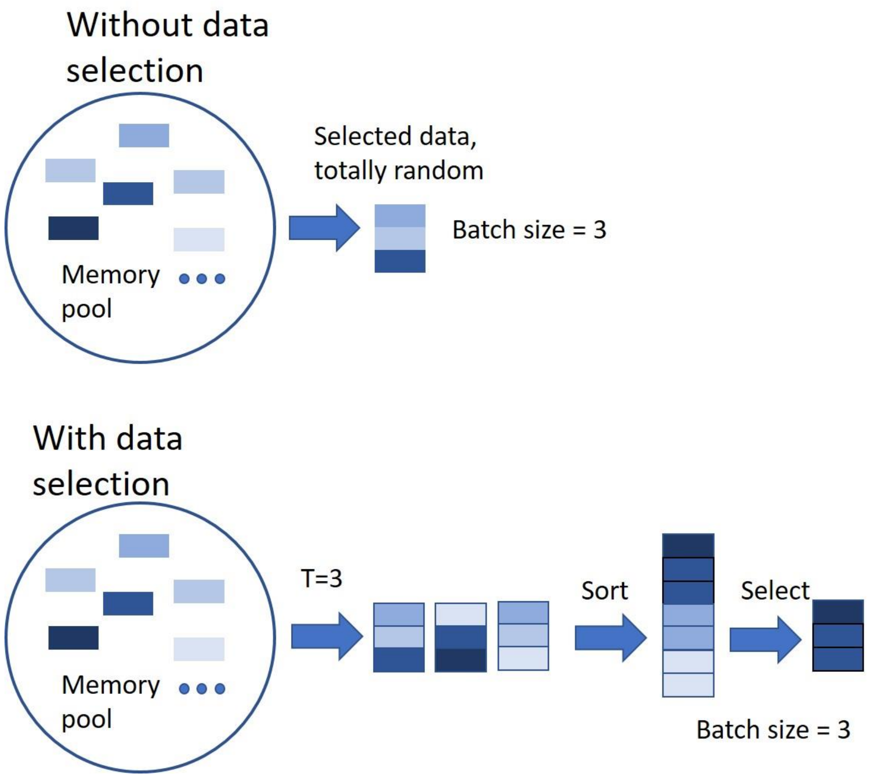


Fig. 3. Illustration of data selection process.

We also considered that filtering all training data with large reward values might be biased when training the network. To compromise, we also tried interval data selection method, meaning sampling with data selection and sampling without data selection take turns to be applied to get training data. This method will be more moderate since data with smaller reward will also be sampled but data with large reward value will be given more priority. This interval data selection method is illustrated in Fig. 4.

The motivation of applying data selection method according to reward magnitude follows the intuition that agents learns faster from movement with large reward. Especially at the beginning of training, agents are expected to learn from some movements which comes with immediate large reward. However this also has the risk of stuck into local minimum. Therefore we keeps a noise mechanism in our model following a Gaussian distribution. The noise mechanism that we use is very similar to reference [7].

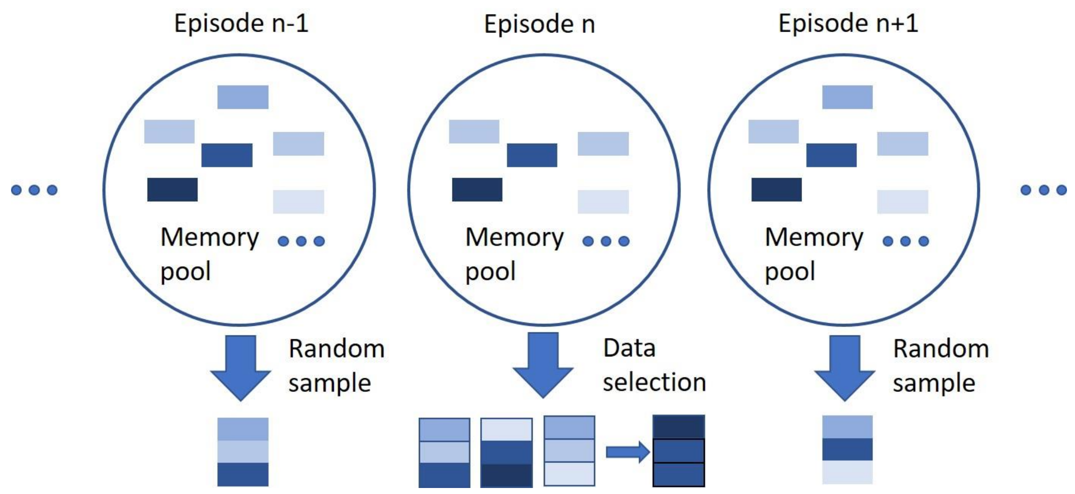


Fig. 4. Illustration of interval data selection.

3. Results

In our experiments, the parameter that we use are listed below:

Parameter	Value
Batch Size	64
Memory Size	500,000
Memory Initial Fill Percentage	10%
Maximum Interaction Episode	100

The actor and critic network structure is the same as reference [7].

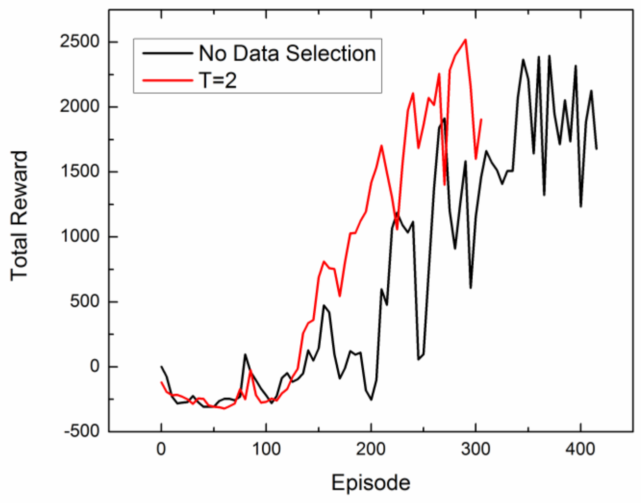


Fig. 4 Comparison between policy gradient method with and without data selection.

Fig. 4 demonstrates the policy gradient method convergence process with and without data selection. Although two processes converges at similar value after about 300 episodes, our new method with data selection according to reward absolute value yields a faster converging process. As mentioned before, T=2 means when sampling from memory pool, we sample twice of the batch size N and sort all the data according to reward absolute value. The top N data have been selected to train our neural network. The total reward is calculated by running the environment 10 times according to the trained policy and using the average total reward in this 10 episodes.

We also conducted experiments on interval data selection method that we mentioned before. Fig. 5 shows the comparison of T=2 with and without interval data selection. It is observed that interval data selection curve lies in between with and without data selection curves. And for T=2 interval and T=3 interval, the difference is not very obvious.

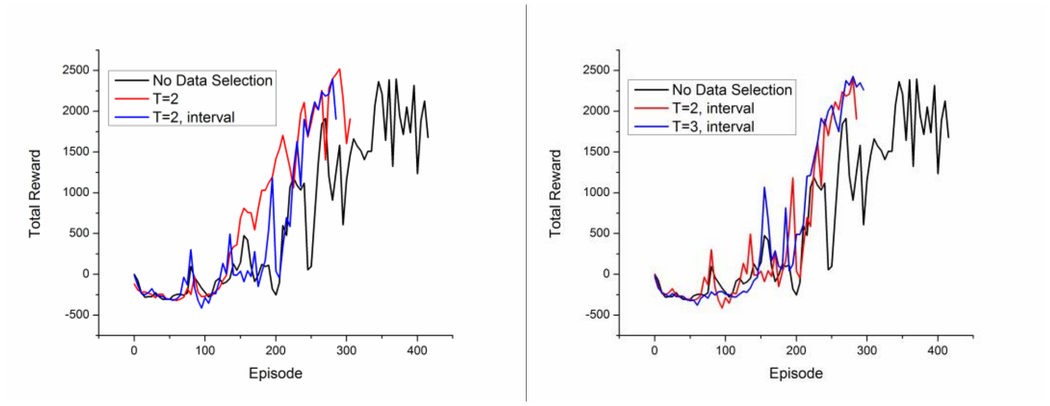
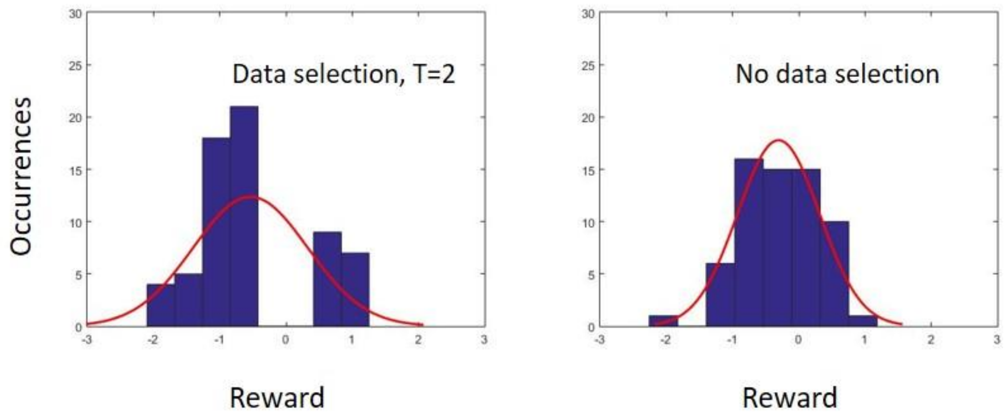


Fig. 5. Comparison of converging process of data selection and interval data selection with different T values.

To better understand the data selection process, we also investigate the reward value in one batch (batch size = 64). The histogram in Fig. 6 demonstrates the rewards distribution in 64 data samples which are sampled from with and without data selection.



Without data selection, the chosen data is more concentrated at smaller reward value. While with the implementation of data selection, data with rewards smaller than 0.5 are entirely filtered out as they are considered to be trivial data which are not efficient for training.

To visualize the training results, we also attached a few screen shots in this report from environment “HalfCheetah-v1” in Fig. 6. It is observed the agent first learns to stand and not move backwards. Then it stuck in a local minima at episode 50 where its head falls but legs pushes the body forward slowly. After about 100 episodes, it is able to behave some finer running movements to achieve much higher reward.

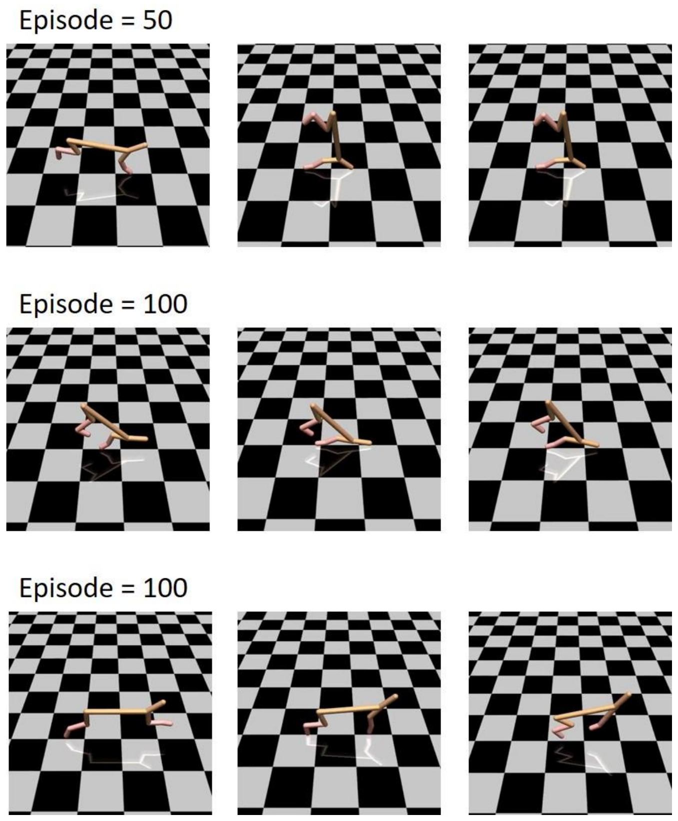


Fig. 6. Environment rendering of 2d half cheetah training.

4. Discussion

The novelty of our work is that we implemented pre-data selection in the existing policy gradient algorithm which accelerates the convergence as shown in Fig. 4. Our data selection method is according to the reward absolute value magnitude. Data with large rewards are more likely to be selected or in other words, they are given more priority when selecting. With all other parameters remain unchanged, simply adding data selection could make the training process more efficient especially at the beginning. Although the final performance cannot be further improved for policy gradient, data selection could possibly save some computation time.

In our work, policy gradient with data selection shows even better results after 300 training episodes than without data selection after 400 training episodes. Due to the limited computational resource we have (the entire training process is conducted by CPU but not GPU), we cannot wait for longer training time. But the observation so far proves the positive effect of this data pre-selection. And reward magnitude standard also proves to be an effective way to filter efficient data. However one thing we need to note is that the T value cannot be too large. When we did experiment on large T values such as 3, 4 and 5, the convergence is even slower. This is due to data repetition since only data with very large reward value have been left, and they are very repetitive which leads to the training to be even less efficient.

Due to the limited time and computation resource, we did not experiment on different policy gradient methods. In our program we only test on Actor-Critic method which cannot cover the entire policy-based reinforcement learning. So testing of this data selection on other policy-based method will be an interesting direction for our future work. And also, given enough time and computation resource, we would like to try other robot control environment especially in some complicated 3-d simulator. A systematic study about the hyper-parameter T tuning will be another interesting follow-up in our data selection implementation.

References

[1] Sutton, Richard S., et al. "Policy gradient methods for reinforcement learning with function approximation." *NIPS*. Vol. 99. 1999.

[2] Baxter, Jonathan, and Peter L. Bartlett. "Direct gradient-based reinforcement learning." *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*. Vol. 3. IEEE, 2000.

[3] Baxter, Jonathan, and Peter L. Bartlett. "Reinforcement learning in POMDP's via direct gradient ascent." *ICML*. 2000.

[4] Mnih, Volodymyr, et al. "Asynchronous methods for deep reinforcement learning." *International Conference on Machine Learning*. 2016.

[5] Williams, Ronald J. *Toward a theory of reinforcement-learning connectionist systems*. Northeastern University, 1988.

[6] Williams, Ronald J. "Simple statistical gradient-following algorithms for connectionist reinforcement learning." *Machine learning* 8.3-4 (1992): 229-256.

[7] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." *arXiv preprint arXiv:1509.02971* (2015).