Investigation on Deep Reinforcement Learning Methods for Classical Control Problems

Peng Xu
Department of Computer Science
University of Alberta
pxu4@ualberta.ca

Abstract

Through the study in CMPUT 609, I build a solid foundation and comprehensive overview of reinforcement learning. However, we didn't explore too much deeper into the recent advances in the research of reinforcement learning, such as deep reinforcement learning. In this paper, I explore and implement several popular deep reinforcement learning methods along with cross-entroy method. In addition, I apply these algorithms to a classical control problem, i.e. cart-pole problem and its variations to compare their performance. Due to the limited time, I do not tune the parameters carefully enough which may not reveal some algorithm's full power though. We can still get a overall impression about how these algorithms work in practice.

1 Introduction

Throughout the course, we investigate on various methods which focus on the use of value functions to organize and structure the search for good policies, including Dynamic Programming, Monte Carlo methods and TD methods. Furthermore, we explore a lot about linear value function approximators and the corresponding feature construction methods, such as tile coding. During the study of these methods, we form a good overview of reinforcement learning and understand three basic questions about reinforcement learning (what, how and why). In addition, we can solve some practical problems by combining these methods effectively, including one classical control problem, i.e. cart-pole problem. In this paper, I'll focus on the solutions of this problem and its variations, but use some newly emerging methods which are entitled with "Deep Reinforcement Learning".

With the huge success of DeepMind, deep reinforcement learning methods become more and more popular. But what is deep reinforcement learning at first? In general, they're just reinforcement learning methods using nonlinear function approximatiors which are neural networks commonly. And usually, they update their parameters with stochastic gradients. Extended from what we learned from the course, we can use neural networks to learn a value function, such as action-value function Q, which leads to deep Q-learning network (DQN) naturally [2]. Furthermore, we can also learn the policy function $\pi(a|s)$ directly which leads to policy gradient methods. Besides methods using SGD, I also implement a derivative free optimization approach called "cross-entropy method" which also optimizes the policy directly and perform amazingly well in quite a lot tasks [4].

The outline of this paper is as follows. Section 2 describes the cart-pole problem and one of its varations, i.e. multi-pole cart-pole problem. Section 3 introduces the methods I investigate in this paper, including cross-entropy method, policy gradient methods and deep Q-learning network. Section 4 gives the details and results of experiments I've done so far along with my understanding and analysis. Finally I summarize my work and potential further work in section 5.

2 Cart-pole Problems

The prototypical control problem that has been used as benchmark for reinforcement learning is cart-pole [6] (also knonw as pole balancing, broom balancer, or inverted pendulum problem). The problem involves balancing a pole hinged to a cart which is on a finite length track by exerting forces either left or right on the cart, Figure 1a.

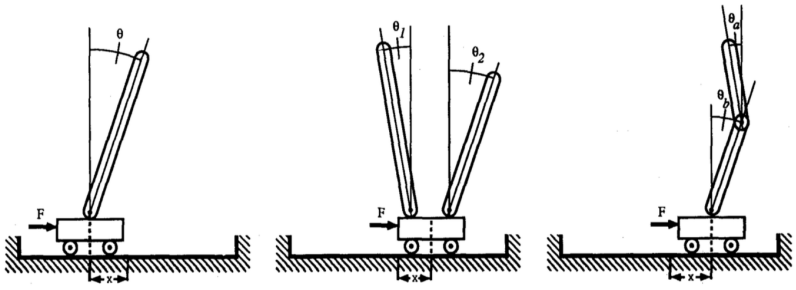


Figure 1: Cart-Pole Problems

An interesting and considerably more challenging of the cart-pole problem involves balancing more than one pole on the same cart, a two pole example is shown in Figure 1b. As long as the poles are of different lengths they will react differently to a force applied to the cart and can therefore be balanced simultaneously.

The final variation of the cart-pole problem considered a jointed pole, shown in Figure 1c. As with the multiple pole problem, as long as the lengths and therefore the natural frequencies of the poles are sufficiently different it was possible to balance the system.

3 Algorithms

3.1 Parameterized Policies

Before I introduce the algorithms I use, let's begin with the concept of parameterized policies which are quite different from the policies we learned from the course. Instead of consulting a value function, parameterized policies can select actions themselves. Formally, they are a family of policies indexed by parameter vector $\boldsymbol{\theta} \in \mathbb{R}^d$. Thus we write $\pi(a|s, \boldsymbol{\theta}) = \Pr\{A_t = a|S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}$ for the probability that action a is taken at time t given that the agent is in state s at time t with weight vector $\boldsymbol{\theta}$. They're analogous to classification or regression with input s, output a, e.g. for neural network stochastic policies.

3.2 Cross-Entropy Method (CEM)

I start with a simple but powerful method which is called Cross-Entropy method. Actually, CEM is a evolutionary algorithm without using neural networks and stochastic gradient descent. It just ignores all other information other than rewards R collected during episode. Its objective function can be seen as:

$$\max \mathbb{E}[R|\pi(\cdot, \boldsymbol{\theta})]$$

The pseudo-code is shown as follows in Figure 2. In practice, this simple method works amazingly even embarrassingly well, for that it outperforms many complicated algorithms designed for reinforcement learning tasks. Actually, it works quite good in my experiments without too much efforts to tune the hyperparameters.

In implementation, the policy function is simply a linear function of states.

Initialize $\mu \in \mathbb{R}^d, \sigma \in \mathbb{R}^d$
for iteration $= 1, 2, \ldots$ **do**
  Collect n samples of $\theta_i \sim N(\mu, \text{diag}(\sigma))$
  Perform a noisy evaluation $R_i \sim \theta_i$
  Select the top p% of samples (e.g. $p = 20$), which we'll call the **elite set**
  Fit a Gaussian distribution, with diagonal covariance, to the elite set, obtaining a new μ, σ.
end for
Return the final μ.

Figure 2: Cross-Entropy Method Algorithm

3.3 Policy Gradient Methods

We consider methods for learning the policy weights based on the gradient of some performance measure $\eta(\boldsymbol{\theta})$ with respect to the policy weights. The policy weights are updated with the following formula:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla \eta(\boldsymbol{\theta}_t)$$

All methods that follow this general schema we call *policy gradient methods*, whether or not they also learn an approximate value function.

The intuition of policy gradient methods is to make the good (maybe lucky) actions more probable after collecting a bunch of trajectories. And push the actions towards good action further and further. The policy gradient theorem gives a mathematical explanation for it, but I'll not give the details here. If interested, the proof can be found in the textbook.

3.3.1 REINFORCE with a Baseline

REINFORCE [3] is a policy gradient method based on Monte Carlo algorithm. The update formula for this algorithm is as follows:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \gamma^t \left(G_t - b(S_t)\right) \nabla_{\boldsymbol{\theta}} \log \pi(A_t | S_t, \boldsymbol{\theta}).$$

In implementation, the policy function is approximated by a neural network with one hidden layer and 20 hidden units. Tanh function is used as the activation function and the baseline function $b(S_t)$ is the averge rewards.

3.3.2 Actor-Critic Methods

Methods that learn approximations to both policy and value functions are often called actor-critic methods [1], where 'actor' is a reference to the learned policy, and 'critic' refers to the learned value function, usually a state-value function. One-step actor-critic methods replace the full return of REINFORCE with the one-step return as follows:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \left(G_t + \gamma v(S_{t+1}, \boldsymbol{w}) - v(S_t, \boldsymbol{w})\right) \nabla_{\boldsymbol{\theta}} \log \pi(A_t | S_t, \boldsymbol{\theta}).$$

In implementation, the policy function and the value function are both approximated by a neural network with one hidden layer and 20 hidden units. Also, Tanh function is used as the activation function.

3.4 Deep Q-Learning Network (DQN)

DQN is a natural extention to the Q-learning we learned in the course. The update formula for action-value function is as follows:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right].$$

The only difference is that DQN use a deep neural network to approximate the action-value function Q. In addition, DeepMind introduces various techniques to improve the performance of DQN, such as memory replay and double DQN [5].

In implementation, the action-value function Q is approximated by a neural network with three hidden layers and each with 128 hidden units. Rectifier (ReLU) function is used as the activation function and I also use the technique of memory replay in my implementation.

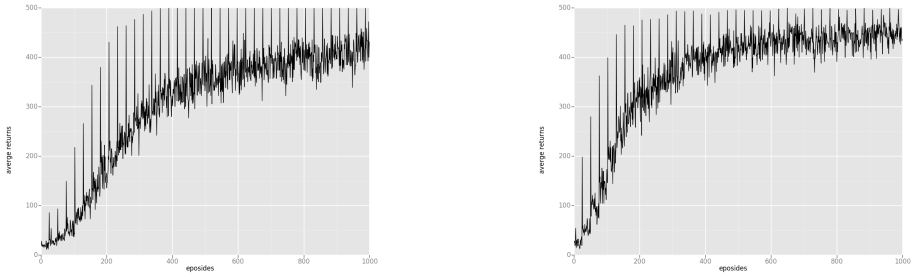
4 Experiments

4.1 Settings

For the environments, I use the **gym** package developed by *OpenAI*. However, there is only classical cart-pole environment. So I develop a new environment for multi-pole cart-pole under the framework of gym. The neural networks are implemented by **tensorflow** which is developed by *Google*.

To compare the performane of different algorithms, I run each algorithm 50 times independently. In each run, there are 1000 episodes in total.

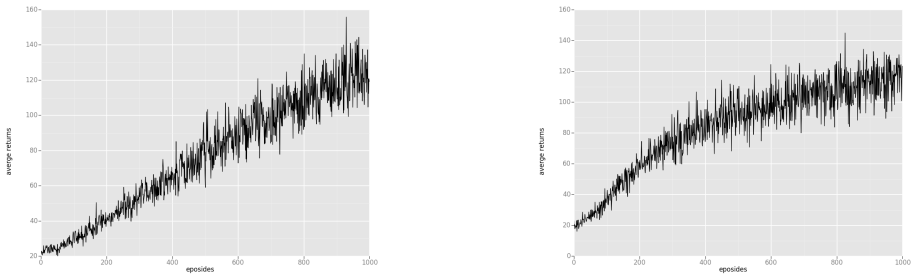
4.2 Results



(a) classical cart-pole (b) multiple-pole cart-pole

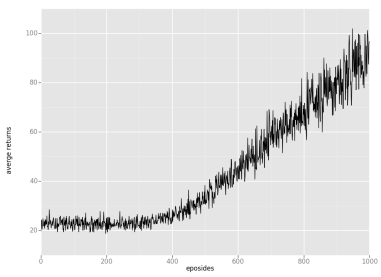
Figure 3: Average Returns over Episodes: Cross-Entropy Method

The reason why there is a periodical spikes in the Figure 3 is that CEM need to collect the returns R using sampled parameters for a certain batch size and then evaluated on the actual learned parameters. As a result, the spikes indicates the real performance of the algorithm and the other episodes can be seen as training episodes.

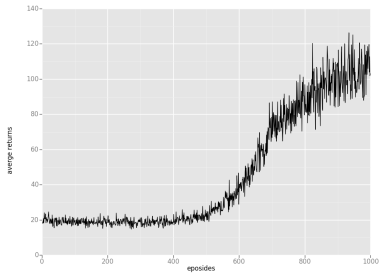


(a) classical cart-pole (b) multiple-pole cart-pole

Figure 4: Average Returns over Episodes: REINFORCE

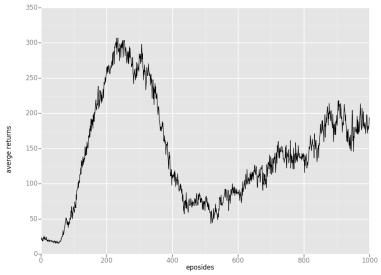


(a) classical cart-pole

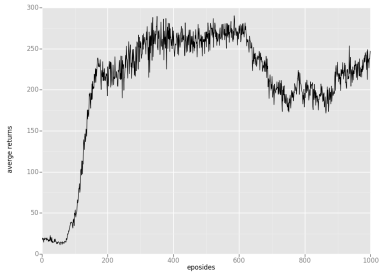


(b) multiple-pole cart-pole

Figure 5: Average Returns over Episodes: Actor-Critic



(a) classical cart-pole



(b) multiple-pole cart-pole

Figure 6: Average Returns over Episodes: Deep Q-Learning Network

From Figure 3-6, we can see that there are not too much difference between cart-pole and multi-pole cart-pole problems. The reason may be that the difficulty between the two problems is not that large. Just as the previous research show, CEM performs amazingly well in these problems from no matter the averged returns or the stability of the algorithms. DQN also perform quite well, but it's not that stable like CEM. In addition, training a deep neural networks is quite time-consuming. I spent several days to complete the 50 runs of DQN, but it only took several minutes to complete the experiments of CEM. Policy gradient methods perform relatively bad in these tasks, and it seems that there is no too much difference in performance between REINFORCE and Actor-Critic. However, their returns are steadily increasing when we have more episodes and I think they can achieve better results with more time.

One thing to be noted is that I didn't spend too much time to tune the parameters and architectures of the neural networks for the deep reinforcement learning methods. As a result, the performances here are just the first impression of how these algorithm behaves and may not reveal their full poentials.

5 Conclusion and Further Work

In this paper, I briefly describe the cart-pole problems and its variations. Then, I introduce three reinforment learning algorithms along with CEM algorithm and apply them to the cart-poles problems. When implementing these algorithms and doing the experiments, I have a basic understanding of these algorithms and how they work. By comparing their performances, I learn more about how they behave in the practical problems. In the future, I'll apply these algorithms to more tasks and gain deeper understanding of these algorithms. In addition, I plan to write a tutorial covering basic deep reinforcement learning algorithms and their applications on my own blog.

6 Acknowledgements

I would like to express my appreciation to professor Richard Sutton. Thanks for his time and efforts on guiding the whole process of my project.

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. *ICML*, 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] Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. *NIPS*, 99(1057–1063), 1999.

[4] István Szita and András Lörincz. Learning tetris using the noisy cross-entropy method. *Neural computation*, 18(12):2936–2941, 2006.

[5] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *AAAI*, 2015.

[6] Alexis P Wieland. Evolving neural network controllers for unstable systems. *Neural Networks*, 2:667–673, 1991.