Deep Reinforcement Learning for MiniGrid

Yi-Pei Chan
Department of Statistics
Columbia University
New York, NY, 10027
yc3700@columbia.edu

Abstract

In this project we plan to study the application of deep reinforcement learning (DRL) in solving problems in the MiniGrid gym environment and a classic control problem CartPole. Specifically, we plan to employ the proximal policy optimization (PPO) algorithm which is a modified version of actor-critic policy gradient method. We choose two testing environments from the MiniGrid environment and the CartPole environment from OpenAI Gym to verify our implementations.

1 Introduction

Recently, reinforcement learning (RL) (1) has been successful in a wide range of tasks, and has demonstrated human-level or super-human level of capabilities in video games (2; 3; 4; 5) and even mastering the game of Go (6; 7). These great achievements led us decide to further study RL in this project. In this project, we study the proximal policy optimization (PPO) algorithm (8), which is a variant of the policy gradient methods. The testing environments for this project include the classic control problem CartPole from the OpenAI Gym package (9) and two grid-world problems from the package MiniGrid (10). This report is organized as follows. In Section 2 we introduce the RL algorithm proximal policy optimization (PPO) we plan to use in this project. In Section 3 we describe the experimental settings, including states, actions, and rewards, and the numerical results. In Section 4 we discuss our results and several potential future directions.

2 Method

2.1 Proximal Policy Optimization (PPO)

In the policy gradients method, we optimize the policy according to the *policy loss* $L_{\text{policy}}(\theta) = \mathbb{E}_t[-\log \pi\left(a_t \mid st; \theta\right)]$ via gradient descent. However, the training itself may suffer from instabilities. If the step size of policy update is too small, the training process would be too slow. On the other hand, if the step size is too high, there will be a high variance in the training. The proximal policy optimization (PPO) (8) fixes this problem by limiting the policy update step size at each training step. The PPO introduces the loss function called *clipped surrogate loss function* that will constraint the policy change a a small range with the help of a clip. Consider the ratio between the probability of action a_t under current policy and the probability under previous policy $q_t(\theta) = \frac{\pi(a_t|s_t;\theta)}{\pi(a_t|s_t;\theta_{\text{old}})}$. If $q_t(\theta) > 1$, it means the action a_t is with higher probability in the current policy than in the old one. And if $0 < q_t(\theta) < 1$, it means that the action a_t is less probable in the current policy than in the old one. Our new loss function can then be defined as $L_{\text{policy}}(\theta) = \mathbb{E}_t[q_t(\theta)A_t] = \mathbb{E}_t[\frac{\pi(a_t|s_t;\theta)}{\pi(a_t|s_t;\theta_{\text{old}})}A_t]$, where $A_t = R_t - V(s_t;\theta)$ is the advantage function. However, if the action under current policy is much more probable than in the previous policy, the ratio q_t may be large, leading to a large policy update step. To circumvent this problem, the original PPO algorithm (8) adds a

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

constraint on the ratio, which can only be in the range 0.8 to 1.2. The modified loss function is now $L_{\text{policy}}(\theta) = \mathbb{E}_t[-min(q_t A_t, clip(q_t, 1-C, 1+C)A_t)]$ where the C is the clip hyperparameter (common choice is 0.2). Finally, the value loss and entropy bonus are added into the total loss function as usual: $L(\theta) = L_{\text{policy}} + c_1 L_{\text{value}} - c_2 H$ where $L_{\text{value}} = \mathbb{E}_t[\|R_t - V(s_t;\theta)\|^2]$ is the value loss and $H = \mathbb{E}_t[H_t] = \mathbb{E}_t[-\sum_j \pi(a_j \mid s_t;\theta)\log(\pi(a_j \mid s_t;\theta))]$ is the entropy bonus which is to encourage exploration.

3 Experiments and Results

3.1 CartPole Environment

In the first experiment, we plan to study the classic control problem `CartPole-v0` environment provided by the OpenAI Gym (9). The environment is shown in the Figure1. In this environment, a pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The pendulum starts upright, and the goal is to prevent it from falling over by increasing and reducing the cart's velocity. The RL agent needs to output the appropriate action according to the observation it receives at each timestep.

The cart-pole environment mapping is:

- Observation: a four dimensional vector s_t representing the cart position, cart velocity, pole angle, pole velocity at tip.
- Action: there are two actions $+1$, -1 in the action space.
- Reward: A reward of $+1$ is provided for every timestep that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.

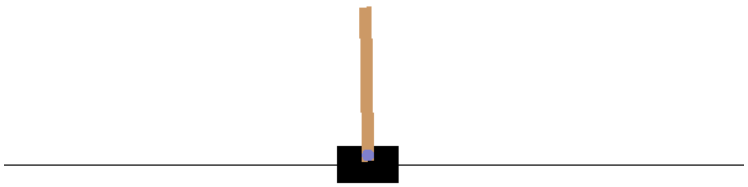

Figure 1: **CartPole environment in OpenAI Gym.** Screenshot from the OpenAI Gym (9).

3.2 MiniGrid Environment

We plan to study the PPO algorithm in solving problems provided by the environment MiniGrid (10). This environment follows the standard OpenAI Gym (9) API thus it is a good choice for the RL benchmarking. In this environment, the RL agent will receive $7 \times 7 \times 3 = 147$ dimensional vector for observation and would need to determine the action from action space $\mathcal{A}$ in which there are 6 possibilities.

The MiniGrid environment mapping is:

- Observation: a 147 dimensional vector s_t
- Action: there are six actions $0, \cdots, 6$ in the action space. Each representing the following,

  - Turn left
  - Turn right
  - Move forward
  - Pick up an object
  - Drop the object being carried
  - Toggle (open doors, interact with objects)
- Reward: Get 1 when reaching the goal with small penalty subtracted from the return for the number of steps to reach the goal.

We plan to study the following two testing cases for this project. They are the *empty-env-6x6* as shown in Figure3 and the *empty-env-8x8* as shown in Figure5. In these two environments, the RL is expected to go from the starting point (upper left in the figure) to the destination (lower right green box). The reward scheme in this MiniGrid environment is as follows. In each of the step, the agent will receive a small negative reward as penalty. The ideas behind this design is to encourage the agent to take the shortest possible path. The agent gets reward of 1 when it succeed, and 0 for failure.

The RL agents will be implemented with a recent released python package `TF-Agents` (11).

3.3 Results and Performance Analysis

Our implementation is based on the official tutorial jupyter notebook on REINFORCE agent provided by the `TF-Agent` (12). The design of neural networks are inspired by the open-source project (13). In the following three experiments, we use the same hyperparameters of the optimizer and the neural network architecture. We use the Adam (14) optimizer with learning rate $\eta = 0.01$. The PPO setting follows the default values on `TF-Agent`. The actor network and the critic network are the single hidden layer network with 100 neurons. The input layer depends on the environment. For example, in the CartPole environment, the input is 4-dim while in the MiniGrid environments, the input is 147-dim.

3.3.1 CartPole

The first experiment we perform is the `CartPole-v0`. In this environment, the maximum steps allowed is 200, therefore 200 is the maximum value of total return. The training results are in Figure2. We can see that after around 20 training steps, the agent successfully reaches the optimal policy. In addition, the learned policy is stable and no sub-optimal policy is observed in the experiment.

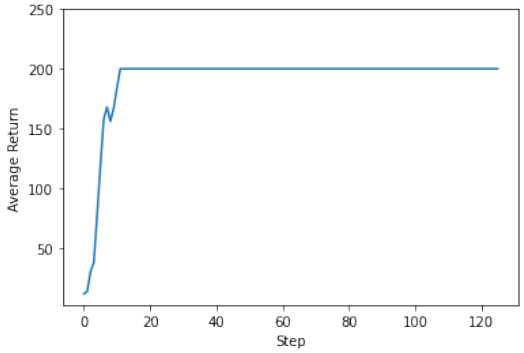

Figure 2: **Results: CartPole in OpenAI Gym.** Results: PPO on CartPole.

3.3.2 MiniGrid EmptyEnv-6x6

The second experiment we perform is the `MiniGrid-Empty-6x6-v0`. The environment is shown in Figure3. In this environment, the agent is expected to find the shortest path from upper left corner to the lower right corner. As mentioned before, there is a small negative reward on each step used to encourage the agent to use fewer steps. The training result is in the Figure4. In training steps between 20 and 80, the agent sometimes reach the optimal while sometimes does not. One of the possible

reason is that in PPO training, we rollout the episode to collect the trajectories. Before the agent actually converges, there is possibility that the many episode rollouts are sub-optimal, making the training non-stable. When we test the agent at this stage, we may observe that the agent does not perform very well. It can be observed that after 80 training steps, the agent reaches the optimal policy stably.

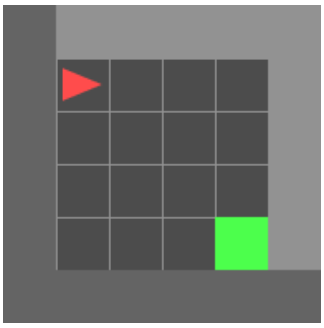

Figure 3: **Empty environment** `MiniGrid-Empty-6x6-v0` **in MiniGrid gym.** Screenshot from the MiniGrid env (10).

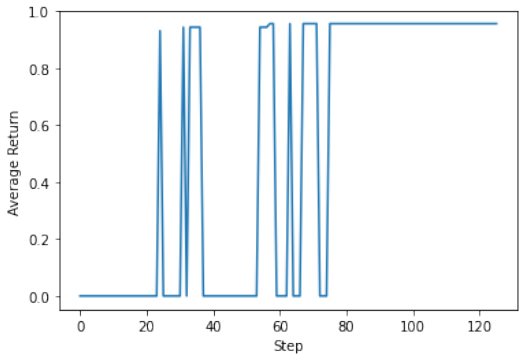

Figure 4: **Results: Empty environment** `MiniGrid-Empty-6x6-v0` **in MiniGrid gym.** Results: PPO on MiniGrid Empty Env 6x6.

3.3.3 MiniGrid EmptyEnv-8x8

The third experiment we perform is the `MiniGrid-Empty-8x8-v0`. The environment is shown in Figure5. Similar to the previous 6x6 environment, the agent is expected to find the shortest path from upper left corner to the lower right corner. As mentioned before, there is a small negative reward on each step to encourage the agent to use fewer steps. This environment is more difficult than the previous one since there is a larger search space for the agent to explore. The training results of this experiment is shown in Figure6. We can see that the RL agent converges to optimal policy much slower than the previous 6x6 environment.

4 Discussion

4.1 More Complex Environments

RL agent is in general hard to train, especially with large state and action spaces and sophisticated neural networks architectures. For example, playing video game is not easy. The state or observation is in raw-pixel format which needs considerable processing to extract relevant information for inference. One of the notable example is the Atari games. To successfully train a RL agent on such environment, it may require more power encoders and more expressive policy networks. Take other more challenging environments in `MiniGrid` for example, the `Unlock` environment expects

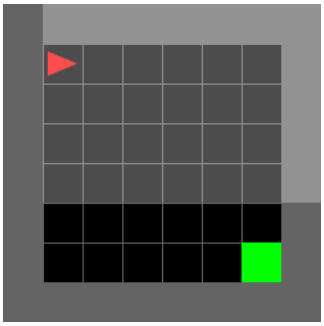

Figure 5: **Empty environment** `MiniGrid-Empty-8x8-v0` **in MiniGrid gym.** Screenshot from the MiniGrid env (10).

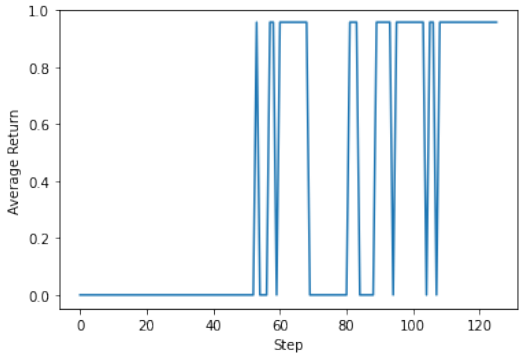

Figure 6: **Results: Empty environment** `MiniGrid-Empty-8x8-v0` **in MiniGrid gym.** Results: PPO on MiniGrid Empty Env 8x8.

the agent to learn the fact that it needs to first pick the key and then unlock the door. What makes this environment difficult is that the location of the key will change each episode. To successfully solve this task, the RL agent needs to learn the fact that it needs to first search for the key and then choose the shortest path to the door after picking up the key. There are some potential solutions for this problem, for example, the RL agent can be equipped with a recurrent policy to keep a temporal memory in an episode. We will discuss this later.

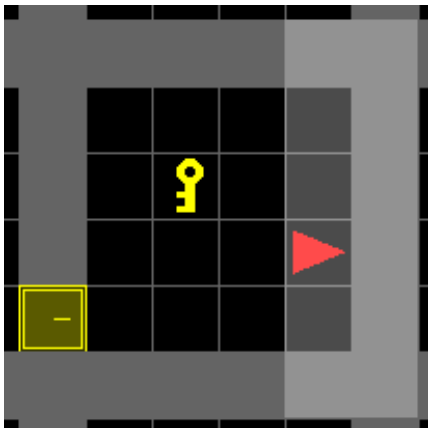

Figure 7: `Unlock` **environment in MiniGrid gym.**

4.2 Convolutional Encoder

To deal with more complex environment like video game or 3D navigation, it is inevitable for the RL agent to process the raw pixel values of each frame directly. It is non-trivial to extract information from such noisy observation. Convolutional neural networks (CNN) (15; 16), one of the greatest achievements in modern machine learning, may be helpful in such circumstances. Indeed, recent progress in game-playing RL has largely depended on CNN to encode the state or observation. For example, the famous results of AlphaGo (7; 6) use the CNN to compress the information of Go state. Then, highly successful RL applications in playing Atari video games are also employing CNNs in the RL agents (2; 3; 4). These sophisticated RL implementations however are computationally intensive and is not practical for personal computers. Nevertheless we expect our experiments in this project can be extended to raw-pixel values given the sufficient computing capabilities. For example, the RL agents are expected to successfully solve the same tasks while seeing only the pixel values on each time-step. We also expect that the RL agent would converge to optimal policy slower as learning from raw-pixel values is considered much harder.

4.3 Recurrent Policy

In this work, we use the MLP to serve as the state encoder, actor and critic. Although this setting is rather enough for our purpose, it can be expected that such simple function approximators may not be with enough capacity when dealing more complex problems. For example, certain environments may need the RL agent to keep temporal memories for a long range. For example, the work (17; 18) needs the RL agent to learn how to learn. In that work the agent is equipped with a long short-term memory (LSTM) which is a variant of recurrent neural networks (RNN) capable of keeping long-range information. In our implementation on PPO algorithm, it is also possible to optimize the agent with recurrent networks. For example, it may be possible to train a RL agent to solve an environment in which the agent needs to pick up different objects in a unknown sequence in order to finally reach the goal. We reserve this interesting direction for future study.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2 2015.

[3] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *arXiv preprint arXiv:1911.08265*, 2019.

[4] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, D. Guo, and C. Blundell, "Agent57: Outperforming the atari human benchmark," *arXiv preprint arXiv:2003.13350*, 2020.

[5] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney, "Recurrent experience replay in distributed reinforcement learning," in *International conference on learning representations*, 2018.

[6] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[7] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[9] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[10] M. Chevalier-Boisvert, L. Willems, and S. Pal, "Minimalistic gridworld environment for openai gym." `https://github.com/maximecb/gym-minigrid`, 2018.

[11] S. Guadarrama, A. Korattikara, O. Ramirez, P. Castro, E. Holly, S. Fishman, K. Wang, E. Gonina, N. Wu, E. Kokiopoulou, L. Sbaiz, J. Smith, G. Bartók, J. Berent, C. Harris, V. Vanhoucke, and E. Brevdo, "TF-Agents: A library for reinforcement learning in tensorflow." `https://github.com/tensorflow/agents`, 2018. [Online; accessed 25-June-2019].

[12] "REINFORCE toturial." `https://github.com/tensorflow/agents/blob/master/docs/tutorials/6_reinforce_tutorial.ipynb`. Accessed: 2020-12-19.

[13] "RL Starter Files." `https://github.com/lcswillems/rl-starter-files`. Accessed: 2020-12-19.

[14] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[16] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, vol. 1. MIT press Cambridge, 2016.

[17] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick, "Learning to reinforcement learn," *arXiv preprint arXiv:1611.05763*, 2016.

[18] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "Rl ^{2}: Fast reinforcement learning via slow reinforcement learning," *arXiv preprint arXiv:1611.02779*, 2016.