**Gaming Reinforcement Learning: Value-based vs Policy-based**

**(Proximal Policy Optimization Part)**

by

Nan Zhenghan (Student Name)

1098476 (Student ID Number)

June, 2022

# Introduction

## 1.1 Reinforcement Learning Review

Reinforcement learning (RL) that intends to maximize the rewards by evolving the policy of agents models the environment in terms of the Markov Decision Process (MDP). MDP, whose output is nonunique deterministic optimal policy, is used to model the decision-making in a stochastic environment. There are two families in RL exploring the environment, which are model-based and model-free. The essential difference between them is whether the state-transition and reward probability functions are known. If they are known, the problems are relatively simple due to the finite number of states, thus being easy for agents to study the environment.

As for model-based families, there exist two categories. One is that the actual model is known, and the other is establishing a simulated model in light of the past experience from the interaction between agents and the environment. The simulated model will play a vital role as the real model in generating the simulated trajectories. In both cases, q values that depict the value of state and action pair are stored in a table by virtue of a finite number of states. Consequently, the tabular methods value iteration (model the environment) and policy iteration (model the agents) can find the best decision chain by optimizing the Bellman expectation equation and Bellman optimality equation. It is noteworthy that dynamic programming (DP) is applied to evaluate the policy since the model or equation is known. Then the value and policy iteration will update the policy according to the evaluation. However, model-free, they are not known, so Monte-Carlo sampling (MC) and Temporal-Difference learning (TD) are desirable to approximate the action and state values, thus evaluating the policy. More discussion of the relationship among them will be presented in the following paragraphs.

Unfortunately, most real-world problems are NP-Hard problems because of an infinite number of states or the complexity of modeling the environment according to MDP. For instance, Game of Go, Atari Game, and Portfolio Management. That’s why model free is preferable in practice. Firstly, it’s important to discuss the policy evaluation, based on which value-based and policy-based algorithms will update the policy. As mentioned before, both MC and TD are applied to evaluate the policy for model free. The MC method samples many trajectories, computes the actual returns for all the trajectories and then averages them to evaluate the policy. The Epsilon-Greedy will trade off the exploration and exploitation of the MC method. As for TD, it can be regarded as the combination of MC and DP, which can learn from incomplete sequences, whereas MC must sample complete trajectories. That’s why TD allows the agents to get an instant reward so that it can update the policy faster. TD consists of on-policy and off-policy. Sarsa, which uses the same policy to collect trajectories and improve policy, is an excellent example of on-policy. Also, whether the policy to collect samples and improve policy is the same or not is the criteria to distinguish on-policy and off-policy. In off-policy, q-learning is a wonderful instance, exploring the environment much more radically than Sarsa. Therefore, q-learning outperforms Sarsa in most cases. In state-of-the-art, off-policy is preferable to on-policy because off-policy has higher sample efficiency. In other words, off-policy uses two different policies, so the policy update will not be interfered with by data collection. Consequently, off-policy can take advantage of history experience or even the samples from other agents. That’s why q-learning and deep q-learning set the replay buffer to learn all previous trajectories. In addition, off-policy allows agents to reuse the samples, which can maximize the value of each sample. The agent can progress a lot with less sample, referring to a high sample efficiency. (Enhancing Sample Efficiency in Reinforcement Learning with Nonparametric Methods | NVIDIA Technical Blog, n.d.) To sum up, DP is applied only when the model or equation is known, whereas MC and TD techniques are suitable for model-free algorithms. Furthermore, TD is equivalent to MC+DP. DP determines the width of the update because it must consider all previous states in accordance with the recursive Bellman equation. Nonetheless, MC determines the depth of update because it samples complete trajectories. If both width and depth attributes are maximized, it will downgrade to an exhaustive search.

After understanding how to evaluate policy in model-free, this section will introduce how to update policy in terms of the evaluation. For most practical problems, the predominant difficulty is the infinite number of states, so it’s tough to model either the environment by value or the agent actions by the policy. Therefore, researchers proposed an intelligent approach to approximate the true value and agent actions, which is the neural network. Thus, the key point of the value-based and policy-based is to learn parameterized values or policies so that the unknown samples can be approximated with the trained function. (Bootcamp Summer 2020 Week 4 – Policy Iteration and Policy Gradient, n.d.) For both, it’s significant to design the function parametrizing the value and policy. For instance, the dependent variable of the value function can be the value of a state or action, and the independent variables can be the states and actions. Policy, however, an action probability distribution can be parametrized by Softmax or Gaussian Distribution. After designing the function, the loss function can be determined as well with MC and TD methods, as mentioned before, to evaluate the parametrized policy. Then the neural network will optimize the loss function with front and backpropagation. Interestingly, there are two kinds of optimization methods in the neural network called derivative-based and derivative-free since the objective function is sometimes not differentiable. Eventually, the parameter θ can be acquired to construct either value or policy function to predict unknown samples.

Nevertheless, the model-free has the problem of unstable training trace because of function approximation, bootstrapping, and off-policy training. Firstly, as discussed before, model-free tries to approximate unknown samples with function, bringing the approximation errors to the training. Secondly, since some update rule is based on DP or TD instead of exclusively relying on actual rewards and returns provided by MC, the update targets include existing estimates. In other words, it’s biased to use current estimates to estimate themselves, which will trap to overfit issue. Lastly, the behavior policy used to collect samples and update the target policy is different in off-policy, so it will generate noise during the training.

Besides, the results of model-free or even the whole RL domain have high variance. Whether we can find the optimal solutions based on problem size. RL typically can’t guarantee that we can find the optimal solution attributed to numerous states.

## Super Mario Bros with Proximal Policy Optimization (PPO)

This project intends to use one of the policy-based methods called Proximal Policy Optimization (PPO) to train the agent in Super Mario Bros. Although training an agent to learn how to play a game is meaningless at first glance, the trained agent can be generalized to other real-world problems. For instance, a popular multi-player collaborative game called League of Legends (LOL) has similar logic as intelligent cities. Hence, the model trained with RL in LOL takes effect when solving traffic management systems. This paper will first introduce the environment of the game. Secondly, illustrate the mechanism of the PPO model. Thirdly, present the code. Finally, discuss the results.

# Environment (Problem) Analysis

1. Game: gym-super-mario-bros 7.3.2 in Open AI Gym Platform
2. Simulator: Nes-py to build a virtual joypad for python to be able to play the game
3. RL Algorithm Library: Stable-Baseline3
4. Action:

Here, a library Nes-py will be utilized to project the action space (Figure. 1.) into the combination of buttons on the joypad. (Figure. 2.) As such, there are seven simple movements. The advantage is speeding up the training by decreasing the initial 256 discrete actions into only seven units.

图形用户界面, 文本, 应用程序

描述已自动生成

Figure. 1. Action Space before Projection

文本

描述已自动生成

Figure. 2. Joypad Action Combination

图片包含 图形用户界面

描述已自动生成

Figure. 3. Action Space after Projection

There are six buttons: up, down, right, left, A, and B, on the joypad. Up, down, right, and left buttons are used to control the agent's direction. As for A, it represents Jump action, and B represents Rush action. For instance, the action “Right, A, B” means jump one time and rush towards the right.

1. State:

Figure. 4. Shows how state was shaped in this game. The shape of state is (240，256，3) the first dimension refers to height; the second dimension refers to the length; the third one refers to RGB channels. Therefore, each frame of the game is regarded as the state. And the Convolutional Neural Network (CNN) will train the model with these frames.

图形用户界面, 文本, 应用程序

描述已自动生成

Figure. 4. State

1. Reward:

图形用户界面, 文本, 应用程序

描述已自动生成

Figure. 5. Default Reward Function

(Source: (*Gym-Super-Mario-Bros · PyPI*, n.d.))

Figure. 5. Shows the default reward function, consisting of three indicators that are velocity (v), time (c) and death (d). The formula of reward function is

More specifically, the sign of each variable will influence the value of the reward function. The higher the reward function, the better the policy. The variable ‘v’ is positive when the agent moves towards the right. Otherwise, its left movement will be penalized. The design is reasonable because the termination locates on the right. The second variable, c, refers to the time penalty. The agent can prevent punishment by minimizing the time so that the agent can pass the game as fast as possible. The third variable, d, quantifies the impact of death on the reward. The agent will die when it encounters the trap or monsters.

1. Game Parameters

表格

描述已自动生成

Figure. 6. Game Parameters

(Source: (*Gym-Super-Mario-Bros · PyPI*, n.d.))

These parameters are vital when customizing the reward functions.

# Model Description (PPO)

The PPO, a policy gradient model, will be introduced in this section. PPO was proposed to improve standard policy gradient through multiple epochs of minibatch updates. Also, it speeds up the training of Trust Region Policy Optimization (TRPO) (Schulman, Levine, et al., n.d.) with the unconstrained objective function, resulting in only the first-order optimization. (Schulman, Wolski, et al., n.d.)

## 3.1 Policy Gradient

The policy Gradient technique utilized gradient descent to address the objective function parameterizing the policy with probability distribution concerning expected return (long-term cumulative rewards). The target of the policy gradient is to acquire the optimal distribution function parameters θ, which can optimize the objective functions. Here is the formula.

This informs that the optimal parameter endeavors to maximize the expected returns of a random trajectories sample under a specific policy with parameters θ. And the process of sampling is exactly the idea of MC, and the algorithm wants to estimate the policy based on samples. What’s more, the policy gradient has many forms

The primary difference among them is how to formalize the sample rewards. The objective function reveals that the policy performs well when the expectation is high in any forms listed here.

However, there are some limitations: step size, determination difficulty, and low sample efficiency. First, since the policy gradient is an on-policy technique, the policy update will directly affect the subsequent sampling. Suppose the update step size is too large, the policy will probably collapse into an awful policy, and the successive sample will contain a high bias. More significantly, it’s almost impossible to recover from the bad policy. As a result, it’s crucial to constrain the step size properly. Secondly, because it’s an on-policy approach, the collected sample isn’t reusable. In other words, each environment sample only demonstrates the value of one gradient step update. In fact, useful information from samples hasn’t been exploited yet. That’s why it has low sample efficiency.

## Natural Policy Gradient

An enhanced version of the policy gradient comes up to make the model update much more stable, which is Natural Policy Gradient. This technique introduces the fisher information matrix into the standard policy gradient, which can describe the curvature of the model parameterization.

Here F is the Fisher information matrix, which is also the second-order derivative of the KL-divergence measuring the difference between two distributions. Consequently, the update process will be model-invariant and stable.

## 3.3 Trust Region Policy Optimization (TRPO)

As for TRPO, it improves the idea of natural policy gradient by adding importance sampling to transfer the original on-policy to off-policy. As such, it tackles the problem of low sample efficiency. Furthermore, the trust-region automatically determines the update step size. Accordingly, the learning rate doesn’t have to be defined manually, solving the difficulty of step size selection. Here is the pseudo-code::

文本, 信件

描述已自动生成

Source: (Achiam & Achiam Berkeley, 2017)

Firstly, the model defines a novel surrogate loss function with importance sampling.

The old policy, the denominator, samples the trajectories from the environment, according to which the numerator (new policy) updates. This approach transfers on-policy to off-policy by differentiating between the new and old policy. Then the probability ratio of new policy and old policy is introduced to recalibrate the results.

Secondly, a trust-region is introduced to constrain the step size, formed as KL divergence.

The reason why it takes KL divergence instead of Euclidean distance as a measurement of the difference between the new and old policy is that it’s a united and problem-independent indicator.

Thirdly, to efficiently solve such a constrained optimization problem, the linear-quadratic approximation is applied to solve it as an unconstrained optimization problem. Then it will get the same results as the natural gradient step mentioned before.

Fourthly, to avoid calculating the time-consuming fisher information matrix, the conjugate gradient method is applied to approximate the value of the matrix. Finally, we get the TRPO.

## 3.4 Proximal Policy Optimization (PPO)

Despite the outstanding performance of TRPO in state of the art, one of its predominant drawbacks is high time complexity when calculating the Fisher information matrix. Thus, PPO is proposed to reduce the time complexity. PPO, simplifying TRPO, has a comparable performance with lower time complexity than TRPO. Mainly, PPO aborts the constrained form. Instead, it designs an unconstrained objective function to adaptively adjust the weight of the KL penalty to update the parameters. Here is the surrogate function with the KL penalty.

Without generating a fisher information matrix from a constrained optimization problem, PPO tremendously reduces the computational time. The more widespread form of surrogate function is the clipping function because it empirically has a better performance. Here is the formula.

When the advantage is positive, the action is encouraged. However, to avoid a large update, the threshold is set to clip the out of bounds value.

When the advantage is negative, the objective function should add penalty. Similarly, a threshold is set to clip the out of bounds value.

In Figure 7, the parallel line is the boundary limiting the update step size.

箱线图

中度可信度描述已自动生成

Figure. 7. Clipping

(Souce:(Schulman, Wolski, et al., n.d.))

Here is the pseudo code of PPO with clipping.

文本, 信件

描述已自动生成

Source: (*Proximal Policy Optimization — Spinning Up Documentation*, n.d.)

## 3.5 PPO Exploration & Exploitation

PPO algorithm explores stochastic environment according to evolving policy which randomly samples an action for each state initially. The randomness relies on both initial conditions and the course of training. More specifically, the update date rule will encourage the agent to exploit more the rewards that have been found. Meanwhile, the exploration rate will decrease along with the increase in exploitation. In PPO with clipping, the stochastic gradient ascent with Adam will trade-off the exploration and exploitation of PPO.

# Code Explanation

1. Set up Mario environment

First of all, a customized environment is defined as the following. Mainly, we modify the default reward function. Besides velocity, time, and death, the current reward function also includes the score, which is the cumulative in-game score as indicated in Figure. 6. Beyond that, if the agent gets the flag, meaning that it passes the game, it’ll get an enormous reward. Otherwise, it’ll receive a penalty of 50 for each failure. The step() function is used to random sample an action to interact with the environment, so the reward is defined inside this function. For each iteration, the agent only has three chances. If he dies, the reset() function will Reset all the environments. The render() is used to render the game, which is optional and doesn’t affect the training. Finally, the close() function is used to close the environment.

文本

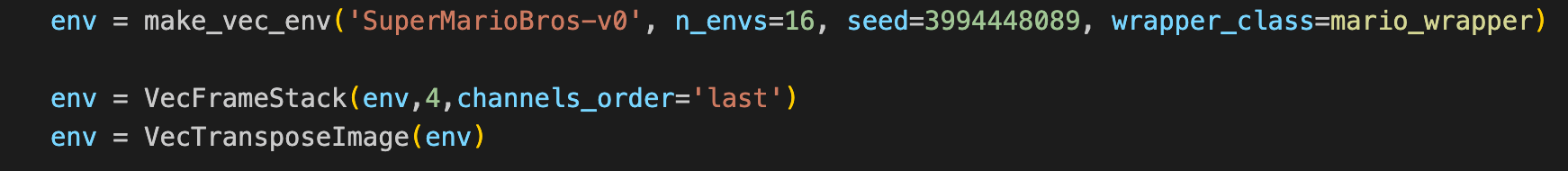
描述已自动生成

Then the customized environment is encapsulated into a wrapper for the convenience of the subsequent processing.

文本

描述已自动生成

1. Preprocess the environment



This part intends to vectorize the environment to decrease the training time. According to the stable-baseline3 document example, the vectorized environment has two benefits. One is that “agent experience can be collected more quickly”, and the other is that “The experience will contain a more diverse range of states, it usually improves exploration”. (Rl-Tutorial-Jnrr19/3\_multiprocessing.Ipynb at Sb3 · Araffin/Rl-Tutorial-Jnrr19, n.d.) After that, the VecFrameStack function is applied to stack several frames so that the agent can understand the dynamic of movements.

1. Tune hyperparameter

In this part, we use the optuna tool to tune the hyperparameters, but it needs at least several days to tune the hyperparameter, so we didn’t implement this part.

1. Set up callback

文本

描述已自动生成

“A callback is a set of functions that will be called at given stages of the training procedure. You can use callbacks to access internal state of the RL model during training. It allows one to do monitoring, auto saving, model manipulation, progress bars, …”(*Callbacks — Stable Baselines 2.10.2 Documentation*, n.d.) Thus, we use the callback to save the model at a certain stage. For instance, the callback will save the best model per 100000 steps.

1. Train model

文本

描述已自动生成

The PPO algorithm from stable baseline3 is built to train the model.

1. Test model

文本

描述已自动生成

Finally, we use render the trained model and observe the results.

# Results

In this study, two different environments are trained and compared. Here are the training results. (The video results are in the attached file)

图形用户界面, 图表

描述已自动生成

Figure. 8. Tough Environment Training Results

图形用户界面, 应用程序

描述已自动生成

Figure. 9. Easy Environment Training Results

The two plots on the top are vital indicators to evaluate the training results. In a tough environment, the life of an agent is much more difficult than the easy one to get a satisfying performance. The reason is that the tough environment contains more high pillars and traps. (Figure. 10&11)

图形用户界面

中度可信度描述已自动生成

Figure. 10. High Pillar

图形用户界面

中度可信度描述已自动生成

Figure11. Trap

By contrast, there are few high pillars and fewer traps in the easy one, and the road is flat in most cases. (Figure. 12.)

图形用户界面

描述已自动生成

Figure12. Easy Environment

Also, I tried to run the agent trained in the challenging environment in an easy one and vice versa. Interestingly, in a similar landscape, both agents perform well in their unfamiliar environment. Agents can’t execute the untrained state despite remarkable generalization for similar terrain. For example, the agent trained in an easy environment will be hindered by the high pillar in the challenging environment.

Actually, the displayed training plots are the best, which are stable. However, for most experiments, the training is unstable, as stated by researchers. The experiment results correspond to the analysis at the end of section 1.1. Moreover, the reward function design is predominant. The customized reward function in the paper radically reduces the training time, whereas the default one converges slowly and has worse performance. Finally, increasing the number of agents is conducive to the training. Initially, we only set one agent to collect the samples, and the agent easily trapped into a local optimum. The agent is typically stuck by the blocks, but multiple agents will enormously increase the probability of escaping the local optimum, contributing to faster convergence.

# Conclusions

This project successfully trained an intelligent agent with one branch of the model-free technique called policy-based algorithms. Meanwhile, I realized that reinforcement learning is a cost-consuming field, including time, hardware, and knowledge. Incidentally, it’s environment-sensitive because a tiny modification of the reward function will significantly impact training. Here are several future works that we can do:

1. Since this is an individual report, we don’t get the opportunity to compare various algorithms in the same environment. Therefore, it’s necessary to compare different RL algorithms' performance.
2. We can study the connection between RL and Meta-heuristic. In my opinion, the mechanism of the meta-heuristic is also related to state value. By quantifying the state value in the meta-heuristic, we can understand why the apparent random exploration process works well. I speculate that the meta-heuristic also chooses the action with high value indirectly.
3. We can model the environment differently and study the other category of RL in model-free, which is constructing a simulated model.
4. Also, effectively preprocessing the environment is vital for reinforcement learning because each epoch takes a long time. An advanced preprocessing technique is desirable.

# 7. Reference

Achiam, J., & Achiam Berkeley, J. U. (2017). *Advanced Policy Gradient Methods*.

*Bootcamp Summer 2020 Week 4 – Policy Iteration and Policy Gradient*. (n.d.). Retrieved May 31, 2022, from https://core-robotics.gatech.edu/2020/12/16/bootcamp-summer-2020-week-4-policy-iteration-and-policy-gradient/

*Callbacks — Stable Baselines 2.10.2 documentation*. (n.d.). Retrieved May 31, 2022, from https://stable-baselines.readthedocs.io/en/master/guide/callbacks.html

*Enhancing Sample Efficiency in Reinforcement Learning with Nonparametric Methods | NVIDIA Technical Blog*. (n.d.). Retrieved May 31, 2022, from https://developer.nvidia.com/blog/enhancing-sample-efficiency-in-reinforcement-learning-with-nonparametric-methods/

*gym-super-mario-bros · PyPI*. (n.d.). Retrieved May 31, 2022, from https://pypi.org/project/gym-super-mario-bros/

*Proximal Policy Optimization — Spinning Up documentation*. (n.d.). Retrieved May 31, 2022, from https://spinningup.openai.com/en/latest/algorithms/ppo.html

*rl-tutorial-jnrr19/3\_multiprocessing.ipynb at sb3 · araffin/rl-tutorial-jnrr19*. (n.d.). Retrieved May 31, 2022, from https://github.com/araffin/rl-tutorial-jnrr19/blob/sb3/3\_multiprocessing.ipynb

Schulman, J., Levine, S., Moritz, P., Jordan, M., & Abbeel, P. (n.d.). *Trust Region Policy Optimization*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Openai, O. K. (n.d.). *Proximal Policy Optimization Algorithms*.