Mario Playing Reinforcement Learning Agent

AKSHAY KAMLOO
801210539

ITCS 5156 FALL 2022 – APPLIED MACHINE LEARNING
DR. LEE

Primary Paper

REINFORCEMENT LEARNING TO PLAY MARIO

Authors: Yizheng Liao, Kun Yi, and Zhe Yang

Year: 2012 [Online]

Available: https://cs229.stanford.edu/proj2012

ITCS 5156 Final Report
Mario Playing Reinforcement Learning Agent

Akshay Kamloo
Department of Computer Science
University of North Carolina Charlotte
akamloo@uncc.edu

***Abstract* — In this paper, I study how Reinforcement Learning can be used to create an automated Super Mario Bros agent to play the game. The complexity of the game environment is one of the difficulties. I accomplish tractable computation time and quick convergence by abstracting the game environment into a state vector and employing Double Q learning, a method unaware of transitional probabilities.**

I. INTRODUCTION

It has been extensively debated and researched to employ artificial intelligence (AI) and machine learning (ML) algorithms to play video games, as comparisons between the ML play pattern and that of a human player provide for insightful discoveries that may be used to enhance the algorithms.

For developing agents that can replicate player behavior, Reinforcement Learning (RL) is one widely studied and promising ML strategy. In this project, I'm going to investigate how to create a Mario controller agent that can absorb information from the gaming environment. Determining what constitutes a state, an action, and a reward is one of the challenges of RL. Furthermore, since real-time response is necessary to play the game inside the framework, the state space cannot be very big. I employ a state representation like to, which condenses the entire environment description into a handful of discrete-valued key attributes. The decision-making process that tries to maximize the reward is evolved using the Double Q-Learning algorithm.

One of the major challenges with RL is efficiently learning with limited samples. Sample efficiency denotes an algorithm making the most of the given sample. Essentially, it is also the amount of experience the algorithm must generate during training to reach efficient performance. The challenge is it takes the RL system a considerable amount of time to be efficient. For instance, DeepMind's AlphaGoZero played five million Go games before beating the world champion in it.

A. Motivation

A fascinating subject that has recently received a lot of attention is artificial intelligence in video games. In this situation, Mario AI Competition alters a Super Mario Bros. game to serve as a benchmark program for those who program AI controller to guide Mario and help him get beyond the various stages.

To create intelligent computer-controlled characters, the PlayStation game development team is working with Sony AI, the company's artificial intelligence research branch. They are creating AI game agents that can compete with or cooperate with players by utilizing reinforcement learning through trial and error, an AI can effectively educate itself how to behave through reinforcement learning Simply, these characters will mimic real players. They'll think to some extent.

II. BACKGROUND

In this section, I'm going to summarize the methods that researchers used before.

In [4] project, they created a Mario-playing autonomous agent using Q-learning. Their learning system shows quick convergence to the ideal Q-value and a high success rate. The best course of action continuously outperforms the level and has a high kills rate. Furthermore, their findings demonstrate that the state description is sufficiently broad so that the ideal policy can deal with a variety of unpredictable circumstances. The researchers also note that long-term reward maximizing outperforms short-term reward maximization. They conclude by demonstrating that adopting a decaying learning rate leads to a better policy than one that uses a fixed learning rate.

The 2009 Mario AI Competition introduced the benchmark used in the competition, and the rules for competitors to design agents. The competition's most unexpected outcome is that the top three agents use the A* search algorithm, outperforming agents that use rule-based, genetic programming, or neural network methods. As the environment becomes more complex, learning-based techniques have the advantage of changing and modifying the controllers.

A rule-based system evolved using genetic algorithms is called REALM (Rule-Based Evolutionary Computation Agent that Learns to Play Mario) by Bojarski and Congdon [6]. To control Mario, REALM features learnt and hard-coded versions of rule-based agents.

A. Game Mechanic and AI Interface

The object of the agent is to guide Mario to the finish line while collecting items and defeating foes to earn a high score. Six keys are utilized to control Mario: DOWN (not used in this interface), LEFT, RIGHT, JUMP, and SPEED. SMALL, BIG, and FIRE are the three game modes available to Mario. He may improve by consuming specific items (such as mushrooms and flowers), but if he is attacked by opponents, he will lose the current mode. The world contains many enemies (Goomba, Koopa, Spiky, Bullet Bill, and Piranha Plant), platforms (hill, gap, cannon, and pipe), and items in addition to Mario (coin, mushroom, bricks, flower). Once SMALL Mario is assaulted or if Mario falls into a gap, the game is finished.

When performing each step, the Mario AI framework interface call could return current scene from the environment. After applying the four wrappers to the environment, the final wrapped state consists of 4 gray-scaled consecutive frames stacked together, as shown above in the image on the left. Each time Mario makes an action, the environment responds with a state of this structure. The structure is represented by a 3-D array of size [4, 84, 84].

III. MARIO AGENT DESIGN USING DOUBLE Q-LEARNING

Using *gym-super-mario-bros* library, which is built on top of the OpenAI gym. gym is an extremely popular Python library that provides ML enthusiasts with a set of prebuild set of environments for reinforcement learning including actions, rewards, and states.

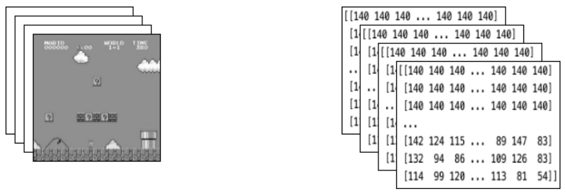
A. Preprocess the Environment

After performing an action to the current state of the environment, it returns the data to the agent as next state. Each state is represented by *[3, 240, 256]* size array. Often that is more information than our agent needs; for example, Mario's actions does not depend on the color of the pipes or sky. A nice part of gym is that we can use gym's Wrapper class to change the default settings originally given to us. Below I have defined a few wrappers that will help our agent learn faster. [2]

GrayScaleObservation is a common wrapper to transform an RGB image to grayscale; doing so reduces the size of the state representation without losing useful information. Now the size of each state: [1, 240, 256]

ResizeObservation down samples each observation into a square image. New size: [1, 84, 84]

SkipFrame is a custom wrapper that inherits from gym.Wrapper and implements the step() function. Because consecutive frames don't vary much, we can skip n-intermediate frames without losing much information. The nth frame aggregates rewards accumulated over each skipped frame.



```
[[140 140 140 ... 140 140 140]
 [140 140 140 ... 140 140 140]
 [140 140 140 ... 140 140 140]
 ...
 [142 124 115 ...  89 147  83]
 [132  94  86 ... 109 126  83]
 [114  99 120 ... 113  81  54]]
```

Figure 1: This image shows the final output of preprocess step. Final size: [4, 84, 84]

FrameStack is a wrapper that allows us to squash consecutive frames of the environment into a single observation point to feed to our learning model. This way, we can identify if Mario was landing or jumping based on the direction of his movement in the previous several frames.

B. Mario Agent Framework

I have created a class Mario to represent our agent in the game. The agent should be able to do following things:

i. Act:

For any given state, an agent can choose to do the most optimal action (exploit) or a random action (explore). Mario randomly explores with a chance of exploration rate. And when it exploits, it relies on learned parameters of MarioNN (Neural Netowork) to provide the most optimal action.

ii. Remember

There are 2 functions serves as Mario's memory process: cache and recall.

cache(): Mario records every action he takes in his memory after doing it. His experience consists of the game's current state, the action taken, the reward received from the action, the next state, and if the game is complete.

recall(): Mario randomly samples a batch of experiences from his memory and uses that to learn the game.

iii. Learn

Mario uses the DDQN algorithm under the hood. DDQN uses two ConvNets – Q_{online} and Q_{target} that independently approximate the optimal action-value function. I further explain the details of DDQN.

C. Deep Reinforcement Learning with Double Q-learning [3]

Finally, let's discuss what makes deep Q-learning so "deep."[5] A state can be thought of as a group of 4 consecutive 84×84 pixel frames, and there are 2 different actions based on how my environment has been set up. Since there are 84×84×4×2 pixels in a state, each pixel has an intensity between 0 and 255, and there are 2 possible actions for each state, the Q-table we would create for this scenario would have $2\times256^{84\times84\times4}$ values. We must utilize function approximation because it is difficult to store a Q-table so large. To do this, we will use a neural network to map a state to its state-action values to approximate the Q-table. [1]

Two values are involved in learning:

- TD Estimate - the predicted optimal Q* for a given state **s**

$$TD_e = Q^*_{\text{online}}(s, a)$$

- TD Target - aggregation of current reward and the estimated Q* in the next state s′

$$a' = \text{argmax}_a Q^*_{\text{online}}(s', a)$$

$$TD_t = r + \gamma Q^*_{\text{online}}(s', a')$$

Updating the Model:

$$\Theta_{\text{online}} \leftarrow \Theta_{\text{online}} + \alpha\nabla(TD_e - TD_t)$$

θ_{target} does not update through backpropagation. Instead, we periodically copy Θ_{online} to Θ_{target}

$$\Theta_{\text{target}} \leftarrow \Theta_{\text{online}}$$

α: the learning rate in the range [0,1]. This value controls the percentage of the feedback and the old state's Q-value that mix to each other.

γ: the discount rate in the range [0,1]. This value controls how much importance of future Q-value we treat.

In the implementation, I shared feature generator across Q_{online} and Q_{target}, but maintain separate fully connected classifiers for each. Θ_{target} (The parameters of Q_{target}) is frozen to prevent parameters updating by backprop. Instead, it is periodically synced with Θ_{online}.

IV. EXPERIMENTS AND RESULTS

After training for 10000 episodes the DDQN algorithm converges fast to get maximum reward, but in order to increase win probability of a Mario level, the model has to train for at least 40000 episodes.

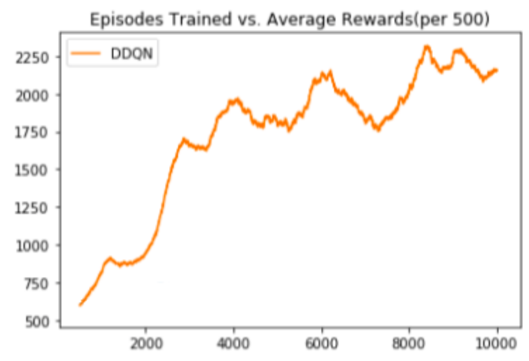


Figure 2: Avg reward per 500 episodes for 10,000 episodes in total

A. Future Work

Due to unavailability of powerful resources and computational limitation, the AI had limited action space in my case I set only two actions: RIGHT and JUMP, in future one I can train the RL agent to learn on all the action space.

Again, due to computational limitation, the current model takes 40+ hours to train for 40,000 episodes.

Mario levels has different backgrounds and huddles in each level, so the agent must train on each of such level in order perform its best behavior in the environment.

Extend our state to allow grabbing coins and upgrading.

We believe that our work provides a good introduction to this problem and will benefit the people with interests in using reinforcement learning to play computer games.

V. CONCLUSION

During this project, I personally had a lot of fun learning and implementing the Reinforcement Learning agent on Mario. Especially understand the concept of Deep Reinforcement Learning using Double Q Learning also known as DDQN. I can use the same methods to train an AI to play any of the games at the OpenAI gym.

There are other ways to expand on my RL strategy. For instance, I didn't design the state for Mario to grasp mushrooms and flowers in my learning algorithm. In addition, the algorithm focuses on optimizing the successful rate. Possible future work may include but is not limited to:

- Extend our state to allow grabbing coins and upgrading.
- Vary the reward function to realize different objectives (e.g., killer-type Mario)
- To tackle the position rounding problem, improve the state design.

I believe that my work provides a good introduction to this problem and will benefit the students with interests in using reinforcement learning to play computer games.

VI. APPENDIX

Source Code: https://github.com/akamloo/ITCS-5156/blob/main/MARIO_RL.ipynb

Video Presentation: https://www.youtube.com/watch?v=SoiCNTHo7Ug

Demo: https://drive.google.com/file/d/1766BVV4URjv2iI5EWE1_WUSseVIEWGJ0/view?usp=sharing

VII. REFERENCES

[1] Jyh-Jong Tsay, Chao-Cheng Chen, and Jyh-Jung Hsu, "Evolving Intelligent Mario Controller by Reinforcement Learning", 2011, 2011 International Conference on Technologies and Applications of Artificial Intelligence

[2] Yuansong Feng, Suraj Subramanian, Howard Wang, and Steven Guo, "TRAIN A MARIO-

PLAYING RL AGENT”, [Online]. Available: https://pytorch.org/tutorials/intermediate/mario_rl_tutorial.html

[3] Hado van Hasselt, Arthur Guez, and David Silver, “Deep Reinforcement Learning with Double Q-learning”, AAAI'16: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence February 2016 Pages 2094–2100

[4] Yizheng Liao, Kun Yi, and Zhe Yang, “CS229 Final Report Reinforcement Learning to Play Mario”, [Online]. Available: https://cs229.stanford.edu/proj2012

[5] Andrew Grebenisan, “Play Super Mario Bros with a Double Deep Q-Network”, [Online]. Available: https://blog.paperspace.com/building-double-deep-q-network-super-mario-bros/

[6] Bojarski and C. B. Congdon, REALM: A Rule-Based Evolutionary Computation Agent that Learns to Play Mario, Proceedings of the 6th international conference on Computational Intelligence and Games. Dublin, Ireland, IEEE Press, 2010, pp. 83–90