CS229 Final Report
Reinforcement Learning to Play Mario

Yizheng Liao
Department of Electrical Engineering
Stanford University
Email: yzliao@stanford.edu

Kun Yi
Department of Electrical Engineering
Stanford University
Email: kunyi@stanford.edu

Zhe Yang
Google Inc.
Email: rabbiest@gmail.com

***Abstract*—Abstract here.**

I. Introduction

Using artificial intelligence (AI) and machine learning (ML) algorithms to play computer games has been widely discussed and investigated, because valuable observations can be made on the ML play pattern vs. that of a human player, and such observations provide knowledge on how to improve the algorithms. Mario AI Competition [1] provides the framework to play the classic title Super Mario Bros, and we are interested in using ML techniques to play this game.

Reinforcement Learning (RL) [2] is one widely-studied and promising ML method for implementing agents that can simulate the behavior of a player [3]. In this project, we study how to construct an RL Mario controller agent, which can learn from the game environment. One of the difficulties of using RL is how to define state, action, and reward. In addition, playing the game within the framework requires real-time response, therefore the state space cannot be too large. We use a state representation similar to [3], that abstracts the whole environment description into several discrete-valued key attributes. We use the Q-Learning algorithm to evolve the decision strategy that aims to maximize the reward. Our controller agent is trained and tested by the 2012 Mario AI Competition evaluation system.

The rest of this report is organized as follows: Section 2 provides a brief overview of the Mario AI interface and the Q-Learning algorithm; Section 3 explains how we define the state, action, reward to be used in RL; Section 4 provides evaluation results.

II. Background

In this section, we briefly introduce the Mario AI frameword interface and the Q-learning algorithm we used.

A. Game Mechanics and the Mario AI Interface

The goal of the game is to control Mario to pass the finish line, and gain a high score one the way by collecting items and beating enemies. Mario is controlled by six keys: UP (not used in this interface), DOWN, LEFT, RIGHT, JUMP, and SPEED. In the game, Mario has three modes: SMALL, BIG, and FIRE. He can upgrade by eating certain items(mushroom and flower), but he will lose the current mode if attacked by enemies. Other than Mario, the world consists of different monsters (Goomba, Koopa, Spiky, Bullet Bill, and Piranha Plant), platforms (hill, gap, cannon, and pipe) and items (coin, mushroom, bricks, flower). The game is over once SMALL Mario is attacked, or when Mario falls into a gap. For more specific descriptions, please see [3] and [1].

When performing each step, the Mario AI framework has a call that returns the complete observation of 22 x 22 grids of the current scene, as shown in Figure 1. That is, an array containing the positions and types of enemies/items/platforms within this range. This is the whole available information for our agent.

The benchmark runs the game in 24 frames per second. The environment checking functions are called every frame. Therefore, while training we have to train and update the Qtable within 42 milliseconds.

B. Q-Learning

Q-learning treats the learning environment as a state machine, and maintains a reward value, denoted by Q, for each pair of (state, action) in a table. Here, $Q(s,a)$ denotes the Q-value, where s is the state and a is the action. For each action in a particular state, a reward will be given. The Q-value is updated based on reward by following rule:

$$Q(s_t,a_t) \leftarrow (1-\alpha)Q(s_t,a_t)+\alpha_{s,a}(r+\gamma \max(Q(s_{t+1},a_{t+1}))) \tag{1}$$

In the Q-learning algorithm, there are four main factors: current state, chosen action, reward and future state. In (1), $Q(s_t,a_t)$ denotes the Q-value of current state and $Q(s_{t+1},a_{t+1})$ denotes the Q-value of future state. $a \in [0,1]$ is the learning rate, $\gamma\,[0,1]$ is the discount rate, and r is the reward. (1) shows that for each current state, we update the Q-value based on the maximum Q-value of the next state.

Note we use the decreasing learning rate $\alpha_{s,a}$ [**?**] is different for different (s,a) pairs. Specifically,

$$\alpha_{s_t,a_t} = \frac{1}{\text{\# of times action } s_t, a_t \text{ has been performed}} \tag{2}$$

the decreasing learning rate allows for better convergence.

III. Mario Controller Design Using Q-Learning

A. Mario State

The state consists of the following fields:

- Mario Mode: 0 - small, 1 - big, 2-fire

- Direction: 8 directions + stay. Total of 9 possible values;
- If stuck: 0 or 1. Set true if Mario doesn't make movement over several frames.
- If on group: 0 or 1.
- If can jump: 0 or 1.
- If collided with creatures: 0 or 1.
- Nearby enemies: denoting whether there is an enemy in 8 certain directions in 3x3 window (or 4x3 window in large/fire Mario mode)
- Midrange enemies: denoting whether there is an enemy in 8 certain directions in 7x7 window (or 8x7 window in large/fire Mario mode).
- Far enemies: denoting whether there is an enemy in 8 certain directions in 11x11 window (or 12x11 window in large/fire Mario mode).
- If enemies killed by stomp: 0 or 1.
- If enemies killed by fire: 0 or 1.
- Obstacles: 4-bit boolean indicating whether there exist obstacles in front of Mario.

Note the nearby, midrange, and far enemies attributes are exclusive.

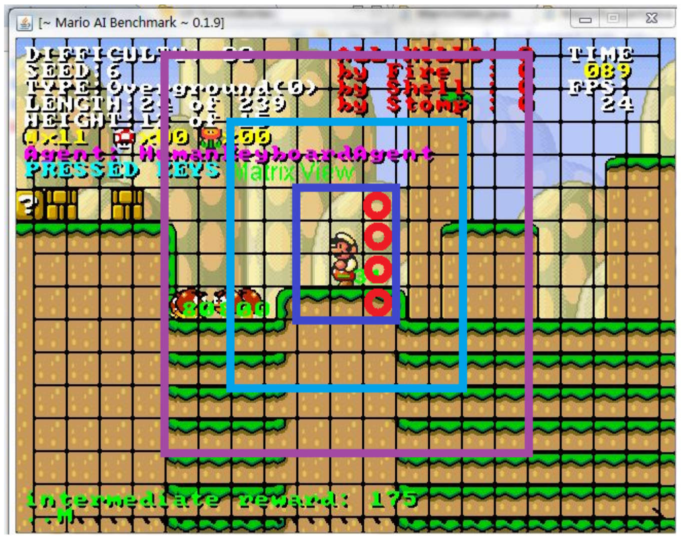

Fig. 1. Mario Scene

The figure 1 shows a typical scene of the Mario environment consisting of the platform and enemies. The area within dark blue box indicates the range of nearby enemies. The area between dark blue and lighter blue boxes indicates the mid-range enemies. The area between purple and lighter blue enemies indicates the far enemies. For instance, in the figure we have both mid-range and far enemies. The four red circles denote the obstacles array.

B. Actions

The Mario agent performs one of 12 actions. The combination {LEFT, RIGHT, STAY} x {JUMP, NOTJUMP} x {SPEED(FIRE), NOSPEED}.

C. Rewards

Our reward function is a combination of weighted state attribute values and the delta distance/elevation it performs from the last frame. We also let the reward of moving forward decrease when nearby enemies are present. Note our reward function is "approximately" only a function of our state only. Had us included the magnitude of speed, it will be completely determined by the state.

IV. Evaluation Results

For training the Q-learning algorithm, we firstly initialized the Q-table by a uniform distribution, i.e. $Q \sim \mathcal{U}(-0.1, 0.1)$. Then we trained the Q-learning algorithm by 5000 iterations for fixed episode setup and seed. For every 20 training iterations, we evaluated the algorithm performance by running 100 episodes and use the average metrics as the performance indicators. The evaluation episode parameters are identical to the training parameters.

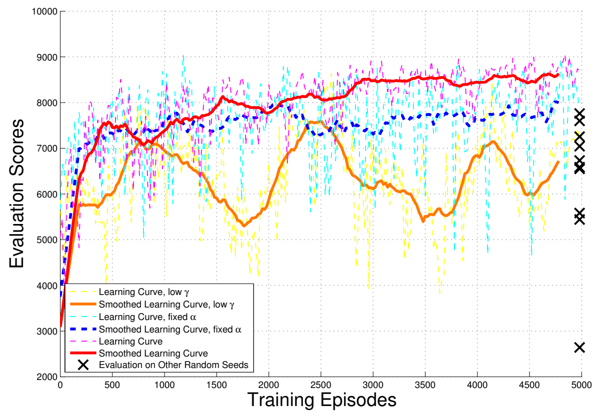

Fig. 2. Evaluation Score

Figure. 2 shows the learning curves of evaluation score. For our optimal parameter sets, we had the initial learning rate to be $\alpha = 0.8$ and then reduced it by following (2). We also set the discount factor, γ, to be 0.6, which means that the Q-learning algorithm tried to maximize the long-term reward. Obviously, our algorithm demonstrates a learning curve and quickly converges to the optimal solution after about 3000 training iterations. At the end of training cycles, our average evaluation score is around 8500. For some evaluation cycles, we can even achieve 9000 points, which is nearly the highest score human can achieve in one episode. In order to show the generalization, we also tested the trained Q-learning algorithm with random episode seeds. The results show that for most random seeds, our trained algorithm gives a very high score. The reason why one test performs bad is that there always be some unknown situations, which Mario is not trained.

In addition, we plot the learning curves with fixed learning rate and low discount factor. As discussed above, in our training algorithm, we keep decreasing the learning rate. The figure indicates that with a fixed learning rate, when it converges, the converged solution is not optimal. A low discount factor means that the learning algorithm maximizes the short-term reward. In our learning algorithm, we gave positive reward for right movement and negative reward for left movement. If the algorithm try to maximize short-term reward, the Mario will always move the right. However, in some situations, the Mario should move to left to avoid monsters. In this case, the short-term reward maximization is not the optimal solution.

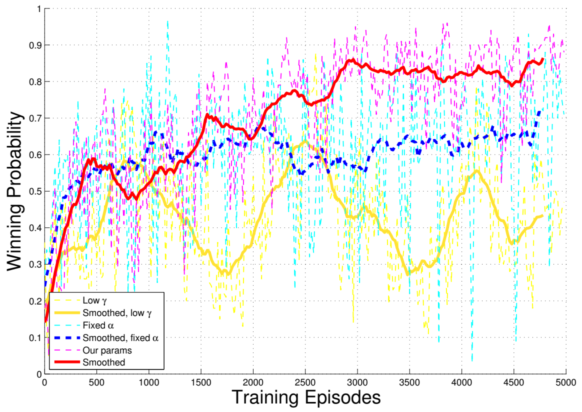

Fig. 3. Winning Probability

Figure. 3 shows the winning probability. As the early stage of learning process, the winning probability is as low as 0.3. With the increase of training cycles, the winning probability increases and converges to around 0.85. For the low γ learning, even the average probability is increasing but the variance is very high. For the fixed α learning, the learning curve converges but the converged value is not optimal.

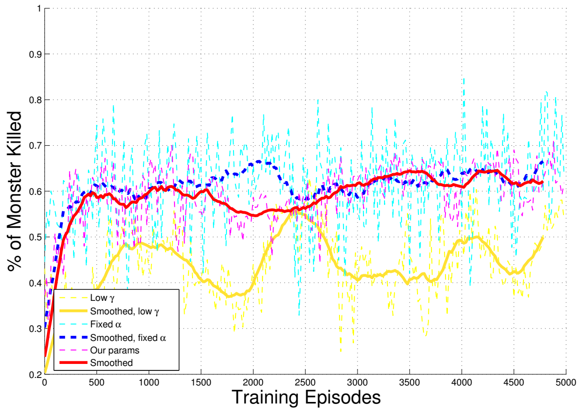

Fig. 4. Percentage of Monster Killed

Figure. 4 shows the percentage of monster killed. Since we generated the training episode by the same seed and setup, the total number of monsters within a episode is the same over training, and therefore, it is comparable. At beginning, the Q-values are generated randomly. Therefore, the killing percentage is very low. In the learning algorithm, we give a very high reward for killing, e.g. 60 for killing by fire and stomp. Hence, the Mario has the motivation to keep more monsters. The learning curve shows fast convergence of the killing probability within a few training episodes. There are two reasons that we give high reward for killing. Firstly, the killing action is given high score in evaluation and therefore, we can achieve a high score in evaluation. Secondly, the Mario is safer when he kills more monsters.

In this figure, the learning curve with fixed α shows a similar performance to our optimal learning curve. The learning curve with low γ has very low mean and high variance.

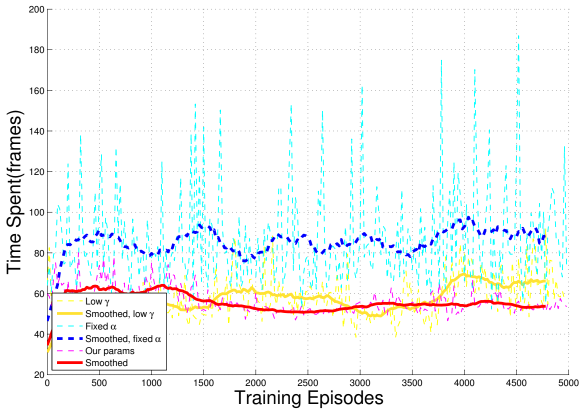

Fig. 5. Time Spent in Frames

Figure. 5 shows the time spent for the Mario to pass a episodes successfully. The optimal learning curve shows a fast convergence within 250 iterations. The learning curve with low γ has a similar performance as the optimal learning curve. The reason is that the short-term reward maximization forces the Mario to keep moving rightward. However, as we discussed above, it is not optimal since the Mario will collide creatures with high probability. The learning algorithm with fixed α needs more time to win because the converged policy is not the best.

V. Conclusion

In this project, we used Q-learning algorithm to control the Mario. Our learning algorithm demonstrates fast convergence to the optimal Q-value with high successful rate. The optimal policy also has high killing rate and needs less time to be successful. In addition, our results show that the optimal policy is generalized to tackle different and random environment. Further, we observe that long term reward maximization over-performs the short term reward maximization. Finally, we show the the decaying learning rate algorithm converges to a better policy than the fixed learning rate algorithm.

There are many future works regarding using machine learning to play Mario. For example, in our learning algorithm, we did not design the state for the Mario to grab mushroom and flowers. In addition, our algorithm focuses on optimizing the successful rate. Other design directions include maximizing the coin collection and maximizing the killing rate. 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 game.

References

[1] J. Togelius, S. Karakovskiy, and R. Baumgarten, "The 2009 mario ai competition," in *Proceedings of the IEEE Congress on Evolutionary Computation*, Citeseer, 2010.

[2] R. Sutton and A. Barto, *Reinforcement learning: An introduction*, vol. 1. Cambridge Univ Press, 1998.

[3] J. Tsay, C. Chen, and J. Hsu, "Evolving intelligent mario controller by reinforcement learning," in *Technologies and Applications of Artificial Intelligence (TAAI), 2011 International Conference on*, pp. 266–272, IEEE, 2011.