Reinforcement Learning: Video Game

William Minchew
Engineering and Computer Science
University of Texas at Dallas
Richardson, USA
wmm170030@utdallas.edu

Fatima Parada-Taboada
Behavioral and Brain Sciences
University of Texas at Dallas
Richardson, USA
fnp170130@utdallas.edu

Daniel Le
Engineering and Computer Science
University of Texas at Dallas
Richardson, USA
daniel.le@utdallas.edu

Yeswanth Bogireddy
Engineering and Computer Science
University of Texas at Dallas
Richardson, USA
yrb160030@utdallas.edu

***Abstract*— In this paper, we will be using a frozen lake game that implements a reinforcement learning algorithm. The frozen lake game consists of a character that is controlled by a set of directional actions aiming to maximize the reward by continuously reaching a "goal" at the end of the frozen lake without falling into "holes" or off the edges. Using Q-learning we are able to optimize the movements the character takes to be able to get to the other side of the lake by finding the best action to take in any state. By making adjustments to discount rate and learning parameters, the model is able to maximize reward and achieve the goal as best as possible.**

Keywords— *reinforcement learning; frozen lake; Q-learning*

I. INTRODUCTION

For the purpose of this project, we will be using reinforcement learning. This project has served as an example of implementation of reinforcement learning into video gaming. By using reinforcement learning, we will try to achieve the optimal manner of using machine learning in real life and in this case the game.

Reinforcement learning is an area of machine learning that concerns how our agent, which takes actions, reacts to its surrounding environment and based off of those variables to take its action [1]. In other words, it takes in information and tries to observe and create patterns within the environment and tries to maximize the probability of it being correct in future predictions. It is one of the three main learning types alongside supervised learning and unsupervised learning in machine learning.

A. Basics of Reinforcement Learning

Like every machine learning algorithm, reinforcement learning tries to achieve the optimal value in the most efficient way that the worker wants, then that information is processed and used for its own purpose. In short, the way we can understand our algorithm works is our agent will act randomly at first based on a set of finite actions then it will look at the environment to see the results. After those observations, the agent will now have all the necessary inputs for its next steps, and it will continue to take actions based on those inputs (Fig. 1). In this assignment, we will try to implement reinforcement learning in the form of a short video game.

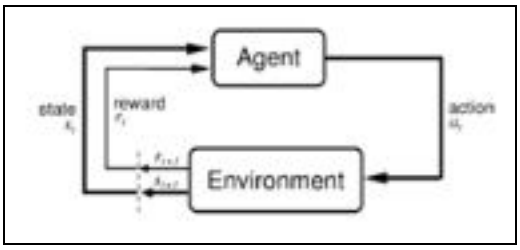


Fig. 1. A demonstration of how the reinforcement learning algorithm occurs in terms of agent taking actions, environment giving observations, and the updating reward and state values that maximize.

II. IMPLEMENTATION OF ALGORITHM

The implementation of our algorithm will be based on a simple frozen lake game [2][3]. The main object of the game is to get our agent- the character, to reach the end goal as accurately it can possibly make based on a set of four actions it can take. The character at first will make some basic moves and we will have to set our reward for whenever it does what we want it to do. Doing this, we will be able to control its action and reward or "penalize" its actions that it took.

A. The task

Our task is to learn an optimal policy that maps the states of the environment to the actions of our agent. For example, if we are hungry, we eat. If we are thirsty, we drink (1).

$$\pi : S \rightarrow A \tag{1}$$

π: the policy
S: the state
A: Action

B. The Reward

In order to control our agent, we have to set a parameter to give it feedback that measures the success or failure rate (2). Based on that feedback, the agent then will

be able to adjust itself for the next right action in the next state that we want it to take. It is like training a dog, whenever the dog behaves well or does what we want, we'd like to encourage it and give the reward so that it will do it again or punish it by decreasing the reward or not giving it at all. So, what our agent is trying to optimize is the total future discounted reward.

$$V^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \qquad 0 \le \gamma < 1$$

(2)

$V^{\pi}(S_t)$: total discounted reward with policy at state of time t

r_t: reward at time t

γ: discount rate

Example: If you have a choice to choose between receiving $5 now or $5 sometime in the future. What would you choose? You would probably choose the first option to receive the money right now as you want to maximize the profit and the chance you would get the money.

III. DEFINING THE Q FUNCTION

To maximize the reward, we have to find the algorithm that computes to get the maximum expected reward for an action taken at any given state (3).

$$Q(S_t, a_t) = E[R_t | S_t, a_t]$$

(3)

Q: Expected total future reward

S_t: state at time t

a_t: action at time t

What this means is that if an agent in state S time t and take an action "a" at time t, the total amount of reward that it can obtain if it takes an action at that state is the result of Q. Therefore, higher Q means more desirable action in that state so we would like to take that action which can maximize our Q value.

IV. EXECUTION

Currently, we have two choices for our agent to take feedback and prepare for its next action in the next state. The first option is our agent will input a move-one of our finite set of moves, then it observes the environment. The state changes, and the rewards will be given. Based on that, the next move will be decided. Keep doing this and we will have our model. But is it efficient to go down this way? Is the actual Q value optimized and we get to choose the highest one? The answer is no.

A. *Generic*

In this execution step, we want to feed our agent all the actions that we can execute at that time. Then based on that, we choose the action that nets us the optimized value or the highest Q that we expect (Fig. 2).

In short, the agent needs a policy, $\pi(s)$, to infer the best action it should take to maximize Q (4).

$$\pi(s) = \arg Q(s, a)$$

(4)

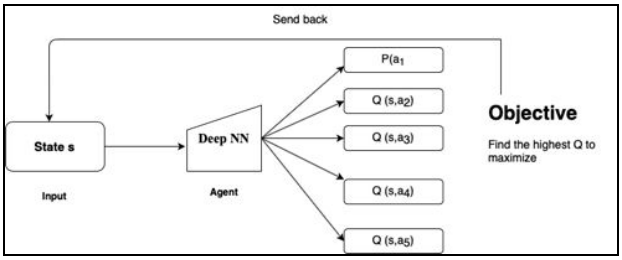


Fig. 2. This chart shows the DQN model for inferring policy.

B. *Probability*

We can also apply a probability gradient (Fig. 3) into our model. This can be done when the best and optimized Q will be given a probability to boost and then is sent back as an input for our agent to process the next step. The undesirable ones' probability will eventually be lowered so that our agent will not be likely to take the same action in the future. This would be the most advantageous model because directly outputting the optimized policy would prevent there from being any inferences of the policy solely based on the Q function.

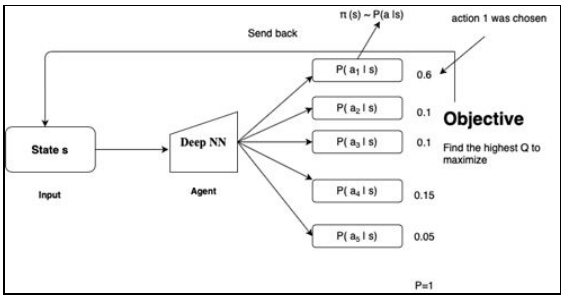
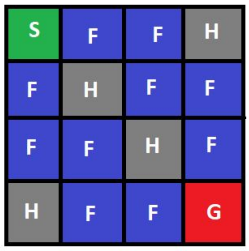


Fig. 3. This chart demonstrates the policy/probability gradient model.

C. *Implementation*

Using the frozen lake model, we can implement reinforcement learning using an off-policy model that allows the character to use a greedy action that allows it to explore future movements [4]. The environment used in our project consists of a 4x4 grid. There are four possible types of blocks within the grid: the start (S), frozen (F), hole (H), and goal (G) blocks (Fig. 4). The character moves around this grid until it reaches the end goal. If the agent or character reaches a hole, they will "fall" and have to restart with a reward of 0.



S: start

F: frozen

H: hole

G: goal

Fig. 4. This is an example of the environment used in the frozen lake game with each blocks' respective type.

Our agent has the option to move up, down, left, or right on the grid with random actions being taken after every few episodes or trials. This greedy action allows the agent to explore different options to make it to the goal block and minimize loss after correcting errors observed in the random action episodes. With this information there are 16 possible locations or states for the agent to be at any given time with 4 different possible directions or actions for the agent to take. Using these numbers we can construct our Q-table consisting of our agents' current state and action in a 16x4 matrix.

D. *Quality Table*

A quality table, commonly referred to as a Q-table, is used to represent the agent's memory of what it has learned [5] [6]. The matrix does not initially begin as a 16x4 matrix, but it is initialized to zero and works its way there through the process of finding new states as it goes through each episode. The Q-table adds new states to the matrix using the transition rule of Q-learning (5):

$$Q(s_t,a_t) \leftarrow (1-\alpha)\cdot \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}} \quad (5)$$

Using this formula, the value given to the current location in the Q matrix is given by the comparable value in the R matrix added by the learned value containing the maximum Q value times reward and discount rate, multiplied by the learning rate.

Using this function, our agent can learn unsupervised based on new information they gain and focusing on reaching the goal at every episode.

V. RESULTS AND ANALYSIS

By using the frozen lake game to be able to observe the reinforcement learning model, we were able to modify parameters in the transition function given in the implementation section to further optimize the game's functionality. Tracking the model's performance is measured best looking at the sum reward and loss value. Generally, a smaller loss value and higher sum reward is preferred. Using these values, different trials can be observed and the best parameters can be chosen.

```
++++++++++++++++
    Move 34
++++++++++++++++
  (Right)
SFFF
FHFH
FFFH
HF G
++++++++++++++++
    Move 35
++++++++++++++++
  (Right)
SFFF
FHFH
FFFH
HFF
Finished after 35 moves
```

Fig. 5. This figure demonstrates what the last moves of an episode looks like and where the agent is during that state in the game.

The frozen lake game environment was created using python and implementing the gym library. We implement the reinforcement learning algorithm using the Q-learning method and looking at progress values. Each

session in every episode is also outputted with the number of moves taken in each session and specific actions taken while the game is running (Fig. 5). Our code's output demonstrates each trials' Q-table along with a reward and loss value for that given trial (Fig. 6).

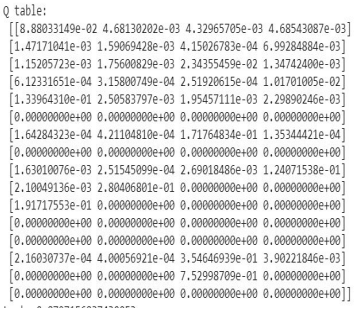


Fig. 6. This is an example of how the entire model's final weights for the model are displayed. This is the quality table for one of our experimental runs.

Parameters modified include the learning parameter and the discount rate with a consistent number of 4000 episodes for each trial (Fig. 7). There were 6 different combinations of parameter values that were used to determine which were going to work to create the best outcome for our model.

TABLE I. TABLE OF EXPERIMENT TRIALS

Experiment Number	Parameter Chosen	Results
1	Learning rate = 0.7 Discount rate = 0.8 Episodes = 4000	Sum reward = 0.3035 Loss =1.02805
2	Learning rate = 0.8 Discount rate = 0.9 Episodes = 4000	Sum reward = 0.63875 Loss = 1.2644
3	Learning rate = 0.5 Discount rate = 0.8 Episodes = 4000	Sum reward = 0.1785 Loss = 0.9459
4	Learning rate = 0.6 Discount rate = 0.9 Episodes = 4000	Sum reward = 0.32375 Loss = 0.9998
5	Learning rate = 0.8 Discount rate = 0.6 Episodes = 4000	Sum reward = 0.076 Loss = 0.9722
6	Learning rate = 0.4 Discount rate = 0.3 Episodes = 4000	Sum reward = 0. 029 Loss value = 0.3118

Fig. 7. This table includes all parameter values during each experiment trial. The results column includes the outputted sum reward and loss values. Experiment 4 is highlighted because that is the trial that showed the most favorable conditions for this model.

The first trial and second trials were both given high learning and discount rates and both experiments resulted in higher loss values in the results. Although the reward for the second experiment was higher, it was not high enough to be considered the optimal option. In the third trial, we tried to increase the gap between the two parameters by making the learning rate 0.5 and the discount rate 0.8. This actually resulted in a super low reward outcome and a high loss value, as well. Trials 5 and 6 approached the idea of making the learning rate higher than the discount rate and also tried to look at differences in higher parameter values and lower parameter values. Both yielded a low sum reward value. Trial 5 received a high loss value while the 6th trial actually received a really low loss value.

The fourth trial used a higher discount rate and a lower learning rate to yield an okay reward and loss value. Although the reward was not as large as in the second trial, the loss was as low as it could be to get a good reward value. Otherwise the second trial parameter options would have been the next option.

VI. CONCLUSION

Our project's purpose was to implement reinforcement learning using the frozen lake game. We also wanted to maximize the reward outcome for the model by using Q-function learning and off-policy learning for this specific game.

By experimenting with the combination of values in our experiment trials, we were able to optimize our code to find the best balance of sum of reward and loss. We found that the best parameters were seen in our fourth trial, allowing there to be moderate reward and loss for our model.

VII. FUTURE WORK

For this project there are quite a few other models we would have liked to use but were unable to do to time and difficulty constraints. Here we decided to go with a much simpler form of reinforcement learning than originally planned due to these constraints. Our main future work concern would be to explore new models and apply our method or more advanced methods built off of it to them.

Such a model would be an attempt at using a snake game for reinforcement learning. We would have liked to develop a more complex algorithm built from the current and applied it to the snake game. Which would have been

fairly similar yet had a larger “play” area and attempt to have multiple goals per iteration.

An even more in depth model that we considered was an attempt to have a racing game simulation where we taught a car to drive through. These methods all would follow the same principles and methods derived for our current project but would have more variables added to the algorithm giving them far more complexity.

Lastly we have a couple ideas for how to improve this specific program. By potentially increasing the “play” area for the program we could have increased the complexity of it without needing to create a brand new model. There are ideas of adding new obstacles that could move or shift around like an iceberg where it could block potential paths or break up the path removing the ability for the agent to move. Or even have it where the agents best results are stored and then can be used in a test mode where the user can create a map and then allow the agent to traverse it potentially with turn by turn editing to force the program to respond.

REFERENCES

[1] M. Kana, “This is how reinforcement learning works,” unpublished.
[2] J. Bowen, “Frozen lake with q-learning!” unpublished.
[3] R. Mendes, “Gym tutorial: the frozen lake,” unpublished.
[4] A. Violante, “Simple reinforcement learning: q-learning,” unpublished.
[5] J. McCullock,‘“A painless q-learning tutorial,” unpublished.
[6] K. Maladkar, “Frozen lake: beginner’s guide to reinforcement learning using openAI gym,” unpublished.