Minecraft Maze Runner

Xiaolei Jiang Yuxiang Qian Jeffrey Eric Su

February 2020

1 Abstract

Our project is about solving mazes of different types of difficulty. There are 3 difficulties of mazes (easy, medium, and hard) and we are attempting to use Q Learning to solve these challenges.
We have accomplished solving simple and medium mazes. We implemented a Tabular Q Learning algorithm to solve simple mazes, but we found that it was difficult for the agent to solve medium mazes and so we implemented a Deep Q Learning algorithm to solve medium level mazes and hopefully hard level mazes.

2 Introduction

Our project is focused on solving mazes mazes in a minecraft environment. Our first milestone is to be able to solve simple mazes. We defined simple mazes as mazes which are small in size and have few random obstacles like random lava placements. Our second goal is to be able to solve medium mazes which we defined as larger mazes, and more obstacles like zombies and lava. Our reach goal is solving hard mazes which involves interacting with the map to open doors in addition to everything described in medium mazes. The approach we took was using reinforcement learning to find the best action based on some state. This problem was interesting to us since we realized that we could add all sorts of different challenges and see if we could have an agent that could learn how to solve those difficult challenges. This task was also interesting since we were randomizing lava and zombies so that it would be difficult for simple algorithms like Q Learning to learn. A problem that we have encountered is that Tabular Q Learning doesn't do a good job of adjusting to randomized lava so we decided to use Deep Q Learning to see if it would be more efficient at adjusting. We are currently running medium level mazes and hyper tuning our Deep Q Learning algorithm and we plan to use our Deep Q Learning algorithm on hard mazes eventually.

3 Background

The platform that we are using is Minecraft Malmo. For code libraries we used some tutorials provided malmo to help us develop our Tabular Q Learning algorithm and we are using PyTorch along with some code from our Tabular Q Learning algorithm. For our Tabular Q Learning algorithm we followed

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

for updating our Q Table. We also added an ϵ for randomness. In which we use a random move $\epsilon\%$ of the time. We added α as the learning rate to control how much the model should change. We added γ as the discount factor which controls how important future rewards are to the state.

$$Action = \begin{cases} \max_a Q(s,a), & R > \varepsilon \\ Random\ a, & R \leq \varepsilon \end{cases}$$

R is random number between 0 and 1, ε is the exploration factor between 0 and 1. If ε is 0.1, then 10% of the times, the algorithm will select a random action to explore corresponding rewards.

For Deep Q Learning we used PyTorch and code from some of the examples in the malmo tutorial.

4 Problem Statement

The problem we are addressing is solving a maze, but in addition we are randomizing obstacles. For the environment, we are currently giving the agent a 3x3 array of the blocks around it. We gave the agent a reward of -1000 for dying, +2000 if it reached the diamond block, and +500 if it reached the lapis block which is the sub-goal.The diamond block represents the end block, and there are 2 lapis blocks in the middle part of the environment. The lapis blocks were added so that the agent could receive some sort of reward for going in the right direction since it only has a 3x3 vision.

5 Method

We first tried to solve the problem using tabular q learning, which works well in the easy-level maze with just several random obstacles and small size map. We are currently implementing deep Q learning algorithm(DQN). In this algorithm, we use a feed-forward neural network to help us calculate the q value. Because the agent can observe 3x3= 9 blocks around it, and there are 5 types of blocks

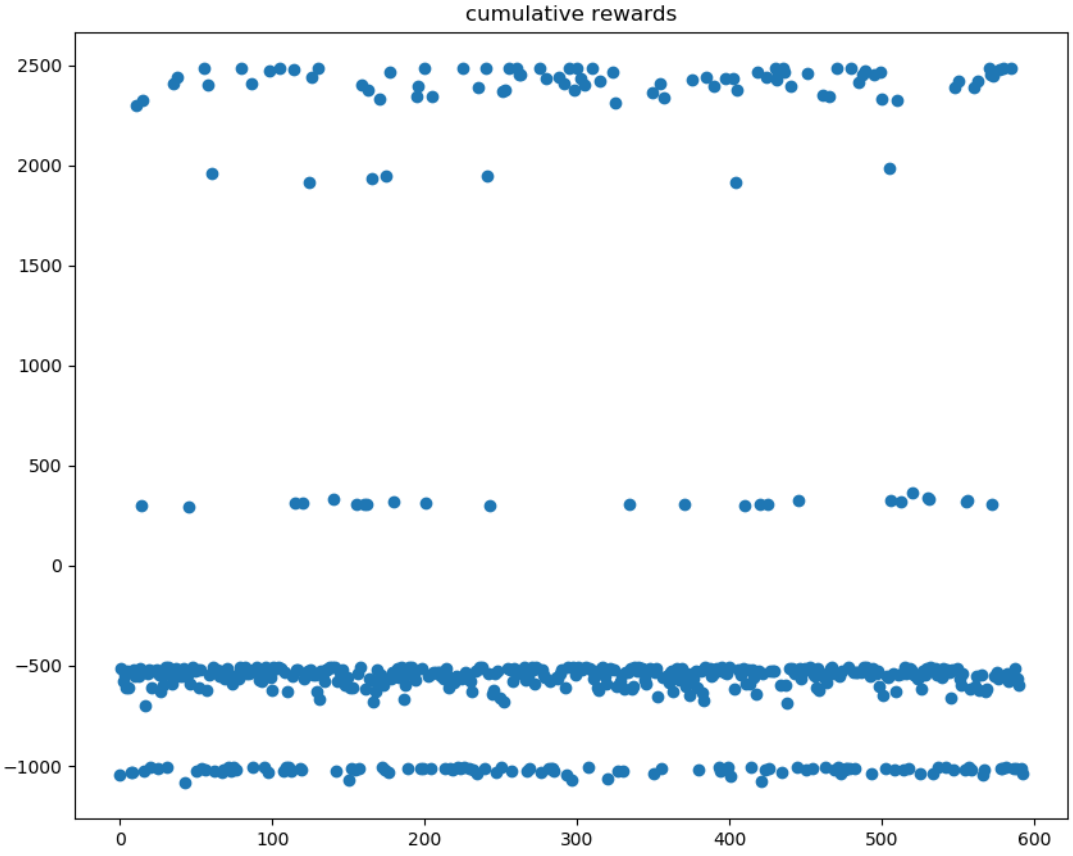
(gold block, lava, stone, diamond block, lapis block) and a replay buffer to store the previous 4 states, the input layer of the neural network is 3x3x5x(1+4)=225 nodes, and we have 2 hidden layer each with 150 nodes, and we have an output layer with 4 nodes which represents our 4 moves (move south, move north, move west, move east).
For hyper-parameters, we choose learning rate= 0.01, epsilon(the exploration rate of the agent) = 0.9, discount factor = 0.99. With this combination of hyper-parameters, the agent can relatively perform better and more easier to find the optimal solution of the medium-level maze after training.

6 Experiments

For tuning hyper-parameters in Deep Q neural network, we first tried to set learning rate= 0.1. After 1000 times of training, it still doesn't perform well with very low cumulative rewards. We realized that the learning rate may be too high so that the neural network didn't find the optimal solution, it may be stuck in a sub-optimal solution. Therefore, we change it the learning rate to 0.01, and after 1000 times of training, it performs better, and the agent could find the optimal way to the endpoint.
We also first set epsilon=1.0, but it didn't perform well in the beginning of the training steps because it explored the environment too randomly in the beginning and easily fell into the lava and died. To reduce the training time, we set it to 0.9.

7 Results

The follow graph is the agent's cumulative rewards for 600 times of evaluation after 1000 steps of training in the environment. The positive reward points mean that the agent reaches the endpoint, and the negative reward points mean that the agent died or didn't find the endpoint within 15 seconds. The result until now is good, but not optimal.



8 References

1. Microsoft's Malmo tabular q-learning example (https://github.com/microsoft/malmo)
2. Pytorch documentation (https://pytorch.org/)
3. Hui, Jonathan. "RL — DQN Deep Q-network" July 16, 2018. https://medium.com/@jonathan_hui/rl-dqn-deep-q-network-e207751f7ae4
4. Deep Q neural network example codes https://github.com/dannym08/Dungeon-Masters/blob/master/deep_q_learning.py