Deep Q-Learning Taxi

Daniel Roberts
Data Science
University of Texas at Dallas
Richardson, United States
daniel.roberts@utdallas.edu

Daoqun Yang
Computer Science
University of Texas at Dallas
Richardson, United States
daoqun.yang@utdallas.edu

In this article, we discuss deep Q-learning for game playing. We built a model by implementing Deep Q-learning on the game called Taxi which is in the gym resource package so that it will reach an end game goal. From the test we can conclude that a model in a scarce reward environment requires some encouragement in order to succeed and win the game.

I. INTRODUCTION

Speaking of machine learning, reinforcement learning should never be ignored since it is an important part of machine learning. In this project, the Deep Q-learning algorithm is applied on a program and it is able to learn by itself during the process of playing the game. For the game, we choose a game from the gym package. In the gym package, there are lots of ready mini games with environments designed to be used for machine learning implementation. In order to apply the reinforcement learning for the games, we compared several different games. For building layers of Deep Q learning networks, we choose the Keras package to construct models and layers. Keras package provides the functions of dense, reshape, embedding and sequential functions to build neural networks conveniently and make the learner's action more effective and faster in the training process. At the beginning of the project phase, we chose Ms. Pacman, however due to some environmental issues we chose to look into other games from the gym package. The main problem with the Ms. Pacman environment that lead us to pick Taxi, was the amount of data Ms. Pacman stored in each state. Because the size of the states were so large, processing time became much longer than expected, so we moved to the smaller, text based environment of Taxi.

II. INTRODUCTORY KNOWLEDGE

A. *Game Information*

In Taxi, there are six different actions: move up, move down, move left, move right, pick up and drop off. At the beginning of the game, the car will be spawned at random position, and each movement without pick up and drop off will decrease score by 1. In Taxi, there is only one way to increase your score, a successful drop off will increase score by 20. However, there is a way to lose a large number of points in the game, as an unsuccessful drop off or pickup will give a 10-point penalty. The original win condition of the game was making one successful drop off, however we adjusted this win condition to having a score of 500 points in order to give the learner a harder challenge. There was no original end of game condition besides winning, so we also added a death condition of achieving a score of -500.

B. *Basics of Q-Learning*

Deep Q-Learning can be broken into two main theories, Q-learning and a neural network. Q-learning is a less complex reinforcement learning algorithm than the deep Q-learning algorithm, but it is still quite powerful. Q-learning uses previous environment, action pairs of information to output the best action to select given the current environment. It does this using a table of all environment, action pairs, so the complexity and memory requirements of the algorithm grow drastically as the action space and number of potential environment states grow. The Q-learning model uses the following equation in order to decide which action to take using the current reward value and max reward of the next environment state. It computes this for all actions and picks the action with the highest Q value.

$$Q(s,a) = r(s,a) + \gamma \max_{a} Q(s',a)$$

Fig 1. Q value equation [1]

With enough environmental and action analysis the equation will converge so that the model picks the best action for each environment state. The gamma, or discount factor of this equation is used to help the algorithm choose between focusing on current rewards or prioritizing long term goals.

C. *Basics of Deep Q-Learning*

Deep Q-Learning, as stated before, combines the theories of neural networks and Q-learning by using a neural network to approximate Q values rather than having to use a table of values. The model works by inputting the state of the game into the neural network, which then approximates the Q values of each action. The algorithm then uses the action with the highest approximated value as the next action and inputs the new environment as the new input. Once enough

data is collected, we refit the model using the current state and a target value calculated by the Q function above. This method of generating Q values comes at the cost of some accuracy, but with the added advantage of having a much lower memory cost, as well as it is much more scalable than the Q-learning algorithm.

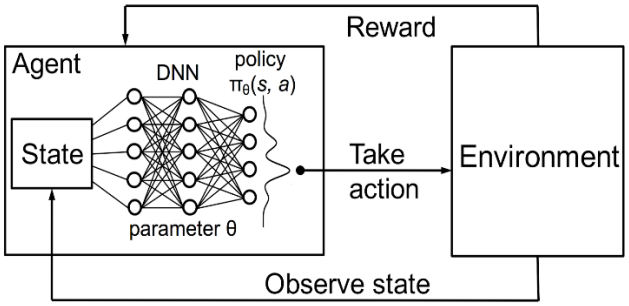

Fig 2. Picture of deep Q-learning process [2]

D. *Scarce Reward Problem*

The scarce rewards problem is an issue that occurs in reinforcement learning when the rewards of a game are few and far between. This problem can be solved in a couple of different ways, including adding more ways to earn rewards or allowing the model to take more greedy actions via a decaying epsilon value. Both of these methods come with upsides, however they also both have disadvantages. If you add too many rewards to the environment the model may become distracted or be able to cheat its way to a successful result. In regard to greedy actions, the more greedy actions a learner takes, the less it is actually using what it has learned, causing it to make unproductive decisions later on.

E. *Problems with Exploration*

Along with the scarce rewards problem, some environments, especially those that are more complex, run into problems with the learner not exploring enough to find a solution to the problem. One solution to this problem happens to coincide with the greedy solution from the scarce rewards problem. A greedy algorithm is used to take random actions at random times in order to continually cause random movement and avoid the learner falling into repetition of a safe action. The greedy algorithm works by having an initial epsilon value (usually 1) and decreasing each time the model learns, until it reaches the lowest epsilon value.

III. Version Changes

A. *Version 1*

In the original version of our model we did not take into account any of the problems discussed earlier. We found that the model would fall into repeat actions as almost all actions resulted in -1 reward being given. We had not yet set the new win condition yet, so the model was able to consistently make a drop off at around -200 score. We used a neural network with two hidden layers each with 250 neurons and a Relu activation function, an embedding input layer, a reshape layer, and a dense output layer. We used 500 maximum games, a gamma of 0.9, a starting epsilon of 1 with a 0.95 decrement and a 0 end value for epsilon.

B. *Version 2*

In version 2 of our model we wanted to improve its chances of winning, so we added both a -3 point reward for taking 3 or more repeat actions. This was meant to help encourage exploration. We also introduced a new win condition to the game so rather than winning by making a successful drop off, the learner would win by achieving a score of 500. We left the neural network the same in this version as the last. We used the same hyper parameters for this version as the last. In this version we updated the max games to 200 because we wanted to be able to run more simulations.

C. *Version 3*

In this version we took steps to solve the scarce reward problem by adding a 0 reward when the taxi was within one block of the pickup if a pickup was needed, and within one block of the drop off if a drop off was needed. This was meant to encourage the model to move towards these spaces when needed. We also added a +20 point reward to a successful pickup as another hint at what it should be doing. We also played around with the hidden layers, making them have 20 and 15 neurons respectively. We updated the epsilon related hyper parameters to encourage more random actions, changing epsilon decrement to 0.99 and epsilon end to be 0.05. This allowed for random actions to always occur. We kept these hyper parameters for the rest of the versions.

D. *Version 4*

We realized that because we made the numbers for picking up a passenger too high, the model was able to win the game simply by making a pickup, an illegal drop off (resetting the passenger), then picking the passenger back up to gain a net score of 10 points. In order to quell this behavior, we upped the close bonus to +0.5 and halved the pickup to +10. This was meant to continue to encourage the model to move toward the pickup and drop off points while the main incentive was still a successful drop off. We left the neural network the same for this version.

E. *Final Version*

In this final version of the game we only updated the repeat action punishment to -6 points over 5 repeat moves, but mostly played with the neural network to see if we could make the game win in a fewer number of steps. We tried adjusting the neural network to have 55 neurons and a linear activation function for the first hidden layer, and 95 neurons and a Relu activation for the second layer.

IV. Anlysis and Results

After running eight trials in each version of our model, we found the results to improve with each change we made, with the one exception of the model finding an unintended win method in version 3 of the code. We focused on tracking the high score, number of games, number of steps per game, and final score of each game. We found the following results by analyzing this data.

A. *Tracking Step Count*

We tracked the step count of each game within the training scenario. We did this in order to gauge when the model began to survive long enough to make a sucessful drop off as well as see how long it could survive in general.

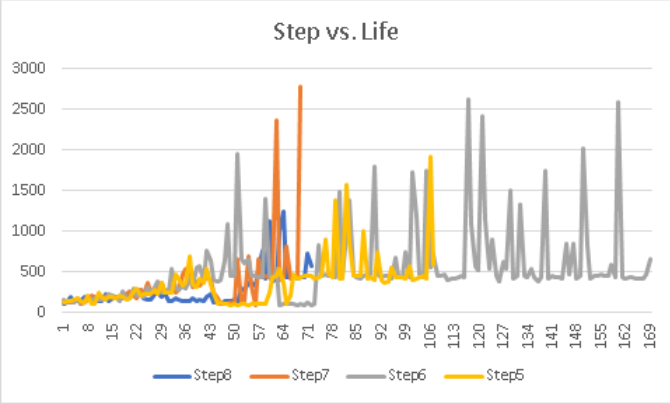

Fig 3. Number of Steps per life across 4 different models of the same type

As seen in the above graph dervived from four different version 5 models, the number of steps in a given life starts out very small but then grows larger with time. This is due to the models increased survival skills as it begins to learn more and more. We then see the model's steps peak in two places, a few lives before achieving victory, and when achieving victory in some cases. We see from this that as the model learns to survive in its enviornment, that it also finds the nesecary tools for victory.

B. *Highscore in each game*

We tracked the high score achieved in each game in order to find out when the model began to go past survival and begin to achieve points. We see in the following graphs that in most models a score above 0 (the starting high score) is not achieved until very close to completion of the game. The only exception to this is models that require more games to complete the task such as model 6. This is most likely because the learner spends the first few moves of a game exploring and losing points, then the rest of the game trying not to die while still remaining in the negative point values. So even though it is gaining many points, it is losing more than it is gaining. This is most likely caused because the punishment for an incorrect drop off or pickup occurs more often than the reward for a successful one.

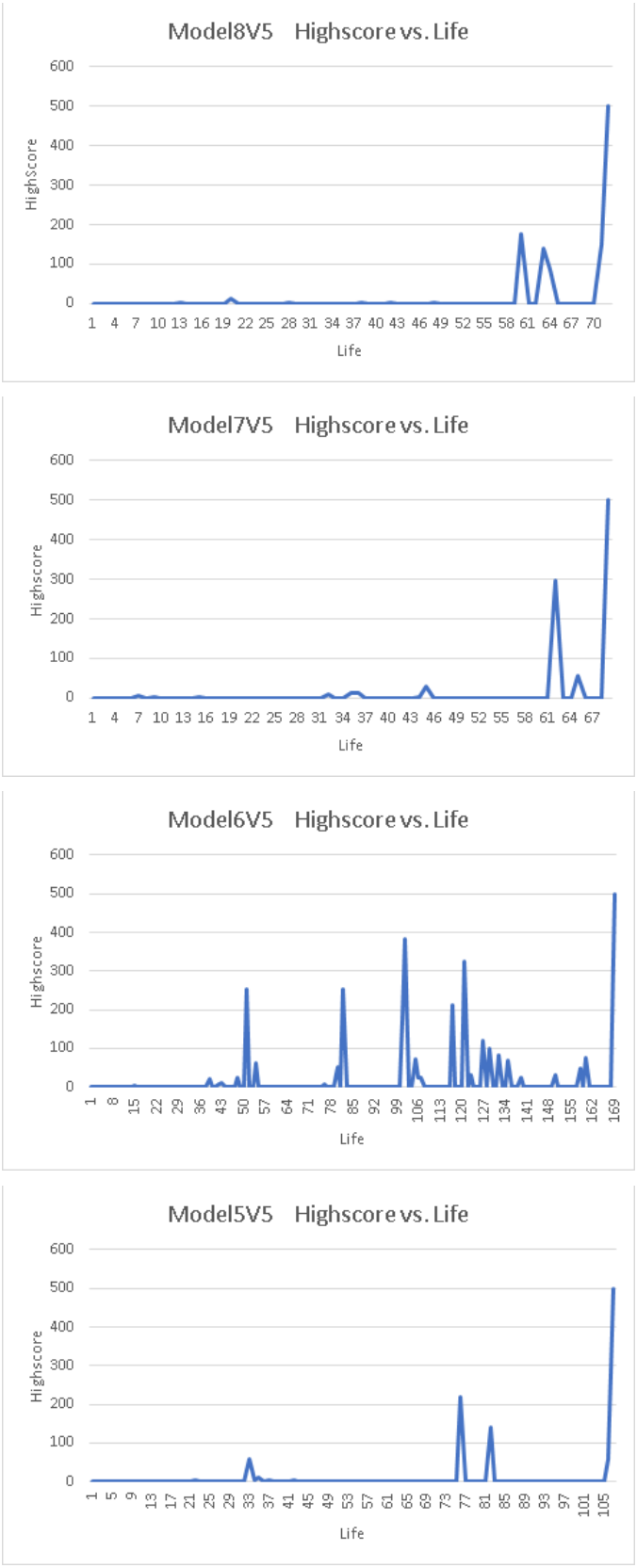

Fig 4. High score per life in four models

V. Conclusion and Future work

A. *Conclusion*

From the deep Q-learning model we built we can conclude that the game taxi is indeed winnable by using a reinforcement learning algorithm. We also found that the deep Q-learning algorithm is able to find unique solutions to problems that humans may not necessarily think of since people would tend to focus on the drop offs rather than a way to achieve the max score. The model successfully learned how to plan for future steps in the Taxi game by taking in the environment and previous actions. While the model was successful, there are improvements that could be made since the model takes so many steps to win in the final game.

B. *Future Work*

In the future we plan to try and continue upgrading the Taxi learner to be able to complete the task in less moves, as none of our models were able to complete the game quickly. Along with improving the Taxi learner, we plan to try to expand the model to play other games such as our original goal of playing Ms. Pacman.

References

[1] Campos, L., 2018. *Deep Reinforcement Trading ⋆ Quantdare*. [online] Quantdare. Available at: <https://quantdare.com/deep-reinforcement-trading/> [Accessed 2 August 2020].

[2] Choudhary, A., 2019. *Deep Q-Learning | An Introduction To Deep Reinforcement Learning*. [online] Analytics Vidhya. Available at: <https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/> [Accessed 2 August 2020].

[3] Mitchell, T., 1997. *Machine Learning*. New York: McGraw Hill.