Rocket Landing with Reinforcement Learning

University of California, Los Angeles

Aaron Zhang

March 10, 2023

Abstract

We are motivated in training an AI agent that is capable of landing a rocket through Reinforcement Learning. We aim to solve this problem by adapting Q-Learning, a widely-used Reinforcement Algorithm. Applying Q-Learning algorithm with well tuned hyperparameter and discretized observation space, our algorithm adequately prevents the rocket from crashing. We introduce a more performant algorithm that combines Q-Learning with neural networks, known as Deep Q-Learning algorithm. We further compare other modification to the network of Deep Q-Learning algorithm such as Double Deep Q-Network, and Dueling Deep Q-Network with previous methods.

1 INTRODUCTION

One of the primary reason to solve a problem like rocket-landing is the amount of resources it preserves. Quoting from the official website of SpaceX, "a fully and rapidly reusable rocket is the pivotal breakthrough needed to substantially reduce the cost of space access. The majority of the launch cost comes from building the rocket, which historically has flown only once"(SpaceX, 2023). Consequently, we are motivated to solve this problem with a focus on Reinforcement Learning.

Q-Learning algorithm is known for its simplicity. It is a model-free algorithm without following a specific policy. It is a good starting algorithm that we can later build upon for future improvements. To further simplify our problem, we use a fixed seed for terrain regeneration.

Training an AI model requires some forms of an environment. For the sake of simplicity, we pick OpenAI Gym API for the implementation of Q-Learning algorithm.

The environment has an action space of Discrete(4), and an observation space of an 8-dimensional vector. In this setup, six of the eight states are continuous, making the implementation of Q-table more complicated than usual. The rest of the states are boolean, making the dimension of the observation space (∞,∞,∞,∞,∞,∞,2,2).
The action space consists of integers, 0, 1, 2, and 3,
0: do nothing
1: fire left engine
2: fire main engine
3: fire right engine
and the observation space consists of x/y coordinates, x/y velocities, angle, angular velocity, whether each leg is touching the ground or not, adding up to a total number of 8 states (hence 8-dimensional vector).
Finally, the problem is said to be successfully solved if the agent gets a total reward over 200 within an episode.

An obvious approach to construct a Q-table with finite number of states is to discretize the state value of each state variable. There are number of ways to discretize an observation space, but the method we used required Q-table to have a dimension of $(n^6 \times 2^2) \times 4 : n \in \mathbb{Z}$, which may require a lot computing resources depending on n. The row of Q-table symbolizes the dimension of our observation space whereas the columns are simply our Discrete(4) action space. The choice of n is also commonly interpreted as n bins.

Another issue of Q-Learning algorithm is it being an off-policy algorithm. We will demonstrate our attempt on solving exploitation vs exploration. We found that the standard way of setting a constant ϵ has the potential to disrupt a training, or even break a well trained model.

We are able to partially solve most of the identified problems in our research. Our core idea was to build more components on the preexisting Q-Learning model. We further looked into popular adaptation of neural network such as Deep Q-Network, Double Deep Q-Network, and Dueling Deep Q-Network.

2 BACKGROUND

We know Q-Learning can be extremely resource hungry as number of states we discretize gets larger. In fact, the Q-table will eventually become infeasible for updating the Q-value.

To address this problem, we looked into the implementation of a variant of Q-Learning known as Deep Q-Learning, which was developed by a team of researchers at Google DeepMind (Mnih et al., 2015).

While Deep Q-Learning prevents the state from exploding, it has its issues. We found two variants that improves different perspectives of Deep Q-Learning that were both published at Google DeepMind a year after it was developed. Double DQN mainly aims to solve the overestimation behavior of DQN (van Hasselt et al., 2016), and Dueling Deep Q-Learning reduces ambiguity of a set of similar actions during training. (Wang et al., 2016).

3 METHODS

Q-Learning

To apply Q-Learning for our problem of interest, we need to first discretize the observation space into discrete states. A Q-table can then be created to function as the brain of the agent.

The first approach we came up was to reduce the observation space into Discrete(2), where 0 indicates $\theta < 0$, and 1 indicates $\theta > 0$. We believe that the angle of the rocket produces normal to the ground is important in balancing and landing the rocket. This method does prevent Q-table from exploding in its size, however its stability may be very sensitivity to the change of environment.

Q-table has a dimension of (number of states) $\times$ (number of actions). This approach aims to capture every state. Instead of simply reducing the number of state, it aims to discretize individual continuous states into Discrete(n). Hence, the original observation space (∞,∞,∞,∞,∞,∞,2,2) becomes (n,n,n,n,n,n,2,2), 8 discrete state variables.

We initially let our agent to pick the most optimal action at all time (greedy algorithm). We then use ϵ-greedy to balance exploitation and exploration of the model. One issue with the value of $\epsilon > 0$ is that the model tend to break even after it consistently reaching scores over 200. Hence, we believe it is necessary to fully eliminate ϵ after the model is successfully trained. Our approach in solving this is to set ϵ to zero after x episodes.

We further adjust our model by changing the number of discrete value for each states.

Given the discretized states, Bellman equation can be easily implemented to update Q-table for choosing the most optimal action.

Deep Q-Learning

To solve the dimensional exploding issue in our original Q-Learning model, we attempt to adapt the Deep Q-Learning algorithm. The idea of Deep Q-Learning is to fully replace Q-table with a Deep Neural Network (DNN) that can be trained during each episode, where its Q-function does not increase exponential increase in size like Q-table (Mnih et al., 2015).

Double Deep Q-Learning uses the online network to select the action and the target network to evaluate its value, thereby reducing the overestimation of action values (van Hasselt et al., 2016).

Dueling DQN uses a shared convolutional layer to extract features from the input, and then splits into two separate streams for the state value and the advantage function. The outputs of the two streams are combined to obtain the final Q-value estimates for each action (Wang et al., 2016).

4 THEORY

Q-Table

Q-table is a matrix with dimension (number of states) $\times$ (number of actions) that evaluates expected future reward by transitioning from a state to an action. The initial approach that consists of two states can be easily illustrated.

–	0	1	2	3
0	r_{00}	r_{01}	r_{02}	r_{03}
1	r_{10}	r_{11}	r_{12}	r_{13}

Discretization

Q-Learning cannot be directly implemented to continuous observation space because a Q-table must have a finite number of dimensions. The map from observation space to 8 discrete variable can be as simple as a linear transformation.

We can formalize this transformation as $L : \mathbb{R} \to \mathbb{N}$
We use $\mathbb{N}$ instead of $\mathbb{Z}$ simply because implementing Q-table in python requires the indices to be integers that are greater than or equal to zero.
Let $x_1, x_2, x_3, x_4, x_5, x_6 \in \mathbb{R}$
and $x_7, x_8 \in \mathbb{N} : x_7, x_8 < 2$
Let $y_1, y_2, y_3, y_4, y_5, y_6 \in \mathbb{N} : \forall y_i < 8$
and $y_7, y_8 \in \mathbb{N} : y_7, y_8 < 2$

$$L[(x_i)] = (y_i) : i \in \mathbb{Z}, D = 1 \leq i \leq 8$$

In fact, $\forall x_i \in D$ is uniquely bounded, we could simply map them to $\mathbb{N}$ with the given definition by solving for

$$y_i = n \times \frac{x_i - x_{\text{lower bound}}}{x_{\text{upper bound}} - x_{\text{lower bound}}}$$

, where $n = 8$ for this case.

We could use a smaller n, or a larger n. This number for discretization represents how sensitivity the agent reacts to the change of states.

Bellmen Equation

Q-table is trained using Bellman Equation, it can be expressed as
Let Q := Q-table
Let s := state
Let a := action
Let γ := exploration rate
Let x' := next x

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

The reward $Q(s, a)$ the agent receives from taking a (action) and applying it to s (state) is being updated by r (reward) plus the maximum reward for s' (next state). Overtime, each state will have the most optimal action entry maximized.

Exploration

Q-Learning does not converge on its own, so a policy that encourages exploration will greatly increase the performance of the model. Our choice of policy is ϵ-greedy, an additional randomness layer over the well known Greedy algorithm.

If classic greedy policy, which involves in taking the most optimal route at all time was used instead of ϵ-greedy, the learning of Q-table will be heavily dependent on its initialization. Therefore, exploration is required for Q-Learning to succeed. However, It is also necessary to fully eliminate ϵ later on to reduce any possibility for the agent to act randomly instead of optimally.

Deep Q-Network

The previous sections are mostly applicable to Deep Q-Learning, except that Deep Q-Learning uses different strategies to update the Q-value.

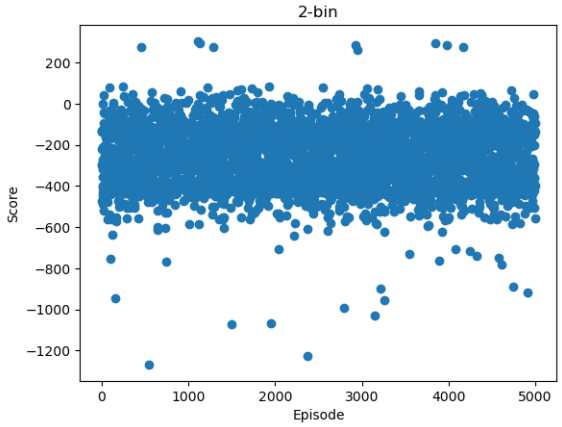
The structure of Deep Q-Learning has a deep convolutional neural network that takes input of current state and output the Q-values of all possible actions. DQN can simply take the 8-dimensional vector as input, which means it requires no discretization.

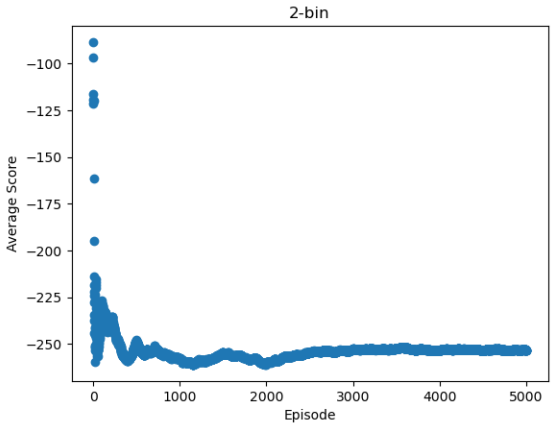
The theory of other Deep Q-Learning algorithms are fairly difficult to explain in rigorous mathematical terms. This paper will mainly focus on their results.

5 RESULTS

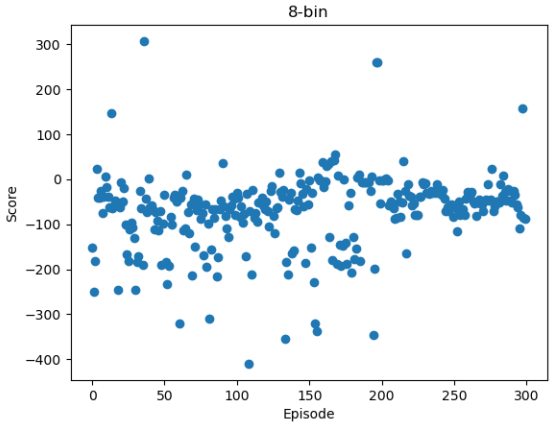
Initial approach

Our first attempt to use a Discrete(2) state was unsuccessful. The training showed no sign in improving within 5000 episodes. We believe the model is very unlikely to be trined successfully with only two states.



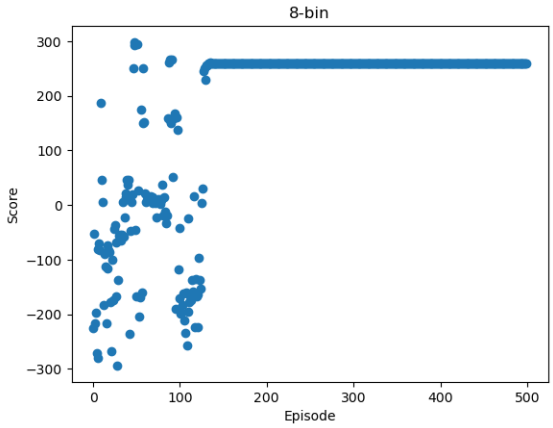


The first diagram where score versus episode suggests that all values of scores are randomly distributed over the period. Looking at the second diagram, the average score shows no sign of improvement over 5000 episodes, which further hints that the model is failing. Currently, only the angle is being considered. One possible improvement that doesn't involve in using too much computing resources is to discretize different angle values into much more finite discrete states instead of just positive angle and negative angle.



Discretization approach

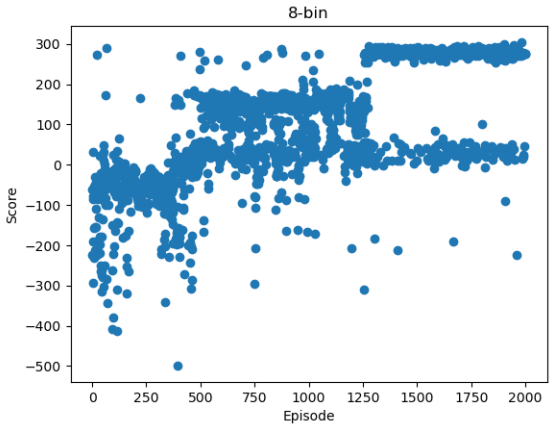
By choosing $n = 8$ for discretizing the observation space, the number of states is now finite, and we can construct a Q-table with dimension $(8^6)(2^2) \times 4 = 4194304$. Since the dimension of the Q-table have expanded exponentially due to this approach, the amount of time for completing the training also increased drastically.

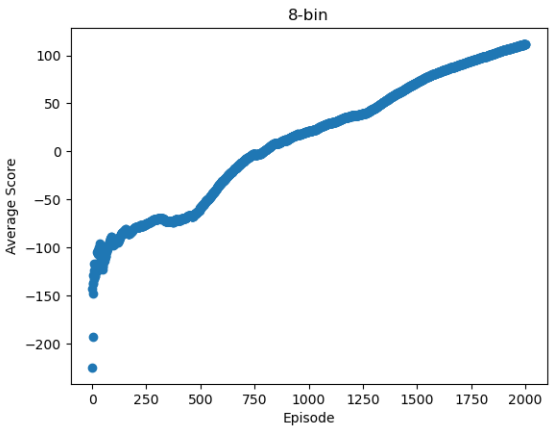


On the above graph, we can see the agent does not make a single mistake after around 140 episodes due to the elimination of ϵ. In this test, ϵ was assigned to zero after a predetermined amount of episodes. In fact, the same diagram can be reproduced when $\epsilon = 0$ from the beginning due to having a luck seed (We fixed terrain regeneration). This is however unreliable because the score may converge to a much lower value, reference to Appendix A.

Now, we again focus on the results on setting ϵ to zero from the start, hence using a pure greedy algorithm. The convergence of the score's value can significantly depend on the number of bins we take for discretization. This is also demonstrated to Appendix A.

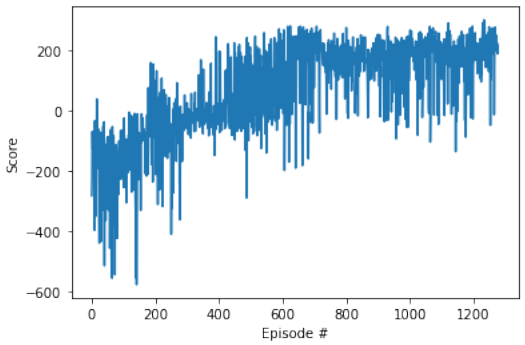
Most of the time, the scores show a trend of converging to a value that isn't optimal but also suggest that the rocket is not brutally crashed into the ground. We did not stay with the idea of having ϵ to be time dependent because it may also slow down the converge period, purely base on luck.





Q-Learning with ϵ-greedy policy does show an overall increase in its average score. Without letting ϵ to be zero after a period of time, the model is more applicable to every settings. However, the top diagram shows that even if the average score is increasing, it is still mostly jumping back and forth

between two possible converging candidates. It is tested that the average score will eventually reach a value of 200. This is illustrated in Appendix B.



We also present our implementation of DQN to solve this problem. The result showed an overall increases in its score. The stopping condition for this implementation depends on wether the average score is over 200. It is important to notice that DQN trains on random environment whereas our model uses the same seed in each episode. Therefore, DQN showed a much more impressive result.

6 CONCLUSION

We identified rocket landing as a Reinforcement Learning problem and Q-Learning with various tweaks to train the model. Although Q-Learning algorithm is a simple and straight-forward algorithm, it comes with challenges with real-world applications. It requires a lot efforts on fine tuning the perfect hyperparameter, and may still be outperformed by other Q-Learning derivatives such as DQN, and etc. Deep Q-Learning in this experiment showed significant boost in performance, which was expected from the start.

7 FUTURE RESEARCH

Other approaches such as changing the exploration policy may show different results. Especially because we have failed to deal with ϵ's "aftershock". Additionally, we could try to implement other Q-Learning motivated algorithms that doesn't require any implementation of a deep neural network. Greedy Gradient Q-learning is able to avoid the computation of Q-table with Temporal-Difference Learning (Szepesvári et al., 2010).

8 CONTRIBUTIONS

Aaron Zhang

Throughout the project, I utilized OpenAI Gym API in python for the simulation of the rocket landing. Specifically, I researched on Q-Learning as well as more advanced techniques such as Deep Q-Learning, Double Deep Q-Learning, and Dueling Deep Q-Learning. I regularly collaborated with my teammates on various tasks such as core implementations of the code, researches, and theorization of the model. After finishing this project, I have gained a better insight on various tasks and topics of machine learning, mainly the challenges with algorithm implementation.

Zhuoxian Lin

In this project, we all contributed to the success of this project. For many of the parts, all three of us collaborated and we also did parts independently. Overall, the collaboration was smooth for the team, and everyone on the team took responsibility for what they are assigned to do. Moreover, we not only finished what was assigned but also took initiative to explore and bring new ideas to the team every time we get a chance to meet. Aaron was a teammate that is always looking to explore new ideas and tries different methods. When I hear about his ideas, I would then start researching online and look to get an implementation out of his idea. Lastly, when we are faces obstacles, Yafei Dong never fails to find an alternative solution or different approach to tackling the problem. I am more than happy to be able to be in this team and work with my teammates. Below, I have bullet points for some more concrete contributions to the project, feel free to ask us questions.

- Early topic resource gathering: Aaron Zhang, Zhuoxian Lin, Yafei Dong
- Implementation of Classic DQN: Mostly online resource, minor tuning from all of us
- Implementation of Hyperparameter tuning for classic DQN: Zhuoxian Lin
- Implementation of Duo DQN: Zhuoxian Lin, Yafei Dong Bug fix: Aaron Zhang, Zhuoxian Lin

Yafei Dong

I was involved in exploring and gathering early topic resources related to the Lunar Lander problem and

reinforcement learning techniques. This likely involved researching existing literature, papers, and code repositories related to the topic to help inform the research and implementation process. Moreover, I was involved in the implementation of the Dueling DQN reinforcement learning technique as part of the research project. This likely involved writing code, testing and evaluating the model's performance, and fine-tuning the parameters of the model to achieve optimal results.

References

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2015). Playing atari with deep reinforcement learning.

SpaceX. (2023). *Mission*. https://www.spacex.com/mission/

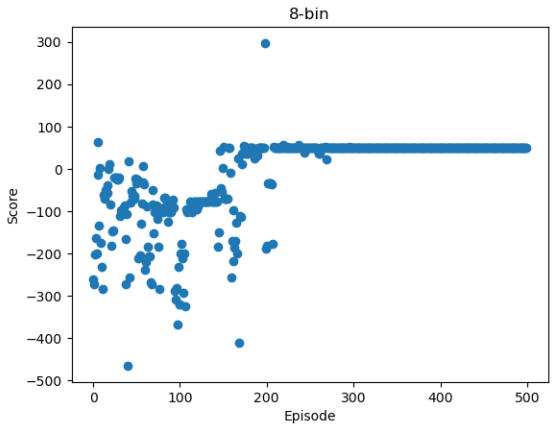
Szepesvári, C., Csapó, T. G., & Hegedűs, I. (2010). Control of temporal difference methods using model predictive control and its application to the game of tetris. *International Conference on Machine Learning (ICML)*, 1031–1038.

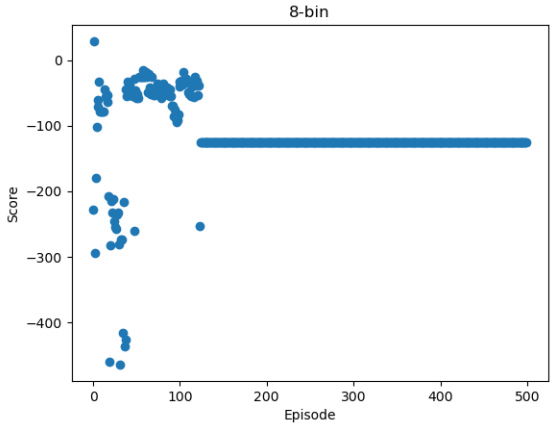
van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. *Thirtieth AAAI Conference on Artificial Intelligence*.

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., & de Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. *International Conference on Machine Learning*, 1995–2003.

9 Appendix

A Pure Greedy Algorithm





B Q-Learning