Solving the Lunar Lander Problem with Multiple Uncertainties using a Deep Q-Learning based Short-Term Memory Agent

Håkon Guttulsrud, Mathias Sandnes, and Raju Shrestha
Department of Computer Science, Oslo Metropolitan University (OsloMet)
Oslo, Norway
{s360394,s361766,raju.shrestha}@oslomet.no

ABSTRACT

Efficient space travel requires intelligent and robust control mechanisms during spacecraft landing scenarios. Developing a control mechanism for a rocket trajectory problem is inherently complex. This paper introduces a novel approach using Deep Q-Learning (DQL) with Short-Term Memory (STM) to address the intrinsic challenges of this task. Unlike traditional Q-Learning methods, our DQL STM agent performs in an environment with uncertainties such as starting position, gravity, and wind in both training and simulation, allowing for enhanced robustness in difficult environmental conditions. This adaptation enables the agent to observe *n*-previous state-action pairs, offering a more accurate estimation of environmental dynamics. Experiments demonstrate that this new approach yields better results under stricter testing conditions compared to previous methods. Moreover, we establish the innovative aspects of our methodology through systematic comparisons with basic Q-Learning, highlighting the merits of the DQL STM agent.

CCS CONCEPTS

• **Computing methodologies → Reinforcement learning**.

KEYWORDS

Lunar Lander, Reinforcement learning, Deep Q-learning (DQL), Uncertainties, Short-Term Memory (STM), Non-Markovian environments, Robust control

ACM Reference Format:
Håkon Guttulsrud, Mathias Sandnes, and Raju Shrestha. 2023. Solving the Lunar Lander Problem with Multiple Uncertainties using a Deep Q-Learning based Short-Term Memory Agent. In *2023 12th International Conference on Computing and Pattern Recognition (ICCPR 2023), October 27–29, 2023, Qingdao, China.* ACM, New York, NY, USA, Article 4, 7 pages. https://doi.org/10.1145/3633637.3633641

ICCPR 2023, October 27–29, 2023, Qingdao, China
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0798-8/23/10.
https://doi.org/10.1145/3633637.3633641

1 INTRODUCTION

LunarLander-v2 is a two-dimensional environment developed by OpenAI in the Gym toolkit [1]. The Lunar Lander problem aims to successfully land a rocket-propelled spacecraft in moon-like conditions as quickly and safely as possible. Previous attempts to solve the Lunar Lander problem without additional uncertainties have been successful with heuristics and Reinforcement Learning (RL) techniques such as Q-learning, Deep Q-learning (DQL), Sarsa, and model-based methods [2, 3, 5, 10, 16]. These approaches, however, fail to address the uncertainties inherent in the environment, making them unsuitable for more complex scenarios.

The introduction of uncertainties into the Lunar Lander environment makes it non-Markovian, meaning a single observation does not necessarily contain all the information necessary to understand the entire state of the environment [2]. While some researchers have proposed methods to handle individual uncertainties, they have not accounted for multiple uncertainties combined. This limitation highlights the need for a more sophisticated approach that can address complex situations with various uncertain elements.

We propose a novel approach using Deep Q-Learning with Short-Term Memory (DQL STM) to develop an advanced and robust control mechanism capable of landing a spacecraft on any planet in any environment with uncertainties. Unlike previous work, our method introduces uncertainties such as gravity, wind direction and speed, and starting position into the training process in the lunar lander environment. This innovation forces the agent to adapt to unseen challenges, improving its robustness and allowing for a more accurate estimation of the environment dynamics and its trajectory. Our approach is rigorously evaluated systematically and controlled to demonstrate its performance across a wide range of uncertainty combinations, thus proving its merits over traditional techniques.

The rest of the paper is organized as follows. We introduce the Lunar Lander environment in Section II. Section III describes related works. Section IV presents the proposed uncertainties and our novel model. Experimental setup and experiments are described in Section V. Results are presented and discussed in Section VI. Finally, Section VII gives a conclusion to the paper.

2 THE LUNAR LANDER ENVIRONMENT

In the LunarLander-v2 environment, the spacecraft has three controllable engines, a powerful main engine located at the base of the spacecraft, and two lighter engines on each side. In the discrete version of LunarLander-v2, only one engine can fire at once, and the engines can be turned completely on or completely off. However, there also exists a continuous version of the environment where these restrictions are not present. By default, the gravitational force is comparable to the forces on the moon. Figure 1 shows a screenshot of the running environment. The state and action space and reward in the environment are described below.

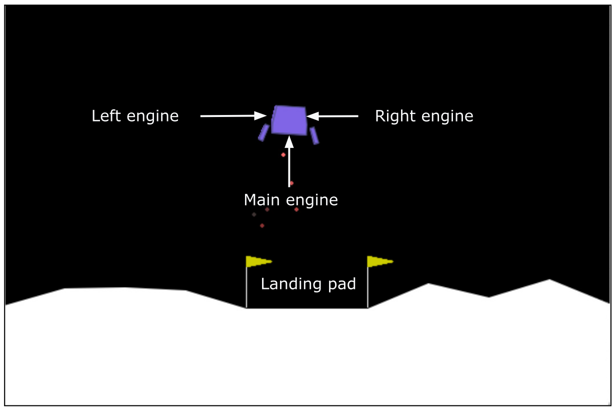

Figure 1: Lunar Lander environment in the OpenAI Gym.

2.1 State and action space

The state space of the environment contains information about the spacecraft itself, shown in Equation 1. The agent observes its position and speed in the two dimensions, angular momentum, and velocity, and which legs are in contact with the ground.

$$State\ variables \longrightarrow \begin{cases} x: & x\ position \\ y: & y\ position \\ v_x: & x\ velocity \\ v_y: & y\ velocity \\ \theta: & angular\ orientation \\ v_\theta: & angular\ velocity \\ r: & right\ leg\ ground\ contact \\ l: & left\ leg\ ground\ contact \end{cases} \tag{1}$$

The agent can perform one of four possible discrete actions. The actions are to do nothing, fire the left orientation engine, fire the right orientation engine, or fire the main engine. Firing an engine is a binary operation, meaning the engine must be turned completely on or off. Firing the main engine exerts 10x the force as the side engines. After a successful landing, the agent must be stationary for a set amount of timesteps to terminate the episode.

2.2 Reward

The reward function for the environment is calculated with Equation 2.

$$r_t = \begin{array}{l} -100 * \sqrt{x\ position^2 + y\ position^2} \\ -100 * \sqrt{x\ velocity^2 + y\ velocity^2} \\ -100 * |angularvelocity| \\ -100 * has\ crashed \\ -0.3 * bottom\ engine\ on \\ -0.03 * side\ engine\ on \\ +10 * right\ leg\ ground\ contact \\ +10 * left\ leg\ ground\ contact \\ +100 * has\ landed \end{array} \tag{2}$$

The agent receives a positive reward for landing safely, while unsafe behavior and moving away from the target receive a negative reward. The spacecraft must land within a specific range along the X-axis to achieve the highest possible reward. The spacecraft may land outside the range, but the reward is lowered. The episode is terminated if the spacecraft crashes or landing is not successfully performed within the required time limit. The reward is measured in points, and a reward of 200 points or more is considered solved. If the spacecraft moves away from the landing pad, the reward decreases. Additionally, if the spacecraft crashes, it receives a negative reward. Firing the main engine decreases the reward by 0.3 points, and firing the smaller side engines decreases it by 0.03 points. The spacecraft has two legs; if either leg comes into contact with the ground, the reward is increased by 10 points for each leg. If the legs lose contact, the respective 10 points for that leg are subtracted from the score. Additionally, the reward is determined by the angular velocity of the spacecraft. The reward function does not consider the spacecraft's angular position.

After calculating the reward for the current timestep, the previous reward is subtracted, resulting in the received reward R_t, expressed in Equation 3.

$$R_t = r_t - r_{t-1} \tag{3}$$

3 RELATED WORK

Using multiple timesteps in reinforcement learning has previously been studied and can be categorized into two approaches. The first approach is the forward view, where each state is updated by looking forward to n future rewards and states. The other approach is the backward view, where each state is updated by looking back to previous n rewards and states [13]. The forward view method has been researched extensively, showing improved performance over the 1-step method [6]. However, the use of a backward view has not been as thoroughly researched for DQL in RL problems.

Mnih et al. [7] proved that DQL could handle complex problems without looking at the inner workings of the environment. They continued the work proposed two years earlier and caused a boom in DQL approaches for RL [8]. This work proved that DQL could perform at a superhuman level in perfect information environments. Using DQL and experience replay, Yu [16] solved the default problem. They found that the Lunar Lander problem favors wide but not deep neural networks.

Lu et al. [5] solved the Lunar Lander problem using a model-based approach where instead of learning the system's dynamics, a model directly learns the optimal parameters for controlling the spacecraft. Gou and Liu [3] proposed a novel approach by combining a model-based and model-free agent, evaluating the performance in two Gym environments, Mountain Car and Lunar Lander. They propositioned that transitions stored in the replay memory have little informative correlation to the reward signal. The transitions can be used to learn the environment dynamics by posing the problem as a supervised learning approach. Instead of dealing with the stochastic nature of the epsilon greedy algorithm [11], they proposed one-step planning during exploration that explored environments better and faster by predicting the next state. Although the method improved over model-free DQL for Mountain Car, the performance did not transfer to the Lunar Lander problem.

Other researchers have compared the performance of Deep Q-Networks (DQN) to Policy Gradient (PG) in the Lunar Lander environment [12]. They found that the stochastic PG method performed relatively disadvantageously compared to DQN, as the episodic Lunar Lander environment is deterministic. The PG method reached upwards of 160 points in the environment, while the DQL-based method reached 200 points. In their concluding remarks, they note that the validity of their results is limited to the deterministic Lunar Lander environment and could potentially observe different results in the real world under uncertainty. Nugroho [10] used DQL to solve the problem and compared the performance to a heuristic approach using classical Q-Learning [15]. They found that the heuristic approach outperformed the DQL. However, the heuristic approach is carefully hand-crafted to the default Lunar Lander environment, which suggests that the approach would not be able to handle environments with added uncertainty. In the case of Lunar Lander, introducing multiple uncertain elements such as variable gravity and wind, causing the environment to be non-markovian, would argumentatively reduce the performance of a heuristic approach.

Gadgil et al. [2] have solved the problem with three different uncertainties, random force acting on the agent, noisy observation data, and stochastic engine failure. The researchers sought to solve the problem with several approaches, including a Sarsa agent that reached average rewards of 170+. The Sarsa agent is an on-policy algorithm that chooses the highest Q value using an exploration policy, in their case, the epsilon-greedy algorithm. They show that the agents can adapt to the environment under uncertainty and attain rewards of 100+. However, they found that the Sarsa agent performed poorly without reducing the dimensionality of the environment by implementing a state-discretization scheme. It could be argued that their Sarsa agent could see an increased reward if they were to combine the state discretization with the supervised learning approach proposed by Gou and Liu [3]. Their DQL-based agent attained marginally better results and solved the problem with rewards above 200. However, they tested the agents only on individual uncertainties in isolation and did not combine multiple uncertainties to assess agent performance in more complex conditions. In their concluding remarks, the researchers say that further work should be done to combine uncertainties to assess how agents would adapt to the increased uncertainty.

4 PROPOSED UNCERTAINTIES AND MODEL

4.1 Uncertainties

We introduce three new uncertainties to the Lunar Lander problem and combine uncertainties for a more challenging and realistic landing situations. The uncertainties are the start position of the lander, gravitational force, and wind speed and direction. These uncertainties change the properties of the environment, causing it to become non-Markovian. At the time of our research, only gravity and wind were implemented in the Lunar Lander environment. Therefore, we had to customize the source code of the Lunar Lander environment to allow for variable start positions. The uncertainties are described below.

Start position: The start position determines where the spacecraft spawns along the X-axis for each new episode. The varying start position allows the flight controller to be initiated in a broader range of situations instead of directly above the landing pad as in the default environment. The environment's default value of the start position is 300. Here, we used the range 0 to 550, covering the full width of the environment as shown in Figure 2.

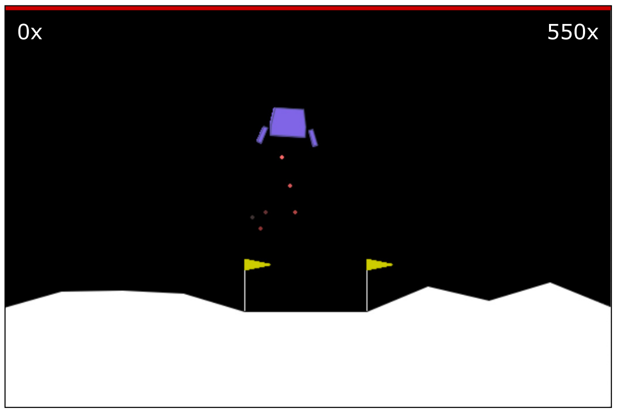

Figure 2: Uncertain start position.

Gravity: Changing the gravitational force ensures the agent can land on a broader range of moons and planets. Gravity will heavily influence the velocity and behavior of the agent and how it needs to utilize the engines. The environment default for gravity is -10, while we define a gravity range between -15 and -5. With gravitational force stronger than -15, the legs of the spacecraft can no longer support the body's weight. In addition, the engines are too powerful for gravitational forces weaker than -5. Figure 3 illustrates uncertain gravity in the Lunar Lander environment.

Wind: Wind has been introduced so that the landing procedure can function in variable climates. The wind will affect the spacecraft's trajectory. The direction of the wind is selected at random and can originate from any angle in the 2D space (see Figure 4). Wind is enabled with wind speed values between but not including 0 and 20.

The three uncertainties have been combined, and the problem has to be solved without changing the spacecraft's body, engine positions, number of engines, and engine thrust.

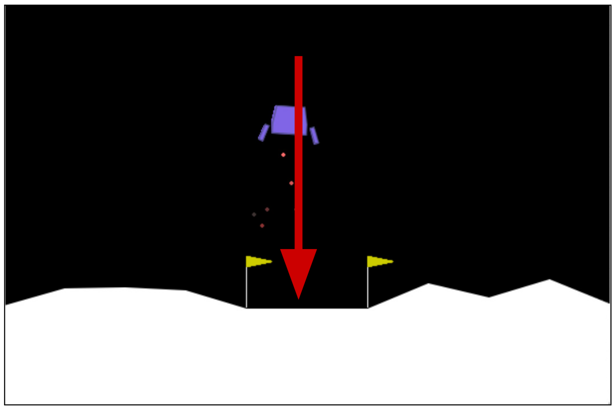
Figure 3: Uncertain gravity.

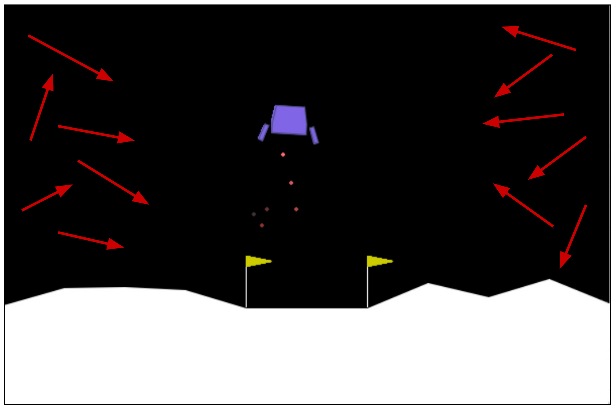
Figure 4: Uncertain wind.

4.2 Model

We propose a Short Term Memory DQN-based learning agent that observes its environment through a queue containing the current state and n number of previous state-action pairs, making the agent a backward-looking multiple timestep agent. Our experiments found the ideal value to be the current and five previous state-action pairs, totaling six timestep states that are used as input to the agent. The DQN consists of an input layer, two hidden layers, and an output layer (see Figure 5). The input layers contain one-hot encoded current state, previous states, and previous actions. Both hidden layers use ReLU [9] as the activation function and have 128 nodes. This network architecture is inspired by the DQL agent proposed by Gadgil et al. [2]. Both hidden layers in their DQN have 64 nodes. Using this as a starting point, we have run Hyperparameter Optimization (HPO) schemes to test various hyperparameters and network architectures. Architectures with different depths and breaths were explored, resulting in the final model architecture. The network has four linear output nodes, one for each action. The output of each output node is an estimated expected return for performing each action given the input state. The loss is calculated with Mean-Squared-Error (MSE) [14], and the network is updated with the Adam optimizer [4] and the learning rate 0.001. The exploration rate is set to 0.2, and the discount factor is set to 0.99.

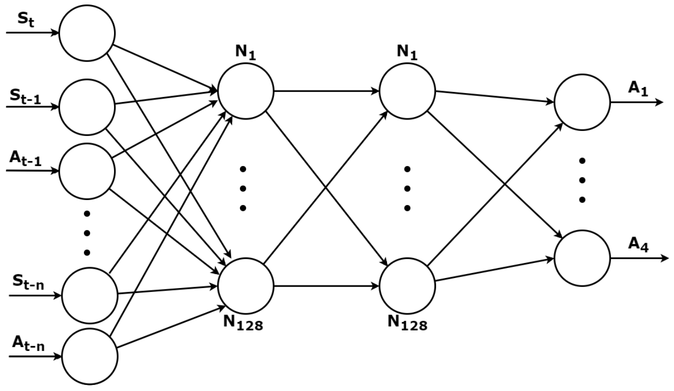

Figure 5: Deep Q-Network used in the proposed DQL model.

Algorithm 1 Training of the agent.

```
initialize Agent
for every episode do
    load uncertainty values
    build environment
    for every timestep do
        decide action
        step environment
        save experience
        train agent on random batch
    end for
    evaluate agent
end for
```

The model's training is episodic as expressed algorithmically in Algorithm 1. At the start of every timestep, the agent interacts with the environment. The data relating to that interaction is then saved in the experience replay buffer. The relevant data for a timestep is the state, action, new state, reward, and whether or not the episode should be terminated. This cycle of observing, acting, and collecting experience is repeated until the episode terminates.

At every timestep, the model is trained on one batch of data after the current timestep data has been stored. The training data is selected stochastically from the experience replay buffer. During training, the input is the current state, while the desired output is calculated by looking at the received reward for the current state, the next action, the discount factor, and if the episode should be terminated. The network's output estimates the Q-value, which is the expected return for performing a specific action in a specific state.

The agent has a set amount of time before the episode terminates, determined by the amount of maximum allowed timesteps, set at 1000. During training, the agent will periodically choose random actions, determined by a static exploration rate of 20% using the epsilon-greedy algorithm.

The introduction of a backward-looking multiple timestep agent through the Short Term Memory Deep Q-Learning model signifies a novel approach to handling non-Markovian environments. By observing both the current state and previous state-action pairs, our model gains enhanced adaptability and robustness in the face of uncertainties. The optimal configuration of six timestep states, determined through meticulous experimentation, sets our model apart from conventional Q-Learning methods. The well-tuned hyperparameters and network architecture, combined with the episodic training regimen, ensure a balance between exploration and exploitation. The successful handling of multiple uncertainties reaffirms the merits of this innovative approach, extending the boundaries of existing solutions in Reinforcement Learning.

5 EXPERIMENTS

The proposed DQL STM agent and the model are implemented and tested with the three new uncertainties individually and combined. The agent was trained and tested with individual uncertainties by introducing one uncertainty at a time. With combined uncertainties, the agent was tested with all uncertainties enabled simultaneously. The same experiments were conducted with the state-of-the-art model and agent from Gadgil et al. [2], which will be referred to as a Naive agent, to test how this agent would adapt to the new uncertainties and to compare the results from the proposed model. The implementation was done with Python 3.7 using Keras and Tensorflow 2.8. ReLU activation function, Adam optimizer, Mean squared error (MSE) loss function, and batch size of 32 were used in both agents.

As some combinations of uncertainties are more challenging, the evaluation can not be performed with random uncertainty values for each evaluation. In addition, the goal is not to find an agent proficient in one specific combination of uncertainties. Therefore, the selected uncertainty values have to be the same for each evaluation, and each agent needs to be evaluated in multiple episodes. The agent should not be evaluated more than necessary to reduce training time. Because of this, we have used a new testing method when agents are tested with combined uncertainties. The method is described below.

Testing method: The testing method consists of a robust evaluation scheme, as shown in Figure 6. The test has a set amount of unique uncertainty combinations, resulting in eight evaluation episodes with all uncertainties enabled. When an agent performs proficiently, it means that the agent is highly adept at solving the Lunar Lander problem with various combinations of uncertainties.

The robust test is initialized by selecting values periodically from each uncertainty range, ranging from the minimum to the maximum. All possible combinations of the selected values are combined. The evaluation selects two values between the minimum and maximum of each uncertainty range. This can be imagined as a 3D plot, with each uncertainty as an axis. The selected points are the corners of a cube that fits inside the plot. This means that every evaluation is run with 8 unique combinations of uncertainties, sampled evenly.

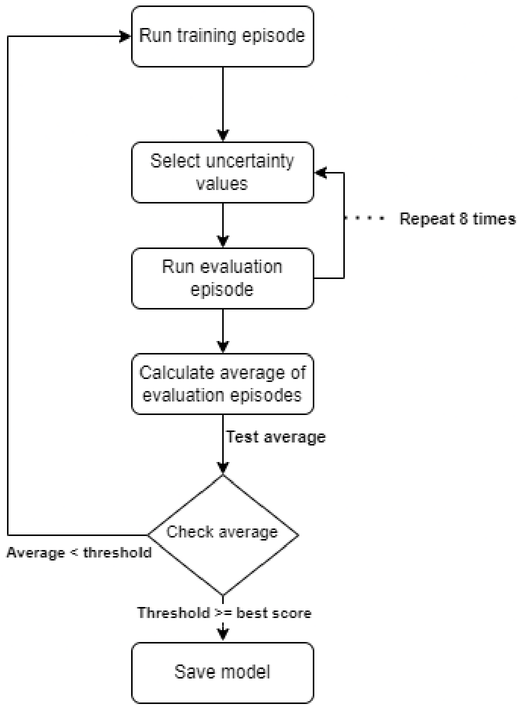

Figure 6: Flowchart depicting the testing method used to evaluate DQL agents with combined multiple uncertainties.

6 RESULTS AND DISCUSSION

At the end of each training episode, the agents are evaluated two times for specific uncertainties and eight times with all uncertainties with uncertainty values selected periodically from their ranges. The results are presented as the average score (reward) received in overall evaluation episodes. Figure 7 shows the results of the Naive agent when evaluated with individual uncertainties.

The results show that the Naive agent could not solve the problem with either uncertainty. However, the best results were obtained with a random start position, and the agent was close to solving the problem. The agent could not deal with uncertain wind and gravity because the introduction of uncertain wind and gravity makes the environment to be non-Markovian, i.e., a single observation does not fully describe the state of the environment. This confirms that these uncertainties are arguably more challenging to handle than what was introduced by Sadavarte et al. [12].

Figure 8 shows the results from the performance of the proposed agent running the environment with individual uncertainties in isolation. As observed, the proposed agent is able to handle the challenge of individual uncertainties more optimally than the Naive agent. The agent is able to reach average rewards of over 200 points multiple times for every 8 evaluations per episode.

Because the Naive agent could not handle the individual uncertainties, they predictably did not succeed with the introduction of multiple simultaneous uncertainties. The evaluation results from the proposed agent and the Naive agent when all three uncertainties are enabled simultaneously are shown in Figure 9.

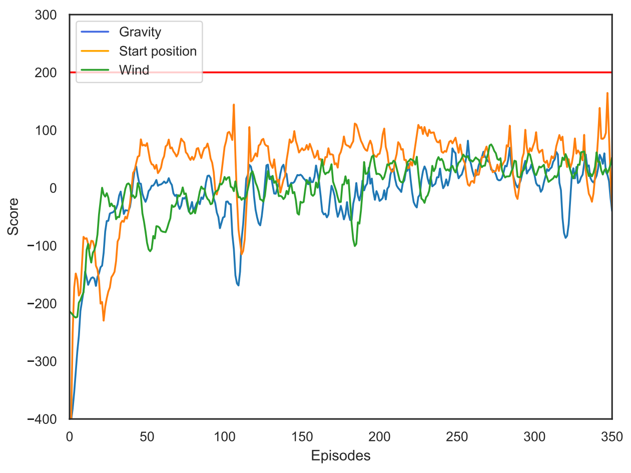

Figure 7: Results from the Naive agent with individual uncertainties in isolation.

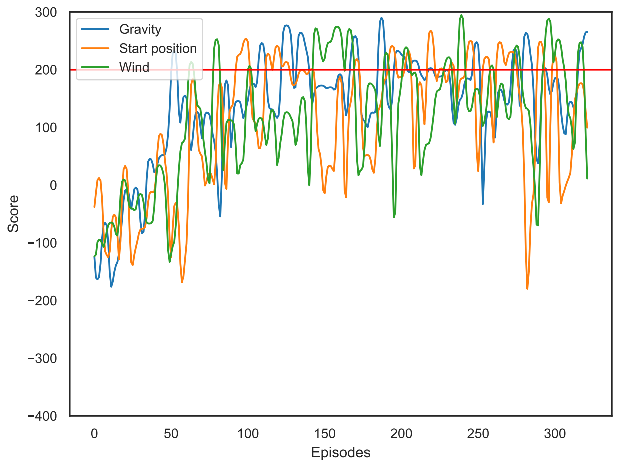

Figure 8: Results from the proposed agent with individual uncertainties in isolation.

From the scores in Figure 9, we observe that the proposed DQL STM agent solved the problem by achieving an average of over 200 points for some evaluations, while the Naive agent never solved the problem. The trend for the former is more positive than the latter over time, and it warrants further research to explore if the proposed agent would be able to perform better given more training time. We believe that the environment context gained from observing previous states and actions is attributed to the improved performance compared to the other agents. The extra context of previous state-action pairs allows the agent to estimate its trajectory better and better understand how the uncertainties affect it.

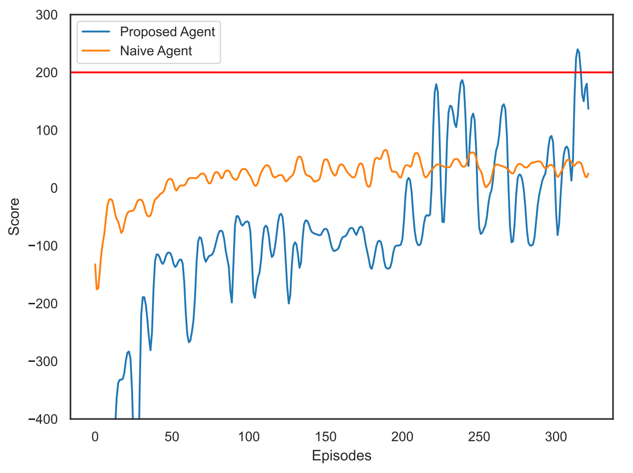

Figure 9: Results from the Naive and proposed agents, with all uncertainties enabled.

7 CONCLUSION

In this study, we tackled the Lunar Lander problem under the influence of complex uncertainties, such as starting position, gravity, and wind. Traditional DQL agents struggle with these uncertainties, as the non-Markovian nature of the environment prevents a complete understanding of the state in a single observation. To address this challenge, we proposed a novel Deep Q-learning model equipped with a Short-Term Memory (DQL STM) agent that observes previous state-action pairs which allows for a more accurate estimation of the environmental state, thus leading to successful landings even under strict testing conditions. The innovative integration of Short-Term Memory into Deep Q-learning improves the capability to perform at a high level in a non-Markovian environment. This achievement distinguishes our method from basic Q-learning, showcasing its merits and robustness in handling multiple uncertainties.

Future work could explore more sophisticated recurrent techniques, including RNN or LSTM-based layers [1], to further refine the estimation of the spacecraft's trajectory. The success of this research offers promising directions for enhancing control mechanisms in stochastic and uncertain environments.

REFERENCES

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI gym. *arXiv preprint arXiv:1606.01540* (2016).
[2] Soham Gadgil, Yunfeng Xin, and Chengzhe Xu. 2020. Solving the lunar lander problem under uncertainty using reinforcement learning. In *2020 SoutheastCon*, Vol. 2. IEEE, 1–8.
[3] Stephen Zhen Gou and Yuyang Liu. 2019. DQN with model-based exploration: efficient learning on environments with sparse rewards. *arXiv preprint arXiv:1903.09295* (2019).
[4] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[5] Yingdong Lu, Mark S Squillante, and Chai W Wu. 2019. A control-model-based approach for reinforcement learning. *arXiv: 1905.12009* (2019).

[6] Lingheng Meng, Rob Gorbet, and Dana Kulić. 2021. The effect of multi-step methods on overestimation in deep reinforcement learning. In *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 347–353.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[9] Vinod Nair and Geoffrey E Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *ICML*.

[10] Larasmoyo Nugroho. 2021. Powered Landing Guidance Algorithms Using Reinforcement Learning Methods for Lunar Lander Case. *Jurnal Teknologi Dirgantara* 19, 1 (2021), 43–56.

[11] Paavai Paavai Anand et al. 2021. A Brief Study of Deep Reinforcement Learning with Epsilon-Greedy Exploration. *International Journal Of Computing and Digital System* (2021).

[12] Rohit Sachin Sadavarte, Rishab Raj, and B Sathish Babu. 2021. Solving the Lunar Lander Problem using Reinforcement Learning. In *2021 IEEE International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS)*. IEEE, 1–6.

[13] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction.* MIT press.

[14] Philip Thomas and Emma Brunskill. 2016. Data-efficient off-policy policy evaluation for reinforcement learning. In *International Conference on Machine Learning*. PMLR, 2139–2148.

[15] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3 (1992), 279–292.

[16] Xinli Yu. 2019. Deep Q-Learning on Lunar Lander game. https://www.researchgate.net/publication/333145451_Deep_Q-Learning_on_Lunar_Lander_Game.