Deep Q-Learning Optimal Parameter Search: Lunar Lander Simulation

Shilpa Kancharla
Department of Computer Science
North Carolina State University
Raleigh, NC
skancha@ncsu.edu

***Abstract*—The purpose of the following reinforcement learning experiment is to investigate optimal parameter values for deep Q-learning (DQN) on the Lunar Lander problem provided by OpenAI Gym. The `LunarLander-v2` is an environment with uncertainty and this investigation explores optimal parameters that will maximize the mean reward over 400 episodes or less. A deep learning network is designed for the agent and various reinforcement learning parameters are used to carry out the simulation. Through the use of a neural network with two hidden layers, the agent was able to converge to a mean average reward score of 200 with $\epsilon = 0.9$, ϵ-decay $= 0.995$, $\alpha = 0.001$, and $\gamma = 0.99$ in a little over 250 episodes. A comparative analysis between different parameters used is also performed. The results and the architecture of the model used from this experiment are also compared to other similar experiments that employ the DQN method for the Lunar Lander problem.**

I. Introduction

A. Reinforcement Learning

In reinforcement learning, there exist two components: an agent and the environment it acts in. The agent takes actions while the environment gives out rewards or penalties based on those actions. Each action an agent takes results in it transitioning to a new state, as shown in Figure 1. In this process, we attempt to find the optimal actions that lead to the highest reward [1].

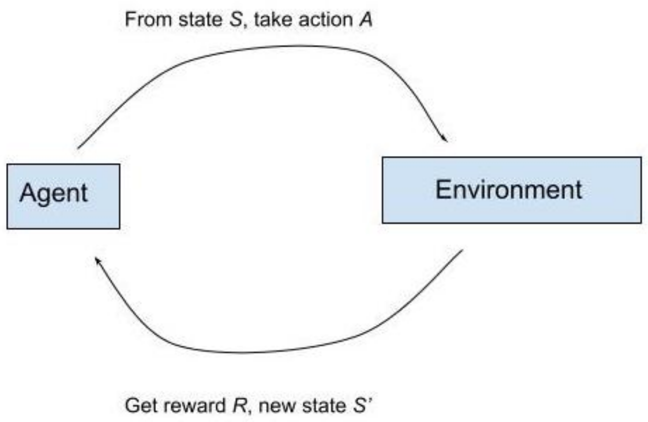
From state *S*, take action *A*
Agent
Environment
Get reward *R*, new state *S'*

Fig. 1. Interaction between agent and environment

Reinforcement learning solves a particular kind of problem in which decision-making is sequential and the goal is long-term. An agent that uses reinforcement learning should aim to maximize its reward. This is typically done by exploring an environment and gaining rewards or incurring penalties by taking actions. The measure of the reward should be proportional to how beneficial the action is. This information is fed back into the agent, which assesses what action would be beneficial to take next. Taking actions that have beneficial results and can maximize rewards at that moment is known as exploitation. A proper reinforcement learning agent should be able to have an appropriate trade-off between exploration and exploitation. While different types of reinforcement learning algorithms exist and can be applied to the Lunar Lander problem, deep Q-learning is used to find optimal parameters in this investigation.

B. Lunar Lander Simulation

This experiment uses `Box2D` in the simulation and the `LunarLander-v2` environment from the OpenAI Gym in order to visualize the performance of the agent as it interacts with its environment. Gym is an OpenAI toolkit that is used to visualize agent performance from the reinforcement algorithms they run on. The environment simulates the situation in which a lander needs to land at a specific location under low-gravity conditions, and has a well-defined physics engine implemented.

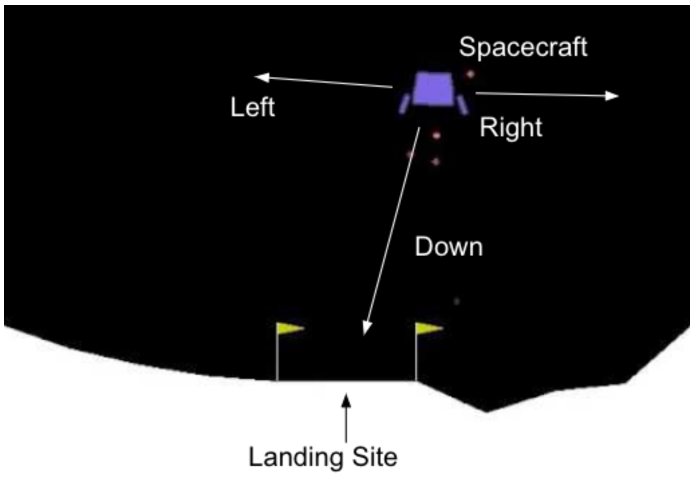

Fig. 2. Visualization of the lunar lander problem

The main goal is to land the spacecraft between two flags in a stable manner. The state space is continuous in real physics, but the action space is discrete. The landing pad, as shown in Figure 2, is always situated at coordinates (0, 0). The reward for moving from the top of the simulation screen to the landing site ranges from 100 to 140. If the lander moves away from the landing site, it loses reward points. The episode finishes when the lander crashes or comes to rest, for which it may receive either -100 or 100 points. Each leg to ground contact instance is worth 10 points.

Firing the main engine is -0.3 points per frame. Solving the simulation is worth 200 points. Moreover, it is possible to land outside the landing site. In this simulation, the amount of fuel available is infinite, so the agent can fly for as long as possible before it lands. The state and actions spaces are described further in the following sections.

C. State Space

There are eight state variables associated with the state space for the `LunarLander-v2` environment [10]. They are described as the following:

- x: horizontal coordinate of the lander
- y: vertical coordinate of the lander
- v_x: horizontal velocity of the lander
- v_y: vertical velocity of the lander
- θ: orientation in space
- v_θ: angular velocity
- Left leg touches the ground (Boolean)
- Right leg touches the ground (Boolean)

The coordinate values are given relative to the landing pad at (0, 0). The position of the landing pad changes at every iteration.

D. Action Space

The discrete `LunarLander-v2` environment is used for this analysis. Four discrete actions are possible [10]: do nothing, fire left orientation engine, fire main engine, and fire right orientation engine. If the left and right engines are fired, these actions introduce torque to the lander which causes it to rotate. This rotation makes stabilizing the lander difficult.

E. Q-Learning

A reward or penalty is expected for actions taken at every episode. If the actions that yield the maximum reward are known in advance, the agent has the ability to choose such actions to perform. It will perform a sequence of actions that eventually generate the highest reward yield. The total reward is also known as the Q-value and can be written as [1]:

$$Q(s,a) = r(s,a) + \gamma \max_a Q(s',a)$$

The above equation contains the following components:

- s: particular state
- a: action
- s': next state
- $r(s,a)$: immediate reward plus the highest Q-value possible from the next possible state
- γ: discount factor (controls the contribution of rewards further in the future)

$Q(s',a)$ in turn depends on $Q(s'',a)$, which will have a coefficient of γ^2. Therefore, the Q-value depends on Q-values of the future states, as displayed in this equation:

$$Q(s,a) \rightarrow \gamma Q(s',a) + \gamma^2 Q(s'',a) + ... + \gamma^n Q(s''^{\dots n},a)$$

Adjusting the γ value will decrease or increase the contribution of future values. With this recursive equation, arbitrary values are assigned to the Q-values. As the iterations continue and experience is collected, the equations converge to an optimal policy. This can be implemented as a Q-value update derived from the Bellman equation (denoted by the following equation) [2]:

$$Q(S_t,A_t) \leftarrow$$
$$Q(S_t,A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)]$$

It is possible to apply Q-learning to this problem because there is a certain amount of uncertainty present [14]. In the Lunar Lander simulation, there are many states present with many actions available per state. It becomes very hard to keep track of all the different states and actions for each state. Two issues are present if just Q-learning is applied [3]:

1) The amount of memory required to save and update the Q-learning table would increase as the number of states increase.
2) The amount of time required to explore each state to create such a Q-table would be incredibly long.

Due to these memory and time challenges, a modified version of Q-learning is applied in this situation known as deep Q-learning (DQN). The DQN model allows us to account for the continuous state space [5].

II. Methodology

A. Algorithm

In DQN, we use a multilayer perceptron to approximate the Q-value function. The inputs into the network are the eight state space values for all the state-action pairs for that state and the Q-value of all possible actions is generated as the output. The steps involved in this algorithm are as follows [5]:

1) All past experience is stored by the user in memory.
2) The next action is determined by the maximum output of the Q-network.
3) The loss function used is the mean squared error of the prediction Q-value and the target Q-value. However, how do we deal with a target whose value is not explicitly known?

$$Q(S_t,A_t) \leftarrow$$
$$Q(S_t,A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)]$$

Referring to the equation above once more, note that $R_{t+1} + \gamma \max_a Q(S_{t+1},a)$ represents the target. We argue that it predicts its own value but since R is the unbiased true reward, the network is going to update its gradient using backpropagation to finally converge to a value.

The pseudocode for DQN is presented below in Algorithm 1 [5]:

Algorithm 1 Deep Q-Learning (DQN) Algorithm

```
Start with $Q_0(s,a)$ for all $s,a$
Get initial state s
for i = 1, 2, ..., till convergence do
    Sample action $a$, get next state $s'$
    if $s'$ is terminal then
        $target = R(s,a,s')$
        Sample new initial state $s'$
    else
        $[target = R(s,a,s') + \gamma \max_{a} Q_k(s',a')]$
    end if
    $\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta E_{P(s'|s,a)}[(Q_s(s,a) - target(s'))^2]|_{\theta-\theta_k}$
    $s \leftarrow s'$
end for
```

There exist several challenges when we compare reinforcement learning to deep learning [6]. Our first challenge deals with encountering non-stationary or unstable targets. As seen in Algorithm 1 above, the target continuously changes (as seen in the $else$ statement part of the pseudocode) with each iteration. However, in deep learning, the target variable does not change and therefore, the training is stable. This is not true for reinforcement learning. Therefore, it is imperative to understand how to map values of a constantly changing input and targets. This challenge can be mitigated with the use of a target network and experience replay [7].

Since the same network calculates the predicted value and target value, there is a possibility for a great amount of divergence to arise between the two values. Instead of using one neural network for learning, two can be used. This separate network can be used to estimate the target. The target network has the same architecture as the function approximator but with frozen parameters. For every a fixed amount of iterations, the parameters from the prediction network are copied to the target network. This leads to more stable training because it keeps the function fixed for a period of time.

In order to perform experience replay, the agent's experiences are stored. Instead of running Q-learning on state and action pairs as they occur during the Lunar Lander simulation, the system keeps a record of data discovered for $[state, action, reward, nextstate]$ in a large table. During training, a batch of 64 records is randomly sampled from the last hundred thousand frames in order to train the network. This provides us with a subset of examples that provide correlation among the samples that are low but provide better sampling efficiency. The experience replay technique is also useful in improving DQN models and adapt reinforcement learning to the scalable machine learning process [12].

B. Reinforcement Learning Parameters

There are several parameters involved in reinforcement learning [8]. In this investigation, four parameters are experimented with:

- ϵ: probability of choosing a random option and not going with the greedy choice
- ϵ-decay: factor by which ϵ decreases and provides a balance between exploration and exploitation
- α: learning rate, which controls how rapidly the model is adapted to the problem
- γ: discount factor that quantifies how much importance is given for future rewards

The value of ϵ can allow us to have a balance between exploration and exploitation. A small value of of ϵ means that the agent chooses to explore, not exploit what has already learned thus far. In the case of exploration, an action is chosen randomly and performed regardless of action-value estimates recorded [1]. In an extreme case, if $\epsilon = 0$, the agent never explores but always exploits the knowledge collected. If $\epsilon = 1$, then the agent always chooses to perform random actions and never relies on accumulated knowledge.

Setting an ϵ-decay value allows the agent to initially take more random actions to explore, and then exploit previous knowledge such that the agent relies on previous knowledge from records kept. If we initially set a high value for ϵ, like 0.9 or 1, it will decrease over time due to this factor [9].

The learning rate, α, is generally set between 0 and 1. Setting $\alpha = 0$ means that Q-values are never updated, and the agent learns nothing from new actions. Setting $\alpha \geq 0.9$ means that learning will occur quickly as the agent may ignore previously collected information and only value the most recent information gained. The discount factor, γ, is also generally set between 0 and 1. If $\gamma = 0$, then the agent will only value immediate rewards. If $\gamma = 1$, then the agent values future rewards just as much as the current reward [8].

C. Model Architecture and Training

A feed-forward network is used as the model. The input consists of the current state which depends on the eight aforementioned dimensions. The outputs of the network are the Q-values for the state-action pairs. The input layer, alongside two hidden layers, and the output layer are shown in Figure 3.

The neural network model for the DQN is shown in Table 1. This model consists of three dense layers and an output layer. There are 14,129 total parameters and 14,129 trainable parameters. DQN approximates the actions using the neural network. The ReLU activation function was used for the hidden layers of this model, while a linear activation function was used in the output layer. Lastly, to train all the models, the model is seeded for consistency.

The import packages that need to be installed in order to visualize the Lunar Lander environment are `gym` either `box2d-py` or `Box2D`. It is important to use Python for the Lunar Lander problem specifically because

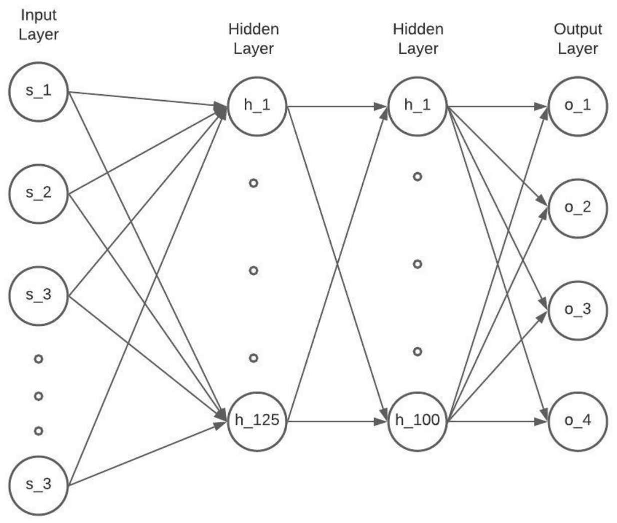

Fig. 3. Neural Network Structure by Layer

gym is only supported by Python. To create the OpenAI Gym environment in order to run, the following command is included in our DQN Python script: `env = gym.make('LunarLander-v2')`. Using gym allows us to avoid building an image interface that would allow us to visualize the reinforcement learning algorithm.

TABLE I
SUMMARY OF NEURAL NETWORK

Layer (type)	Output Shape	Parameter No.
Dense	(None, 125)	1125
Dense	(None, 100)	12600
Dense	(None, 4)	404

D. Parameter Search

There are four parameters that need to be tuned: ϵ, ϵ-decay, α, and γ. After some preliminary trials with running the simulation on different combinations of parameters, the following parameter values were chosen to search from:

- ϵ: $[1.0, 0.5, 0.3, 0.1]$
- ϵ-decay: $[0.999, 0.995, 0.900]$
- α: $[0.1, 0.01, 0.001, 0.0001]$
- γ: $[1.0, 0.5, 0.3, 0.1]$

Initially, there are 192 parameter combinations to search from. The following metric is employed in order to search for the optimal parameter combinations:

$$R(\epsilon, \epsilon - decay, \alpha, \gamma) = \frac{1}{100} \sum_{i=n-100}^{n} R(\epsilon, \epsilon - decay, \alpha, \gamma),$$

where the j^{th} of $R()$ reward function based on the parameters supplied and is the j^{th} reward score calculated. At every step, the reward value is calculated. The average of the reward is taken for the past 100 steps. Once this average exceeds a score of 200, then we will have found a set of optimal parameters. However, if the average continues to become more negative (in particular, a threshold of -700 was set after observing how the agent scored after several runs), training is ceased and the next set of parameters are examined. Another important point in regard to ϵ and ϵ-decay values is that if the minimum ϵ value of 0.01 is reached, the ϵ-decay ceases at that point.

Once these 192 runs have been accomplished, another γ value of 0.99 was introduced as well to see if reducing the γ value just slightly could result in better results. For each combination of parameters, the agent is allowed to run for 400 episodes, unless it manages exceed above a 200 average score or fall below a -700 average score. Once again, it is noted that this threshold of -700 was implemented after running several trials and decided on a lower bound for the reward score average. Some runs, as shown in the results section, converge to a reward mean lower than -700. 400 was chosen for the number of episodes after surveying other investigations which have achieved positive score averages around 350 episodes [10].

III. RESULTS

A. Effects of Parameters

The goal of the Lunar Lander problem is to direct the lander to the landing pad at coordinate (0, 0) between the two flag poles, as shown in Figure 2. The right and left leg of the lander should touch the landing site in a stable manner by maintaining a minimum angular speed, and should not take off again. In the original problem, some uncertainty is introduced via applying randomized forces to the center of the lander at the start of every iteration. This randomized force causes the lander to be pushed in a random direction. The lander must overcome this push and land towards the landing pad [10].

At the start of the iterations, the average reward is negative because the agent is exploring many different actions and keeping record of all information collected from taking these different actions. As the agent learns, the Q-values begin to converge and the agent begins to earn a positive rewards. In Figure 4, the agent is shown to start earning positive rewards more consistently past the 50^{th} step, and is able to converge to an above 200 average reward at the 264^{th} step. The set of parameters that accomplished this best run are $\epsilon = 0.9$, ϵ-decay $= 0.995$, $\alpha = 0.001$, and $\gamma = 0.99$. The training time to reach this convergence took roughly 2.5 hours.

Another set of parameters that have results converge to a positive average score are shown in Figure 5. The set of parameters that accomplished this run are $\epsilon = 1.0$, ϵ-decay $= 0.995$, $\alpha = 0.001$, and $\gamma = 0.99$. It should be noted that just increasing the ϵ value from 0.9 to 1.0 results in a model in which the agent was not able to converge to an average 200 score within 400 episodes. Though the average over the course of the last 100 runs did not exceed 200, it did come very close. The final average score over the last 100 runs was 189.55 at the 400^{th} episode. The training of these 400 episodes took

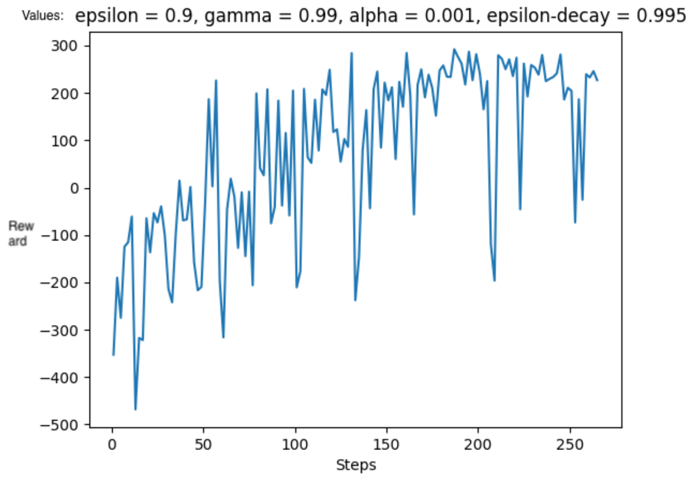

Fig. 4. Plot of reward values for the best set of parameters found

roughly 5 hours. Compared to the results displayed in Figure 4, it seems that using a near-one ϵ value instead of $\epsilon = 1$ is more effective for the Lunar Lander problem, while keeping all other parameter values constant. A possible conclusion to draw from this is that the sooner the agent is able exploit its recorded knowledge, which it should be able to do sooner with a starting $\epsilon = 0.9$ instead of $\epsilon = 1$, the sooner it is able to able to converge to an average reward score of 200. This is an example of the exploration-exploitation trade-off.

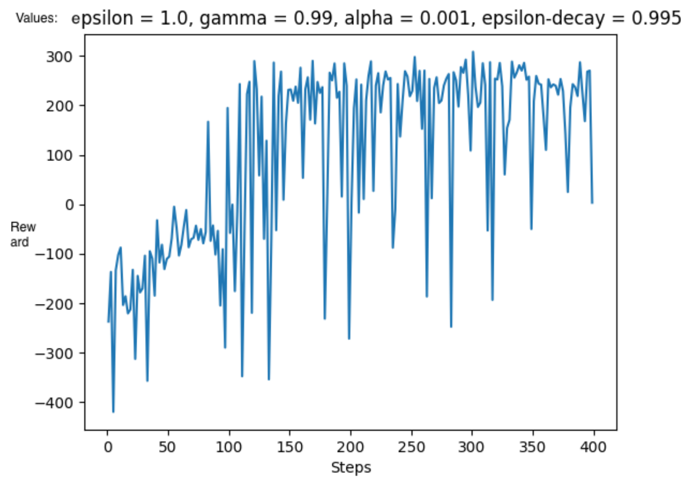

Fig. 5. Plot of reward values for another decent set of parameters

The learning rate of $\alpha = 0.001$ seems to be the most optimal when all the other parameters described above are set to $\epsilon = 0.9$, ϵ-decay $= 0.995$, and $\gamma = 0.99$. However, we examine the performance of the learning rate in other scenarios as well. In Figure 6 which shows the reward values over the course of 400 episodes when $\alpha = 0.001$, these values are generally greater than those in the 400 episodes shown in Figure 7 when $\alpha = 0.01$. The mean of the last 100 reward values when $\alpha = 0.001$ in Figure 6 is -819.53. In contrast, the mean of the last 100 reward values when $\alpha = 0.01$ in Figure 7 is -3194.74. Moreover, in Figure 8, while all other parameters are held constant the α parameter is equal to 0.0001. The mean of the last 100 reward values when $\alpha = 0.0001$ is -356.45. Compared to the model generated in Figure 6, this is a greater mean score. When $\epsilon = 1.0$, ϵ-decay $= 0.999$, and $\gamma = 0.1.0$, setting $\alpha = 0.0001$ seems to produce the greatest mean in the last hundred runs compared to $\alpha = 0.001$ or $\alpha = 0.01$. The lower the value of α is, it does not learn from new actions. It relies of previous accumulated knowledge [1]. Though $\alpha = 0.0001$ produces the greatest reward mean convergence amongst the other α values tested it, it should be noted that the combination of $\epsilon = 1.0$, ϵ-decay $= 0.999$, $\gamma = 0.1.0$, and $\alpha = 0.0001$ is still not as optimal for the Lunar Lander problem as the combination $\epsilon = 0.9$, ϵ-decay $= 0.995$, $\alpha = 0.001$, and $\gamma = 0.99$ shown in Figure 4. It is hypothesized this could be because of the choice of ϵ (using $\epsilon = 0.9$ yields better results than using $\epsilon = 1.0$).

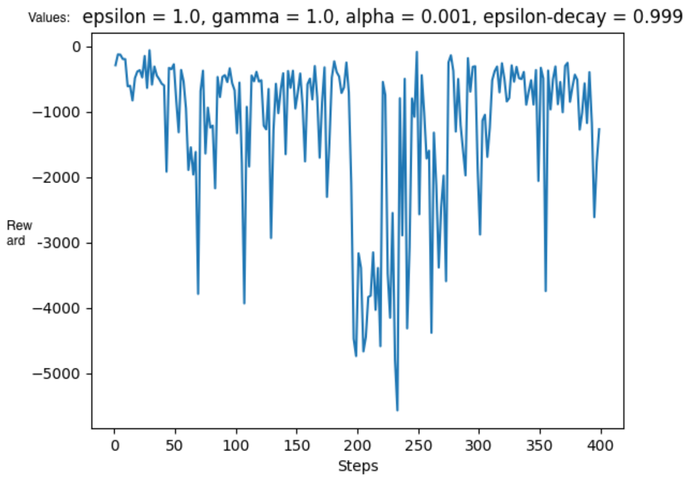

Fig. 6. Plot of reward values for when $\alpha = 0.001$

In general, it seems that using a γ value of either 0.99 or 0.995 seems to be more effective than using a γ value of 0.900. This is evinced when we look at the reward means of the last hundred runs for each model with these parameters. The other three parameters are held constant: $\epsilon = 1.0$, ϵ-decay $= 0.999$, and $\alpha = 0.001$. From Figure 7, we understand the model with $\gamma = 0.999$ is -819.53. Figure 9 displays a model with $\gamma = 0.995$ a reward mean convergence of -750.53. Finally, Figure 10 with $\gamma = 0.900$ has a reward mean convergence of -1111.65. Amongst all these γ values, the model with $\gamma = 0.995$ has the greatest reward mean, and the model with $\gamma = 0.999$ follows second. The closer γ is to 1, the more the algorithm will look for high rewards in the long term [1]. Though $\gamma = 0.995$ produces the greatest reward mean convergences amongst the other γ values employed, it should be noted that the combination of $\epsilon = 1.0$, ϵ-decay $= 0.999$, $\alpha = 0.001$, and $\gamma = 0.995$ is still not as optimal for the Lunar Lander problem as the combination $\epsilon = 0.9$, ϵ-decay $= 0.995$,

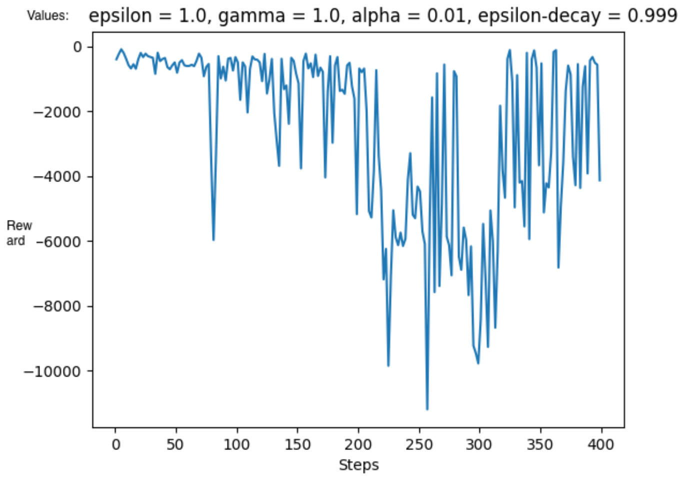

Fig. 7. Plot of reward values for when $\alpha = 0.01$ and $\gamma = 0.999$

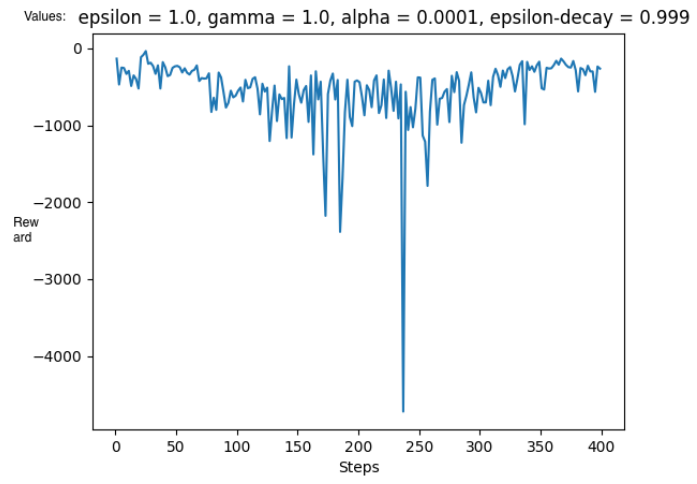

Fig. 8. Plot of reward values for when $\alpha = 0.0001$

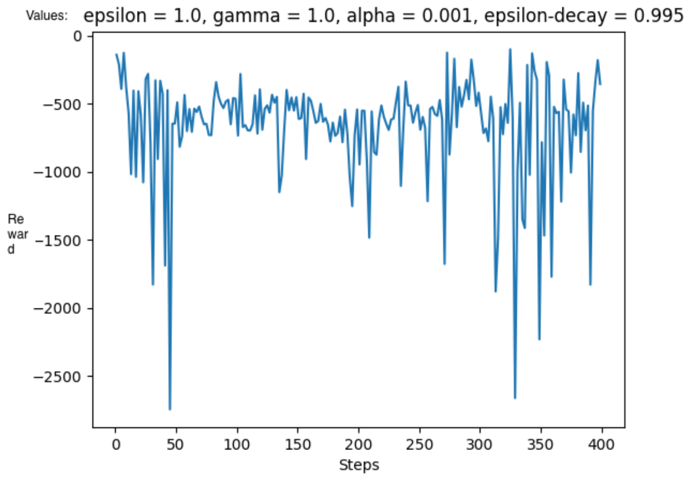

Fig. 9. Plot of reward values for when $\gamma = 0.995$

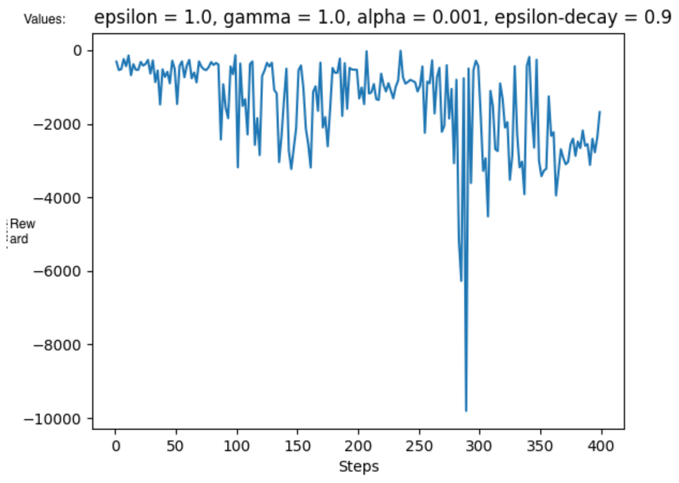

Fig. 10. Plot of reward values for when $\gamma = 0.900$

$\alpha = 0.001$, and $\gamma = 0.99$ shown in Figure 4. It is hypothesized this could be because of the choice of ϵ as well (using $\epsilon = 0.9$ yields better results than using $\epsilon = 1.0$).

B. Comparison to Other Models and Results

It is of interest to compare the results of the DQN created in this investigation to other comparable models that were applied to the Lunar Lander environment and the results obtained from them. A neural network with two hidden layers with 128 neurons each was able to converge to a positive average in roughly 320 iterations. The parameter values used are $\epsilon = 0.5$ and ϵ-decay $= 0.996$ until it reaches ϵ-min of 0.01. In addition, $\alpha = 0.001$ is used; however a γ value is not stated in this study. Moreover, the ReLU activation functions were used on the hidden layers while a linear activation function is used on the output layer. [10]. The model created and trained in this experiment provide a slightly better result, as it able to converge to a mean reward average above 200 in 264 episodes.

In another experiment, it was found that the Lunar Lander problem favors a large hidden layer but not a deeper neural network and a near-one reward discount is necessary for the model to consider a final successful landing. This investigation found that their model could achieve an average score greater than 280 for a trial of 100 landing episodes. The parameters listed included an $\epsilon = 0.5$ to start with and an ϵ-decay $= 0.99$ [11].

IV. Further Directions

Further experiments could include changing the number of neurons in the hidden layers, or employing more hidden layers in the neural network to see if there could a 200 or greater mean reward convergence in fewer episodes than 264 found in this experiment. Moreover, there have been experiments in which long short-term memory (LSTM) architecture was used to learn the internal relations between the agent and the environment. LSTM models have the ability to remember values for both long and short durations [13].

A study done by Erle Robotics used parameters of $\epsilon = 0.9$, $epsilon$-decay $= 0.9986$ $\alpha = 0.2$, and $\gamma = 0.9$ until a minimum value of $\epsilon = 0.05$ is reached [9]. In addition to experimenting with the parameters mentioned in this investigation, ϵ-min could also be another value of interest to experiment with. Perhaps the extent to which ϵ decays may allow for more optimal solutions to the Lunar Lander problem.

While the combination of parameters from this investigation may work for the Lunar Lander environment, this may not be the case for other environments. Therefore, an important extension of this experiment could be to examine how DQN performs in order environments. It may be possible that different sets of parameters could work for different environments. One could use other environments provided by OpenAI Gym to accomplish this. Other methods of reinforcement learning could be applied to the Lunar Lander problem, such as state-action-reward-state-action (SARSA) algorithm.

V. Conclusions

Reinforcement algorithms are a particularly interesting field of machine learning as they do not rely on data to fed into the model in contrast to supervised or unsupervised learning methods. Rather, through exploration and exploitation, this type of learning collects data about possible actions that it could take in an environment that lead to maximal reward. An important part of the exploration-exploitation trade-off are the parameters that determine the agent's ability to use knowledge from its records and what kinds of actions it should take in the iterations of training. While the findings in this experiment are just one possible set of parameters, it is possible there are better options available and therefore optimal parameter combination searches should continue to be explored.

References

[1] Sutton, Richard S., and Andrew G. Barto. *Reinforcement Learning: An Introduction.* Second ed., Cambridge, The MIT Press, 2018.

[2] Christopher J.C.H. Watkins, and Peter Dayan. *Q-learning.* Machine Learning 8, 279-292, 1992.

[3] S. Manju, and M. Punithavalli. *An Analysis of Q-Learning Algorithms with Strategies of Reward Function.* International Journal on Computer Science and Engineering, 279-292, 1992.

[4] Mnih, Volodymyr, et al. *Human-level control through deep reinforcement learning.* vol. 518, Nature, 2015.

[5] Mnih, Volodymyr, et al. *Playing Atari with Deep Reinforcement Learning.* DeepMind Technologies, 2013.

[6] Nguyen, Thanh T., et al. *Deep Reinforcement Learning for Multi-Agent Systems: A Review of Challenges, Solutions and Applications.* arXiv, 2019.

[7] Gao, Juntao, et al. *Adaptive Traffic Signal Control: Deep Reinforcement Learning Algorithm with Experience Replay and Target Network* arXiv, 2017.

[8] Kober, Jens, et al. *Reinforcement Learning to Adjust Robot Movements to New Situations* Robotics: Science and Systems, 2011.

[9] Zamora, Iker, et al. *Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo.* Erle Robotics, 2017.

[10] Gadgil, Soham, et al. *Solving The Lunar Lander Problem under Uncertainty using Reinforcement Learning* IEEE SoutheastCon2020, 2020.

[11] Yu, Xinli *Deep Q-learning on Lunar Lander Game* Temple University, 2019.

[12] Zhao, Dongbin, et al. *Deep Reinforcement Learning with Experience Replay Based on SARSA* Institute of Automation Chinese Academy of Sciences, 2016.

[13] Wang, Pin, et al. *Formulation of Deep Reinforcement Learning Architecture Toward Autonomous Driving for On-Ramp Merge* IEEE 20th International Conference, 2017.

[14] Zhao, Xujiang, et al. *Uncertainty-based Decision Making Using Deep Reinforcement Learning* 2019 22th International Conference on Information Fusion (FUSION), 2019.