Impact of Feedback Modalities on an Agent's State-Action Trajectories: Q-Learning using the OpenAI-Gym Mountain Car Environment

Stephen Wormald

***Abstract*—In Reinforcement Learning (RL), an agent's movement through an environment is related to the expected reward or policy at state-action pairs. The number of state-action pairs increases when an agent has access to more feedback channels. This paper explores how increasing the number of feedback channels influences an agent's ability to reach a goal. The agent's performance is evaluated in terms of number of episodes needed to reach a goal, the mean reward across episodes prior to solving the game, and the evolution of an agent's state-space trajectories as it learns across episodes. The agent is a 2-Dimensional vehicle from the Mountain Car environment distributed through the OpenAI-Gym [3]. Results indicate that there is little to no influence on an agent's state-space trajectory when varying feedback modalities. There is influence of feedback modalities on other performance metrics such as the number of episodes required to train an agent. This finding may depend on the limited environment complexity.**

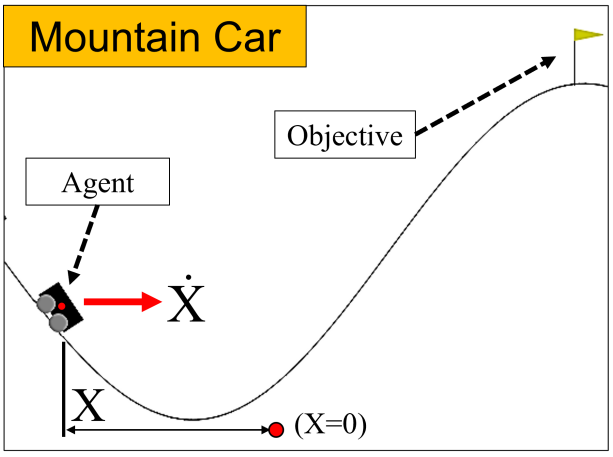

Fig. 1. Depiction of the Mountain Car environment from the OpenAI gym [1], which illustrates the agent and the key state-variables that the environment makes available to the agent.

I. Introduction

THE OpenAI Gym provides a simple interface to easily test RL algorithms [1]. There are multiple 2-Dimensional (2D) and 3D environments with a variety of agent types. This project focuses on the Mountain Car game from the OpenAI Gym where the agent, a 2D vehicle, decides to move left, right, or apply no action in order to meet its goal of reaching an objective flag at the top of a hill. For the baseline feedback modality (FM_0), the agent has access to its location (X) and velocity (X_{dot}) along the horizontal axis (Figure 1). The number of feedback channels can be increased or decreased to study the impact on how agents traverse the environment and make decisions in the state space. The project objective is to quantify how access to different feedback modalities, or sets of feedback channels, influences the agent's ability to achieve its goal.

The following sections describe the results from implementing the ϵ-greedy RL algorithm to solve the objective of the Mountain Car environment. Section II overviews the RL formulation through Q-Learning and expands upon the problem statement. Section III describes the results from exploring how learning parameters impact the agent's ability to reach a goal, and how varying the feedback modalities influences what path the agent takes to reach the goal. Section IV overviews several other projects and modern topics related to the RL paradigm before summarizing conclusions in Section V.

II. Description

This section describes the RL problem formulation through Q-learning (Section (II-A)) and reviews the research question in more detail (Section (II-B)).

A. Reinforcement Learning Formulation

RL often considers how an agent makes actions in an environment to maximize expected rewards. An environment comprises a set of states, where agents transition between states either probabilistically or by performing actions. Each action may be associated with positive or negative rewards. For example, when the agent in the Mountain Car environment moves past the flag, it receives a reward. The action bringing the agent to this state would be reinforced using a policy update so that the agent is more likely to take this action in the future. Action not directly associated with the goal can still receive value if they are connected to other high-values states as determined by the ϵ-greedy RL algorithm.

When an agent is placed into a new environment, it may not know the value associated with given actions. This is called model free reinforcement learning. The agent can move and observe when rewards are given, and use these rewards to update the expected value of the associated state. The process

of updated expected the reward is starts with the Bellman Optimality Equations [4].

$$q_*(S, A) = E_\pi[R_{t+1} + \gamma * max_{A'}(q_*(S', A'))] \quad (1)$$

In Equation 1, $q_*(S, A)$ is the action value function, or the expectation of future reward given some action (A) and state (S). The expectation of reward is calculated given the optimal policy ($*$). The term $max_{A'}(q_*(S', A'))$ is the maximum reward of the next state (S') and action (A') scaled by a discount factor γ, which can help prioritize immediate rewards. Note the q-values can be saved in memory as a table or list for containing the value of each state-action. This data structure is known as a q-table and deals with a finite number of state-action pairs.

The optimal action is often calculated as the action that will give the greatest reward. This approach is known as acting greedily with respect to the policy, and helps the agent maximize future reward. Practically, the value of each action that can be taken from a current state is obtained from the q-table, and the action with the maximum q-value is selected. Note that acting based upon the expected reward does not mean a reward will be received immediately, especially when operating with a policy that is inaccurate. We may encounter unexpected rewards or penalties which may be used to update the q-value in the q-table according to the equation:

$$q_{new}(S, A) = (1 - \alpha)q(S, A) + ... \\ ... + \alpha(R_{t+1} + \gamma * max_{A'}(q(S', A')) \quad (2)$$

Where $q_{new}(S, A)$ is the updated policy of some state-action pair, $q(S, A)$ is the existing q-value, α is the learning rate, and R_{t+1} is the observed reward. This formulation allows the reward of the immediate to be updated, but the reward propagates to upstream states due to the second term. This approach allows existing values in the q-table to be modifies slightly rather than being completely replaced.

Note that agents that act greedily with respect to a policy will always take actions known to give rewards, and may miss taking actions where a potential reward is unknown. One solution is the ϵ-greedy algorithm, which allows an agent to take random actions with some probability ϵ. In other words, ϵ is a probability threshold for determining when to act randomly. The parameter ϵ can be varied over time using:

$$\epsilon_{j+1} = \epsilon_j * \epsilon_{Decay} \quad (3)$$

Where ϵ_{Decay} is the decay rate of ϵ with respect to the number of episodes in a game. Note this formulation converges to zero given enough iterations, though other formulations may converge to some minimum ϵ threshold.

These principles used to program software that operates according to the pseudo code in Algorithm 1. Note I followed a video tutorial to implement this underlying algorithm [2]. Additional software was programmed during this project to streamline the experimentation process. This software is publicly available on Github [5].

Algorithm 1: Pseudo-code for the ϵ-Greedy Algorithm

```
Episode = 0;
Set initial state;
Done = False;
while Done == False AND Episode ≤ maxEpisode do
    if RANDOM ≤ϵ then
        Get random action;
    else
        Determine action of maximal expected reward;
    end
    Update current state given the selected action;
    Get reward of updated state;
    Update policy value in Q-table;
    if Stopping criteria met then
        Done = True;
    else
        ϵ = ϵ*ϵ_Decay;
        Episode += 1;
    end
end
```

B. Problem Description: OpenAI's Mountain Car

The ϵ-Greedy algorithm was used during the full project when studying how an agent's performance varied when altering the learning parameters and feedback modalities. The Mountain Car environment typically allows the agent to perform three actions based upon a state-space composed of two state variables. The actions are move left, move right, and do nothing. The state variables are the agent's position along the horizontal axis (X0) and the velocity along this axis ($X0_{dot}$). Both of these variables are continuous but bounded. Discrete states were determined by dividing the parameter range into a number of bins.

Because the agent starts at the bottom of the valley depicted in Fig. 1, the RL algorithm needs to determine how to move the car back and forth to build momentum. However, there is a time limit forcing the agent to perform quickly. When the game is running, the agent receives a '-1' reward any time it is not at the goal, and a receives a reward of '0' after reaching the goal.

Three experiments were performed. Experiment 1 varied the learning parameters according to Table I to study the impact on a set of performance metrics. These metrics include the number of episodes until solving the game ten times in a row, and the minimum, average, and maximum performance over two epochs. Experiment 2 looked closer at the relationship between ϵ_{Decay} and the agent's ability to solve the game in few episodes.

Experiment 3 varied the agent's feedback modalities to understand the impact on the how the agent performed actions sequentially through the state space (the state-space trajectory). Four feedback channels were added to the Mountain Can environment for Experiment 3. These variables include the position and velocity from the previous state (X1 and $X1_{dot}$, respectively) and the difference between the current and prior

state variables ($\Delta X = X0 - X1$ and $\Delta X_{dot} = X0_{dot} - X1_{dot}$). Five feedback modalities were designed using from these feedback channels as summarized in Table II.

The results from each of these experiments are discussed in Section III.

TABLE I
Learning parameters designed for Experiment 1 to understand the impact on the agent's performance through all episodes. Each combination of these learning parameters were used as a separate experiment. The FM_0 was used during Experiment 1.

Bins	α	γ	ϵ
20, 30	0.1, 0.2	0.85, 0.95	0.25, 0.35

TABLE II
Feedback modalities designed to evaluate cases beside the baseline modality (FM_0) where the agent has added feedback (FM_1 - FM_3) and inhibited feedback (FM_4).

Vars.	FM_0	FM_1	FM_2	FM_3	FM_4
$X0$	✓	✓	✓	✓	✓
$X0_{dot}$	✓	✓	✓	✓	✗
$X1$	✗	✓	✗	✓	✗
$X1_{dot}$	✗	✓	✗	✓	✗
ΔX	✗	✗	✓	✓	✗
ΔX_{dot}	✗	✗	✓	✓	✗

III. Evaluation

This section describes the results and evaluation of the three experiments described in Section II. Section III-A describes results from testing the q-learning software. Sections III-B, III-C, and III-D describe the results of Experiments 1, 2, and 3, respectively.

A. Results from Building Q-Learning Software

Results from a simple test case are depicted in Fig. 2 which shows how the Q-table changes over time. The evolution of the agent's state-trajectory is shown in Fig. 3. This test was run using the following learning parameters: Bins = 10; α = 0.1; γ = 0.95; ϵ = 0.95; ϵ-decay = 0.99. Fig. 2 shows three rows of plots where each row corresponds to episodes 400, 800, and 1200. Each row has four images coloring the 2D state-space where X0 is on the x axis, and $X0_{dot}$ is on the y axis. the first three plot columns correspond to different actions. From left to right, these actions represent the agent moving left, doing nothing, and moving right. The fourth column colors the state space with the action having the maximal q-value. During a given episode, the agent would select these actions when acting greedily.

Fig. 2 shows that the center of the state spaces has a the most negative q-values. This state region represents when the car is in the center of the state (X0=0) and is not moving ($X0_{dot}$ = 0). The surrounding state space shows larger q-values because the state-space was initialized with a uniform random policy between zero and two. Because the agent starts in the middle and does not receive a reward until reaching the goal, every action it takes will receive a penalty. Consequently, unexplored regions will have a higher reward and the agent will give preference to these states until reaching the goal. The emergent agent behavior is exploratory. Note that after 800-1200 episodes, general regions start appearing in the maximum-action column. When X0 is greater than 0, the action with maximum reward is moving right as this action allows the agent to reach the goal. When X0 is less than 0, the action with maximum reward is corresponds to moving left as this action allows the agent to build momentum before going up the hill.

The agent's state was saved for all time-steps and episodes in order to create Fig. 3. This figure shows the agent's sequential actions through the state space. The agent starts at the black dots in the center and moves along a given trajectory before the episode terminates at a red dot. The spiral pattern corresponds to the agent moving left and right across the environment, accelerating in the positive then negative directions. Cases where the red dot crosses the red line represents the agent passing the goal. Note how the the agent did not meet the objective for the first third of the episodes, starts reaching the goal in the second third, and then regularly meets the objective during the last third. This transition represents the agent learning the correct policy per state-action pair. Furthermore, the last third of the episodes show that the agent's performance is more consistent. This trait occurs because the agent is learning more of the optimal policy, and because ϵ will be small during this episode (see Section III-C). These results are typical for each case in Experiments 1 through 3, indicating indicate that the Q-learning algorithm was implemented as expected.

B. Experiment 1: Variation of Learning Parameters

Experiment 1 varied the learning parameters to understand the impact on the agent's performance, as summarized in Fig. 4. This figure shows that it generally takes fewer episodes to solve the game when using a bin size of 10, an α of 0.2, a γ of 0.95, and an ϵ_{Decay} of 0.35. A smaller bin size likely reduces the number of episodes needed because the dimensionality is reduced. In other words, the number of q-values that need to be updated is smaller, which is significant because the dimensionality increases as the square of the bin size. In other words, more time can be spent learning the correct policy rather than visiting new states.

The maximum and mean reward across all metrics is generally increased for the learning parameters above. Consequently, these parameter values were used during Experiments 2 and 3.

C. Experiment 2: Impact of ϵ_{Decay}

Further analysis after Experiment 1 revealed that the range of ϵ_{Decay} was not large enough during the first trial. Fig 7 shows how ϵ changes with the number of episodes for different values of ϵ_{Decay}. The largest value of ϵ_{Decay} testing in Experiment 1 was 0.35, which results in an ϵ curve that is nearly zero for the duration of each test. The three values of ϵ_{Decay} in Fig 7 are examples of parameters that allow for a meaningful difference in agent exploration across a the duration of an experiment with 1200 episodes.

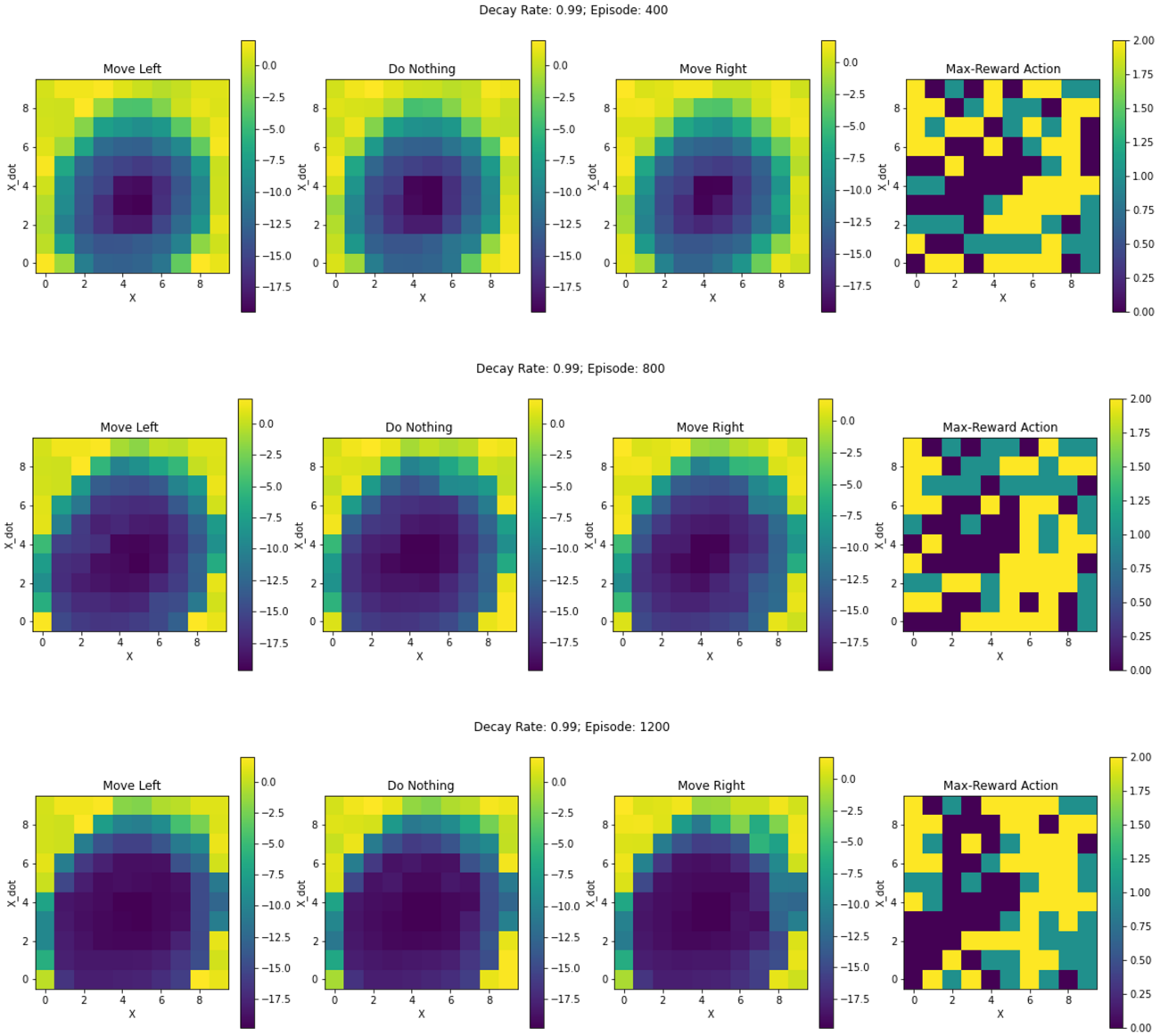

Fig. 2. Evolution of policy function (or Q-table) across 1200 episodes for three actions: Move left (0), Do nothing (1), and move right (2). The action with the maximum q-value for a given state (a pair of X0 and $X0_{dot}$ values) is selected as the action with maximum-reward. Learning parameters: α = 0.1; γ = 0.95; starting ϵ = 0.95; ϵ-decay = 0.99.

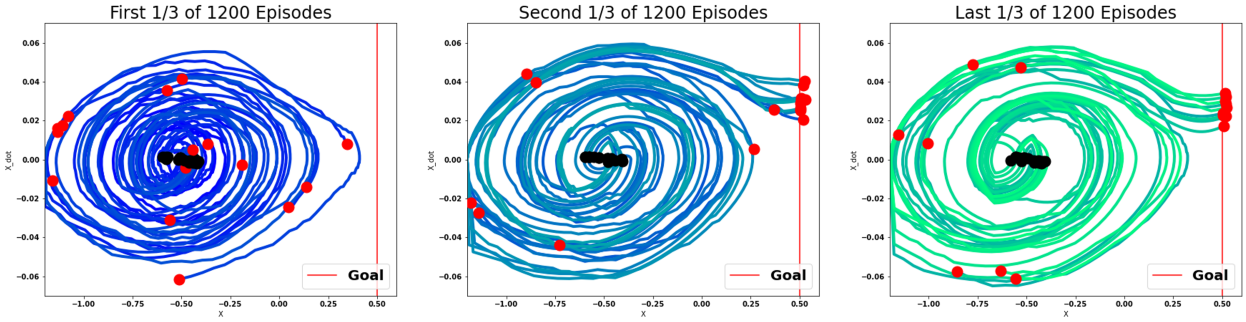

Fig. 3. State trajectories of the car, skipping every 25 episodes. Plots shows how the agent initially fails to reach the goal for the first third of 1200 episodes. The agent begins to reach the goal during the second third of the episodes, but only reaches the goal consistently in the last third of the episodes. Learning parameters: α = 0.1; γ = 0.95; ϵ = 0.95; ϵ-decay = 0.99.

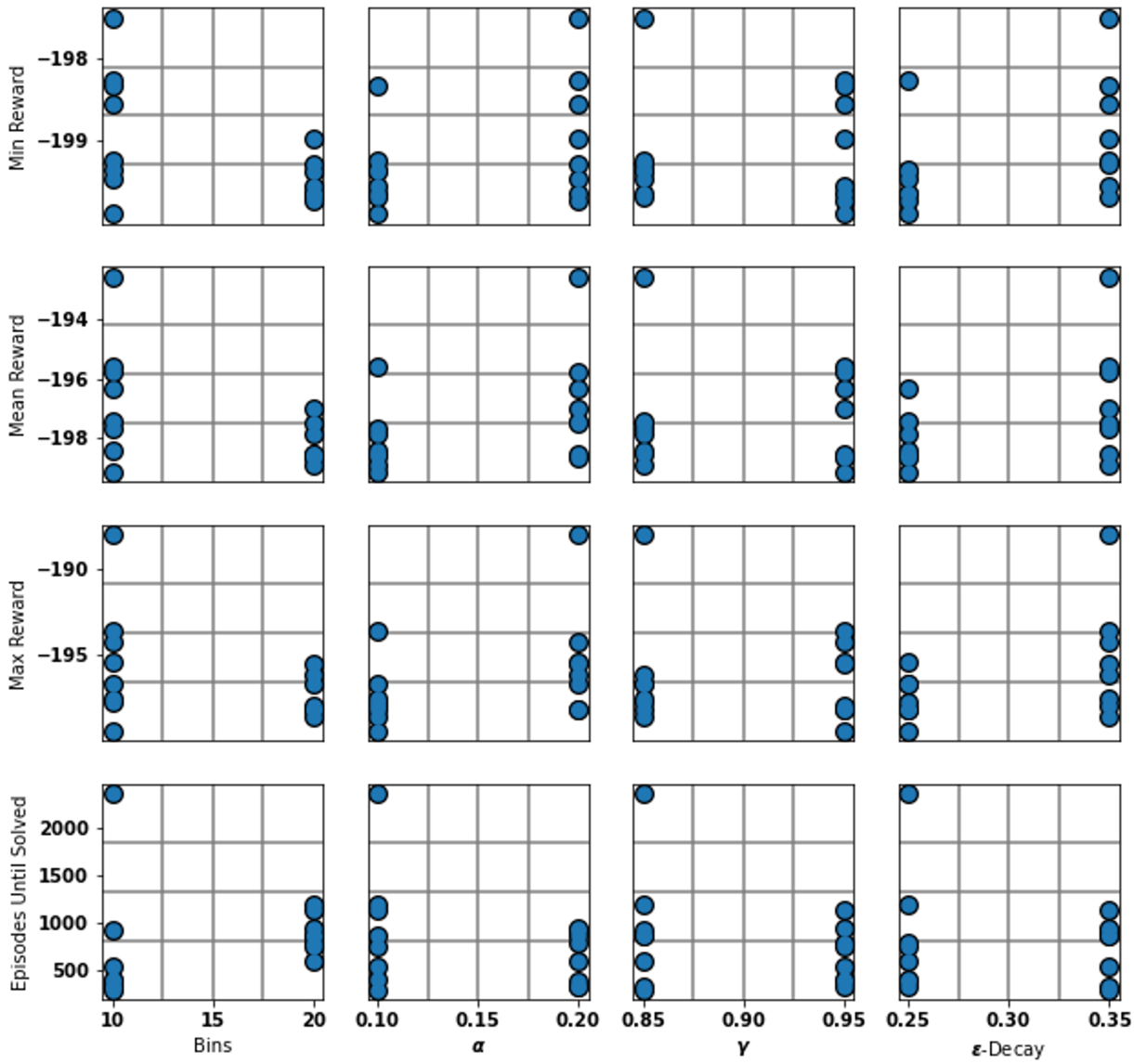

Fig. 4. Results from Experiment 1. The starting ϵ was 0.95. Ten consecutive wins were needed to characterize an agent as having stably solved the game. Generally, a smaller bin size, larger α size, larger γ size, and larger ϵ_{decay} size improved agent's performance.

Fig. 6 shows the results from training five more agents for additional values of ϵ_{Decay}. The mean reward is shown as a function of the episodes or epochs. Note how each agent can learn for many episodes before receiving a reward, at which point many spikes appear at once which represents repeated success. The colored markers shows where each agent successfully solved the game for ten consecutive runs. Larger values of ϵ_{Decay} tended to require more episodes to solve the game (Fig. 7) likely because increasing ϵ_{Decay} allows for more exploration.

While this experiment was performed because ϵ_{Decay} was expected to be too small, it showed that ϵ_{Decay} = 0.35 does result in the least number of episodes required to converge to a winning policy. This value was used in Experiment 3 as a result. However, the larger ϵ_{Decay} values may be associated with other performance metrics, such as stability and policy robustness, that are unexplored in this paper.

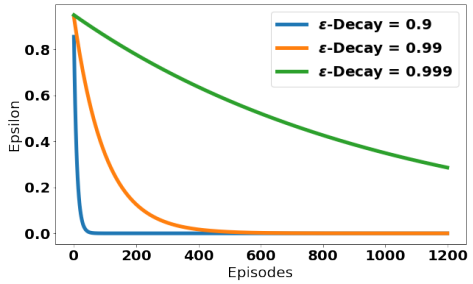

Fig. 5. Relationship between ϵ and the number of episodes as ϵ_{Decay} varies. Starting ϵ = 0.95.

D. Experiment 3: Impact of Feedback Modalities

Experiment 3 was designed to study the influence of different feedback modalities on an agent's performance. The

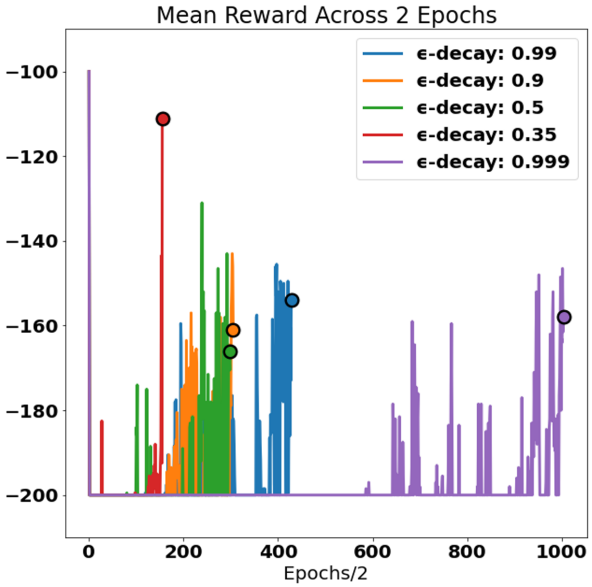

Fig. 6. Relationship between of ϵ_{Decay} and number of episodes as ϵ_{Decay} varies. Learning parameters: α = 0.1; γ = 0.95; Starting ϵ = 0.95. The colored circles indicate the episode where an agent consecutively solved the game ten episodes in a row.

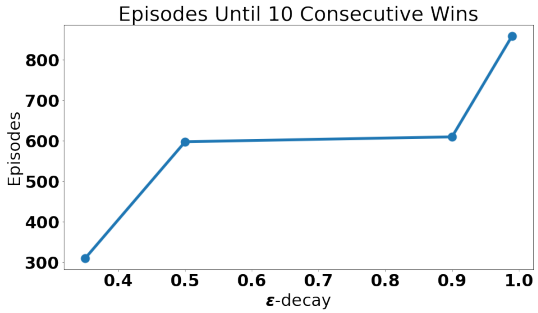

Fig. 7. Relationship between ϵ_{Decay} and number of episodes as ϵ_{Decay} varies. Learning parameters: α = 0.1; γ = 0.95; Starting ϵ = 0.95 The points circles indicate the episode where an agent consecutively solved the game ten episodes in a row. The case where ϵ_{Decay} = 0.999 is not plotted to aid visualization.

performance metrics for comparing each modality are the qualities of the state trajectories and number of episodes required to solve ten consecutive games.

Fig. 8 shows each Feedback Mode (F-Mode or FM) from Fig. II and compares them in terms of the mean reward and total number of episodes. The baseline feedback mode (FM_0) required the smallest number of episodes to solve the game (Fig. 9). While the feedback modalities with more channels (FM_1 through FM_3) added information that could be useful to the agent, this information increased dimensionality significantly. However, while dimensionality increases by about $Bins^D$, where D is the dimensionality of the state space, the number of episodes required to solve the game only increased by about $N * D$, where $1.5 \leq N \leq 8$. This observation is crude, but if true it may indicate that some features in the added channels were useful. However, as D increases, the amount of unused state space also increases, so this prospect is inconclusive.

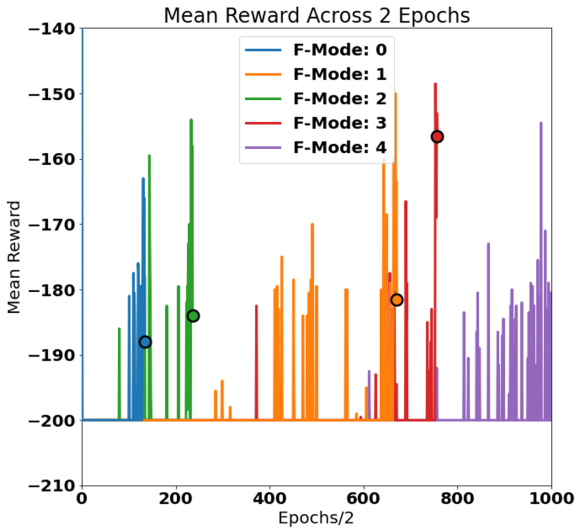

Fig. 8. Learning parameters: α = 0.1; γ = 0.95; ϵ = 0.95; ϵ-decay = 0.35. The colored circles indicate the episode where an agent consecutively solved the game ten episodes in a row.

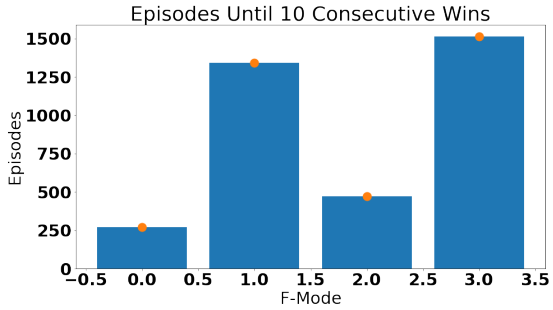

Fig. 9. Relationship between ϵ-decay and number of episodes as ϵ-decay varies. Learning parameters: α = 0.1; γ = 0.95; ϵ = 0.95; ϵ-decay = 0.35.

Note that reducing D also increased the required number of episodes. For example, FM_4 reduced D by excluding $X0_{dot}$, but failed to solve ten consecutive runs in the first 10,000 episodes. This run was marked as unsuccessful. However, this modality produced a maximum number of consecutive wins of 7, making it seems possible for this modality to find an optimal policy. This case illustrates that some information, such as $X0_{dot}$, is worth increasing dimensionality.

Fig. 10 compares the state-trajectories of FM_0 through FM_3. FM_4 was excluded as it failed to converge. While the number of episodes to solve the problem changes per agent, the learned state-trajectories remain largely the same. The similarity likely results from the simplicity of the game, as there is one primary path through the state space that results in a win condition. Varying the feedback modality for games with larger solution spaces may results in more interesting variability in an agent's state trajectories.

The results from this experiment illustrate a trade-off between the benefit of added information yet with the detriment of increased dimensionality. The next steps for this work are twofold: (1) The approach in this paper could be applied to more complex environments with larger solution spaces, and

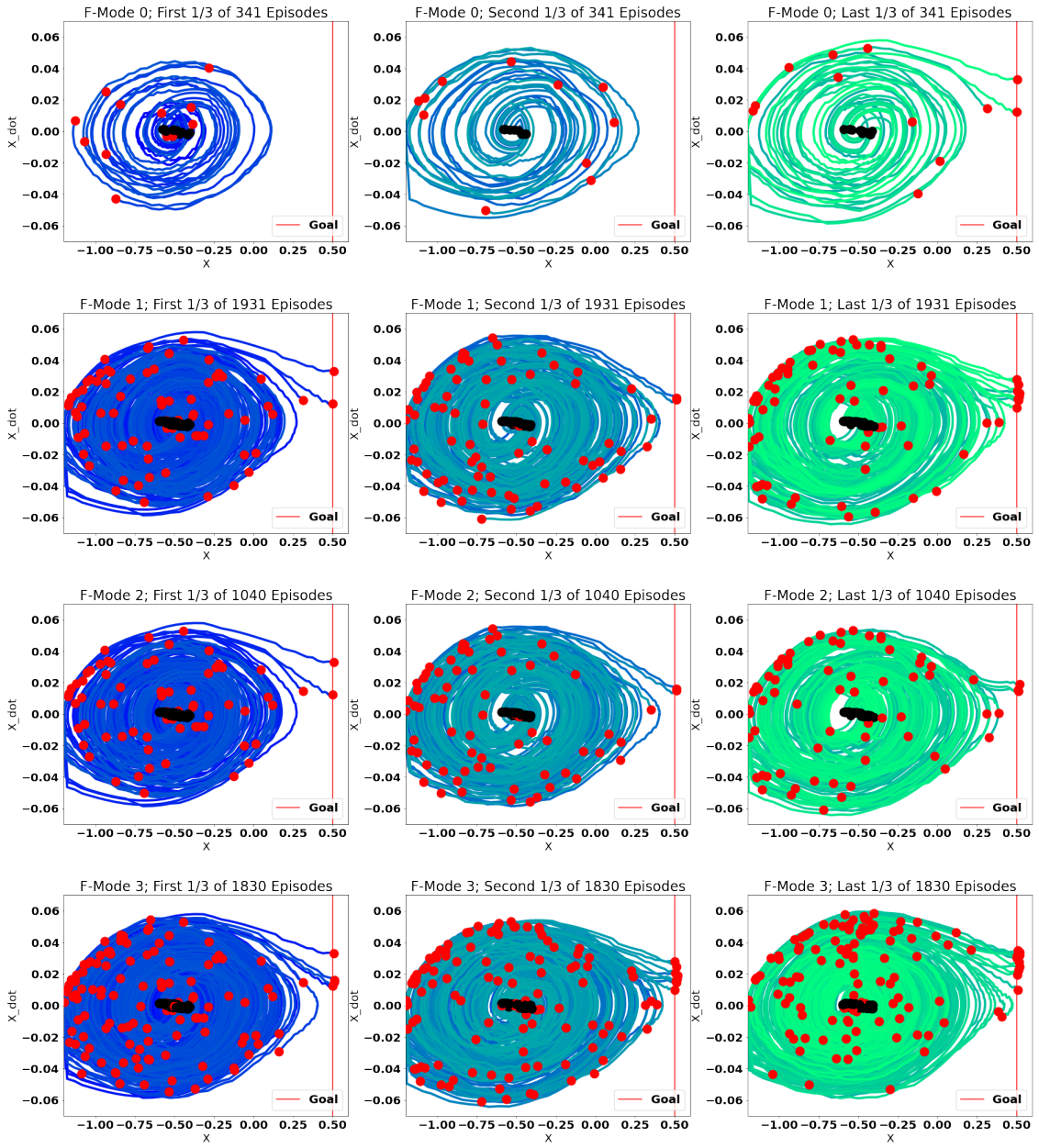

Fig. 10. State trajectories of the car, plotting every fifth episode. These plots show how different feedback modalities change the episode count needed to reach the goal consecutively across ten runs. The goal is depicted as the red line on the right of each subplot. The black points show where the agent started, and the red points depict points where the agent stopped at the end of an episode. Comparing the stopping points and frequency of reaching the goal helps evaluate each feedback modality. Learning parameters: α = 0.1; γ = 0.95; starting ϵ = 0.95; ϵ-decay = 0.35.

for this case (2) dimensionality reduction could be used to simplify the state space. Some combination of increasing feedback dimensionality while applying dimensionality reduction may have potential to improve the model performance and resulting learning rate.

IV. Related Work

Reinforcement learning has been researched for many years, so there are many related works that improve upon the methods used in this project. For example, this work encountered the curse of dimensionality both in the required memory usage and longer training time. This problem comes largely from the need to discretize continuous state spaces into smaller regions. Several solutions to solve this problem include deep Q-learning and RL for continuous state and action spaces.

Deep q-learning replaces the need for a q-table by using deep neural networks to map an input state space to an output action space. This approach often trains the deep neural network in an iterative method as the policy function can change between iterations, thereby changing the predicted class labels [6]. Deep Q-Learning has applications in and beyond the fields of robotics, game theory, natural language processing [7].

Representation of continuous action spaces has been ac-

complished using value-gradient based greedy policies and continuous actor-critic methods [8]. The author Doya mentions that continuous representation of state-spaces with respect to time can improve the continuity and smoothness of control, and says there are benefits in efficiency. An added benefit is removing the need to partition a continuous spaces into discrete representations [8].

Recent advances in the field of RL include evolving RL algorithms and the pursuit of more interpretable models [9]. Applications are growing in partially observable Markov decision processes [10]. Other developments center around inverse and multi-agent RL [11].

V. Summary and Conclusions

The conclusions from work include:

1) Increasing the dimensionality of feedback modalities may give agents helpful information, but at the expense of training time and memory requirements. This trade-off needs to be balanced.
2) It is important that learning parameters such as ϵ_{Decay} be tuned to the given environment and feedback modality these parameters influence the number of episodes required to train an agent reliably.
3) Different feedback modalities can result in the same state-trajectories for an agent. This observation may result from the limited solution space present in the Mountain Car environment. More complex games and environments would serve as a better platform for understanding whether changing feedback modalities significantly influences learned trajectories through a higher-dimensional state spaces.

References

[1] OpenAI, A toolkit for developing and comparing reinforcement learning algorithms. Gym. Available at: https://gym.openai.com/ [Accessed March 18, 2022].
[2] Anon, RL course by David Silver - Lecture 1 ... - youtube. Available at: https://www.youtube.com/watch?v=2pWv7GOvuf0 [Accessed March 18, 2022].
[3] OpenAI, A toolkit for developing and comparing reinforcement learning algorithms. Gym. Available at: https://gym.openai.com/envs/MountainCar-v0/ [Accessed March 18, 2022].
[4] D. Hardik , "Bellman optimality equation in reinforcement learning," Analytics Vidhya, 15-Feb-2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/02/understanding-the-bellman-optimality-equation-in-reinforcement-learning/. [Accessed: 27-Apr-2022].
[5] S. E. Wormald, "qLearning-mountainCar," Github. 26-Apr-2022.
[6] sentdex, "Deep Q Learning w/ DQN - Reinforcement Learning p.5," in Deep Q Learning w/ DQN - Reinforcement Learning.
[7] "The applications of Deep Reinforcement Learning," GetSmarter Blog, 01-Mar-2021. [Online]. Available: https://www.getsmarter.com/blog/market-trends/the-applications-of-deep-reinforcement-learning/. [Accessed: 26-Apr-2022].
[8] K. Doya, "Reinforcement learning in continuous time and space," Neural Computation, vol. 12, no. 1, pp. 219–245, 2000.
[9] X. Xiang, S. Foo, and H. Zang, "Recent advances in deep reinforcement learning applications for solving partially observable Markov decision processes (POMDP) problems part 2—applications in transportation, industries, communications and networking and more topics," Machine Learning and Knowledge Extraction, vol. 3, no. 4, pp. 863–878, 2021.
[10] "Evolving reinforcement learning algorithms," Google AI Blog, 22-Apr-2021. [Online]. Available: https://ai.googleblog.com/2021/04/evolving-reinforcement-learning.html. [Accessed: 26-Apr-2022].
[11] S. Adams, T. Cody, and P. A. Beling, "A survey of inverse reinforcement learning," Artificial Intelligence Review, 2022.