CSE 546 Assignment 1

Mohammed Nasheed Yasin
Department of Linguistics
University at Buffalo, SUNY
Buffalo, NY 14260
m44@buffalo.edu

Abstract

This report explains our[1] experiments with environments in RL. It focuses on the differences between stochastic and deterministic environments, and introduces commonly used rendering methodologies. We also go on to solve these environments with popular RL tabular methods: Q and Double Q Learning.

1 The Environments

There are certain features that are common to both the stocastic and deterministic environments.

Environmental Elements

Figure 1: Environmental elements (from left to right; top to bottom) Agent, Goal, Negative Reward, Reward

[1]Although this work has been written in first person plural (we/ours) this is the work of only the listed author. Such a convention has been adopted to keep in line with paper writing tradition.

The enivironment is a `6x6` grid defined according to the Gym [2] API. With `36` possible positions that the agent can occupy. The goal is to reach the oasis (shown in Figure 1) **after consuming all** the juice (positive reward) within a (configurable) maximum number of time steps. If the agent lands on a juice tile it is awarded +0.99 and the cactus (negative reward) leads to a -1.0 reward. Once all the juice is consumed, the agent must proceed to the oasis to earn a reward of +1.0 The states in this enviroment are a **combination** of the agent's position and the currently available rewards (positive, negative and goal) on the grid. The Formula 1 gives us the number of possible states for the agent.

$$num_{states} = num_{pos} \sum_{k=0}^{c_{reward}} \binom{c_{reward}}{k} \tag{1}$$

Here num_{pos} refers to the number of grid squares, `36` in our case. c_{reward} refers to the number of positive rewards + the number of negative rewards + 1 (for the goal state). We have `6` negative rewards, `3` positive rewards and one goal. Hence, the num_{states} for us is `36864`.

In each position our agent can take 4 potential actions: 1. Left 2. Right 3. Up 4. Down. Resetting the environment will not change the location or distribution of the rewards and goal state. It only alters the initial state of the agent.

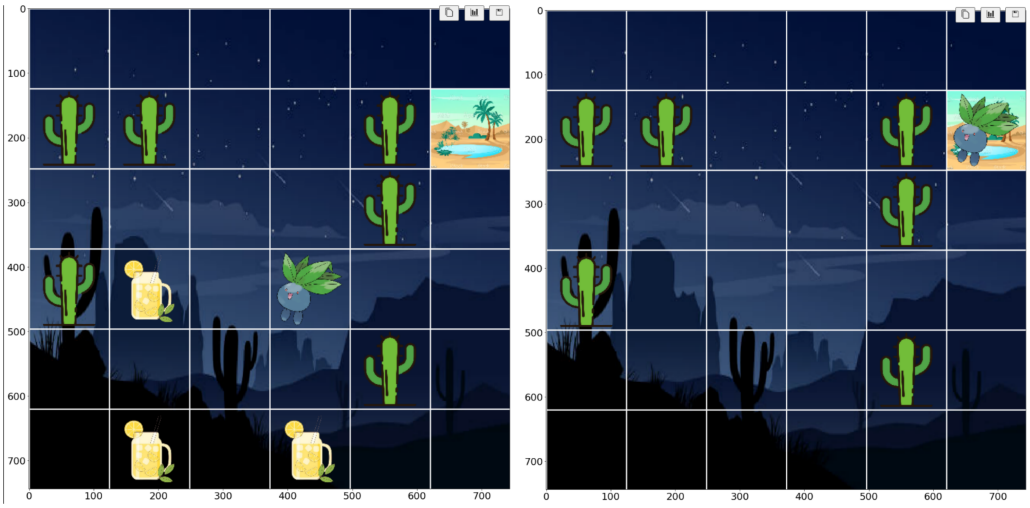


Figure 2: Environment visualizations

1.1 Stocasticity

An enviroment is called stocastic when the result of an action (i.e. its success or its reward) is not guaranteed. Formally, given the same state s_0 and action a the next state s_1 and reward r will be probability distributions.

In our `GridEnviroment` the stocasticity can be described as follows:

- For every action there is a $2/3^{rd}$ chance that the action is performed as expected (`as-is`), a $2/9^{th}$ chance that it is the `double` of what is expected and a $1/9^{th}$ chance of it being a `mirror` action.
- A bonus is distributed in a reciprocal proportions. i.e. `0.1x` the `max_reward` for coming through the `as-is` action, `0.2x` `max_reward` for coming via the `double` action and `0.3x` `max_reward` for the `mirror` action.
- For instance on a 6x6 grid where the reward on each square is 1 (for simplicity):
  If s_0 = `(2,3)` and a = `RIGHT`:
  - $2/3^{rd}$ chance s_1 = `(2,4)` and r =`1+0.1`

- $2/9^{th}$ chance s_1 = `(2,5)` and r = `1+0.2`
- $1/9^{th}$ chance s_1 = `(3,2)` and r = `1+0.3`

1.2 Safety in AI

The following are properties of the environment that ensure valid behavior from the agent:

1. We ensure that the agent consumes *all* the `pos_reward` and reaches the `goal_state` in the fewest number of steps by imposing a penalty of `0.1x` the `max_reward` for every move made.
2. The reward on all squares is consumed once the agent lands in that state, preventing the agent from settling down in a *high-reward* neighborhood.
3. The result of any action (left, right, top, down) are clipped to the min and max values of 0 and `GridSize` (6 in our case) respectively, ensuring that the agent never leaves the environment.
4. If the agent makes a move but remains in the same spot, we impose the *maximum negative reward* (-1.0 in our case). This allows the agent to disincentivize making fruitless moves.
5. We also prevent a *goal rush* (before the agent consumes all the positive rewards) by making the goal square unreachable when there are positive reward squares left. If the agent takes an action to move into a goal square before collecing all the *positive rewards*, they will be kept on the same square, incurring an additional penalty (-1.0 in our case) as detailed in the previous point.
6. The stocasticity of the starting point and limited time steps will nudge the agent to build strategies that accumulate the maximal reward in the shortest time.

2 Tabular Methods

We applied the Q Learning and Double Q learning tabular methods to solve the environment defined in section 1. Both these techniques fall under the Temporal Difference (TD) *model-free* category and we estimate the Q value of each state action pair by considering as target the sum of the reward and the Q value of the next state action pair. Since we only consider the immediately following state's Q value and reward, we call it a TD(0) approach.

$$Q(S,A)_Q = Q(S,A)_Q + \alpha[R + \gamma max_a Q(S',a)_Q - Q(S,A)_Q] \tag{2}$$

Equation 2 is the update function for Q Learning where the α refers to the *learning rate* and the γ represents the *discount factor*. The target value of each state-action pair is represented by $R + \gamma max_a Q(S',a)_Q$. Here R is the reward recieved by taking action A at state S.

Double Q Learning is an extention to Vanilla Q Learning which uses two Q tables instead of one. It targets the over-estimation issue that afflicts Q Learning as it is unlikely that both the Q tables over-estimate the same set of state-action pairs. Such a set-up therefore allows faster convergence of the learning algorithm. Essentially, at each time step we choose the action according to a *mean Q table* (in our case), collect the reward and *randomly* choose to update one of the two Q tables.

For instance if we choose to update table A. Then we choose the greedy action for the state S' according to table A. But the value for this action is taken from table B. Equation 3 formally states the update mechanism. The symbols mean the same as they did in Q learning update (Equation 2).

$$Q(S,A)^A_{DbQ} = Q(S,A)^A_{DbQ} + \alpha[R + \gamma Q(S', argmax_a Q(S',a)^A_{DbQ})^B_{DbQ} - Q(S,A)^A_{DbQ}] \tag{3}$$

Q Learning and Double Q Learning are defined as off-policy methods since the target is estimated using greedy policy regardless of the current selection policy (in our case epsilon-greedy) (refer Equations 2 and 3).

2.1 Reward Function

Framing the reward function correctly is one of the most crucial aspects of environment creation. A good reward function will postively reward ideal behaviour and negatively reward the undesireable actions.

To better promote the objective of avoiding negative rewards (Cacti), we originally designed the episode to terminate immediately upon consumption of a cactus with a negative reward of -1.0. However, this approach had an unintended consequence of incentivizing the agent to rush towards the closest cactus as we also imposed a penalty for every action taken. This penalty was intended to create a sense of urgency in the agent towards achieving the ultimate goal. To address this issue, we have decided to remove the termination of episodes when a cactus is consumed.

In our set up we have the agent start from random positions in the grid every episode. We noticed that whenever the agent spawned close to the goal, it rushed towards it, ignoring the other positive rewards on the map. Hence, in order to compel the agent to collect all the reward before proceeding towards the goal, we **block** the agent from moving into the goal grid square **before** consuming all the positive rewards on the board. This meant that the agent would continue to remain in the same state and incur a penalty of -1.0 should it choose an action that takes it to the goal when there are other positive rewards on the board.

This ensured that the agent found the **quickest way** to collect all the rewards and only then proceeded to take the **shortest** path to the goal state.

Another behavior of note occured in our stocastic environment. Based on the way our reward function was initially set up, if the agent reached a state using a less probable route (refer section 1.1), the agent began to prioritze reaching positions via the less probable action (higher reward), such actions did cause it to collect substantial negative reward in the process. For instance to reach the grid square (1,5), the agent would repeatedly get to grid square (5,1) and carry out random actions in the hope that the mirror action (1/9 probability) is achieved. To supress this behavior we reduced the bonus that came with arriving at a location via a less probable action.

2.2 General Observation

Initially, we encountered challenges in ensuring that our model could successfully collect all available rewards before proceeding towards the goal. After implementing the safety measures outlined in section 1.2, however, we found that our agent was getting stuck in loops and eventually using up all the allowed time steps (`max_time_steps`).

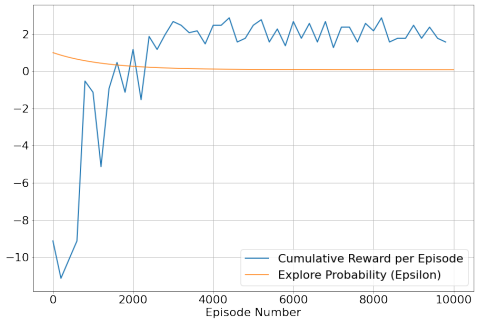
Upon further investigation, we determined that the issue was due to insufficient detail in the state information provided to the agent. Initially, we had only included the agent's position on the grid in the state. To overcome this limitation, we decided to augment the state information with details regarding the current rewards available on the grid.

This additional information proved to be crucial in enabling the agent to differentiate between tiles that had already been rewarded and those that still held rewards. For instance, turning left into a tile before and after the reward on that tile had been collected should intuitively have different action-values. By providing the agent with the necessary contextual information, we were able to help it make more informed decisions and ultimately improve its performance.

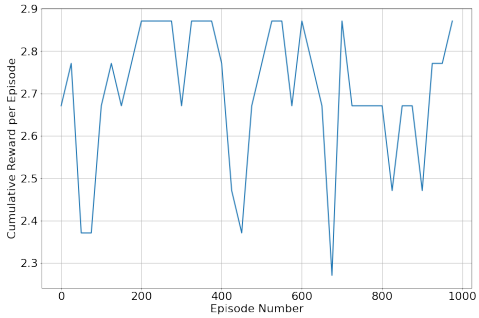
Accross all the experiments, epsilon decay was calculated using the equation:

$$\epsilon = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) * e^{-\lambda * index_{current\_episode}} \tag{4}$$

Here, λ is the decay rate which we set to $8e-4$, ϵ_{max} and ϵ_{min}, were set to 1.0 and 0.08 respectively. Figure 11 is a graphical representation of ϵ v/s $num_{episodes}$.

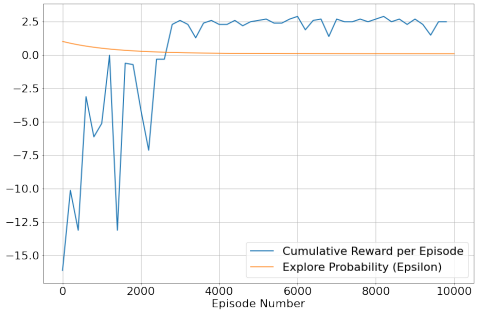


(a) Q-Learning Training ($\gamma = 0.90; \alpha = 0.45$)

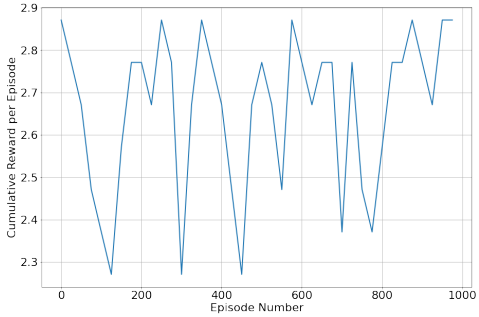


(b) Q-Learning Evaluation

Figure 3: Cumulative Reward History



(a) Double Q-Learning Training ($\gamma = 0.90; \alpha = 0.45$)



(b) Double Q-Learning Evaluation

Figure 4: Cumulative Reward History

2.3 Results–Deterministic Environment

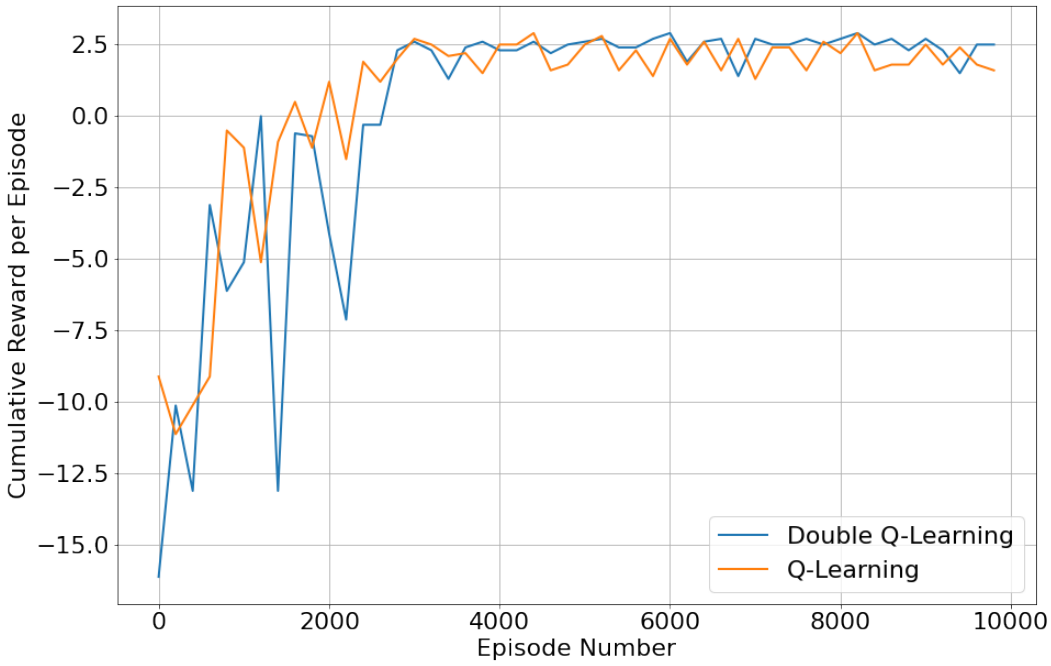


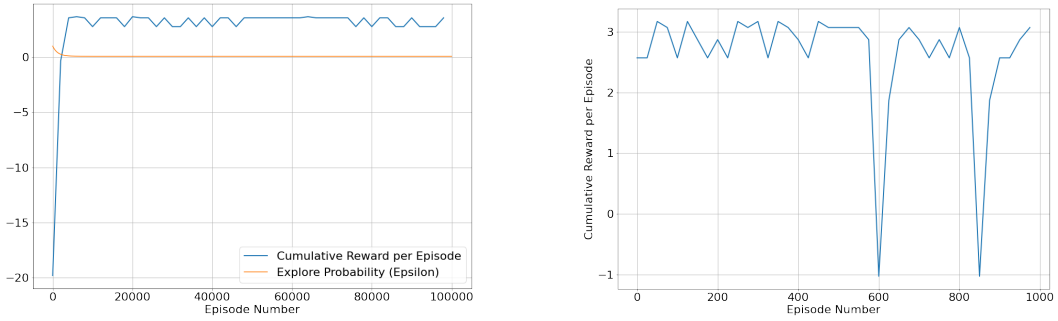
Figure 5: Q v/s Double- Q learning training cumulative reward curve comparison

When looking at the data in Figures 4a, 3a, and 5 that show the Cumulative per Episode Reward v/s Episode, we found that as was expected, the agent trained using Q learning converged much faster than the one trained using Double Q Learning. This is because the environment is relatively simple and there is no variance in the reward distribution for different actions in a given state.

An interesting observation is the Double Q Learning agent's stable cumulative per episode reward. This stability can be attributed to the better estimates of the action-value function that Double Q Learning provides, thanks to its maintenance of two Q Tables.

The reason why there are slight variations in the cumulative reward per episode for both agents during evaluation runs is that the starting position of the agent is chosen randomly at the beginning of each episode.

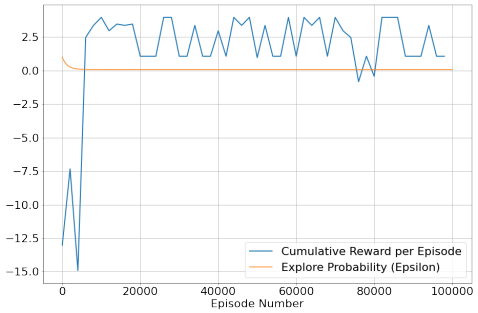
2.4 Results–Stocastic Environment



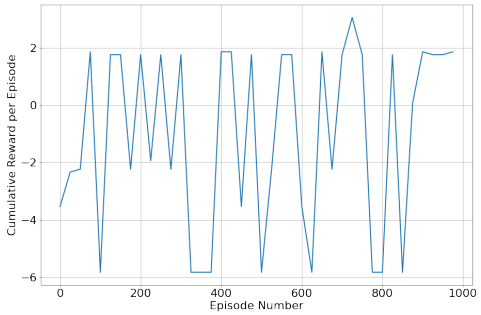
(a) Q-Learning Training ($\gamma = 0.90; \alpha = 0.45$)

(b) Q-Learning Evaluation

Figure 6: Cumulative Reward History



(a) Double Q-Learning Training
($\gamma = 0.90; \alpha = 0.45$)



(b) Double Q-Learning Evaluation

Figure 7: Cumulative Reward History

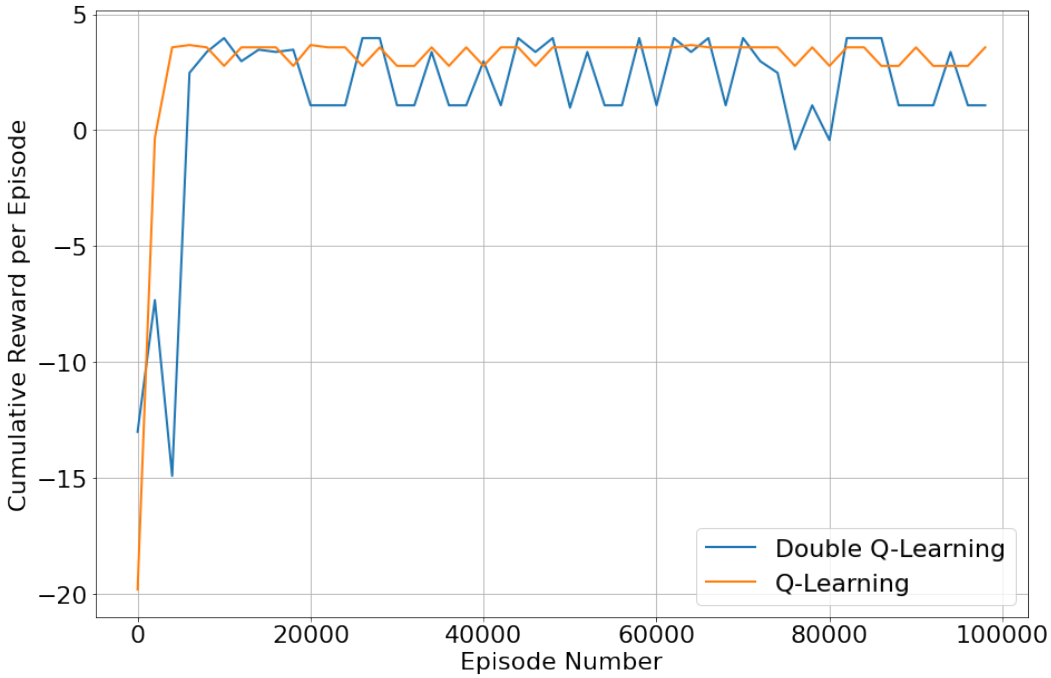


Figure 8: Q v/s Double- Q learning training cumulative reward curve comparison

The Figures 7a, 6a, and 8 suggest that the Double Q Learning agent's cumulative per episode reward is relatively unstable as compared to the agent trained via Q learning. This loss in stability can be attributed to the lower sample effiency (Double Q learning requires twice as many estimations to converge). It can be expected that given more training episodes the double Q learning agent will eventually out perform the Q learning agent.

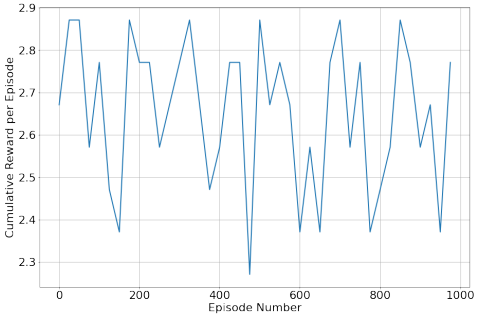
The cumulative reward per episode for both agents varies slightly during evaluation runs. This is because the starting position of the agent is randomly chosen at the beginning of each episode. Additionally, the stochastic nature of the environment may cause the agent to take longer paths to reach rewards and goals in order to avoid potentially dangerous tiles with high negative reward neighborhoods.

2.5 Hyperparameter Tuning

We use the Optuna Library [1] to tune the γ (discount factor) and α (learning rate) for our *Deterministic Environment's Q-Leaning Agent.* We vary the γ in the range `0.8-1.0` and the α in the

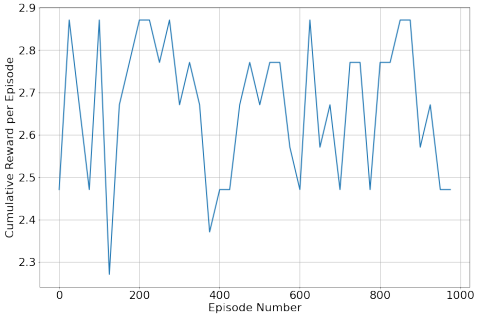
range `0.1-0.5`. We ran optimization for 100 trials and the best hyperparameter pair was chosen based on the **highest** *average cumulative reward per episode* during **evaluation**.

(a) Training $\gamma = 0.90; \alpha = 0.45$



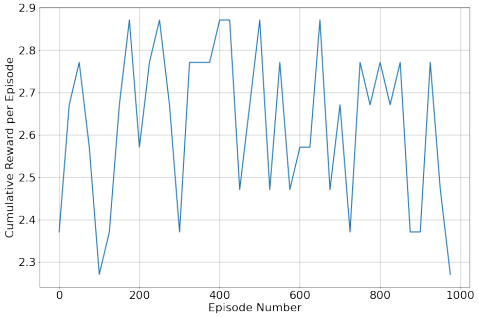
(b) Evaluation $\gamma = 0.90; \alpha = 0.45$

(c) Training $\gamma = 0.86; \alpha = 0.47$



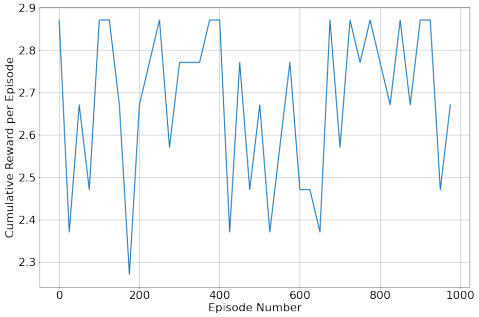
(d) Evaluation $\gamma = 0.86; \alpha = 0.47$

(e) Training $\gamma = 0.90; \alpha = 0.27$

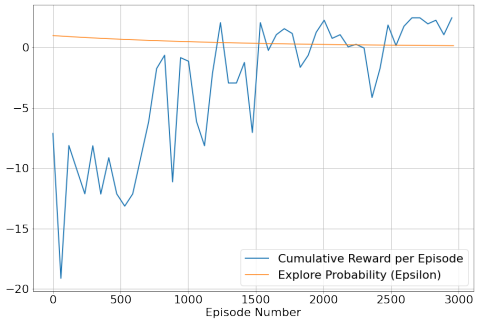


(f) Evaluation $\gamma = 0.90; \alpha = 0.27$

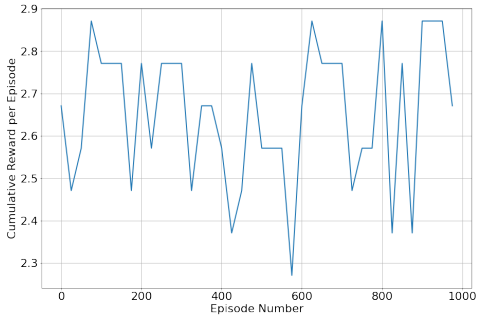
(g) Training $\gamma = 0.97; \alpha = 0.44$



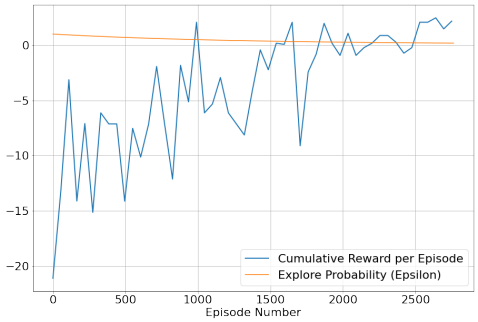
(h) Evaluation $\gamma = 0.97; \alpha = 0.44$



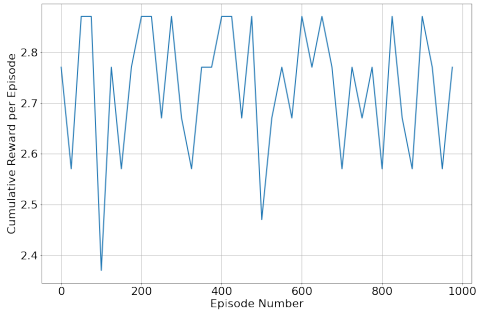
(a) Training $\gamma = 0.97; \alpha = 0.20$



(b) Evaluation $\gamma = 0.97; \alpha = 0.20$



(c) Training $\gamma = 0.81; \alpha = 0.50$



(d) Evaluation $\gamma = 0.81; \alpha = 0.50$

Figure 10: Tuning Results

The tuning results (as shown in Figure 10) suggest that the best hyperparameter pair is $\gamma = 0.90; \alpha = 0.45$. Using these values we were able to have the Q-Learning agent learn an optimal policy in 953 episodes as opposed to the previous 2000 (see Figure 3a). The slightly lower discount factor coupled allows the agent to prioritize finding the **shortest** path to nearby positive rewards and the higher learning rate allows the agent to learn the optimal policy in fewer iterations.

We suggest using these values ($\gamma = 0.90; \alpha = 0.45$) to train future agents to solve this environment as they achieved the best textitaverage cumulative reward per episode during **evaluation** of 2.81.

3 Stock Market Environment

3.1 Training and Evaluation Result

We trained a Q Learning agent to maximize account worth trading the nVidia stock. The agent took into consideration the market trends for the preceding `10` days when making buy, sell and hold decicions. We train our agent for `10,000` episodes.

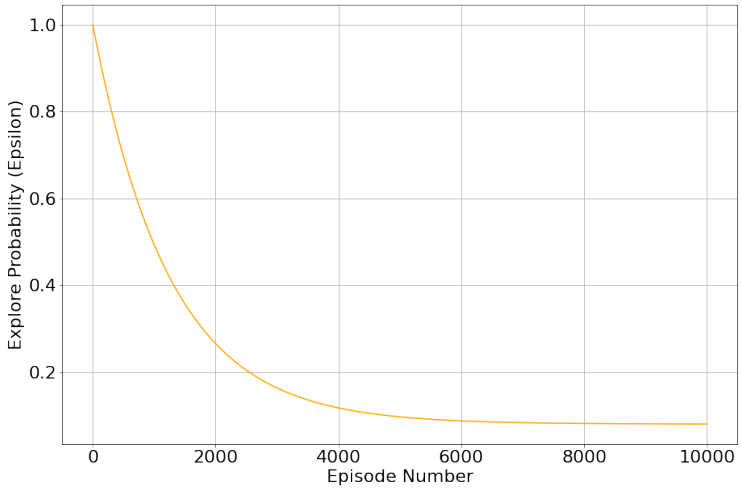
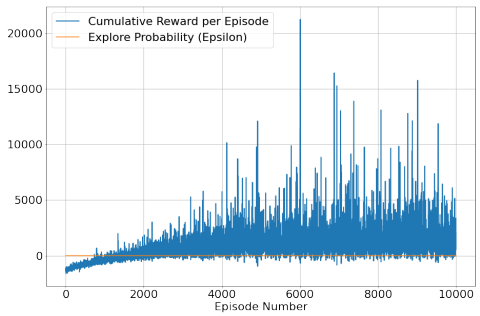
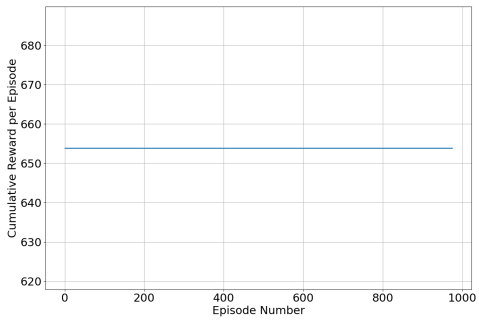


Figure 11: Explore Probability ϵ (decay=$8e-4$)



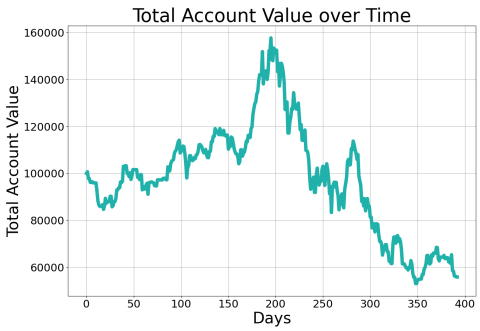
(a) Q-Learning Training $\gamma = 0.95; \alpha = 0.40$



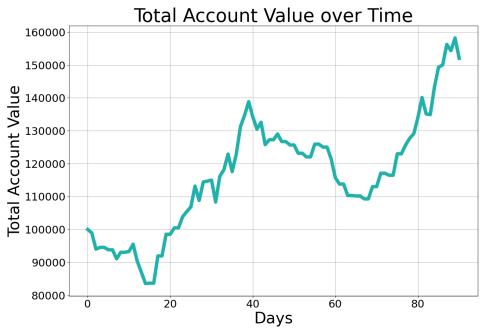
(b) Q-Learning Evaluation

Figure 12: Cumulative Reward History

As shown in Figure 12a, the agent learns the optimal policy in about `2000` training episodes and is able to maintain a profitable portfolio. The trained agent is able to maintain a constant reward rate of 654 accross the 1000 evaluation episodes.



(a) Training Networth accross 400 days



(b) Evaluation Networth accross 100 days

Figure 13: Agent Networth Graph

The evaluation networth chart 13b suggests that the Agent has learnt the optimal way to hold, sell and buy stock. Buy penalizing buy actions when the agent already has stock, we disincentivize hoarding

and the *sunken-cost-fallacy* where the agent may try to recover losses buy purchasing more stock in a bear market with a downward trend.

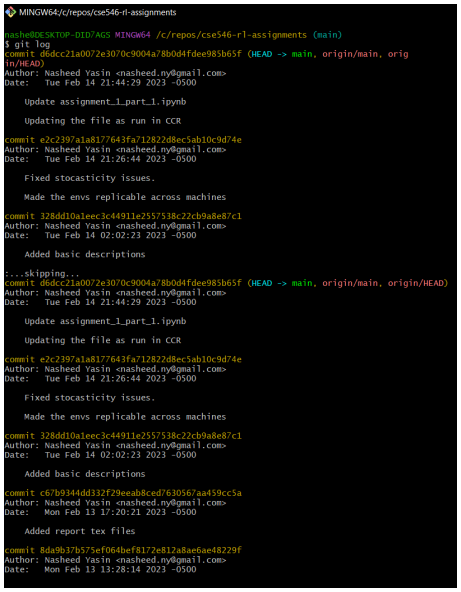
References

[1] Takuya Akiba et al. “Optuna: A next-generation hyperparameter optimization framework”. In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 2623–2631.

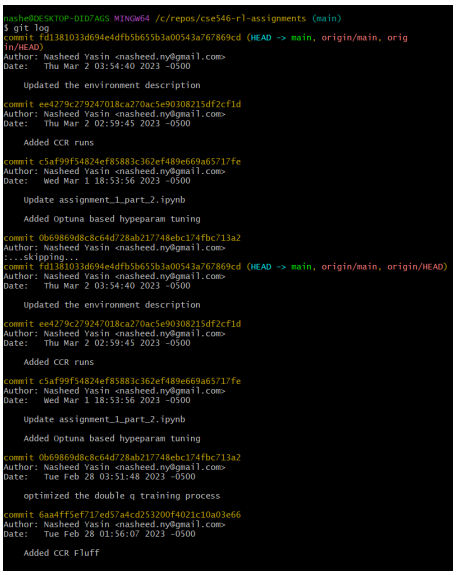
[2] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: `arXiv:1606.01540`.

Addendum

The code for this assignment has been uploaded to `https://github.com/nasheedyasin/cse546-rl-assignments`



(a) Checkpoint



(b) Final

Figure 14: Commit history