Project 2: Q-learning with Wang-Landau Exploration

Abstract

This project is a preliminary exploration work of a research idea of using Wang-Landau sampling for exploration in reinforcement learning. Traditional exploration techniques such as ϵ-greedy often struggle in non-stationary environments due to exploration being too close to the greedy policy. This can lead to an agent being stuck in outdated policy when the environment changes for the better. We propose incorporating Wang-Landau sampling as the behaviour policy in Q-learning. This approach encourages a uniform exploration in the state-action space by sampling actions based on the density of state-action values, overcoming the getting-stuck issue of ϵ-greedy exploration. We compare the proposed method, Q-learning with Wang-Landau sampling with the traditional Q-learning method in a grid world environment that improves. The results show that our method is able to learn the new and better policy, whereas the traditional Q-learning method is stuck with the outdated policy. This highlights the potential of the research idea.

1 Introduction

In reinforcement learning, an agent aims to learn how to act in an environment– through trial and error–such that it maximizes its cumulative reward (Figure 1a for an example simple RL task). How an agent acts in any given situation is determined by its policy. Exploration is crucial for an agent to discover improved policies. It is the counterpart to exploitation, where the agent exploits its experience to maximise rewards. Without proper exploration, an agent may continue exploiting a suboptimal policy without ever discovering better ones. This challenge is particularly challenging in non-stationary environments, where better policies can emerge over time.

Traditional methods of learning a policy like `SARSA` and Q-learning use ϵ-greedy or decaying ϵ-greedy for exploration. While ϵ-greedy exploration works well in many scenarios, it can struggle in these non-stationary environments because ϵ-greedy explores areas close to the greedy policy (see Figure 1b for illustration). Therefore, these methods often fail to adapt quickly enough to changes, potentially getting agents "stuck" in outdated strategies as the environment evolves.

In this project, we propose an off-policy method called Q-learning with Wang-Landau sampling (`QWL`), which uses the Monte Carlo technique of Wang-Landau sampling for exploration. We want to encourage a more uniform exploration of policies (both good and bad) to discover improved policies that may appear far away from the trajectory of the greedy policy. The Wang-Landau sampling archives this by exploring policies uniformly by doing a non-Markovian random walk in the state-action-value space. The idea is that the Wang-Landau exploration will prevent agents from getting "stuck" and, thus, lead to quicker discoveries of improved policies than ϵ-greedy in changing environments.

Objective The objectives of this project are as follows:

1. To incorporate Wang-Landau sampling into the exploration process such that the histogram of state-action values visited is approximately uniform.
2. To explore whether the proposed method discovers improved policy quicker than ϵ-greedy in an environment that changes for the better.
3. To determine whether the proposed method is worth further research.

The rest of the project report is outlined as follows: Section 2 introduces the basics of reinforcement learning and Section 3 introduces the Wang Landau Algorithm. Section 4 describes the proposed method. Section 5 covers the experimental setup and results. Finally, in Section 6, we discuss the results and conclude.

2 Reinforcement Learning

In Reinforcement Learning (RL), an agent faces a sequential decision problem and receives rewards based on its performance. This sequential decision problem can be formalized as a Markov Decision Process (MDP), characterized

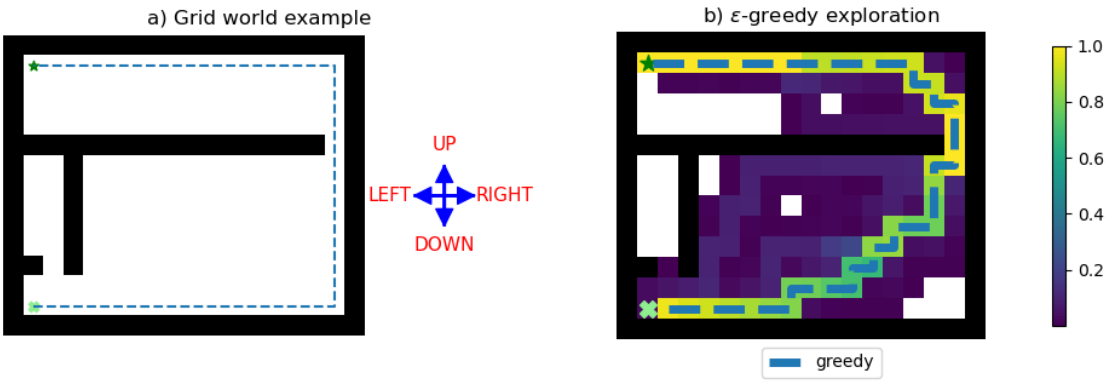

Figure 1: a) A grid world environment where the task is for the agent to learn to find the shortest path from the start position (bottom left) to the goal (top left) by moving left, up, right, or down at each time step. The dotted line highlights one of many equally optimal paths. b) Show the number of states visited by 1000 runs of ϵ-greedy after learning the optimal policy as a heatmap using $\epsilon = 0.1$. The white areas are not being visited. If, now, a shorter path emerges due to an opening in the barrier far from the trajectory of the greedy policy, ϵ-greedy would struggle to discover it and risk being stuck on a suboptimal policy.

by a 4-tuple: $(\mathcal{S}, \mathcal{A}, \mathcal{P}_T, \mathcal{R})$. Here $\mathcal{S}$ represents the set of states, $\mathcal{A}$ the set of actions, $\mathcal{P}_T$ the transition probabilities, and $\mathcal{R}$ the reward function. The agent learns by interacting with its environment taking actions from a set $\mathcal{A}$, observing states from $\mathcal{S}$, and receiving rewards from $\mathcal{R}$. At each step, t, the agent observes a state S_t, selects an action A_t based on its policy $\pi(A_t|S_t)$, receives reward R_{t+1}, and transitions to a new state S_{t+1} with a probability $\mathcal{P}_T(S_{t+1}, R_{t+1}|S_t, A_t)$. This interaction forms a sequence or trajectory τ:

$$\tau = S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots.$$

An agent aims to learn a policy π that maximizes its discounted cumulative reward, known as the return $G_t = \sum_{t=0}^{T} \gamma^t R_{t+1}$, where γ is the discount factor. The value function, $V^\pi(S)$ represents the expected return from state S under policy π, while the state-action value function $Q^\pi(S, A)$ specifies the expected return upon taking action A in state S and thereafter following π.

2.1 Learning an optical policy with SARSA and Q-Learning

SARSA and Q-Learning are temporal-difference (TD) learning methods. TD is similar to Monte Carlo methods in RL, Both use the pattern of generalized policy iteration (GPI) (see Figure 2) to learn optimal policy (Sutton & Barto (2018)) and both update estimates using sampled trajectories from a policy π. The difference between MC and TD is that MC waits until the end of an episode to update its estimates based on the actual return G, whereas TD methods update their estimates every timestep using bootstrap estimates of the return:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[\underbrace{R_{t+1} + Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)}_{\text{TD error/difference}}].$$

Here α is the learning rate.

SARSA is an on-policy, whereas Q-Learning is an off-policy method. On-policy methods learn the policy used to sample the trajectories (behaviour policy), whereas off-policy methods learn a different policy (target policy) than the behaviour policy. The behaviour policy is often an ϵ-greedy policy, where a random action is selected with probability ϵ and the greedy action is selected with probability $1 - \epsilon$. The target policy, in Q-Learning, is typically the greedy policy ($\epsilon = 0$). See Algorithm 1 for more details.

As mentioned in the introduction, ϵ-greedy exploration may get stuck in outdated strategies since it explores areas close to the greedy policy. To mitigate this, techniques like optimistic initialization can help in the early stages of the exploration. By initializing Q-values to higher-than-expected values, the agent is biased to explore untried actions early in the learning process, as they initially appear more rewarding.

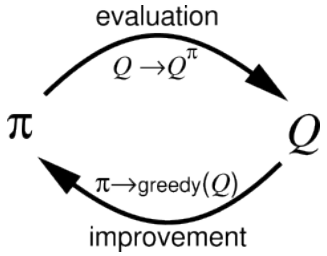

Figure 2: Generalized policy iteration: Value and policy functions interact until they are optimal and thus consistent with each other (Sutton & Barto (2018)). It consists of a policy evaluation step $Q = Q^\pi$ and a policy improvement step $\pi = \text{greedy}(Q)$. We alternate between the evaluation improvement steps until Q and π converge are optimal Q^* and π^*.

Algorithm 1 Q-learning using ϵ-greedy exploration

Initialize $Q(S, A)$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$ arbitrarily, except that $Q(\text{terminal}, \cdot) = 0$
Input: step size $\alpha \in (0, 1]$, discount factor $\gamma \in (0, 1]$, exploration probability $\epsilon \in (0, 1)$
for each episode **do**
    Initialize state S
    while state S is not terminal **do**
        Choose action A from state S using ϵ-greedy
        Take action A, observe reward R and next state S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
        $S \leftarrow S'$
    end while
end for

3 Wang Landau Algorithm

The Wang-Landau (WL) algorithm (Wang & Landau (2001))[1] is a Monte Carlo method that is used to estimate the density of states $g(E)$ of a system characterized by a cost function E. The WL quickly estimates a system's density of states via a random walk in the cost space, which simulates a uniform sampling across all cost levels (i.e., flat histogram in cost space). This ensures rapid exploration of both high and low-probability states.

Algorithm 2 Wang-Landau algorithm

Initialize the density of states $g(E) = 1$ for all energy levels E
Initialize the histogram $H(E) = 0$ for all energy levels E
Choose an initial modification factor $f > 1$
Choose an initial configuration r with energy E from Ω
while modification factor f is not small enough **do**
    Propose a move to a new configuration r' from Ω using proposal distribution $h(r \to r')$
    Calculate the energy E' of the new configuration
    if rand() $< \min\left(1, \frac{g(E)}{g(E')} \frac{h(r' \to r)}{h(r \to r')}\right)$ **then**
        Accept move: $r = r'$ and $E = E'$
    end if
    Update $g(E) \leftarrow g(E) \cdot f$
    Update $H(E) \leftarrow H(E) + 1$
    if Histogram $H(E)$ is "flat" **then**
        Reset $H(E) = 0$ for all E
        Update $f \leftarrow \sqrt{f}$
    end if
end while

The Wang-Landau algorithm is non-Markovian since it relies on keeping track of the historical data of density of states for all cost through $g(E)$ and and its corresponding histogram $H(E)$ to guide the random walk. It works as follows

[1] See also https://en.wikipedia.org/wiki/Wang_and_Landau_algorithm and https://en.wikipedia.org/wiki/Multicanonical_ensemble.

(see also Algorithm 2): Consider the state space Ω of a system with a bounded cost function $E = \Gamma = [E_{\min}, E_{\max}]$. First, we initialize the density of states for all cost levels and the corresponding histogram to one and zero ($g(E) = 1$ and $H(E) = 0$), respectively. Then we start the random walk from a random state $r \in \Omega$ with cost E and propose a new state r' with cost E' according to a arbitrary proposal distribution $h(r \to' r)$. The move is accepted with an acceptance probability $\min\left(1, \frac{g(E)}{g(E')} \frac{h(r' \to r)}{h(r \to r')}\right)$. If accepted r' and E' becomes the current state r and cost level E. Then $g(E)$ is updated by multiplying by a modification factor $f > 1$ and $H(E)$ is incremented. The process continues until the histogram is approximately "flat," indicating that all cost states have been uniformly explored. At this point, the histogram is reset, and the modification factor is reduced by taking its square root. This refines the density of states further. The algorithm repeats this process until the modification factor becomes sufficiently small, suggesting convergence of the density of states.

In its simplest form Wang-Landau uses a uniform proposal distribution, but it also exists more sophisticated proposal distribution such as the gradient based Monte Carlo proposal of Liu et al. (2023).

4 Q-learning with Wang-Landau Sampling (QWL)

The proposed method, QWL, introduces the Wang-Landau algorithm as the behaviour policy of the Q-learning to encourage uniform exploration in the state-action space, Q. We draw the connection between the density of states and Q-values. An action A in state S can be seen as a configuration r with cost E given by $Q(S, A)$. The density of states is then the number of state-action pairs that lead to the same expected Q-value.

Algorithm 3 QWL: Q-learning using Wang-Landau Sampling

Input: step size $\alpha \in (0, 1]$, discount factor $\gamma \in (0, 1]$, bin size bin_size
Initialize $Q(S, A)$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$ arbitrarily, except that $Q(\text{terminal}, \cdot) = 0$
Initialize the density of states $g(Q) = 1$ for all state-action values Q
Initialize the histogram $H(Q) = 0$ for all state-action values Q
Initialize the initial modification factor $f > 1$
function WLACTIONSAMPLING(S)
    Choose an initial action A with $E = \lfloor \frac{Q(S,A)}{\text{bin\_size}} \rfloor$
    accepted $\leftarrow$ False
    while not accepted **do**
        Propose action A' sampled uniformly from $\mathcal{A}(S)$
        Get $E' = \lfloor \frac{Q(S,A')}{\text{bin\_size}} \rfloor$
        if rand() $< \min(1, \frac{g(E)}{g(E')})$ **then**
            Accept move: $A = A'$ and $E = E'$
            accepted $\leftarrow$ True
        end if
        Update $g(E) \leftarrow g(E) \cdot f$
        Update $H(E) \leftarrow H(E) + 1$
    end while
    return A
end function
for each episode **do**
    Initialize S
    while S is not terminal **do**
        $A \leftarrow$ WLACTIONSAMPLING(S)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
        $S \leftarrow S'$
    end while
end for

QWL updates its Q-values using a greedy target policy but replacing the ϵ-greedy behaviour policy with Wang-Landau sampling. The Wang-Landau sampling selects actions given a state such that state-action values are visited uniformly. This change should allow for quicker and more thorough exploration of both good and bad policies. The proposed exploration method works as follows (See also Algorithm 3): We keep track of the density of states $g(E)$ and corresponding histogram $H(E)$ of how often each Q-value is visited. The density of states ratio $\frac{g(E)}{g(E')}$) is used to weigh the action sampling, through the acceptance ratio, which ensures approximately uniform exploration of the state-action

space. Actions are proposed repeatedly until one is accepted. The agent then takes the accepted action, observes a new state, receives a reward and updates its Q-value $Q(S, A)$.

Note that the method uses a uniform proposal distribution, simplifying the calculation of the acceptance ratio. Additionally, since $Q(S, A) \in \mathbb{R}$, binning is necessary, where the bin size is a hyperparamer. For simplicity, the adaptation of Wang-Lanadu algorithm does not include flatness check and decaying the modification factor.

5 Experiments

5.1 Experimental Setup

Enviromnent. We evaluate the proposed method in an environment similar to the environment in Example 8.3: Shortcut Maze in the book of Sutton & Barto (2018). The environment, presented in Figure 3, is a 15 by 20 grid world where the agent's task is to find the shortest path from start to goal. Each **state** is defined by the agent's (i, j) coordinates in the grid, and at each state, the agent can select between four **actions**, move left, right, up or down. The white cells are states the agent can move to and the cells are the barriers. If the agent tries to move into a black cell, it hits the barrier and remains in the current state. The agent receives a -1 **reward** each time step unless it has reached the goal (0 reward).

The setup. The setup aims to illustrate an example where ϵ-greedy methods struggle to correct their policy when the environment improves. The goal is to learn the optimal path over 5000 episodes. Initially, the optimal path is to go around the barrier on the right. After 3333 episodes, a shortcut opens up on the left side and the agent has to adapt.

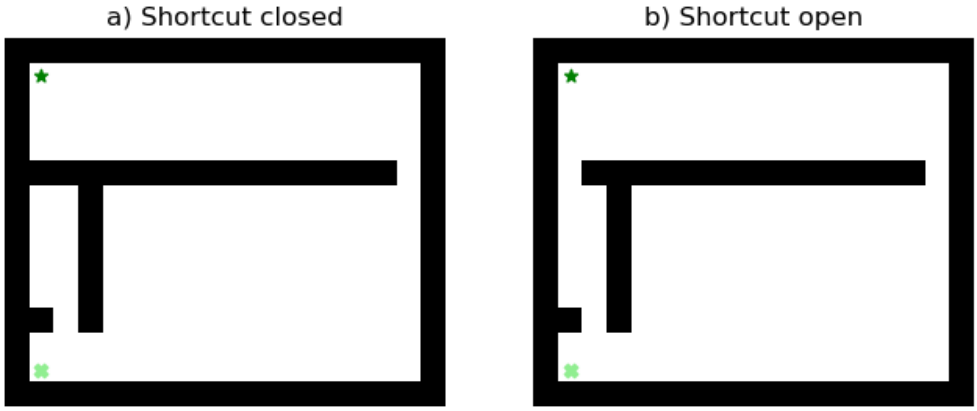

Figure 3: The environment has two stages a) before and b) after opening the shortcut. The agents start at the light-green cross and the goal is to find the shortest path to the goal marked with the green star. At each step, the agents can move left, right, up or down, and receive a -1 reward unless they have reached the goal (0 reward). White cells are navigable states, while black cells are barriers. If an agent attempts to move into a black cell, it hits the barrier and stays in its current state.

Performance measure. The goal is to discover the shortcut and learn the new optimal path. Thus, in each episode, we measure whether the agent found (+1) the shortcut or not (0) and plot the cumulative sum averaged over 10 repeated experiments. The shortcut is considered found if the agent discovers a path shorter than 30 steps (the optimal path length is 42 steps when the shortcut is closed and 14 steps when it is open). This metric provides the agent with some flexibility to identify the shortcut while ensuring it does not take too long to reach the goal.

Agents. We run the experiment for 3 agents, the `QWL` and two Q-learning agent with $\epsilon = 0.1$ and $\epsilon = 0.5$, respectively.

5.2 Experimental Results

Figure 4, shows that `QWL` is able to discover the new and shorter path more frequently than ϵ-greedy based Q-learning agents, hereafter referred to $Q_{\epsilon=0.1}$ and $Q_{\epsilon=0.5}$. After the shortcut opens, the log cumulative number of times `QWL` agents find the shortcut grows drastically faster than $Q_{\epsilon=0.1}$ and $Q_{\epsilon=0.5}$. The $Q_{\epsilon=0.5}$ finds the shortcut more frequent than $Q_{\epsilon=0.1}$, but far from frequency of `QWL`. This highlights `QWL` ability to discover new policies and adapt to a changing environment.

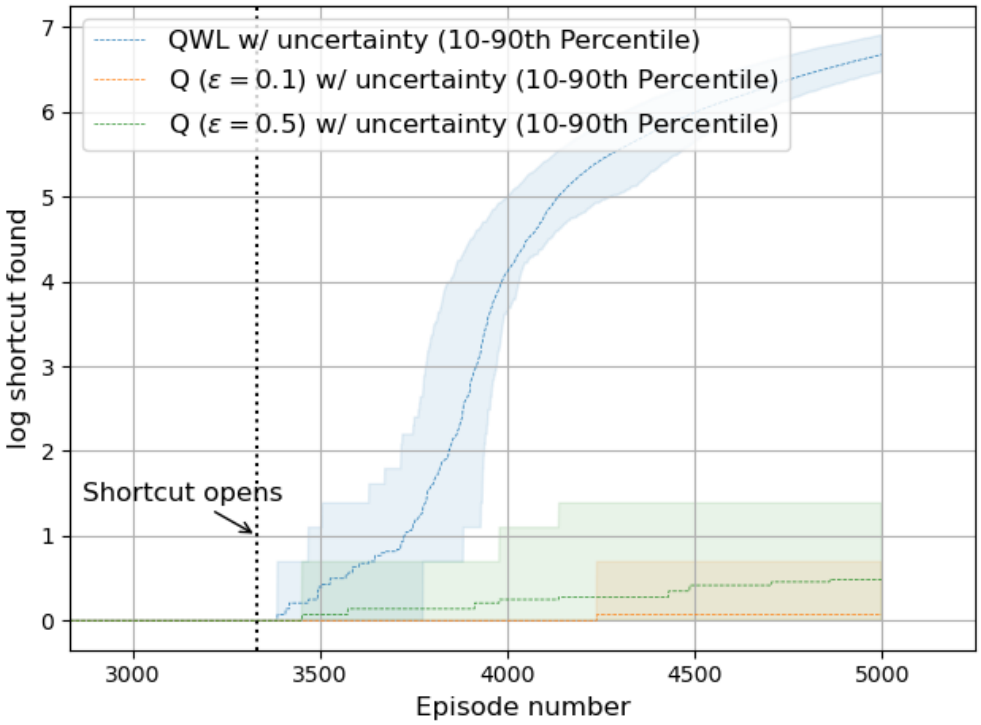

Figure 4: The log cumulative sum of whether the agents found the shortcut (+1) or not (0) in an episode averaged across 10 repeated experiments. The shaded areas represent the uncertainty (10th-90th percentile). A vertical dotted line at episode 3333 marks when the shortcut opens.

Figure 5 shows the evolution of Q-values over time. Initially, all agents have the Q-values (initialized to zero i.e., optimistic initialization). After 3332 episodes of training with the shortcut closed the methods Q-values remain similar. At episode 3333, the shortcut opens and QWL quickly adapts, reflecting higher Q-values (yellow) along the new optimal path. In contrast, both Q-learning agents show slower adaptation, indicated by lower Q-values (more purple) along the new optimal path. While the $Q_{\epsilon=0.5}$ shows slightly better adaptation than $Q_{\epsilon=0.1}$, it is still poor. This is shown by higher Q-values of only one cell further along the new optimal path. The evolution of Q-values demonstrates `QWL`'s better ability to discover and learn the shortcut compared to the ϵ-greedy-based Q-learning agents.

Figure 6 shows how well the target/greedy policy performs at specific episodes, measured by path length. The figure shows that all methods learn the optimal policy when the shortcut is closed. But when the shortcut opens, only `QWL` adapts and learns the new optimal policy, whereas ϵ-greedy-based Q-learning agents are stuck in their outdated policies. The results are consistent across all 10 repetitions, resulting in no uncertainty bands.

6 Discussion

The experimental results show that `QWL` is better at adapting to changing environments than the ϵ-greedy-based Q-learning agents, at least for this particular setup. The `QWL` finds the shortcut earlier and more frequently than ϵ-greedy-based methods, enabling it to update Q-values accordingly and learn the new optimal policy. In contrast, the ϵ-greedy-based methods fail to adapt because they rarely find the shortcut, leading them to be stuck in their outdated policies. This suggests that there is something to the `QWL` method and I should explore this method more, potentially leading to a paper.

Although the results heavily favour `QWL`, it is important to reflect on when `QWL` may be preferable over ϵ-greedy-based Q-learning and vice versa. The `QWL` is suitable when extensive exploration is reasonable and beneficial, such as changing environments where discovering the optimal policy is important. However, this level of exploration is not always practical or necessary. In many scenarios, having a good, though not necessarily optimal, policy is sufficient. The ϵ-greedy-based Q-learning can be more appropriate in these scenarios or scenarios where the environments do not

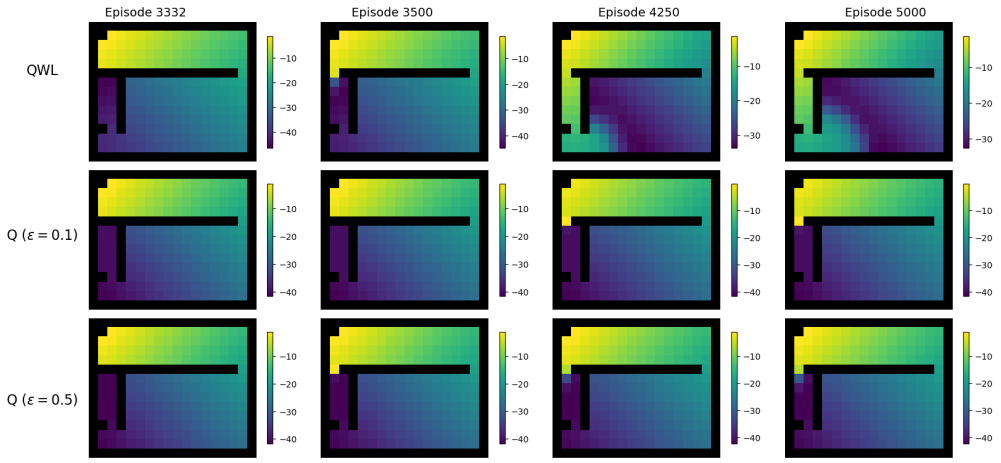

Figure 5: The evolution of Q-values over time of QWL and Q-learning with two different $\epsilon = 0.1$ and $\epsilon = 0.5$. It plots the averaged Q-values over actions for QWL and SARSA after different episodes. The rows represent the averaged Q-values for QWL (top row) and SARSA (bottom row). Each column shows the averaged Q-values at specific episodes (3332, 3500, 4250, and 5000). The colour scale indicates the value of Q-values, with yellow representing higher values and purple representing lower values.

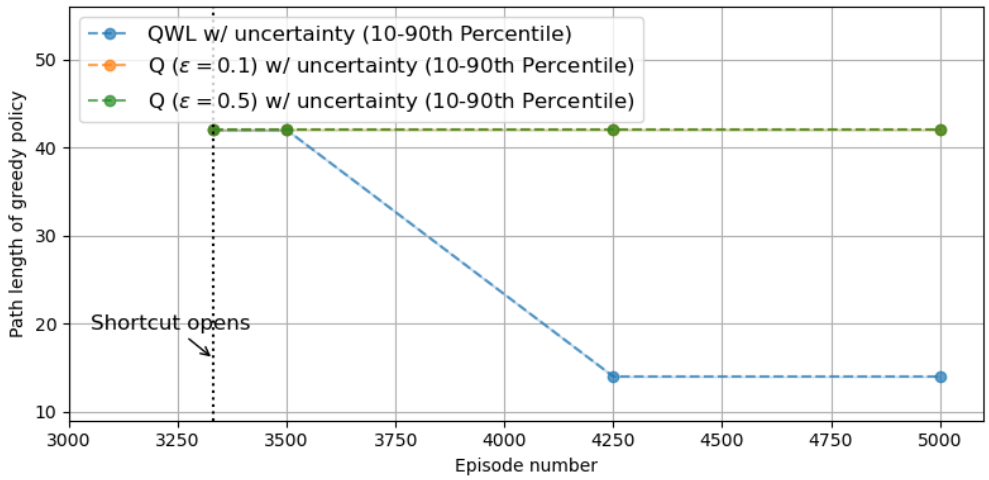

Figure 6: The path length of the greedy policy over episodes for QWL and Q-learning with $\epsilon = 0.1$ and $\epsilon = 0.5$. The vertical dotted line at episode 3333 marks when the shortcut opens.

change or the changes happen close to the trajectory of the current policy. The simplicity of ϵ-greedy over QWL is also worth mentioning.

One practical application of the QWL method is its potential role as an experience collector for other agents that cannot afford extensive exploration. For example, for agents that deliver items, extensive exploration may result in customer complaints. Here QWL can be used as a separate exploration agent that can discover new, efficient routes and share this information with the main delivery agent, improving overall performance without disrupting its primary task. This is similar to the method by Christianos et al. (2021), where multiple agents share experiences to learn more effectively, or the A3C method by Mnih et al. (2016), which runs multiple independent instances of the environment in parallel, with each thread sharing its experience to update a shared model asynchronously.

This project is a preliminary exploration of a potential paper. Due to the limited time of the project and given that I am developing a new method, there are many limitations and ideas left for future work. To start with, a lot of work went into developing the method and making sure that it worked as intended, resulting in less time for a more comprehensive experiment. Given more time, I want to test the method over multiple different environments. Additionally, I want to do an experiment where I run the same environment with gradually increasing the grid size and then plot the result as a function of grid size/ state space. The current experiment could also be improved by comparing QWL to other different exploring techniques, for example, softmax/Boltzmann exploration. A potential issue with QWL is in cases with large action space or many invalid actions (e.g. action that leads to hitting the barrier), I would like to test this. One way of addressing this issue is invalid action masking or seeing if one can use a smart proposal function to sample actions in the Wang-Landau sampling. This should be addressed in future work. The gradient-based Wang-Landau of Liu et al. (2023) sampling may be a good starting point. Extending the QWL from tabular to function approximation is a natural step and is left for future work.

As a side note, QWL might be useful for improving controllability, which is the ability of control systems to reach arbitrary states within an environment. Improving controllability can help address the issue of temporal correlations that can hinder performance in reinforcement learning (Berrueta et al. (2023)).

Overall, this project shows that the idea of using Wang-Landau sampling for exploration in reinforcement learning has potential, showing it can discover improved policies in a changing environment where the traditional ϵ-greedy method fails.

References

Thomas A. Berrueta, Allison Pinosky, and Todd D. Murphey. Maximum diffusion reinforcement learning, 2023.

Filippos Christianos, Lukas Schäfer, and Stefano V. Albrecht. Shared experience actor-critic for multi-agent reinforcement learning, 2021.

Weitang Liu, Ying-Wai Li, Yi-Zhuang You, and Jingbo Shang. Gradient-based wang-landau algorithm: A novel sampler for output distribution of neural networks over the input space, 2023.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Fugao Wang and D. P. Landau. Efficient, multiple-range random walk algorithm to calculate the density of states. *Physical Review Letters*, 86(10):2050–2053, March 2001. ISSN 1079-7114. doi: 10.1103/physrevlett.86.2050. URL http://dx.doi.org/10.1103/PhysRevLett.86.2050.