11/14/2024

Coursework 1

Reinforcement Learning for Bioengineers: BIOE 70077

Erik Garcia Oyono
IMPERIAL COLLEGE LONDON

Contents

Dynamic Programming Agent

Method and Parameter Justification

Dynamic programming (DP) is a method for solving problems by breaking them down into smaller parts, useful in optimisation (Sutton & Barto, 2018). Two common algorithms are **value iteration** (VI) and **policy iteration** (PI). Both find an optimal policy $\pi *$, but differ in approach: **VI** updates $V(s)$ to find $\pi *$, while **PI** implements policy evaluation and improvement until it converges.

The maze size is 13x10, meaning there are 130 locations. However, 32 of these are obstacles and 3 are absorbing states, making the space relatively small. According to Sutton & Barto (2018), PI can be computationally efficient for MDPs with manageable states and actions, as it converges in finite iterations. Furthermore, this process can be easily implemented in smaller state spaces, converging to the optimal policy quicker than VI (Miller, 2023; Tokuc, 2024).

PI directly evaluates and improves a fixed policy, whereas VI explores all possible actions to find the maximum action value (Sutton & Barto, 2018). For this reason, PI requires **fewer iterations** in small state spaces as it can converge to the correct $V(s)$ **faster** (Miller, 2023; Tokuc, 2024). Moreover, PI is **cheaper to compute**, as each iteration of VI involves computationally heavier operations than PI (Tokuc, 2024). However, PI does not scale well as state space grows due to increased complexity in the algorithm, making VI more suitable in these scenarios (NG, 2021).

The policy $\pi(s)$ and value function $V(s)$ were initialised arbitrarily as a zero matrix and zero vector, respectively, according to literature (Sutton & Barto, 2018). Theta (1e-6) controls the stopping condition for policy evaluation, defined by Sutton & Barto (2018) as a small positive threshold to determine the accuracy of estimation. Finally, gamma (0.82) was calculated, giving relatively higher priority to future rewards rather than immediate.

PI assumes the maze environment has a **finite** state and action space, with discrete states and actions, which in this case are given for evaluation and improvement. The algorithm also assumes that the environment's transition probabilities and reward functions are **known**, allowing updates based on the expected returns. This characteristic aligns with the description of dynamic programming by Sutton & Barto (2018).

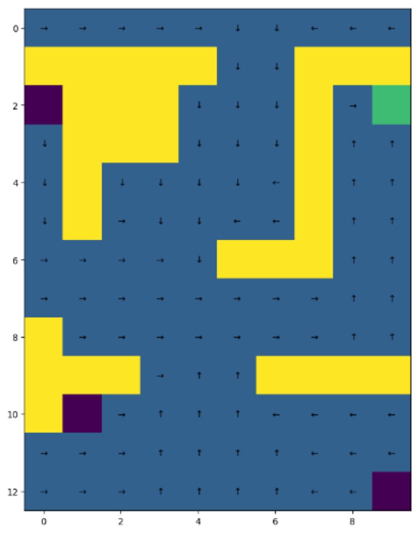

Figure 1.1 Optimal Policy Representation – DP Agent

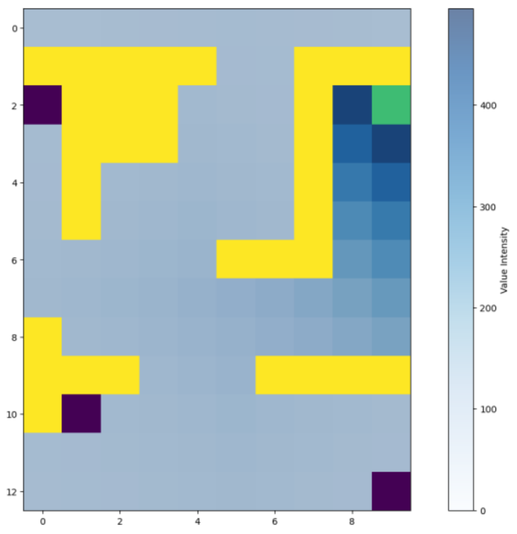

Figure 1.2 Value Function Representation - DP Agent

Effect of Gamma and Transition Probability

The **discount factor (γ)** determines the agent's importance to immediate vs long-term rewards:

- **γ < 0.5:** agent prioritises short-term rewards. The optimal policy $\pi *$ implements actions that result in immediate rewards, even if they do not lead to the best long-term outcome, leading to **suboptimal paths**. Figure 1.3 (a) shows that the value function reflects isolated high values only near the goal state, while distant states have low, slightly negative values.
- **γ > 0.5:** agent prioritises future rewards over short-term gains. The optimal policy leads to a smoother, **optimal path** to the goal state with higher total reward. The value function in Figure 1.3 (b) shows gradual decrease from high-reward states to less rewarding states, as the influence of rewards decreases with each step.

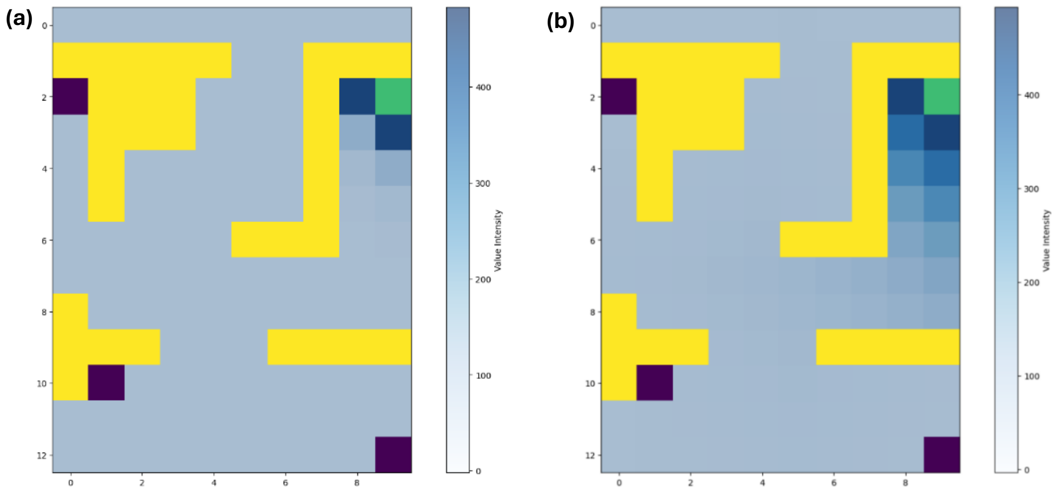

Figure 1.3 Heatmaps of the value function when (a) γ < 0.5 and (b) γ > 0.5

The **transition probability (p)** affects the likelihood of the agent moving in the intended direction:

- **p < 0.25**: agent moves away from goal state due to **higher stochasticity**. Even though the value function $V(s)$ shows higher expected returns near the goal state (Fig. 1.5), the optimal policy $\pi *$ is conservative as it shifts toward the absorbing states.
- **p = 0.25**: agent's probability of moving in any direction is the same. Hence, it takes a **random** action going north only, as it is the first action available. The value function $V(s)$ is still higher closer to the goal state (Fig. 1.6), but the optimal policy does not follow this.
- **p > 0.25**: agent has higher probability of moving in the intended direction (**deterministic**). The optimal policy leads to a smoother path and higher values in the value function (Fig. 1.7).

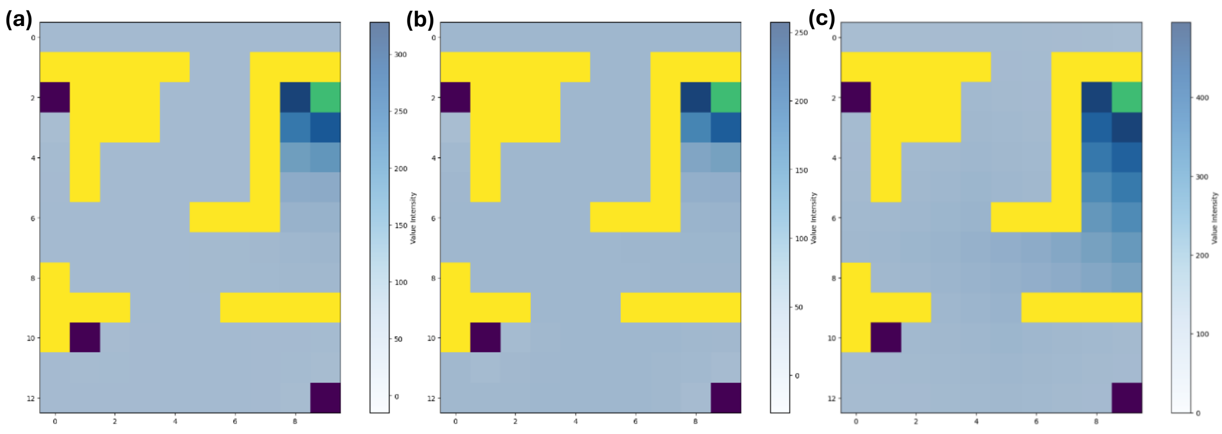

Figure 1.4 Heatmaps of the value function when (a) p < 0.25, (b) p = 0.25 and (c) p > 0.25

Extended Task

To design the reward function such that, for any state s, its value equals the probability of reaching the goal state, the discount factor should be 1 or close to 1 to ensure long-term rewards are prioritised. Moreover, the reward for this state should equal 1, while that for other absorbing states set to 0. This would demonstrate that when the agent reaches the goal state, its expected value would be 1, showing 100% probability of reaching the goal, while at other states, it would be 0%, as it cannot move any further. In the value function, states closer to the goal would have higher values representing the probability of reaching the goal state, as they would be more likely to lead the goal with fewer steps.

Monte-Carlo (MC) Agent

Method and Parameter Justification

First-visit and every-visit MC are methods used to estimate the value function $V(s)$ in episodic tasks. First-visit MC does this by averaging the returns following the first visits to a state s within an episode, whereas every-visit averages the returns after all visits to s (Sutton & Barto, 2018).

The agent implements a vanilla **online**, first-visit algorithm with **decaying** epsilon and learning rate. It estimates the action-value function by updating the Q-values of a state-action pair only the first time encountered during an episode. The Q-values and policy are updated after each episode, using the experience gained during that episode, proving the online learning approach.

This method also encourages **exploration** by assigning greater weight to the early visits of a state-action pair, which often occur during this phase (P. Singh & S. Sutton, 1996). Furthermore, this method provides **unbiased** estimates of state values when the agent revisits states within an episode (Sutton & Barto, 2018). This is because repeated updates within an episode in every-visit MC can make it more sensitive to the environment, leading to higher variance (Szepesvari, 2009).

The MC agent implements **optimistic initialization** by setting Q-values to 1, which encourages early exploration due to high expected rewards. Even though this design may introduce some bias, it is advantageous for small environments because exploration is crucial early on and can lead to faster convergence (Lobel et al., 2022). This is also supported by an **ε-greedy** approach, where epsilon was initialised to 0.7 to ensure the agent explores most of the actions and minimises bias toward specific actions.

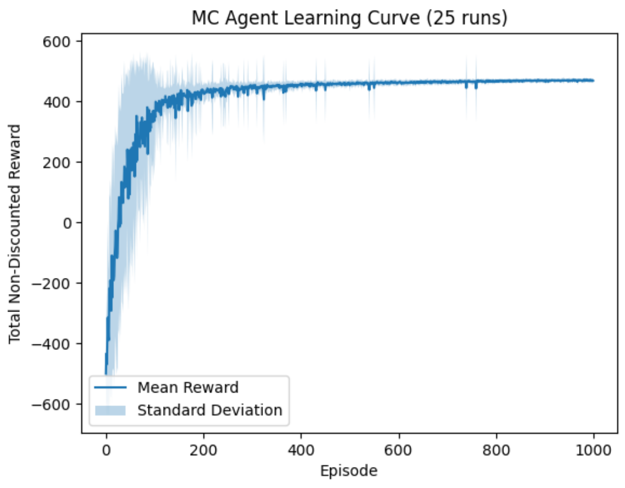

Figure 2.1 Learning Curve of the MC agent

With probability epsilon, the agent takes a random action (exploration), and with probability 1 - epsilon, it exploits its current knowledge by choosing the action with the highest Q-value. This balance is adjusted dynamically by gradually **decaying** 'ε' using a decay rate defined through experimentation, which allows the agent to gradually exploit its learned knowledge after exploration. This behaviour is demonstrated in Figure 2.1, where the standard deviation from the mean return gradually decreases as the agent gains knowledge.

The learning rate, alpha, was set to 0.5 after testing and is decayed after each state-action pair visit using decaying_alpha = 1 / self.visit_count[state, action]. This decay helps stabilise learning by allowing the agent to make larger updates to its Q-values in early episodes and smaller, more refined updates in later episodes to converge to the optimal policy. The discount factor (gamma) was previously set to 0.82, prioritising long-term gains instead of immediate rewards. Finally, the number of episodes was set to 400 so the agent can refine its Q-values, balance exploration and exploitation, and learn an effective policy while reducing computational cost and time.

This design assumes the task is **episodic** and terminates within finite steps in a **stationary** environment, with constant state transitions and rewards. Moreover, it is assumed the state and action spaces are manageable for storing Q-values in a table, which is suitable due to the relatively small state space. Finally, the **Markov property** underlying assumption enables the agent to make decisions based on the current information without needing to consider the past.

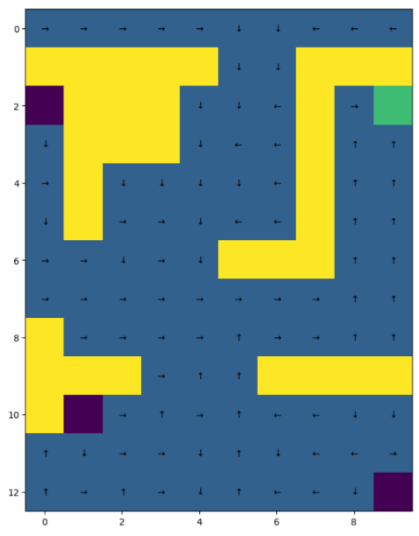

Figure 2.2 Optimal Policy Representation – MC Agent

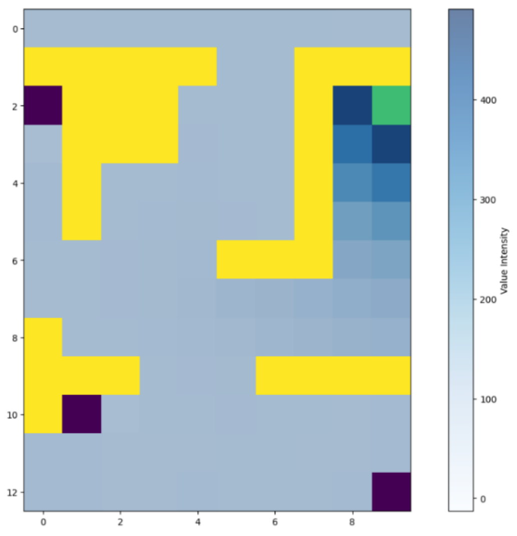

Figure 2.3 Value Function Representation – MC Agent

Extended Task

Ignoring trajectories that do not reach terminal states within 500 steps would negatively impact the value estimates for distant states due to **reduced exploration** and introduce **bias** towards states closer to terminal states. This is because trajectories that could provide useful information about distant states are not learned by the agent, which could result in less accurate estimates for those states (undersampling). Moreover, the algorithm would prioritise updating values for states near terminal states as they are more likely to be sampled in shorter episodes. As a result, the agent's overall performance could be limited by a suboptimal policy and the value estimates would not reflect the true value of all states, especially for those that require longer paths to reach.

Research and Comparative Analysis

Comparative Analysis

The learning speed of the agent was similar for all the strategies. Fluctuations are observed at the beginning due to the model-free environment, and the agent plateaus at around 60 episodes in every case, reducing the deviation significantly in decaying- ε and SoftMax.

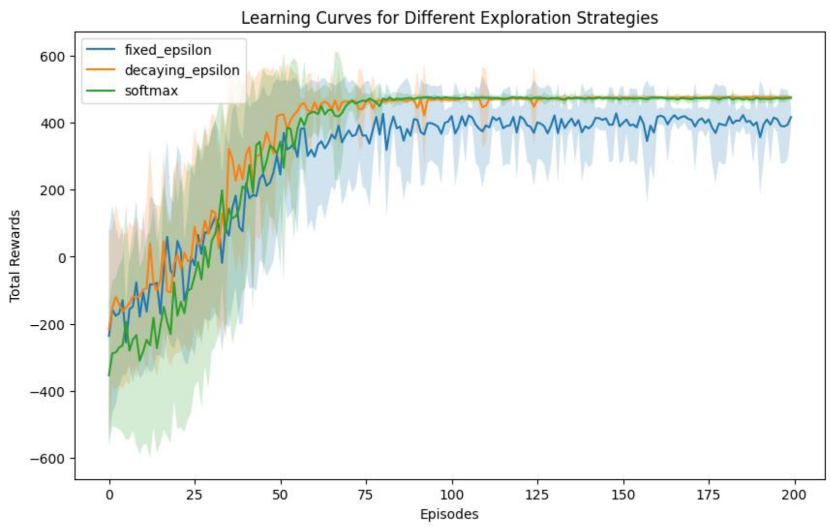

Figure 3.1 Learning curves for epsilon-greedy (fixed/decaying epsilon) and SoftMax exploration strategies.

Fixed epsilon showed high deviation from the mean total rewards and fluctuations even after 200 episodes, as well as the lowest returns. This method did not converge during testing, meaning that it needs further episodes to optimise the policy, as seen in Figure 3.2 (a). The main reason for this is the epsilon being fixed at 0.7, as the agent will consistently choose a random action, limiting exploitation. As discussed by Thrun (1992), maintaining exploration can be advantageous in non-stationary environments, but in this case it leads to suboptimal behaviour.

Decaying-epsilon showed slightly higher long-term rewards and rapid improvement along SoftMax, peaking around the same level, but with fluctuation (Figure 3.1). The decay rate helps balance exploration/exploitation by encouraging the first initially and narrowing down to optimal paths over time, which was an effective approach for the maze environment (Pack Kaelbling et al., 1996). However, the periodic fluctuations within 75 and 125 episodes indicate longer time to converge. In this case, epsilon decreases until the policy converges, so the optimal policy can be executed without exploring further (Figure 3.2 (b)). Due to early exploration and the decay rate until it becomes greedy, it operates more optimally than fixed epsilon (Sutton & Barto, 2018).

SoftMax provided the best convergence performance, showing the best learning curve and quickest strategy to reach the highest total rewards. Even though high deviations were observed during exploration, they gradually decreased to provide the most stable total rewards. A reason for this could be the temperature parameter, set to 2.0 to balance exploration and convergence efficiently. A higher temperature could have increased exploration but would have slowed convergence, whereas a lower value would have encouraged exploitation (Pack Kaelbling et al., 1996). Moreover, this strategy uses the least iterations and time to converge. As supported by Sutton & Barto (2018), this is likely caused by the use of a deterministic policy, as the distribution between actions during exploration is not uniform, encouraging action preferences to converge to the optimal policy in Figure 3.2 (c).

Final Policy and Value Function

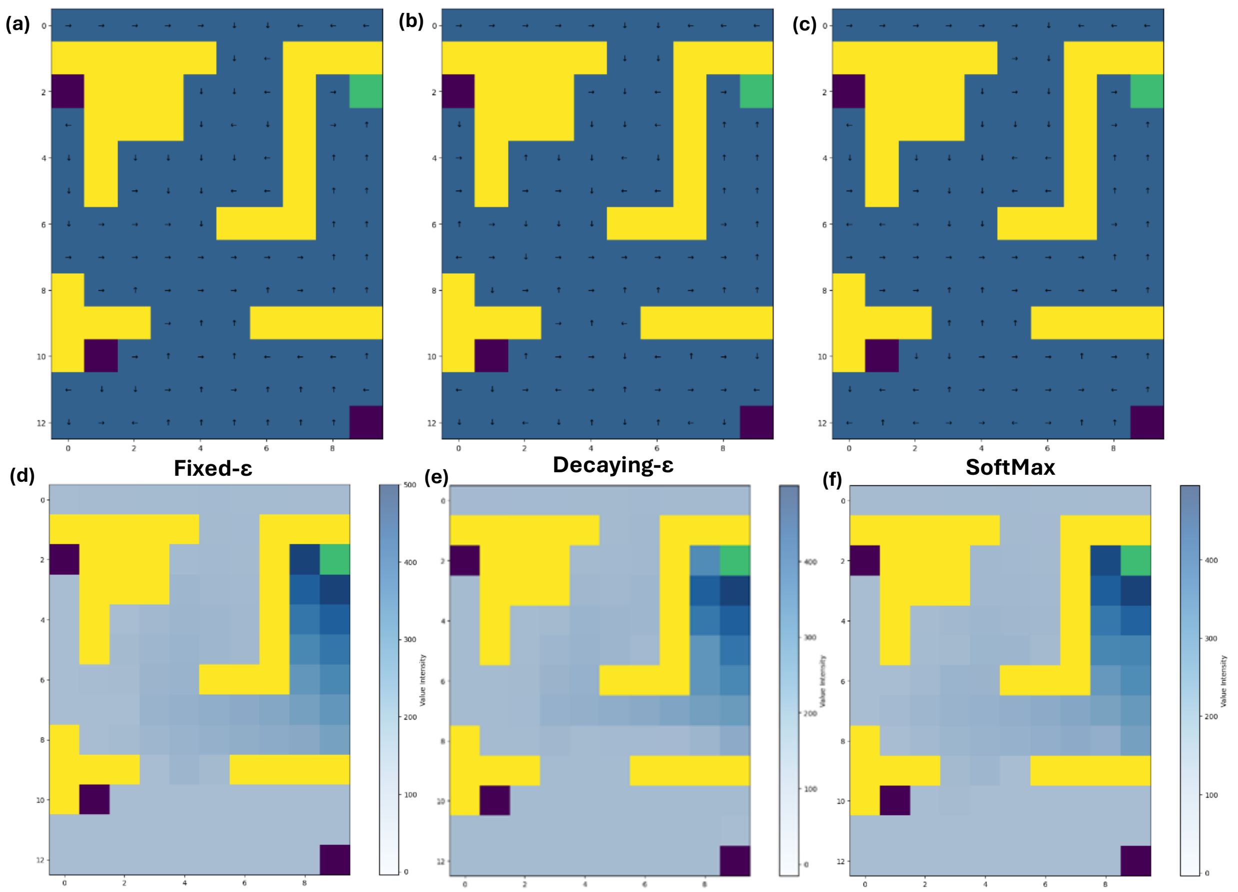

Figure 3.2 Final policy and value function graphs, respectively, for fixed-ε (a, d), decaying (b, e) and SoftMax (c, f).

Literature Table

Authors	Method	Key Findings	Implications
Thrun (1992)	Efficient exploration	Maintaining exploration can be beneficial in some environments	Excessive exploration, even though beneficial in dynamic environments, can lead to suboptimal policies
Pack Kaelbling et al. (1996)	Randomised strategies Measuring learning performance	• Decreasing T works well if best action isolated • An algorithm that tries to achieve optimality fast may incur large penalties during learning	• Lower T decreases exploration • Less aggressive strategy that takes longer is preferable
Sutton & Barto (2018)	Policy Approximation	• Action preferences can be parameterised • Epsilon greedy always probability of choosing random action • Decaying-epsilon improves policy performance by reducing exploration over time	• SoftMax performs best in achieving stable, high returns and convergence efficiency. • T can be reduced to approach determinism • Decaying-epsilon may be optimal for long-term improvement

Figure 3.3 Literature table of the resources used for the comparative analysis.

Sensitivity Analysis

With smaller number of episodes, the agent has limited time to explore the environment and learn optimal policies, which is unable to do due to suboptimal convergence, as seen in Figure 3.4 (a). This leads to a more unstable value function because the agent does not have enough time to retrieve data or accurately estimate the values. Moreover, the exploration induced by SoftMax during earlier episodes causes large fluctuations, as discussed earlier, resulting in a noisier value function. Figure 3.4 (b) supports this, as the total rewards are close to reaching 0 from a negative value, which is insufficient to converge to an optimal policy. The learning speed stays constant among episodes, as the Q-values and temperature are always the same, implementing the same exploration/exploitation strategy in every case. We can see this by observing that in Figure 3.4 (d), (e) and (f) the agent always converges after 75 episodes.

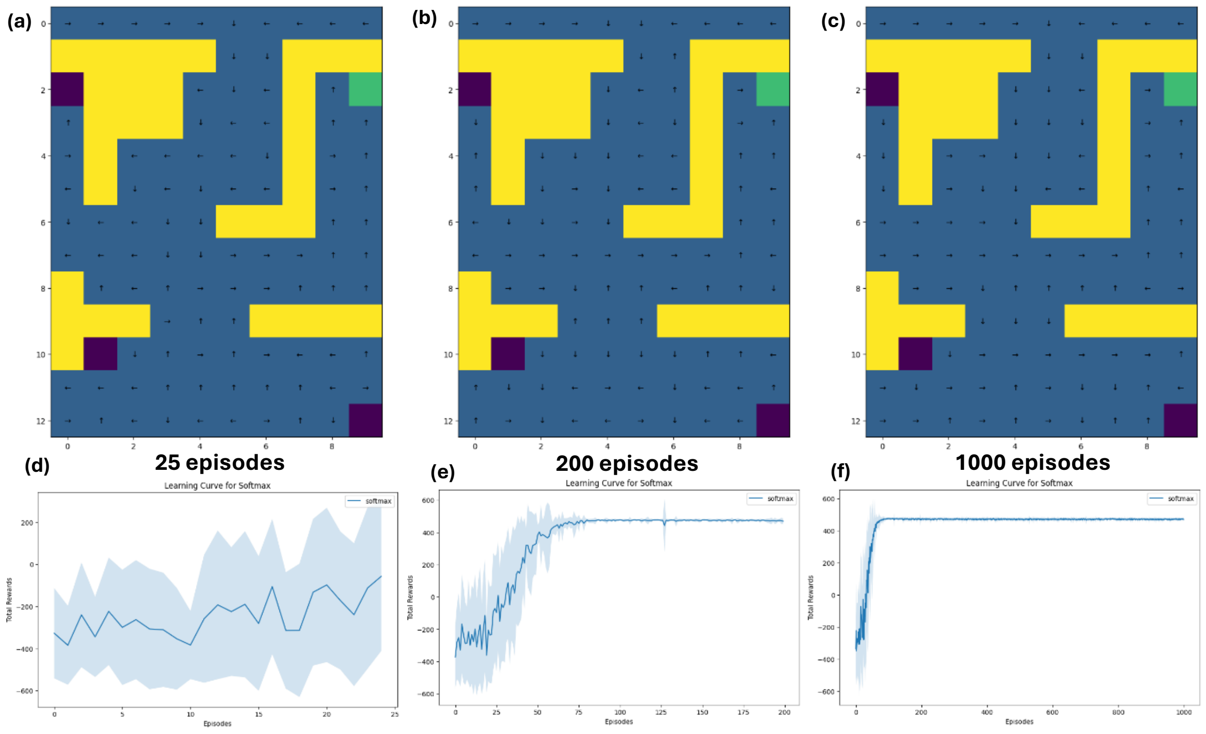

Figure 3.4 Sensitivity Analysis plots for the SoftMax approach after 25 (a, d), 200 (b, e) and 1000 (c, f) episodes.

With higher episode count, the agent is allowed and encouraged to explore more thoroughly, converging to an optimal policy. As shown in Figure 3.4 (e), the agent reaches a maximum total reward of 450, which plateaus and stays stable for, at least, 1000 episodes. Since this agent converges within 75 episodes, the policy and value function are also stable for 200 and 1000 episodes, decreasing the fluctuations as the episode count increases. The main observation is that, since the learning curve plateaus, the agent will not be able to converge to a more refined policy or value function closer to the optimal, as these are already near optimal.

Bibliography

Lobel, S., Gottesman, O., Allen, C., Bagaria, A., & Konidaris, G. (2022). Optimistic Initialization for Exploration in Continuous Control. www.aaai.org

Miller, T. (2023). *Policy Iteration*. Retrieved from Gibberblot: https://gibberblot.github.io/rl-notes/single-agent/policy-iteration.html

NG, A. (2021, October 25). *CS229 Lecture Notes*. Retrieved from Stanford : https://cs229.stanford.edu/notes2021fall/cs229-notes12.pdf

Pack Kaelbling, L., Littman, M. L., Moore, A. W., & Hall, S. (1996). Reinforcement Learning: A Survey. In Journal of Artiicial Intelligence Research (Vol. 4).

P. Singh, S., & S. Sutton, R. (1996). Reinforcement learning with replacing eligibility traces. In S. P. Singh, & R. S. Sutton, *Reinforcement learning with replacing eligibility traces* (pp. 123-158). Boston: Kluwer Academic Publishers.

Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction. In R. S. Sutton, & A. G. Barto, *Reinforcement Learning: An Introduction* (p. 590). Cambridge, MA: The MIT Press.

Szepesvari, C. (2009). *Algorithms for Reinforcement Learning.* Morgan & Claypool Publishers.

Thrun, S. B. (1992). Efficient Exploration In Reinforcement Learning.

Tokuc, A. A. (2024, March 29). *Value Iteration vs Policy Iteration in Reinforcement Learning*. Retrieved from Baeldung: https://www.baeldung.com/cs/ml-value-iteration-vs-policy-iteration#:~:text=In%20policy%20iteration%2C%20we%20start%20with%20a%20fixed%20policy.,iteration%20algorithm%20updates%20the%20policy.