2AMC15- Assignment 2 Report

***The GitHub repository for our project can be accessed here: Repository link.

1. Introduction

Autonomous navigation is a key capability for service robots operating in dynamic indoor spaces such as restaurants, hospitals, and offices. In this project, we investigate a simplified yet representative task: enabling a robot to locate (at least) a single target table inside a small-sized restaurant setting. This requires navigating between obstacles, avoiding walls and planning optimal routes to find its target. In addition, the agent is not aware of its target position. To simulate the restaurant environment, we use a continuous 2D space, including walls and obstacles and now the robot learns how to use information derived from its surrounding environment. The robot starts from a specific starting point (e.g., kitchen) and the episode ends successfully when the target has been reached. Due to the increased complexity of establishing a conitnuous environment, our experimental setup focuses two variants of Q-Learning known as Deep Q-Learning and Dueling Deep Q-Learning.

Related Work

Traditional map-based methods like SLAM and A* require accurate environment models and often struggle in dynamic or partially known settings. In contrast, reinforcement learning allows agents to learn directly from experience, adapting to local cues without needing global knowledge. Deep Q-Networks (DQN) (Mnih et al., 2015) are an extended Q-learning to continuous state spaces using neural networks, enabling robust policy learning even from high-dimensional input. Recent works have shown DQN's effectiveness in robot navigation under limited perception (Zhu et al., 2017; Marchesini and Farinelli, 2020). Inspired by these successes, we apply DQN to our restaurant scenario, where the agent must reach the target table based only on local sensor data. The use of a continuous state space with discrete actions aligns well with DQN's structure, making it a natural and tractable choice for this problem. To extend our baseline algorithm, we implemented Dueling DQN (D-DQN) (Wang et al., 2015). In continuous state spaces, similar states are common. D-DQN is suitable since it can generalize the value of states, independent of specific actions. This design decision improves the learning process, especially in our case where many actions have similar outcomes.

2. Problem Statement

We model the task as a Markov Decision Process (MDP) defined by the a 4-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, R)$:

- **State Space** $\mathcal{S} \subset \mathbb{R}^2$: The agent operates in a continuous space, where its position is represented by floating-point coordinates (x, y). The state includes the agent's current position. No global map or direct table location are provided

- **Action Space** $\mathcal{A} = \{\uparrow, \rightarrow, \downarrow, \leftarrow, \nearrow, \nwarrow, \searrow, \swarrow\}$: a discrete 8-directional action space (up, down, left, right, and diagonals), which better reflects realistic motion in an open environment compared to the traditional 4- directional scheme
- **Transition Probability function** $\mathcal{P}(s'|s, a)$: a transition function indicating the probability that action a will lead the agent form state s to state s'.
- **Reward function** $R(s, a)$: The Reward function function computes a reward based on the agent's behavior and surroundings. If the agent is absent, it returns a strong penalty of -10. A small step penalty of -0.2 discourages inactive actions. In case of failed movement adds -2, while collecting a target gives +20. Task completion yields back a large reward score of +1000. If a target is probably nearby, the agent gains +5; if an obstacle is near, it gets -2; and if neither is close but a target exists, it receives -1 for aimless wandering. The agent's recent positions are tracked, and revisiting the same location too often results in a loop penalty proportional to how frequently it occurs, reducing rewards for repetitive paths.

The state vector: is a 4 dimensional vector that represents the current state of the environment. Each element in the vector corresponds to the agent's normalized position, local obstalce density and a loop signal that indicates how ofter the agent revisits the same area.This state vector is then passed into the neural network of the models, giving it the necessary information to understand the environment and make decisions.

Markov analysis of the defined MDP

To further explain our model implementation, we describe all possible interactions between the robot and its environment, through discrete steps labeled as $t = 0, 1, 2, \ldots t$. At each step t, the agent observes the current state $s \in S$ of the environment $S = \{(x, y) \in \mathbb{R}^2\}$, which contains information, such as the position of the agent, (x, y) coordinates. Then, it chooses an action $a_t \in \mathcal{A}(s_t)$, and receives a reward $r_{t+1} \in \mathcal{R}$ considering possible action's effectiveness. Finally, the agent moves to the next state s_{t+1}. This environment satisfies the Markov property because the transition to the next state s_{t+1} depends on the current state s_t and action a_t, and not on the sequence of previous states or actions. The transition function $\mathcal{P}(s' \mid s, a)$ captures the outcomes of actions under uncertainty such as movement failure or wrong estimation of target's proximity. Although the state space is continuous, the policy and value function can still be learned or approximated under this MDP framework, e.g., using discretization or function approximation techniques. The reward structure introduces time-extended dependencies (e.g., penalties for loops in around safe spaces), but these are encoded into the reward function based on current state-action features, maintaining the formal Markovian structure. Thus, despite behavioral history being implicitly penalized, the model adheres to the requirements of a Markov Decision Process.

Design Justification

The Markov Property holds in this scenario, since all the needed information for the agent to make the optimal decision exists in the current state. Hence, this makes our problem a proper case of a finite MDP. The agent operates in a continuous state space defined by its position; it selects from a discrete set of movement actions; transitions are deterministic based on the agent's current positiom; and the reward function encourages efficient navigation by penalizing each step and rewarding when reaching the target.

3. Approach

3.1 Selected Method: Dueling Deep Q-Network (Dueling DQN)

This project tackles a navigation task for a delivery robot operating under partial observability, with a continuous state space and sparse rewards. To address these challenges, we model the environment as a MDP, where the state is represented by a vector encoding key features such as the agent's position and nearby context.
We implemented both DQN and Dueling DQN agents, selecting Dueling DQN as the primary method due to its improved stability and performance in partially observable, redundant environments(Wang et al. (2016)). Unlike standard DQN, which directly estimates Q-values, Dueling DQN separates state value and action advantage, enabling more efficient learning of valuable states(Hessel et al. (2018)). This structure enhances sample efficiency and robustness, particularly in low-dimensional tasks such as 2D navigation. While methods like PPO and A3C often require extensive tuning, and DDPG and SAC target continuous action spaces(Lillicrap et al. (2016); Haarnoja et al. (2018)), Dueling DQN is better suited for discrete control in compact state spaces, making it an effective choice for our setting.

3.2 Dueling DQN Architecture and Learning Objective

The standard Q-function gives the expected cumulative return for taking action a in state s, and following a policy thereafter:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t \,\middle|\, s_0 = s, a_0 = a\right] \tag{1}$$

Dueling DQN decomposes this Q-value into two parts. The V(s) stands for the Value of state s and A is the Advantage of doing action a while in state s. The Value of a state is independent of action. It means how good is it to be in a particular state

$$A(s,a) = Q(s,a) - V(s) \tag{2}$$

To resolve the problem pf the unidentifiability of $A(s,a)$ and $V(s)$ in (1), we can force the advantage function estimator to have zero advantage at the average advantage of all possible actions. In fact, the last module of the network implements the forward mapping Wang et al. (2016):

$$Q(s,a) = V(s) + \left(A(s,a) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a')\right) \tag{3}$$

This structure improves stability in situations where actions offer similar short-term rewards. We also use ϵ-greedy exploration and a bracket-based strategic reset mechanism when no success is detected for a fixed number of episodes.

Optimization Process in Dueling DQN

The optimization in Dueling DQN is based on minimizing the temporal difference (TD) error between the predicted Q-value and a target computed via the Bellman equation.

Loss Function

The loss function is Mean Squared Error (MSE) over a minibatch $\{(s_i, a_i, r_i, s'_i)\}_{i=1}^{N}$ is:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} (y_i - Q_\theta(s_i, a_i))^2 \tag{4}$$

The parameters θ are updated using gradient descent:

$$\theta^* \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\theta) \tag{5}$$

Experience Replay: To update the network, experiences gathered through interaction with the environment are stored in a fixed-size replay buffer. Each experience is represented as $e_t = (s_t, a_t, r_t, s_{t+1})$, where s_t is the state, a_t is the action, r_t is the reward, and s_{t+1} is the next state. Mini-batches of experiences are randomly sampled from the buffer to compute the temporal difference (TD) error and to optimize the model parameters using stochastic gradient descent.

3.3 DQN Overview and Comparison

Deep Q-Learning is a reinforcement learning algorithm that uses a deep neural network to approximate the Q-function, which determines the optimal action for a given state. The Q-function represents the expected cumulative reward of taking a specific action in a state and following a policy. The Q-function is updated iteratively, creating a Q-table with Q-values for each state-action pair. The input of the neural network will be the state or the observation and the number of output neurons would be the number of the actions that an agent can take. For training the neural network the targets would be the Q-values of each of the actions and the input would be the state that the agent is in.
Dueling DQN addresses this by introducing separate value and advantage estimation, allowing better discrimination of useful states. It improves convergence speed and final policy performance while maintaining low implementation complexity.

4. Empirical Study

Experimental Setup, Environments, and Hyperparameters

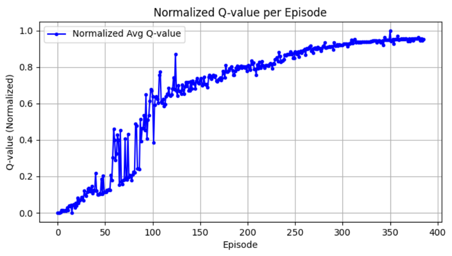
To evaluate our navigation agent, we designed two simulated environments: **env_easy** and **env_hard**. Both represent a 2D restaurant-like space with a single goal table and partial observability. The agent operates with 8 discrete movement actions and receives input as a continuous state vector. Each episode is bounded by a fixed number of steps.

We also introduce an optional experimental mechanism called **System Assist (SA)**, controlled via the `--sa` flag. When enabled, the environment resets every `br` episodes unless a successful run occurs within that bracket. This mechanism helps prevent overfitting to early failures and encourages more directed exploration.

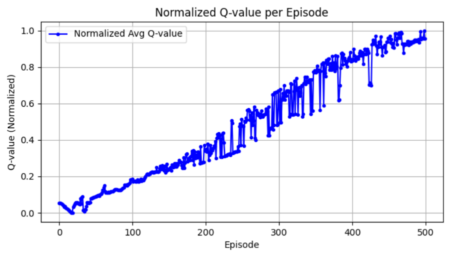
The default hyperparameter settings used across experiments were $\gamma = 0.95$, **learning rate** $= 1 \times 10^{-3}$, and **500 training episodes**. To explore model sensitivity and improve performance, we varied key parameters including the **learning rate** (e.g., 1e-4, 5e-3, 1e-2), **discount factor**(e.g., 0.7, 0.9, 0.99), **number of training episodes**(e.g., 250, 500, 700), **episode step limit**(e.g., 150, 200, 250), and **sa bracket size**(e.g., 5, 10, no-sa). Experiments were conducted in both the **easy** and **hard** environments. Agent performance was evaluated using **success rate** during training, **efficiency** (percentage deviation from the shortest unobstructed path), and **cumulative reward score** .

Validation and Convergence Analysis

In order to assess convergence of the algorithms, we employed two strategies. First, the algorithm logs the total reward and a success flag after each episode. We observed that, after a few episodes, the success rate stabilizes, indicating convergence in task performance. Second, we plotted the normalized average Q-value over time.



(a) Standard DQN Convergence



(b) Dueling DQN Convergence

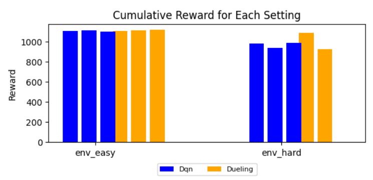
Figure 1: Comparison of Convergence Between Standard and Dueling DQN

Performance Comparison and Ablation Studies

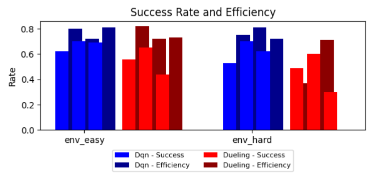
We evaluate the performance of our DQN agent across multiple metrics and compare it against a DuelingDQN agent. To understand the contributions of individual components, we perform ablation studies. Removing the loop penalty resulted in significantly more repetitive agent behavior, ultimately lowering overall success rates. Additionally, simplifying the reward function by excluding proximity bonuses resulted in greedier and less consistent behavior. These findings suggest that our reward shaping and state design effectively balance guidance and flexibility, enabling stable and efficient learning without overconstraining the agent.

Situational Analysis and Robustness

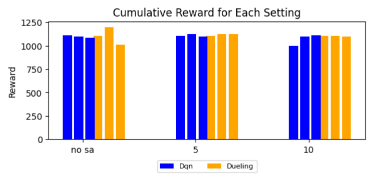
To evaluate the robustness of the models, we perform controlled experiments where one parameter is varied while others remain fixed. Each setting is repeated three times to ensure consistency. subcaption



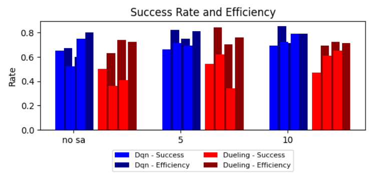
(a) Reward (env)



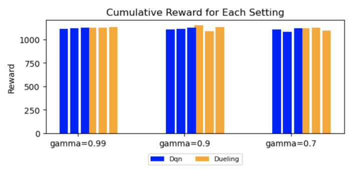
(b) Success Rate (env)



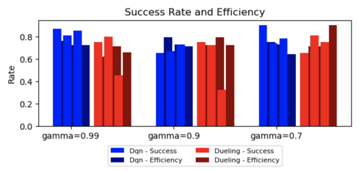
(c) Reward (sa)



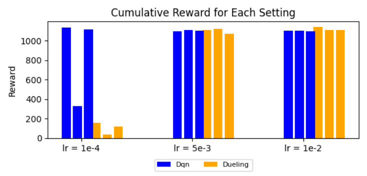
(d) Success Rate (sa)



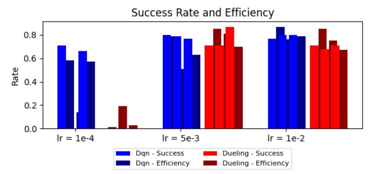
(e) Reward (γ)



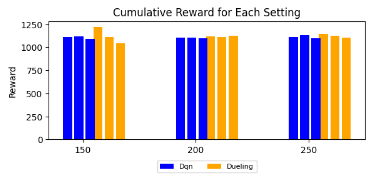
(f) Success Rate (γ)



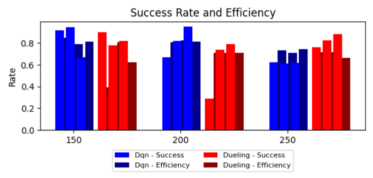
(g) Reward (learning rate)



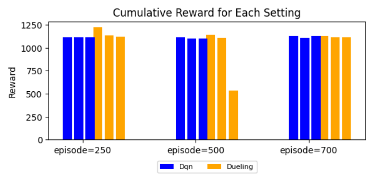
(h) Success Rate (learning rate)



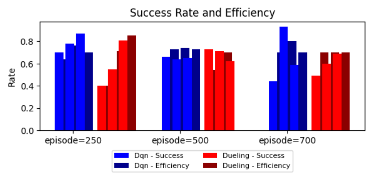
(i) Reward (steps)



(j) Success Rate (steps)



(k) Reward (# episodes)



(l) Success Rate (# episodes)

Figure 2: Performance comparisons across different hyperparameter and environment settings. Each row shows **reward** (left) and **success rate** (right).

Statistical Confidence and Final Results

Each experimental configuration was run three times to assess consistency. Although no formal averaging was performed, reward trajectories and success rates across trials showed similar patterns, suggesting stable learning.

Our experiments showed that the more challenging environment posed a significantly greater difficulty for the agents. Low learning rates led to slower or suboptimal convergence, although most reward values remained consistent across runs, indicating that both algorithms were generally able to master the environment. Although the dueling DQN is designed to offer performance gains, it did not outperform the standard DQN in our experiments. A possible explanation is that the environment was too simple or the training period too short for its architectural strengths to take effect. Prior work has demonstrated that the advantages of Dueling DQN become more apparent in environments where the relative value of actions differs substantially across states (Wang et al., 2016). In contrast, in environments with low action-value variance, such as the one used in this study, standard DQN may perform comparably or even better.

In the standard environment, changes to hyperparameters such as the discount factor, training duration, and episode limit had minimal impact. Most configurations resulted in reliable convergence. In the harder variant, however, performance was more sensitive to these settings, and longer training consistently improved results. System Assist (SA) did not influence the final policy directly but helped agents converge faster and achieve higher success rates in settings with fewer training episodes.

5. Conclusion

In this project, we investigated a navigation task for service robots in a simplified restaurant environment using deep reinforcement learning. The problem was formulated as a Markov Decision Process (MDP) with a continuous state space and a discrete action space, emphasizing partial observability and sparse rewards. We implemented both DQN and Dueling DQN agents, initially experimenting with PPO before favoring Dueling DQN due to its more stable convergence and better alignment with the task's characteristics.

However, several limitations were identified during the experiments. The simplicity of the environment likely constrained the performance of the Dueling DQN, which is designed to take advantage of value-function decomposition in more complex scenarios. Additionally, the current state representation lacks informative features that would help the agent better interpret the environment, and the reward function remains relatively basic. Future work should focus on developing a more sophisticated environment with greater variability, as well as enhancing the state vector and reward design to support richer learning signals and better generalization.

Despite these limitations, the strengths of reinforcement learning are evident in tasks like kitchen navigation and object collection. Unlike traditional machine learning methods, which often require extensive labeled datasets and retraining when faced with even minor changes, RL agents learn directly through interaction. This enables them to generalize more effectively to new or changing environments, making RL a powerful tool for real-world robotic applications.

References

T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1861–1870, 2018. URL `https://openreview.net/pdf?id=HJjvxl-Cb`.

M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2016. URL `https://arxiv.org/pdf/1509.02971.pdf`.

E. Marchesini and A. Farinelli. Discrete deep reinforcement learning for mapless navigation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 46–55, 2020. doi: 10.1109/ICRA40945.2020.9196893.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015. URL `https://arxiv.org/abs/1511.06581`.

Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1995–2003. PMLR, 2016.

Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3357–3364, 2017.

Individual Contributions

- **Thanos Vaios**: Problem formulation,developed the baseline enviroment, developed DQN agent,developed the sa logic, performed sa and env experiments, developed a py script to plot the results, implemented the metrics worked on various parts of the report(Approach, Empirical Study, Conclusions, optimized reward function and state vector through experimentation for the DQN agent, consulted others. Simulating the environment for experimental results. Added some comments and cleaned the code.
- **Nikolaos Louridas**: Suggested problem formulation idea,developed the environment, partially developed DQN agent, worked on the reward function and take action. Developed the Dueling DQN agent, that we used in the assignment. Attempted to implement PPO agent, (without having good performance, we decided not to use it). Simulating the environment for experimental results. Created README file. Worked on the report (Introduction, Related Work, Problem Statement, Markov analysis and Approach). Added comments and cleaned part of the code.
- **Matthijs van Vroenhoven**: Problem formulation ideas, initial report set-up, provided an initial DQN agent, wrote partially report sections for introduction, problem statement and conclusion
- **Cezar Suciu**: Problem formulation ideas. Tried to build the environment, but we later switched to a simpler one. Implemented a PPO agent, but it didn't perform well, so we didn't use it. Helped with parts of the experimental implementation for the report.
- **Dan Liu**: Problem formulation ideas, discussed environment setup feedback, contributed an initial version of a converging PPO agent and participated in hyperparameter tuning, Helped with parts of the experimental implementation for the report, initial report set-up, wrote partial report section for the approach.
- **Reuben Roy**: Problem formulation ideas, developed duelling dqn agent, hyperparameter tuning for the developed agent (the agent still performed poorly, hence decided not to use it).
- **Nouha Bannour**: Contributed to the discussion of the problem formulation. Helped optimize the Dueling DQN agent by tuning hyperparameters and refining the reward setup. Assisted with running the experimental setup. Reviewed final code. Wrote the Statistical Confidence and Final Results section of the report.