Reinforcement Learning Final Project

Reichman University, 2024

Introduction

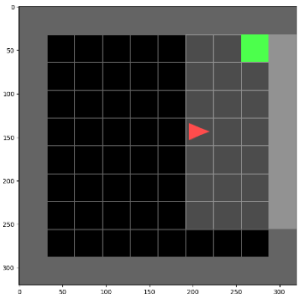
This report presents the comprehensive work and findings of the final project for the Reinforcement Learning course, focusing on solving two variations of the MiniGrid environment using Deep Reinforcement Learning (Deep RL) techniques. The MiniGrid environment presents a lightweight, 2D grid-world setting with goal-oriented tasks, challenging an agent to solve different maze maps and interact with various objects such as doors and keys. Unlike the mid-semester project, this final assignment emphasizes pixel-based observations, adding an additional layer of complexity and requiring more sophisticated approaches.

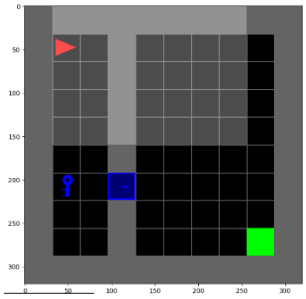
Environment Description

Two distinct MiniGrid environments formed the basis of our exploration:

1. **RandomEmptyEnv_10**: An empty grid environment where the goal is to reach a specific location.

2. **RandomKeyMEnv_10**: A more complex environment where the agent must find a key to open a door before reaching the goal.





Both environments feature a triangle-like agent operating within a discrete action space, showcasing the challenges of navigating and interacting within pixel-based, goal-oriented tasks.

Methodology

Deep Reinforcement Learning Algorithms

Our approach involved implementing and comparing four major variations of Deep Q-Networks (DQNs):

1. **Standard DQN**: The foundational model, utilizing a simple Q-network to estimate the action-value function.

2. **Double DQN**: An enhancement over DQN, addressing the overestimation bias by decoupling selection and evaluation of the action in the Q-update.

3. **Dueling DQN**: Introduces a novel network architecture that separately estimates state values and action advantages, facilitating more efficient learning.

4. **Dueling Double DQN**: A combination of Double DQN and Dueling DQN, aiming to leverage the advantages of both methods.

Reward Shaping Strategy

Optimizing the reward system was essential for guiding the agents through their tasks in both environments, enhancing their learning efficacy and performance. The strategies deployed for each environment were carefully designed, considering their unique challenges.

Environment 1: RandomEmptyEnv_10

In the **RandomEmptyEnv_10** environment, the agent's goal is to navigate to a predetermined location efficiently. The reward structure was formulated to encourage this behavior:

- **Penalty for Each Step**: A (-1) penalty for every step was introduced. This approach incentivizes the agent to find the shortest path to the goal, thus minimizing the number of steps and encouraging efficient exploration.

- **Reward for Reaching the Goal**: Completing the task rewards the agent with (+100). This significant increase from an initial reward of (+1) for reaching the goal considerably accelerates the learning process, as it clearly defines the primary objective for the agent.

This reward system significantly improved learning speeds compared to an approach that offered a minimal reward for reaching the goal. The combination of step penalties and a substantial completion reward proved effective in guiding the agent towards optimal performance.

Environment 2: RandomKeyMEnv_10

The **RandomKeyMEnv_10** presents a more complex scenario, where the agent must acquire a key, unlock a door, and then find its way to the goal. The reward strategy here was more nuanced:

- **Minor Penalty for Each Step**: To discourage inefficient exploration without stifling necessary exploration due to the environment's increased complexity, each step incurs a (-0.1) penalty.

- **Rewards for Key Interactions**: Picking up the key and opening the door each grant the agent a (+10) reward. These positive reinforcements emphasize the importance of these actions towards achieving the overall objective.

- **Penalties for Counterproductive Actions**: To deter behaviors that hinder task progress, actions like dropping the key or closing the door result in a (-10) penalty.

- **Completion Reward**: Successfully reaching the goal after completing the prerequisites rewards the agent with (+100), aligning with the ultimate task objective.

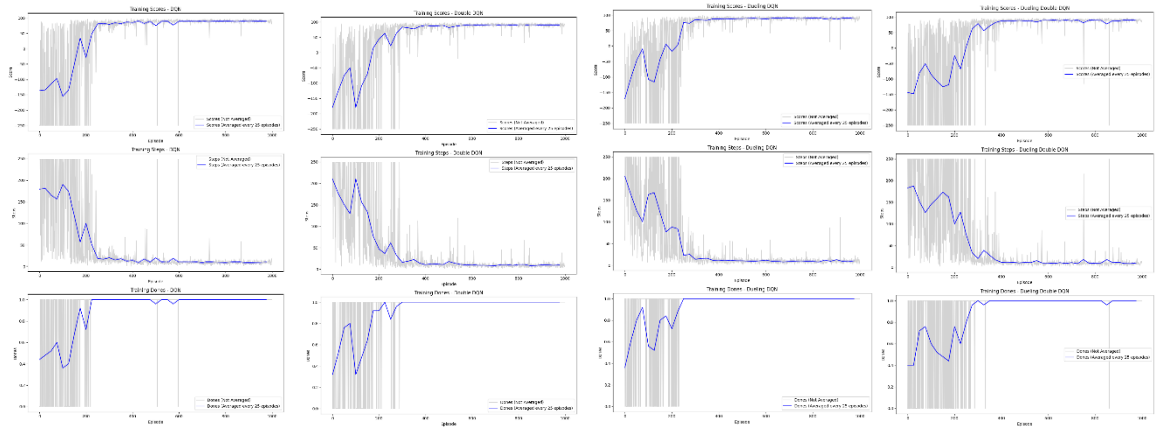
This refined reward shaping approach for the **RandomKeyMEnv_10** significantly guides the agent through the sequential steps necessary to complete the task efficiently. By carefully balancing rewards and penalties associated with key task milestones and actions, the strategy promotes effective learning and task-focused behavior in more complex scenarios.

Overall, the tailored reward systems for each environment were pivotal in steering the agents towards success, demonstrating the importance of a well-considered reward structure in the training and development of efficient, goal-oriented behaviors in reinforcement learning tasks.

Training and Evaluation

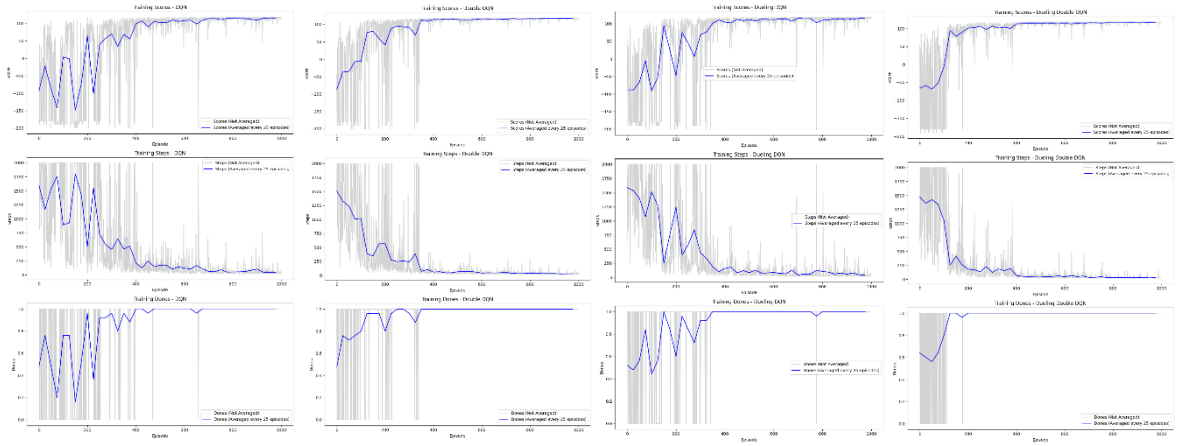
For both environments, the training regimen was tailored to accommodate the distinct challenges and complexity each presented. Over the course of 1,000 episodes, agents were rigorously trained, with variations in the maximum number of steps per episode to better suit the specific environment:

- **RandomEmptyEnv_10**: Due to its relatively simpler nature, a maximum of 250 steps per episode was set. This boundary encourages the agent to learn efficient paths to the goal within a constrained environment.



Training logs (scores, steps, dones) for DQN, Double-DQN, Duel-DQN, Duel-Double-DQN

- **RandomKeyMEnv_10**: Reflecting its increased complexity, the training for this environment allowed up to 2,000 steps per episode. This extension affords the agent ample opportunity to explore and solve the more intricate puzzles presented, such as key retrieval and door unlocking.



Training logs (scores, steps, dones) for DQN, Double-DQN, Duel-DQN, Duel-Double-DQN

Throughout the training phase, the learning process was closely monitored through three principal metrics: scores (cumulative rewards), steps (taken per episode), and dones (rate of task completion). These metrics offered valuable insights into each agent's learning efficiency and behavior, highlighting progress and areas for improvement.

Evaluation for each trained model was conducted over 50 episodes to reliably assess their performance. By averaging the scores, steps, and completion rates across these episodes, a comprehensive overview of each model's effectiveness and consistency was obtained, allowing for an informed comparison of their capabilities.

An epsilon-greedy strategy was employed for action selection in both environments to balance exploration and exploitation. Initially set to 1, epsilon gradually decreased to a final value of 0.05, with the decay rate adjusted to align with the specific demands of each environment:

- **RandomEmptyEnv_10**: A decay rate of 0.995 was chosen, facilitating a relatively quick convergence to optimal actions in this less complex setting.

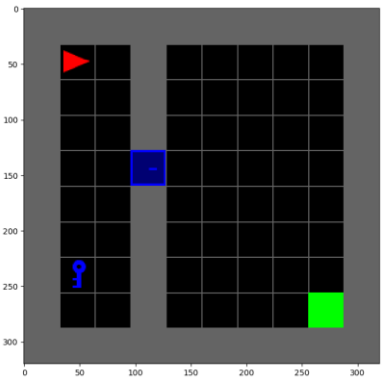
- **RandomKeyMEnv_10**: A slower decay rate of 0.997 was applied, accounting for the need for more extensive exploration to navigate the environment's additional complexities effectively.

This careful calibration of the epsilon decay rates ensured that each agent had a suitable balance between exploring new strategies and exploiting known successful behaviors, optimizing their learning trajectory within the constraints of each unique environment.

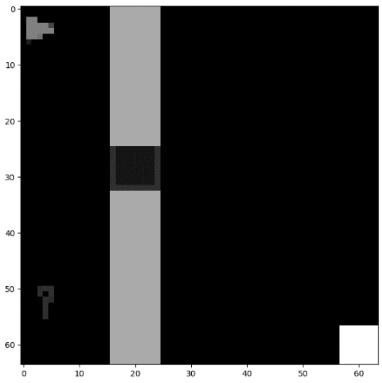
Implementation Details

Preprocessing

To ensure the neural networks could effectively interpret the pixel-based observations from the MiniGrid environments, a preprocessing method was applied to the raw pixel inputs. This method involved several steps aimed at simplifying and standardizing the input data:



Before Preprocessing



After Preprocessing

- **Edge Cropping**: The outer edges of the image were cropped to remove extraneous parts of the environment that hold little value for the learning process. The crop size was set to 35 pixels from each edge.

- **Grayscale Conversion**: The cropped image was converted to grayscale to reduce the complexity of the input, as color information is not critical for the tasks at hand.

- **Normalization**: Pixel values were normalized to a range of [0, 1] by dividing by 255.0, improving the stability and efficiency of neural network training.

- **Grid Line Removal and Resizing**: The image was resized to 125x125 pixels using nearest neighbor interpolation to remove the grid lines, which could introduce unnecessary noise into the learning process. Subsequently, it was downscaled to a 64x64 resolution to maintain computational efficiency without sacrificing necessary detail.

- **Channel Dimension Addition**: A channel dimension was added to the processed image, formatting it correctly for input into the convolutional neural network layers.

Replay Buffer

A Replay Buffer was implemented to store and replay past experiences, allowing the agents to learn from a diverse sample of past actions. This method helps to break the correlation between consecutive samples, leading to more stable and effective learning. The buffer:

- Stores experiences as tuples of (state, action, reward, next_state, done).

- Has a capacity limit, with old experiences being replaced by new ones once the capacity is reached.

- Samples a random batch of experiences for the learning process, facilitating mini-batch gradient descent.

Q-Network

The Q-Network, designed as a convolutional neural network (CNN), takes preprocessed state images as input and outputs Q-values for each possible action. The network comprises three convolutional layers followed by two fully connected layers:

- **Convolutional Layers**: Extract and learn features from the input state. The first layer uses a 8x8 kernel with a stride of 4, the second a 4x4 kernel with a stride of 2, and the third a 3x3 kernel with a stride of 1.

- **Fully Connected Layers**: Map the extracted features to the action space, providing a Q-value for each action. The first layer outputs 512 features, which are then passed to the second layer that maps to the number of possible actions.

Dueling Q-Network

An extension of the standard Q-Network, the Dueling Q-Network separates the estimation of state values and action advantages. This architecture allows the network to learn which states are valuable without having to learn the effect of each action for each state, enhancing learning efficiency. The network splits into two streams after the convolutional layers:

- **Value Stream**: Estimates the value of a given state.

- **Advantage Stream**: Estimates the advantage of taking a particular action in the given state.

- The final Q-values are computed by recombining the value and advantage streams.

DQNAgent

The DQNAgent manages the interaction with the environment, the learning process, and the updating of Q-Network parameters. It supports variations including standard DQN, Double DQN, Dueling DQN, and Dueling Double DQN through its design:

- **Epsilon-Greedy Policy**: Determines the balance between exploration and exploitation, gradually shifting from exploring the environment to exploiting learned strategies.

- **Learning from Experiences**: Utilizes batches of experiences sampled from the Replay Buffer to update the network weights based on the loss between predicted Q-values and target Q-values.

Learning Methods

- **learn_DQN**: In the standard DQN approach, the agent updates the Q-Network by minimizing the loss between the predicted Q-values and the target Q-values, where the target is based on the maximum predicted reward for the next state as estimated by the target network.

- **learn_Double_DQN**: To address the overestimation bias present in DQN, the Double DQN method uses the local network to select actions and the target network to evaluate their value. This approach decouples the action selection from the evaluation, resulting in more accurate value estimates.

These components work together to form a robust system for training agents in complex, pixel-based reinforcement learning environments. By combining preprocessing, a replay mechanism, advanced neural network architectures, and sophisticated learning algorithms, the agents can efficiently learn optimal strategies for navigating and interacting within the MiniGrid environments.

Hyperparameter Configuration

The optimization and performance of our Deep Reinforcement Learning models were critically dependent on the careful selection and tuning of hyperparameters. The configuration employed across all variations of the Deep Q-Network (DQN) models was derived from empirical testing and literature benchmarks to ensure both stability in learning and robustness in performance. The following hyperparameters were standardized for uniformity across all model variants to facilitate a fair and consistent comparison:

- **Learning Rate (`lr`)**: Set to 0.001 (1e-3), this parameter controls the rate at which our models adjust their weights in response to the estimated error. A learning rate that is too high may lead

to erratic learning, while too low a rate may result in prolonged training times without significant improvement in performance.

- **Discount Factor (`gamma`)**: Assigned a value of 0.99, the discount factor quantifies the importance of future rewards compared to immediate ones. A high value encourages strategies that optimize for long-term gain, crucial for the complexity and depth of decision-making in the MiniGrid environments.

- **Soft Update Rate (`tau`)**: Set to 0.01 (1e-2), this parameter facilitates the gradual blending of the parameters from the local Q-network to the target Q-network. This soft update mechanism promotes stability in learning by ensuring that updates to the target network are incremental and do not introduce sudden shifts in policy.

- **Replay Buffer Size**: Configured at 100,000 (1e5) for the Key Environment and 10,000 (1e4) for the Empty Environment, reflecting the relative complexity and memory requirements of each task. A larger buffer in the more complex environment allows for a more diverse set of experiences to learn from, enhancing the agent's ability to generalize and adapt to new scenarios.

- **Batch Size**: A batch size of 64 was chosen to balance the computational efficiency with the benefits of mini-batch gradient descent. This size is sufficient to provide a representative sample of experiences, facilitating effective learning while maintaining manageable computation times.

- **Update Frequency**: Scheduled every 4 steps, this frequency strikes a balance between too frequent updates, which may lead to overfitting on recent experiences, and too infrequent updates, which could slow down the learning process.

- **Epsilon Decay Rate:** Customized to the specific challenges of each environment, with a rate of 0.995 for the RandomEmptyEnv_10 and 0.997 for the RandomKeyMEnv_10. This fine-tuning of the exploration-exploitation balance is critical, with a more gradual decay in the complex Key Environment to encourage thorough exploration of the intricacies it presents.

- **Maximum Steps per Episode:** Set to 250 for the RandomEmptyEnv_10 and 2000 for the RandomKeyMEnv_10, these limits reflect the differing scales of complexity and exploration requirements. The constrained step count in the simpler Empty Environment fosters efficiency and precision in navigation, while the extended limit in the Key Environment accommodates the necessity for detailed exploration and interaction with objects.

This hyperparameter regime was established to optimize the trade-offs between learning speed, stability, and performance. It was consistently applied across all DQN model variants to ensure that any observed differences in performance could be attributed to the model architecture and learning algorithm rather than disparities in the learning environment.

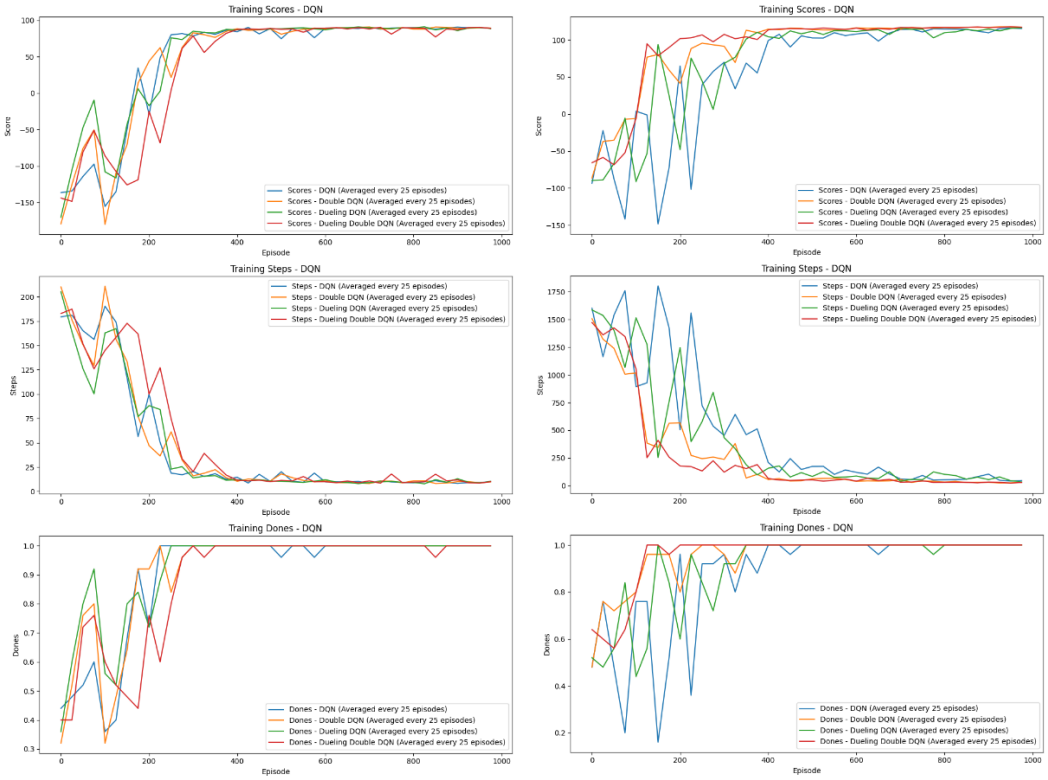
Results and Discussion

The analysis of the performance of the four Deep Q-Network (DQN) variants across the two distinct MiniGrid environments provides a detailed understanding of how each model's nuances influence learning efficacy and task completion. Graphical representations of scores, steps, and completion rates (dones) during both the learning and evaluation phases have been instrumental in elucidating these differences.

Learning Phase Analysis

In the straightforward RandomEmptyEnv_10 environment, convergence was achieved at a similar mark of around 300 steps by all DQN variants. This consistency across models indicates a baseline proficiency when facing less complex tasks. The RandomKeyMEnv_10, however, delineated the models based on their complexity handling capabilities. Both Double-DQN and Dueling-Double-DQN reached convergence notably faster, around 400 episodes, underscoring the critical role of the Double algorithm in stabilizing and accelerating the learning process. The Standard DQN and Dueling DQN, taking over 700 episodes to converge, highlight the challenges faced without the Double modification when addressing more demanding environments.

Simple Env



Key Env

Evaluation Phase Insights

In RandomEmptyEnv_10, negligible differences were seen among the models during evaluation, with each completing the task in about 8 steps. The differentiation became pronounced in the RandomKeyMEnv_10:

- **Standard DQN:** Struggled with higher average steps (354), lower scores (44), and completion rate (0.66).

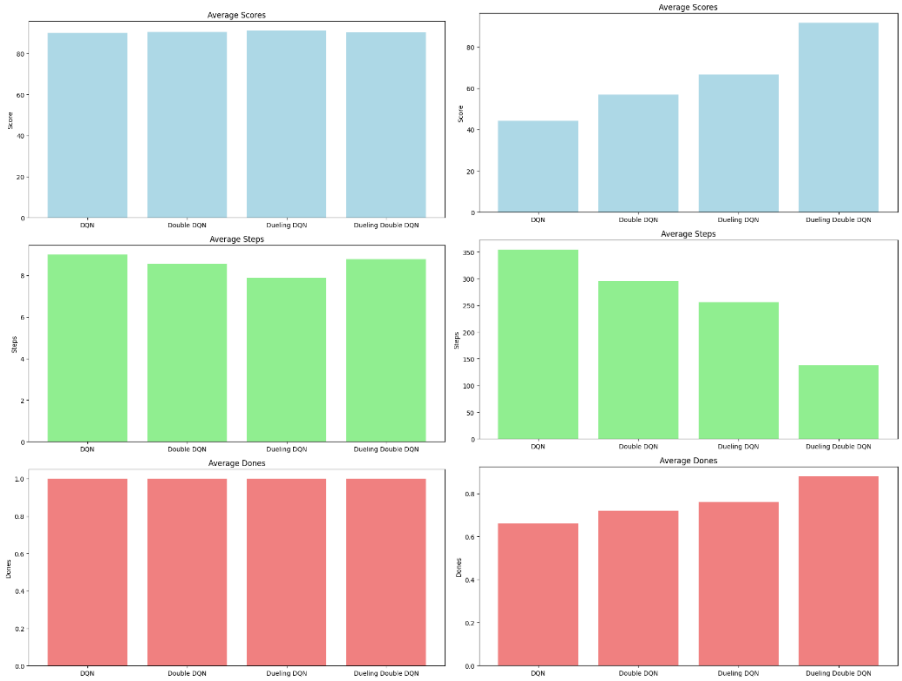
- **Double DQN:** Improved performance, reducing steps (295), increasing scores (56), and completion rate (0.72).

- **Dueling DQN:** Further gains with fewer steps (255), higher scores (66), and completion rate (0.76).

- **Dueling Double DQN:** Most effective with significantly fewer steps (138), the highest scores (91), and completion rate (0.88).

These results vividly demonstrate that while all models can learn simple tasks, complex problems require more advanced approaches for efficient and successful completion.

Simple Env



Key Env

Concluding Remarks

The empirical evidence from our project solidly positions the Double-DQN and Dueling-Double-DQN as superior contenders in challenging environments, showcasing their capability to efficiently handle complex scenarios with discernible adeptness. This study also emphasizes the significance of well-calibrated reward shaping strategies; the correct incentivization proved crucial for guiding agent behavior towards successful task completion and expediting the learning process. Moreover, the sensitivity to hyperparameters was palpable in our experiments—minor adjustments had substantial impacts on learning dynamics and agent performance, signifying the importance of meticulous hyperparameter tuning. Ultimately, the findings reiterate the essence of a nuanced approach in reinforcement learning, combining advanced algorithmic strategies with precise reward structures and hyperparameter configurations, to cultivate agents that are not only competent but also proficient in their operational environments.