Reinforcement Learning with Deep-Q Learning in Grid-Worlds

Abhinav Jain, Sumukh Bharadwaj Koride
Department of Computer Science, Rice University
aj70@rice.edu, sk160@rice.edu

Abstract

Traditional Reinforcement Learning involves an agent interacting with an environment to learn through a reward-based system. In this study, we investigate the performance of Deep Q-Learning in a grid world game. Our goal is to assess the scalability, robustness, and generalization of Deep Q-Learning in environments of varying complexity. We conduct a set of experiments to evaluate the agent's ability to learn to play the game and solve it. Through these experiments, we aim to gain a deeper understanding of the efficacy of Deep Q-Learning in the context of grid world games. Our findings demonstrate the importance of hyperparameter tuning and reward engineering in achieving optimal performance in Deep Q-Learning for grid world games. Our code is publicly available at Github.

1. Introduction

Reinforcement Learning has proven to be a powerful tool for learning in various applications, such as robotics, autonomous-driving and game-playing. It has been around for quite some time, where an agent learns through feedback. Mostly, an agent gets a reward whenever it accomplishes a task or punished otherwise. In this project, we apply fundamentals of Markov Decision Process in classic Reinforcement Learning.

For our project, we are interested in teaching an agent to navigate a simple grid world to reach a goal state. We use Deep Q-learning [2], which uses trial and error to learn the optimal policy. The agent interacts with the environment by taking actions and receiving rewards based on its performance. The agent's policy is updated using a neural network that approximates the Q-value function, which represents the expected reward for taking a particular action in a particular state. Through experience, the agent will learn the optimal policy that maximizes the expected cumulative reward over time. Deep Q-learning is particularly useful when an optimal policy is not available or when solving complex grid world problems, where the discrete state-action space can be extremely large.

For the purpose of detailed study within the scope of this project, we have defined grid worlds of sizes 3x3 and 5x5 in the Karel Environment [3]. In these environments, the agent must navigate through a maze-like structure, collect markers and reach a goal state while avoiding obstacles. Deep Q Learning allows the agent to learn which actions to take in each state to maximize its rewards and ultimately reach the goal state.

However, there are challenges associated with applying Deep Q Learning in grid world environments. One such challenge is the issue of exploration vs. exploitation, as the agent must balance the need to explore new areas of the environment with the need to exploit its current knowledge to maximize rewards. Another challenge is the risk of overfitting, where the agent becomes too specialized to the training environment and is unable to generalize to new environments. We showcase that proper hyperparameter tuning and reward function design are critical for achieving optimal performance in Deep Q Learning.

2. Model

Background: We represent the Agent behaviour in the Grid World as a Markov Decision process to model decision-making processes in a sequential manner, where the outcome of an action is probabilistic and only influenced by the current state of the environment. MDP is characterized by the tuple $\{\mathbf{S}, \mathbf{A}, \mathbf{P}, \mathbf{R}, \gamma, T\}$ where $\mathbf{S} = I^{H*W*D}$, $I \in \{0, 1\}$ is the many-hot state space representing the grid world; $\mathbf{A}$ is the action space comprising of a set of valid actions from the Karel Vocabulary; $\mathbf{P}$ is the transition probability $p(s'|s, a)$ which in our case is assumed to be deterministic; $\mathbf{R}$ is the reward function; γ is the discount factor and T is the MDP horizon.

Deep Q-Learning: In Deep Q-Learning, the state is represented as an input vector s_t, the action as a function of the state $a_t = Q(s_t, \theta)$, and the optimal policy as a function of the state $\pi(s_t) = argmax_a Q(s_t, a; \theta)$. The Deep Q-Learning algorithm involves learning the optimal Q-values through the Bellman equation 1 and updating the weights of the neural network using a Mean-Squared Bellman Error (MSBE) based loss function 2.

$$Q^*(s,a) = \underset{s' \sim P}{E}\Big[r(s,a) + \gamma \max_{a'} Q^*(s',a')\Big] \quad (1)$$

$$\mathcal{L}(\theta, \mathbf{D}) = \underset{(s,a,r,s',d)\sim \mathbf{D}}{E}\Bigg[\Big(Q_\theta(s,a) - \Big(r + \gamma(1-d)\max_{a'} Q_\theta(s',a')\Big)\Big)^2\Bigg] \quad (2)$$

where, (i) $d \in \{0, 1\}$ is the done signal marking whether the transition ended the episode i.e. s' is the terminal state and (ii) $\mathbf{D}$ is the off-policy Experience Replay Buffer.

3. Experiments

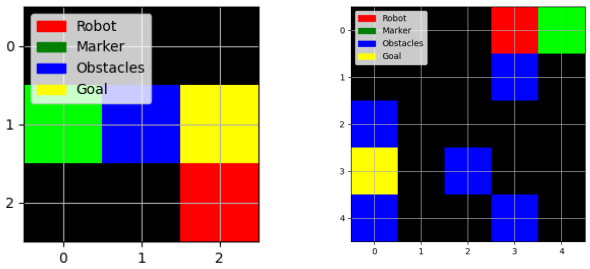
To study Deep Q-learning, we design a set of experiments that explore its performance on various aspects of the task of navigating a grid world to reach a goal state. Here are a few experiments we conduct:

Scalability: We test the scalability by increasing the size and complexity of the grid worlds from 3x3 to 5x5. This will test how well the approach can handle larger and more complex environments.

Exploration vs Exploitation: We compare the performance by varying the balance between exploration and exploitation.

Generalization: Finally, we test the generalization ability by evaluating the performance through success rate and average episode length on a set of unseen grid worlds.

3.1. Data



(a) 3x3 Grid World (b) 5x5 Grid World

Figure 1: Karel Grid Worlds of different sizes marked with grid positions for robot, marker, obstacles and goal

The Karel domain is a popular domain for testing reinforcement learning algorithms. It is a simple, two-dimensional world where an agent, represented by a robot named Karel, moves around and performs actions to accomplish specific tasks. The agent has a set of sensors that allow it to perceive its environment, such as the position of walls, markers, and other objects. The agent also has a set of actions it can perform, such as moving forward, turning left or right, picking up or putting down markers, and checking for walls or other obstacles. One of the advantages of using the Karel domain for these experiments is that it allows us to control the complexity of the environment by changing the size of the world, the number of markers, and the position of the obstacles. This makes it easier to compare the performance of algorithms under different conditions.

To simplify the problem, our task involves an agent that is required to select a single marker and transport it to a designated goal position. The environment is represented by a 3-dimensional grid with dimensions HxWxD, where the depth is a 4-dimensional attribute indicating the occurrence of the agent, obstacle, marker, and goal positions in the grid. The action space is simplified to four directions, namely Up, Down, Right, and Left, by removing actions related to picking up or putting down markers. The agent is designed to automatically pick up a marker when it arrives at the marker's position and drop it when it reaches the goal position.

3.2. Implementation Details

We base our Deep-Q implementation using widely used Stable Baselines library.

Environment Parameters: We define the following parameters to control the complexity of environment:

- World Size $(= H = W)$: The size of the square-grid
- Max Obstacle Occupancy $= o$: max fraction of total grid cells obstacles can occupy. We set it to 0.4
- Environment type $\in$ {*fixed*, *random*}: Instantiating initial state s_0 of the environment by either keeping the obstacles fixed or randomly shuffling them. In each setting, the positions of the marker, agent and the goal are randomised.
- Horizon T: Maximum time steps the agent is allowed to run, in other words maximum episode length. For 3x3, we set it to 10 and for 5x5, we set it to 25 as determined by the time taken by the expert policy across 'random' grid-world environments.

Rewards: We reward/control the behaviour of the agent in the following manner:

- $r_{obstacle}$: penalise the agent for hitting an obstacle or wall with -1 and terminate
- r_{pickup} : reward for reaching marker position $(= 0.5)$
- r_{goal} : reward the agent with $+1$ for reaching the goal with the marker and terminate

- $r_{invalid}$: penalise the agent with -1 for reaching goal w/o marker and terminate
- r_{step} : optionally set to -0.05 to encourage agent actions that reach the goal in least number of time-steps

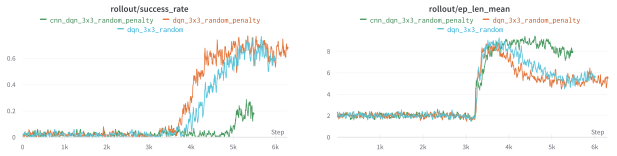
Expert Policy: Instead of training a reinforcement learning expert using PPO [4], we decided to hand-implement the expert agent to avoid sub-optimal behaviours. To implement the expert policy, the grid environment is first transformed into a 2D grid-graph. This involves identifying nodes in the graph that correspond to the agent, marker, obstacles, and goal positions. The nodes corresponding to obstacles are then removed to create a modified graph that enables efficient path planning. The shortest path from the agent to the marker node and from the marker node to the goal is then computed using an algorithm such as Dijkstra's algorithm. This path is considered the expert policy's trajectory for the given task. We use Expert Planning to avoid instantiating the grid-world with states where either goal or the marker are unreachable.

Deep Q Learning: We used a (i) CNN-based Q-Net to encode the state `CONV(32, 1, 1, 0)-LeakyReLU-Flatten-FC(256)-ReLU` and a (ii) MLP-based state encoder `Flatten-FC(512)-ReLU-FC(256)-ReLU -FC(256)-ReLU` for comparison. Discount factor $\gamma = 0.99$, batch size $= 32$ and learning rate $= 1e-4$.

4. Results and Discussion

Average Episode Length v/s Success Rate: To fully explain the behavior of an agent during training, it is necessary to consider both the average episode length and the success rate metrics. During the initial stages of training, we observe a low average episode length, which is primarily due to the episode terminating whenever the agent collides with an obstacle or wall. As the agent becomes more skilled at avoiding obstacles, the average episode length increases, indicating that the agent is spending more time exploring the environment. However, despite this progress, the agent may still struggle to collect the marker and drop it at the goal, leading to a continuous increase in the average episode length. This is because the agent is actively trying to complete the task but is not yet proficient at it. The success rate metric provides additional insights into the agent's performance, as it measures the frequency of successful task completion in the episodes collected for the replay buffer. As the agent further improves, we can see a simultaneous decrease in average episode length and an increase in the success rate (For instance, see it happen at around 4k iteration in Fig. 2, 8k iteration in Fig. 3, 7k iteration in Fig. 4).

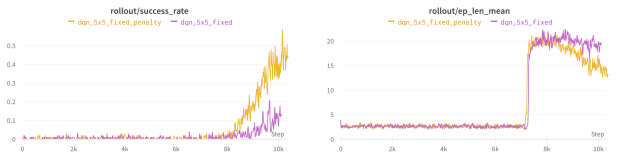
CNN v/s MLP Q-Net: We can observe in Fig. 2 that MLP-based state encoder greatly outperforms CNN-based encoders by achieving a higher success rate and lower avg. episode length (increasing the CNN-mask size to 3x3 did not improve its performance). This suggests that in 3D grid world environments with many-hot representations, MLPs are a better choice than CNNs for state encoding. This could be explained by how each cell in the grid with many-hot representation is in one of several states and is independent of neighbouring cell states. Thus, the CNNs may struggle to extract meaningful features from the input since there is no spatial relationship to exploit. In contrast, an MLP-based encoder can handle many-hot representations more effectively by flattening the input into a 1D vector and passing it through a series of fully connected layers. This allows the MLP to learn the relationships between the different states and extract more meaningful features.



(a) Success Rate v/s TrainSteps (b) Avg. EpLen. v/s TrainSteps

Figure 2: DQN Performance on 3x3 'random' Karel World Environment with and without per-step penalty. Also showing comparison of MLP and CNN-based Q-Net ($T = 10$)

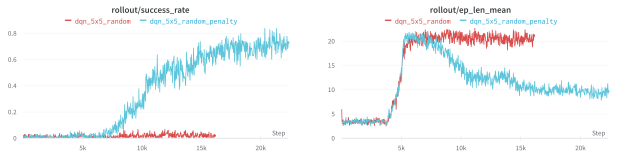
Number of Training Iterations: We observed that we needed more training iterations in DQN to achieve satisfactory performance when increasing the complexity from 'fixed' to 'random' across '3x3' and '5x5' grid worlds primarily because in the 'random' setting, the number of obstacles and positions of obstacles are also randomised in each instantiation of the environment. Thus we used $1e5$ for random 3x3 grid, $3e5$ for fixed 5x5 grid and $1e6$ for random 5x5 grid.



(a) Success Rate v/s TrainSteps (b) Avg. EpLen. v/s TrainSteps

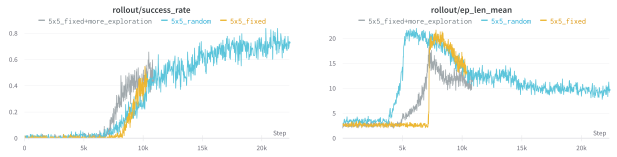
Figure 3: DQN Performance on 5x5 'fixed' Karel World Environment with and without per-step penalty ($T = 25$)

Importance of per-step penalty: When the agent is rewarded with a small reward for picking up the marker and



(a) Success Rate v/s TrainSteps (b) Avg. EpLen. v/s TrainSteps

Figure 4: DQN Performance on 5x5 'random' Karel World Environment with and without per-step penalty ($T = 25$)



(a) Success Rate v/s TrainSteps (b) Avg. EpLen. v/s TrainSteps

Figure 5: DQN Performance on 5x5 Karel World Environment using different exploration methods ($T = 25$)

not penalized for each step taken, we observed that the agent exhibits oscillatory behavior after collecting the marker until the task horizon is reached. This occurs because there is no incentive for the agent to complete the task quickly within the given budget. To overcome this issue, we introduced a small penalty for each step taken by the agent. This encourages the agent to complete the task in fewer steps, resulting in better performance. The penalty term in the reward function serves as a cost function that incentivizes the agent to minimize the number of steps taken to complete the task. As a result, agents trained with such a reward model perform better than those without it. Refer to Figures 2, 3, 4 for higher success rate and lower average episode length for the agent penalised with per-step penalty. We also observed that the gap in the performance widens when the size of the grid increases, which is obvious since such environments present more freedom to the agent to exercise which can be curbed by the penalty.

Exploration v/s Exploitation: In Deep Q Learning for Grid worlds, exploration is a crucial aspect of the learning process. By exploring the environment, the agent can gain a better understanding of the task, learn from its mistakes and develop more robust and adaptive strategies.

Exploration can be achieved in several ways, such as by using an epsilon-greedy policy, which involves selecting a random action with a small probability (ϵ) instead of action with the highest Q-value. This encourages the agent to explore new states, which can help it discover more efficient strategies for completing the task. To achieve higher exploration, we increased the $exploration\_fraction$ from 0.1 to 0.4 (i.e. percentage of total train steps for which agent witnesses a decaying exploration from $\epsilon_{initital}$ to ϵ_{final}) and ϵ_{final} from 0.05 to 0.1.

Another way to promote exploration is by using a more diverse set of training examples, such as by introducing random obstacles. Promoting more exploration during training, leads to more efficient learning and better performance in terms of success rate and average episode length. See Fig. 5 where by increasing exploration either through ϵ-greedy or by changing the environment to 'random', we witness climb in performance much earlier than the vanilla '5x5 fixed' case. However, it is to be noted that prolonged exploration can hinder learning, thus a good balance between exploration and exploitation is a must.

5. Conclusion

In this study, we investigated and demonstrated the key factors influencing the efficiency of Deep Q-Learning in grid-world scenarios, including the selection of encoding networks, exploration coefficient, and reward function components. As a future direction, we suggest exploring the incorporation of Hindsight Experience Replay (HER) [1] to improve the algorithm's convergence and assess its performance in environments with different levels of complexity. HER is an approach that generates additional training data by retrospectively considering a particular state as the goal state and replaying the experience with this new goal, even if it was not the original goal. We believe that the implementation of HER could lead to further improvements in the performance and stability of Deep Q-Learning in grid-world scenarios.

References

[1] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv*, Dec. 2013.

[3] J. D. R. S. P. K. Rudy Bunel, Matthew Hausknecht. Leveraging grammar and reinforcement learning for neural program synthesis, Feb. 2018.

[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.