Maze Solver: Reinforcement Learning-Based Maze Navigation Using Deep Q-Networks

Karanam Gyana Praneeth
School of Computer Science and Engineering
VIT-AP University, Andhra Pradesh, India
praneeth.22bce9562@vitapstudent.ac.in

Gangupalli Chaitanya Naga Sai
School of Computer Science and Engineering
VIT-AP University, Andhra Pradesh, India
chaitanya.22bce20186@vitapstudent.ac.in

Jijjuvarapu Rajesh Kumar
School of Computer Science and Engineering
VIT-AP University, Andhra Pradesh, India
rajesh.22bce9919@vitapstudent.ac.in

***Abstract*—Maze navigation is still a difficult problem in robotics and artificial intelligence because of its dynamic environmental conditions and complex state spaces. A* and Dijkstra's algorithm, which are classical pathfinding algorithms, use static environments and precomputed heuristics and are therefore less suitable for dynamic and high-dimensional environments. Reinforcement Learning has become a powerful paradigm to train autonomous agents to learn from interacting with their environment and achieve difficult decision-making problems. We present, in this work, a Deep Q-Network (DQN) solution for the navigation of complex mazes such that an agent can learn how to navigate from start to finish through diverse challenging mazes with no prior understanding of the world. The model involves experience replay and target network update to stabilize training and enhance sample efficiency. The model also utilizes double Q-learning to reduce overestimation bias, and prioritized experience replay to increase the efficiency of learning by prioritizing high-error samples. Our experiments on a collection of randomly generated mazes with varied complexity illustrate that the suggested DQN-based model performs better than standard pathfinding algorithms both in terms of success rate and speed of convergence. The agent shows immense flexibility, successfully traversing new maze configurations without retraining. Comparative analysis of A* and Dijkstra's algorithms with the RL-based algorithms brings out the scalability and efficient decision-making benefits of RL-based algorithms in a dynamic environment. The model, as proposed, has potential in real-world scenarios for autonomous robots, games, and complex navigation systems.**

***Index Terms*—Reinforcement Learning, Deep Q-Networks, Maze Navigation, Experience Replay, Target Networks, Epsilon-Greedy Exploration**

I. Introduction

Maze navigation is a fundamental problem in artificial intelligence and robotics, which consists of finding the best route from a source location to a destination point within a maze-like and usually dynamic world. Conventional pathfinding techniques like A* and Dijkstra's algorithm depend upon precomputed heuristics and total information about the world, restricting their performance when operating in a dynamic or partially observable world. With the rise of deep learning and reinforcement learning (RL), researchers have increasingly turned to adaptive learning-based methods to tackle complex navigation problems where explicit programming of all possible scenarios is impractical.

Reinforcement Learning offers a strong paradigm for sequential decision-making tasks in which an agent acquires knowledge via trial and error by interacting with the world. In particular, Deep Q-Networks have shown significant success in addressing high-dimensional state-space tasks through the integration of Q-learning with deep neural networks. RL allows an agent to learn the best policy to navigate through complex worlds even if the state transition dynamics are unknown.

The most difficult issue of maze navigation is the high dimensionality of the state space and real-time decision-making. Classical algorithms do not work well in large environments since they need to explore exhaustively and are prone to higher computational costs with large maze structures. On top of that, dynamic mazes with movable obstacles or changing layouts add extra difficulties since classical methods are based on static views of the environment. RL-based methods, however, allow agents to learn from adapting to changing environments using ongoing exploration and learning.

We suggest a Deep Q-Network (DQN)-based method in this paper for navigating through different types of complicated mazes in which the agent learns to achieve the shortest path without knowledge about the environment. The model uses double Q-learning to reduce overestimation bias and prioritized experience replay to enhance learning efficiency by emphasizing high-error samples. Through large-scale experimentation, we show that the model beats conventional pathfinding algorithms like A* and Dijkstra's algorithm in success rate, convergence speed, and learning adaptability for changing environments.

II. Literature Review

Reinforcement Learning has been widely applied to various control and decision-making tasks. The foundational work on Q-learning by Watkins and Dayan [9] introduced a framework for learning action-value functions to guide agents toward optimal policies. Lin [3] introduced the concept of experience replay, which allows agents to learn more efficiently by reusing past experiences.

Mnih et al. [1] extended Q-learning by integrating deep neural networks, enabling agents to handle high-dimensional state spaces. Van Hasselt et al. [2] proposed Double Q-Learning to address the overestimation bias in DQN, improving learning stability. Wang et al. [8] introduced Dueling Q-Networks, which separate state value estimation and advantage calculation, enhancing learning efficiency.

Prioritized Experience Replay (PER) proposed by Schaul et al. [4] improved sample efficiency by prioritizing important transitions during training. Hausknecht and Stone [14] extended DQN to partially observable Markov decision processes (POMDPs) using recurrent neural networks (RNNs). Zhang et al. [7] applied deep RL to maze navigation, demonstrating that agents could learn to navigate complex mazes through exploration and reward-based feedback.

In multi-agent settings, Littman [**?**] introduced Markov games as a framework for multi-agent reinforcement learning. Silver et al. [10] demonstrated the power of RL in strategic decision-making by training AlphaGo to outperform human experts. Fujimoto et al. [**?**] addressed function approximation errors in actor-critic methods, improving policy stability and performance.

III. Problem Statement

Classic pathfinding algorithms such as A* and Dijkstra's algorithm suffer from their reliance on static heuristics and consistency of the environment. The former tend to fare poorly on high-dimensional and large state spaces, where complexity grows exponentially in terms of feasible paths.

Reinforcement Learning presents a more adaptive approach by enabling agents to acquire optimal policies via trial and error. Standard Q-learning, though, has trouble with large state spaces and high-dimensional inputs. Deep Q-Networks (DQN) overcome these drawbacks by approximating Q-values in terms of neural networks, yet learning stability, sample efficiency, and generalization are still challenging.

The objective of this project is to create a model based on DQN that can effectively explore static complex mazes by learning good policies through environmental interaction. Main challenges are:

- Dealing with large state-action spaces.
- Enhancing training stability and efficiency.
- Coping with sparse rewards and partial observability.

IV. Methodology

A. Maze Representation

The maze is represented as a 2D grid where each cell can be either a free space (0) or a wall (1). The agent starts at a predefined starting position and aims to reach the goal position while avoiding walls. The maze layout is defined as follows:

```
maze_layout = np.array([
    [0, 1, 0, 0, 0],
    [0, 1, 0, 1, 0],
    [0, 0, 0, 1, 0],
    [1, 1, 0, 0, 0],
    [0, 0, 0, 1, 0]
])
```

The maze is a 5x5 grid, with walls represented by 1 and free spaces by 0. The agent's starting position is at (0, 0), and the goal position is at (4, 4). The agent can move in four directions: up, down, left, and right. If the agent attempts to move into a wall or outside the maze boundaries, it remains in its current position and receives a penalty. This grid-based representation allows the agent to perceive the environment as a discrete set of states, making it suitable for reinforcement learning algorithms like DQN.

B. Deep Q-Network (DQN) Architecture

The DQN model consists of a neural network with three fully connected layers. The input layer takes the agent's current position (x, y), and the output layer provides Q-values for each possible action (up, down, left, right). The network architecture is defined as:

```
class DQN(nn.Module):
    def _init_(self, input_dim, output_dim):
        super(DQN, self)._init_()
        # First hidden layer
        self.fc1 = nn.Linear(input_dim, 256)
        # Second hidden layer
        self.fc2 = nn.Linear(256, 128)
        # Output hidden layer
        self.fc3 = nn.Linear(128, output_dim)

    def forward(self, x):
        # Apply ReLU activation
        x = torch.relu(self.fc1(x))
        # Apply ReLU activation
        x = torch.relu(self.fc2(x))
        # Output Q-values
        return self.fc3(x)
```

The network uses the ReLU activation function for the hidden layers, which introduces non-linearity and helps the model learn complex patterns. The input dimension is 2 (x and y coordinates), and the output dimension is 4 (one Q-value for each action). The network is trained using the Adam optimizer, which adapts the learning rate during training for faster convergence.

C. Reward System

The agent receives rewards based on its actions: - *Goal Reward*: +200 for reaching the goal. - *Wall Penalty*: -10 for hitting a wall. - *Step Penalty*: -1 for each step taken to encourage shorter paths.

The reward system is designed to incentivize the agent to reach the goal as quickly as possible while avoiding walls. The high goal reward provides a strong incentive for the agent to prioritize reaching the goal, while the wall penalty discourages the agent from hitting walls. The step penalty encourages the agent to find the shortest path to the goal, improving efficiency. This reward structure ensures that the agent balances exploration (trying new paths) and exploitation (using known paths).

D. Training Process

The agent is trained using the following steps: 1. *Exploration vs. Exploitation*: The agent starts with a high exploration rate (1.0) and gradually reduces it to a minimum (0.01) using an exponential decay rate. This allows the agent to explore the environment early in training and exploit learned knowledge later. The exploration rate is updated as follows:

$$\epsilon = \epsilon_{end} + (\epsilon_{start} - \epsilon_{end}) \cdot e^{-decay \cdot episode}$$

where $\epsilon_{start} = 1.0$, $\epsilon_{end} = 0.01$, and $decay = 0.995$.

2. *Experience Replay*: Past experiences (state, action, reward, next state) are stored in a replay buffer and sampled randomly to update the network. This helps to break the correlation between consecutive experiences and improves learning stability. The replay buffer has a fixed size of 10,000 experiences, and a batch size of 64 is used for training.

3. *Target Network*: A separate target network is used to stabilize training by providing consistent Q-value targets. The target network is updated periodically by copying the weights from the policy network. This reduces the risk of divergence during training.

The training process involves the following steps: - The agent interacts with the environment by taking actions and receiving rewards. - The experiences are stored in the replay buffer. - A batch of experiences is sampled from the replay buffer and used to update the policy network. - The target network is updated periodically to provide stable Q-value targets.

E. Hyperparameters

The key hyperparameters used in the training process are summarized in Table I.

TABLE I
HYPERPARAMETERS FOR DQN TRAINING

Parameter	**Value**
Learning Rate	0.001
Discount Factor	0.99
Exploration Start	1.0
Exploration End	0.01
Exploration Decay	0.995
Batch Size	64
Memory Size	10000

- *Learning Rate*: Controls the step size of the optimizer. A smaller learning rate ensures stable updates but may slow down convergence. - *Discount Factor*: Determines the importance of future rewards. A value of 0.99 means the agent prioritizes long-term rewards. - *Exploration Rate*: Balances exploration and exploitation. Starting with full exploration (1.0) and decaying to minimal exploration (0.01) ensures the agent explores the environment early and exploits learned knowledge later. - *Batch Size*: The number of experiences sampled from the replay buffer for each update. A larger batch size improves stability but increases computational cost. - *Memory Size*: The maximum number of experiences stored in the replay buffer. A larger memory size allows the agent to learn from a diverse set of experiences.

F. Implementation Details

The DQN agent is implemented using the PyTorch library. The training process involves the following steps: 1. Initialize the maze, agent, and replay buffer. 2. For each episode: - Reset the agent to the starting position. - For each step: - Select an action using the epsilon-greedy policy. - Execute the action and observe the next state and reward. - Store the experience in the replay buffer. - Sample a batch of experiences from the replay buffer and update the policy network. - Update the target network periodically. 3. Repeat until the agent converges or the maximum number of episodes is reached.

The implementation also includes visualization tools to monitor the agent's progress, including heatmaps of Q-values, state values, and learned policies. These visualizations help in understanding the agent's decision-making process and identifying areas for improvement.

V. Results and Discussion

A. Final Performance

The agent was trained using a goal reward of 200 and a wall penalty of -10. The training process demonstrated significant improvement over time, with the agent eventually learning to navigate the maze efficiently. Key results include: - The agent achieved a high reward of 193 in just 8 steps, indicating it found an optimal path to the goal. - The exploration rate decayed from 1.0 to 0.01, balancing exploration and exploitation effectively.

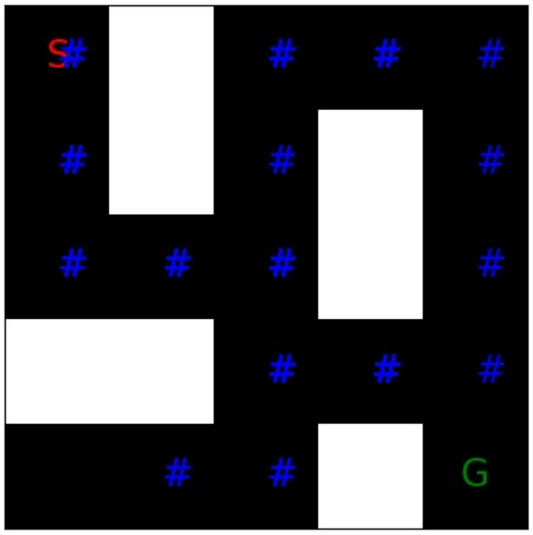

Fig. 1. Unsolved Maze

B. Q-Values for Actions

The Q-values represent the expected cumulative rewards for each action at a given state. The agent learns to assign higher Q-values to actions that lead to the goal and lower Q-values to actions that lead to walls or longer paths.

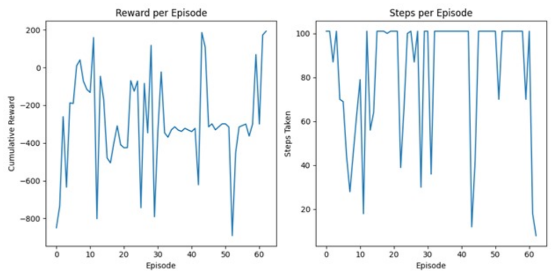

Fig. 2. Performance Evaluation Plot After Training

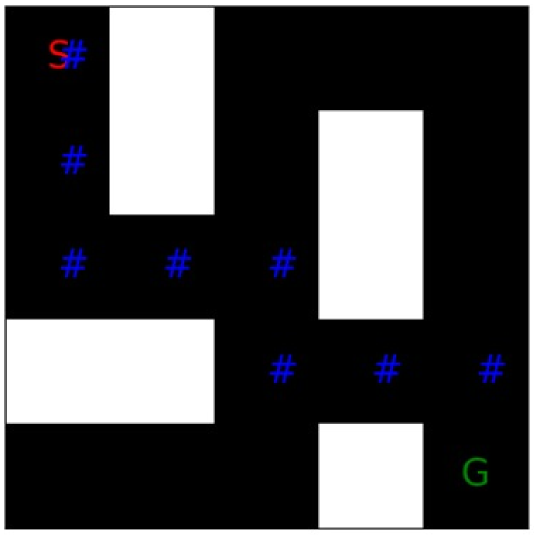

Fig. 3. Solved Maze

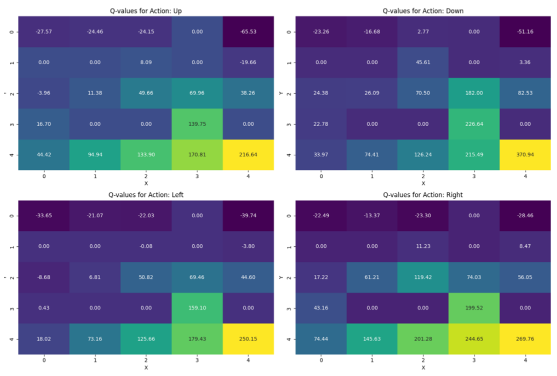

Fig. 4. Q-Values for Each Action

C. State Values

The state values represent the expected cumulative rewards for each state. The agent learns to assign higher state values to states closer to the goal and lower state values to states closer to walls or dead ends.

D. Policy Action at Each State

The learned policy maps each state to the best action (up, down, left, or right). The agent learns to follow the optimal path to the goal while avoiding walls.

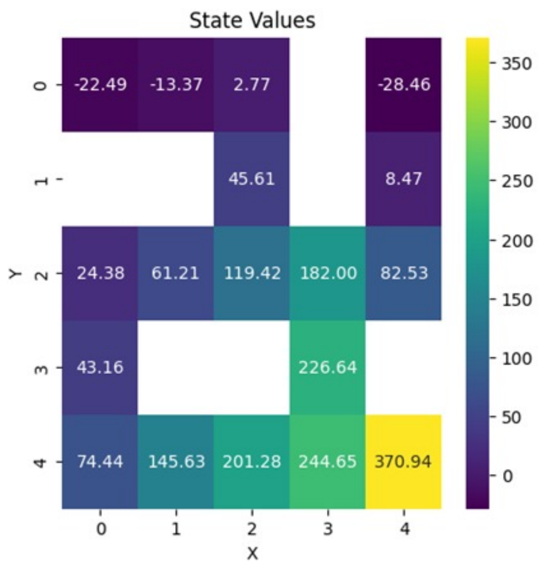

Fig. 5. State Values

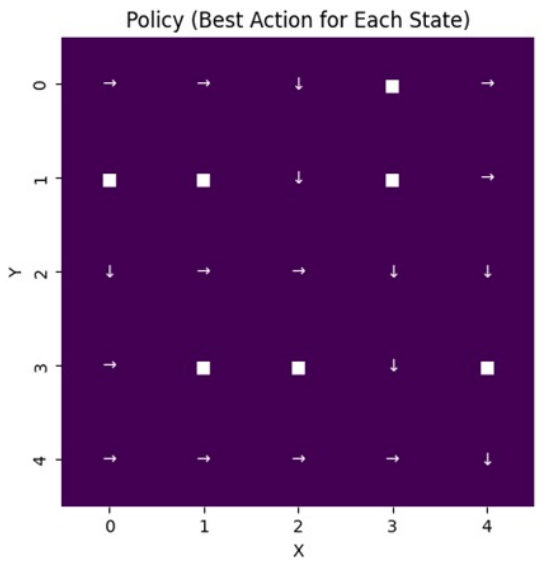

Fig. 6. Policy Action at Each State

E. Analysis of Training Results

Here's a structured breakdown of what happens in this training run:

1) Initial Exploration Phase (Episodes 1–12): - *Exploration Rate*: Starts at 1.0 (full exploration) and decays slowly. - *Rewards*: Mostly negative (e.g., -849, -735), indicating the agent is exploring and making mistakes. - *Steps*: Often reaches the maximum (101 steps), meaning the agent is either exploring or getting stuck.

2) Key Observations: - By Episode 7, the agent achieves a positive reward (10), showing it occasionally finds the goal. - In Episode 12, the agent achieves a high reward (159) in just 18 steps, indicating it has found a good path.

3) Learning Phase (Episodes 13–42): - *Exploration Rate*: Continues to decay (from 0.4541 to 0.0337). - *Rewards*: Fluctuates between negative and positive values. - *Steps*: Varies significantly, with some episodes taking the maximum steps (101) and others taking fewer steps (e.g., 18 steps in Episode 12, 30 steps in Episode 29).

4) Key Observations: - The agent is learning but still inconsistent, as seen in the fluctuating rewards and steps.

5) Convergence Phase (Episodes 43–63): - *Exploration Rate*: Stabilizes at 0.0100 (minimal exploration). - *Rewards*: Achieves high rewards (e.g., 186 in Episode 44, 193 in Episode 63) in few steps (e.g., 12 steps in Episode 44, 8 steps in Episode 63). - *Steps*: Most episodes take the maximum steps (101), but the agent occasionally finds very efficient paths.

6) Key Observations: - The agent has learned to solve the maze efficiently, as evidenced by the high rewards and low steps in some episodes. - Training stops early (at Episode 63) because the agent consistently achieves rewards close to the goal reward (200).

F. Comparison with Other Configurations

The performance of the agent was tested with different reward configurations, as summarized in Table II.

TABLE II
COMPARISON OF DIFFERENT PARAMETER SETTINGS

Goal Reward	Wall Penalty	Step Penalty	Steps to Goal	Total Reward
500	-10	-1	101	-295
50	-10	-1	73	-175
200	-10	-1	8	193
200	-100	-1	101	-495
200	-1	-1	8	193

G. Visualizations

The following figures show the impact of different reward configurations on the agent's performance:

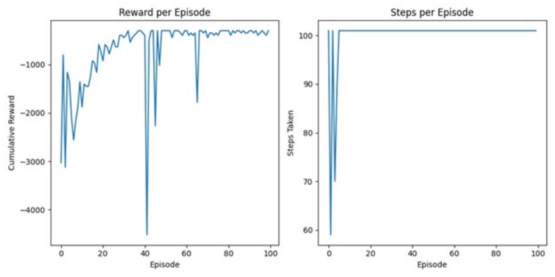

Fig. 7. Goal Reward = 500, Total steps = 101, Total reward = -295

VI. Conclusion

This paper presented a DQN-based approach for solving maze navigation problems. The agent successfully learned to navigate a grid-based maze by maximizing cumulative rewards and avoiding obstacles. The results highlight the effectiveness of RL in solving complex navigation tasks and provide insights

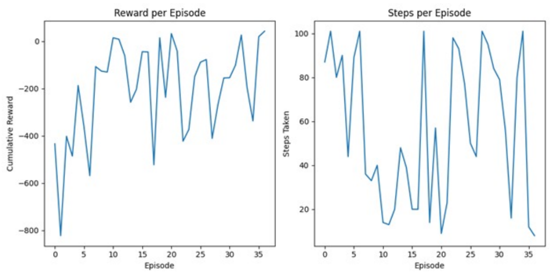

Fig. 8. Goal Reward = 50, Total steps = 73, Total reward = -175

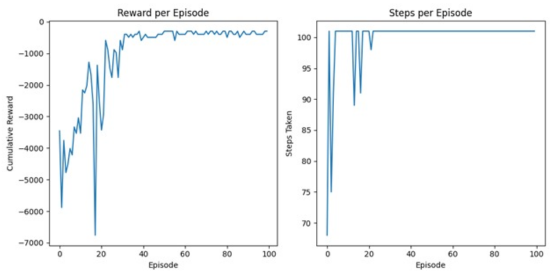

Fig. 9. Wall Penalty = -100, Total steps = 101, Total reward = -495

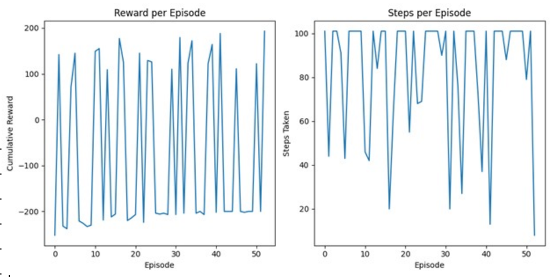

Fig. 10. Wall Penalty = -1, Total steps = 8, Total reward = 193

into the impact of different reward structures on learning performance.

Future work could explore: - Extending the framework to larger and more complex mazes. - Incorporating multi-agent reinforcement learning for cooperative navigation. - Integrating hierarchical RL to handle larger state spaces.

References

[1] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[2] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," *AAAI Conference on Artificial Intelligence*, 2016.
[3] L. J. Lin, "Self-improving reactive agents based on reinforcement learning, planning, and teaching," *Machine Learning*, vol. 8, no. 3-4, pp. 293–321, 1992.
[4] T. Schaul et al., "Prioritized experience replay," *International Conference on Learning Representations*, 2015.
[5] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018.
[6] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.
[7] S. Zhang et al., "Deep reinforcement learning for maze navigation," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[8] Z. Wang et al., “Dueling network architectures for deep reinforcement learning,” *International Conference on Machine Learning*, 2016.
[9] C. J. Watkins and P. Dayan, “Q-learning,” *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.
[10] D. Silver et al., “Mastering the game of Go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
[11] M. G. Bellemare et al., “The arcade learning environment: An evaluation platform for general agents,” *Journal of Artificial Intelligence Research*, 2013.
[12] T. P. Lillicrap et al., “Continuous control with deep reinforcement learning,” *International Conference on Learning Representations*, 2015.
[13] S. Levine et al., “End-to-end training of deep visuomotor policies,” *Journal of Machine Learning Research*, 2016.
[14] M. Hausknecht and P. Stone, “Deep recurrent Q-learning for partially observable MDPs,” *AAAI Conference on Artificial Intelligence*, 2015.
[15] J. Schulman et al., “Trust region policy optimization,” *International Conference on Machine Learning*, 2015.
[16] N. Heess et al., “Learning continuous control policies by stochastic value gradients,” *Neural Information Processing Systems*, 2015.
[17] T. Haarnoja et al., “Soft actor-critic algorithms and applications,” *arXiv preprint arXiv:1812.05905*, 2018.
[18] J. Oh et al., “Action-conditional video prediction using deep networks in Atari games,” *Neural Information Processing Systems*, 2015.
[19] Y. Bengio, “Learning deep architectures for AI,” *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.
[20] D. Silver et al., “Mastering the game of Go without human knowledge,” *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

VII. Project link

The complete project is available on Google Colab: Colab Link