Vivekanand Education Society's Institute of Technology
(An Autonomous Institute Affiliated to University of Mumbai,)
(Approved by A.I.C.T.E and Recognized by Govt. of Maharashtra)

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

A REPORT

ON

Cart Pole Game using Deep Q Network

B.E. (AIDS)

SUBMITTED BY

Mr. Subrato Tapaswi (Roll No. 60)

UNDER THE GUIDANCE OF

Prof. Amit Singh

(Academic Year: 2023-2024)

Vivekanand Education Society's Institute Of Technology, Mumbai

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

V.E.S.

Since 1962

Certificate

This is to certify that project entitled

"Cart Pole Game using Deep Q Network"

Mr. Subrato Tapaswi (Roll No. 60)

have satisfactorily carried out the project work, under the head - Reinforcement Learning Lab at Semester VIII of BE in AIDS as prescribed by the Syllabus.

Mr. Amit Singh **Subject Teacher**	**Mrs. Smita Mane** **Lab Teacher**
Dr.(Mrs.)M. Vijayalakshmi **H.O.D**	**Dr.(Mrs.)J.M.Nair** **Principal**

Date: 04/04/2024
Place: VESIT, Chembur

Declaration

I declare that this written submission represents my ideas in my own words and where other's ideas or words have been included, I have adequately cited and referenced the original source. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

- - - - - - - - - - -
(Signature)

Mr
Mr. Subrato Tapaswi (Roll No. 60)
B.E. AIDS

Contents

1 **1**

1.1 Introduction 1

1.2 Literature Survey 1

1.3 Problem Definition/Statement 2

1.4 Objectives 2

1.5 Proposed Solution 3

1.6 Technology Used 3

1.7 Table Sample 4

2 **5**

2.1 System Design / Road map / Algorithm 5

2.2 Flow Chart 6

3 **7**

3.1 Implementation Snapshots and Training 7

3.2 Code and Evaluation 8

4 **10**

4.1 CONCLUSION 10

4.2 Future Directions 10

List of Figures

2.1 Cart Pole 5
2.2 Observation States 5
2.3 Q Network vs Deep Q Network 6
2.4 DQN Architecture 6

3.1 Final Output 7
3.2 Training 7
3.3 Cart Pole 8
3.4 DQN 8
3.5 Agent 9
3.6 Evaluation Curve 9
3.7 Training Curve 9

List of Tables

1.1 Table 1 4

Abstract

The Cart Pole game has long been a fundamental challenge in the realm of reinforcement learning, captivating researchers and enthusiasts alike with its elegant yet intricate dynamics. In this paper, we delve into the application of Deep Q-Network (DQN), a powerful reinforcement learning algorithm, in the context of the Cart Pole game, aiming to revolutionize the gameplay experience and push the boundaries of autonomous agent performance.

We explore the seamless integration of DQN into the Cart Pole game environment, where the agent learns to balance the pole by interacting with the environment and receiving feedback in the form of rewards or penalties.

The DQN architecture, with its deep neural network components and Q-learning framework, enables the agent to learn effective strategies through trial and error.

Agent Training and Decision-Making: We discuss the training process of the DQN agent, emphasizing the exploration-exploitation trade-offs crucial for discovering optimal policies. The agent's decision-making process, guided by Q-values learned from experiences, showcases how DQN adapts and improves its performance over time.

Keywords-*Reinforcement Learning, Deep Q-Network, DQN, Cart Pole, Gaming environment.*

Chapter 1

1.1 Introduction

Reinforcement Learning (RL) is gaining traction as an effective method for instructing agents to make consecutive choices within ever-changing settings.

The Cart Pole game involves balancing a pole on a moving cart, challenging players to maintain equilibrium through precise control to prevent the pole from falling.

This report investigates the application of Deep Reinforcement Learning, notably Deep Q-Network (DQN), in mastering the Cart Pole game's balancing challenge, emphasizing its relevance in solving dynamic control problems. Beyond its gaming context, the Cart Pole problem mirrors real-life scenarios such as robotic arm control, spacecraft stabilization, and autonomous vehicle navigation, highlighting the practical importance of mastering such dynamic control tasks using advanced RL techniques like DQN.

1.2 Literature Survey

1. **Deep Q-Networks (DQN) and Experience Replay**: A tutorial by Maciej Balawejder [1] provides a detailed explanation of how to use Deep Q-Networks (DQN) and experience replay to solve the CartPole problem [1]. The tutorial explains how DQN combines Q-learning with deep learning to get rid of the Q-table and use neural networks instead to approximate the action-value function [1]. It also discusses the concept of experience replay, which randomizes over the data, thereby removing correlations in the observation sequence and smoothing over changes in the data distribution [1].

2. **Balancing a CartPole System with Reinforcement Learning**: A paper by Swagat Kumar [2] provides a comprehensive guide to implementing various reinforcement learning algorithms for controlling a Cart-Pole system. The paper describes various RL concepts such as Q-learning, Deep Q Networks (DQN), Double DQN, Dueling networks, and (prioritized) experience replay [2]. It is observed that DQN with Prioritized Experience Replay (PER) provides the best performance among all other architectures [2].

3. **Modeling Human Actions in the Cart-Pole Game Using Cognitive**: A study published on Springer [3] discusses giving a human touch to RL by building

it over human behavior data and by developing a cognitive model to work in an ensemble with RL [3]. The study includes the background on the cart-pole problem, followed by the detailed methodology of this study [3].

These resources provide a deeper understanding of the various reinforcement learning methods and their applications in controlling a Cart-Pole system

1.3 Problem Definition/Statement

Creating a Deep Reinforcement Learning (DRL) agent for the Cart Pole game faces unique hurdles. The agent must balance a pole on a moving cart, managing dynamic shifts, maintaining equilibrium, and strategizing to prevent the pole from falling. Challenges include dynamic control, partial state observation, dealing with a high-dimensional state space, and requiring long-term planning for optimal performance. The objective is to develop a DRL agent that excels in balancing the pole and achieves sustained stability throughout the game.

1.4 Objectives

1. **Integrate Deep Q-Learning**: Incorporate Deep Q-Learning (DQL) into the reinforcement learning framework to enhance the agent's decision-making in the Cart Pole game. Leverage the Deep Q-Network (DQN) architecture to approximate the optimal action-value function efficiently.

2. **Train DRL Agent with DQL**: Utilize the integrated DQL approach to train the Deep Reinforcement Learning (DRL) agent in balancing the pole on the moving cart. Employ techniques like experience replay and target networks for stable training and efficient learning.

3. **Optimize Exploration-Exploitation Trade-off**: Fine-tune the exploration-exploitation strategy within the DQL framework to balance between exploring new actions and exploiting learned knowledge effectively. Adjust exploration probabilities like -greedy to ensure a proper balance during training.

4. **Maximize Cumulative Rewards**: Focus on maximizing the agent's cumulative rewards by learning long-term strategies for balancing the pole while navigating the Cart Pole game environment. Emphasize effective pole balancing to prolong gameplay and achieve higher scores.

5. **Evaluate DQL Performance**: Evaluate the trained DRL agent's performance using metrics like average balancing time and stability across multiple episodes. Assess the agent's ability to maintain equilibrium and avoid pole falls, identifying areas for enhancement.

6. **Iterative Refinement**: Continuously refine the DQL-based reinforcement learning framework by adjusting parameters, updating neural network structures, and incorporating advanced techniques. Iterate on training strategies based on performance evaluations to enhance the agent's learning and gameplay proficiency iteratively.

1.5 Proposed Solution

The proposed solution for the Cart Pole game involves implementing a Deep Q-Learning (DQL) algorithm, specifically utilizing the Deep Q-Network (DQN) architecture, to train an agent adept at mastering pole balancing. The DQN architecture comprises a neural network that processes the cart-pole system's state to output Q-values representing future rewards for actions like moving the cart left or right. During training, the agent updates its Q-values based on rewards received, aiming to maximize cumulative rewards over time.

Key components of the DQL approach include experience replay, where past experiences are stored and randomly sampled during training to stabilize learning, and epsilon-greedy action selection, balancing exploration and exploitation by choosing actions based on a mix of exploration and exploitation probabilities determined by epsilon. The goal is to enable the agent to learn optimal strategies for maintaining pole equilibrium, navigating the dynamic environment, and maximizing cumulative rewards.

Through iterative training and refinement, the agent learns to effectively balance exploration of new strategies and exploitation of learned knowledge, showcasing proficient pole balancing and intelligent decision-making in the Cart Pole game's challenging and dynamic control environment.

1.6 Technology Used

In the development of the Snake game reinforcement learning project, the technology stack includes:

1. **Python**: The primary programming language for game logic, DQL algorithm, and overall project development.
2. **Pygame**: Used for GUI creation, graphics rendering, and user input handling for the Snake game.
3. **Google Colab**: Provides GPU resources for efficient DQL agent training.
4. **Jupyter Notebooks**: Interactive development environment within Google Colab for code experimentation and documentation.
5. **Deep Learning Frameworks**: TensorFlow or PyTorch for implementing the Deep Q-Network.
6. **Git Version Control**: Facilitates collaboration and code management throughout the project.
7. **Markdown**: Utilised for documenting project details and results within Jupyter Notebooks.
8. **LaTeX**: Used to create the mini-project report, ensuring a structured and professional format for documenting project details, methodology, results, and conclusions with precise formatting and mathematical notation capabilities.

1.7 Table Sample

Average Score = 477.25
Here's the complete output table:

Episode Number	Score
1	10
100	150
200	220
300	295
400	240
500	100
554	400

Table 1.1: Table 1

Chapter 2

2.1 System Design / Road map / Algorithm

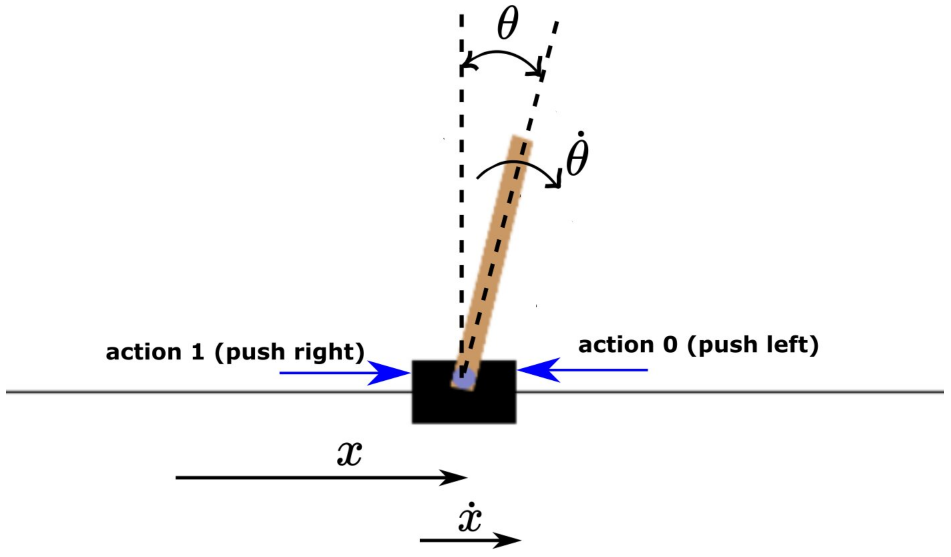


Figure 2.1: Cart Pole

Number	Observation	Minimum	Maximum
0	Cart Position	-2.4	+2.4
1	Cart Velocity	-Inf	+Inf
2	Pole Angle	~-41.8°	~+41.8°
3	Pole Velocity At Tip	-Inf	+Inf

(b) Actions

Number	Action
0	Push Cart to the Left
1	Push Cart to the Right

Figure 2.2: Observation States

2.2 Flow Chart

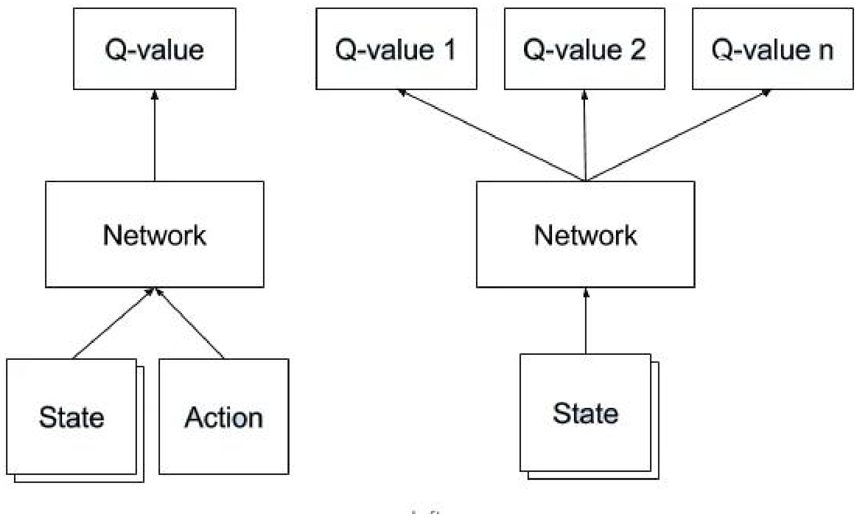


Figure 2.3: Q Network vs Deep Q Network

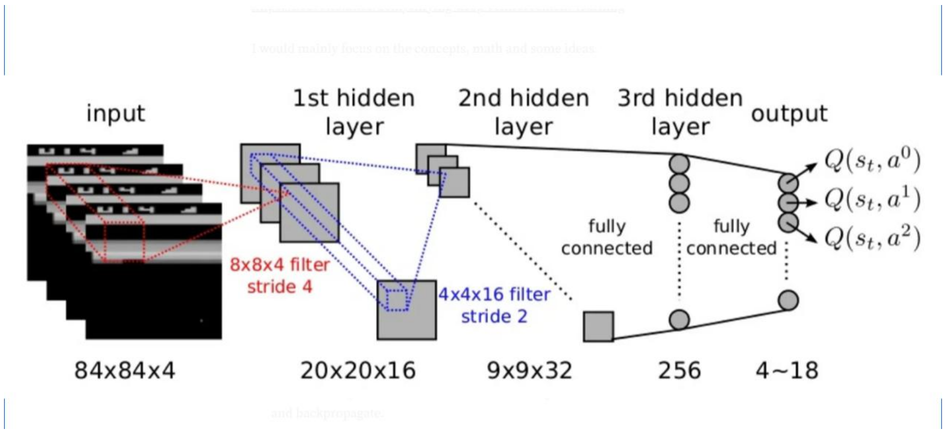


Figure 2.4: DQN Architecture

Chapter 3

3.1 Implementation Snapshots and Training

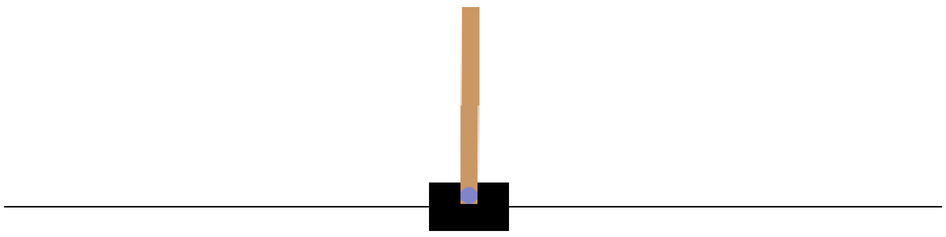
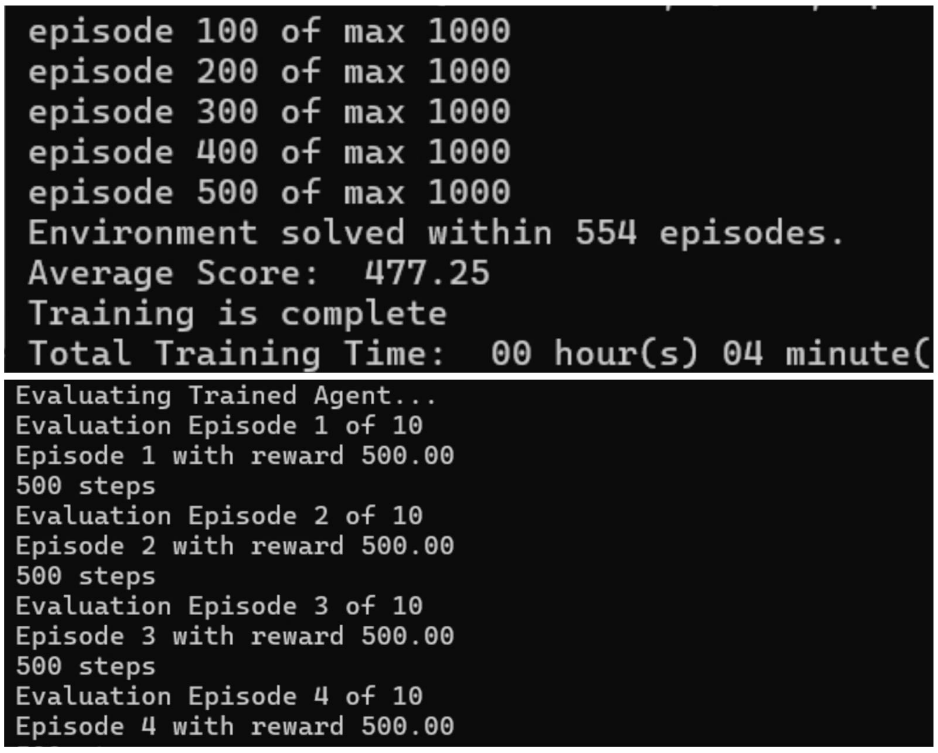


Figure 3.1: Final Output

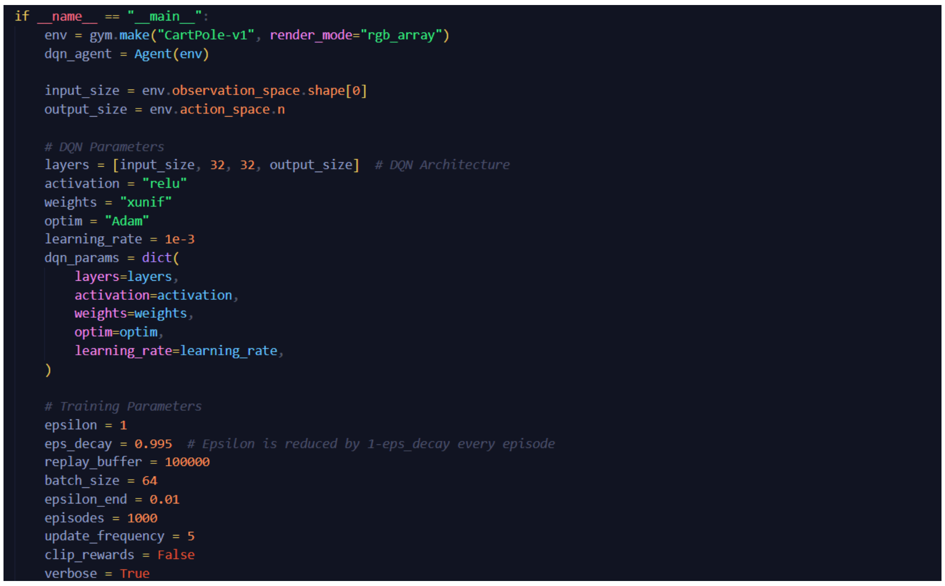


```
episode 100 of max 1000
episode 200 of max 1000
episode 300 of max 1000
episode 400 of max 1000
episode 500 of max 1000
Environment solved within 554 episodes.
Average Score:  477.25
Training is complete
Total Training Time:  00 hour(s) 04 minute(
```

```
Evaluating Trained Agent...
Evaluation Episode 1 of 10
Episode 1 with reward 500.00
500 steps
Evaluation Episode 2 of 10
Episode 2 with reward 500.00
500 steps
Evaluation Episode 3 of 10
Episode 3 with reward 500.00
500 steps
Evaluation Episode 4 of 10
Episode 4 with reward 500.00
```

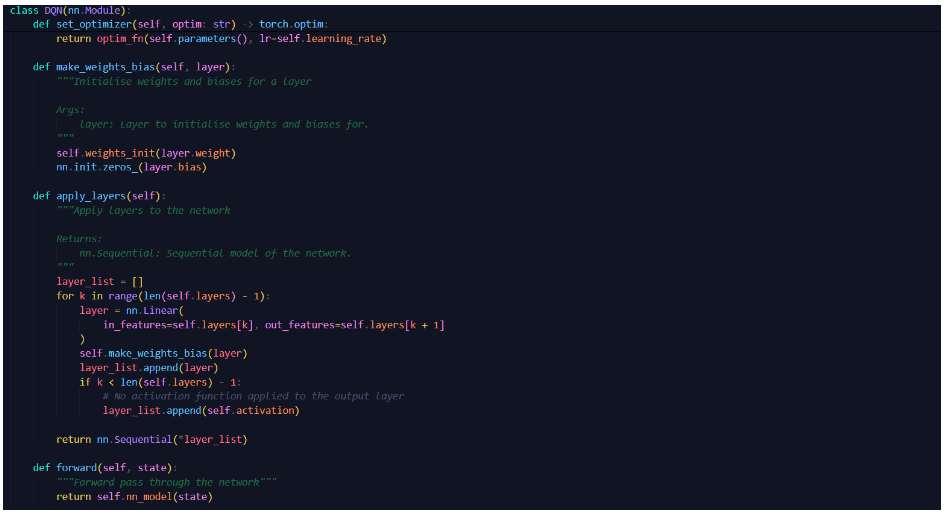
Figure 3.2: Training

3.2 Code and Evaluation



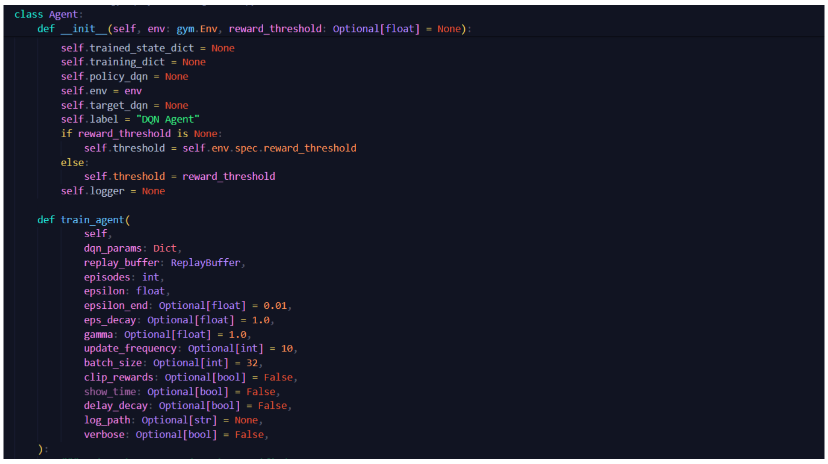
```python
if __name__ == "__main__":
    env = gym.make("CartPole-v1", render_mode="rgb_array")
    dqn_agent = Agent(env)

    input_size = env.observation_space.shape[0]
    output_size = env.action_space.n

    # DQN Parameters
    layers = [input_size, 32, 32, output_size]  # DQN Architecture
    activation = "relu"
    weights = "xunif"
    optim = "Adam"
    learning_rate = 1e-3
    dqn_params = dict(
        layers=layers,
        activation=activation,
        weights=weights,
        optim=optim,
        learning_rate=learning_rate,
    )

    # Training Parameters
    epsilon = 1
    eps_decay = 0.995  # Epsilon is reduced by 1-eps_decay every episode
    replay_buffer = 100000
    batch_size = 64
    epsilon_end = 0.01
    episodes = 1000
    update_frequency = 5
    clip_rewards = False
    verbose = True
```

Figure 3.3: Cart Pole



```python
class DQN(nn.Module):
    def set_optimizer(self, optim: str) -> torch.optim:
        return optim_fn(self.parameters(), lr=self.learning_rate)

    def make_weights_bias(self, layer):
        """Initialise weights and biases for a layer

        Args:
            layer: Layer to initialise weights and biases for.
        """
        self.weights_init(layer.weight)
        nn.init.zeros_(layer.bias)

    def apply_layers(self):
        """Apply layers to the network

        Returns:
            nn.Sequential: Sequential model of the network.
        """
        layer_list = []
        for k in range(len(self.layers) - 1):
            layer = nn.Linear(
                in_features=self.layers[k], out_features=self.layers[k + 1]
            )
            self.make_weights_bias(layer)
            layer_list.append(layer)
            if k < len(self.layers) - 1:
                # No activation function applied to the output layer
                layer_list.append(self.activation)

        return nn.Sequential(*layer_list)

    def forward(self, state):
        """Forward pass through the network"""
        return self.nn_model(state)
```

Figure 3.4: DQN



```
class Agent:
    def __init__(self, env: gym.Env, reward_threshold: Optional[float] = None):
        self.trained_state_dict = None
        self.training_dict = None
        self.policy_dqn = None
        self.env = env
        self.target_dqn = None
        self.label = "DQN Agent"
        if reward_threshold is None:
            self.threshold = self.env.spec.reward_threshold
        else:
            self.threshold = reward_threshold
        self.logger = None

    def train_agent(
            self,
            dqn_params: Dict,
            replay_buffer: ReplayBuffer,
            episodes: int,
            epsilon: float,
            epsilon_end: Optional[float] = 0.01,
            eps_decay: Optional[float] = 1.0,
            gamma: Optional[float] = 1.0,
            update_frequency: Optional[int] = 10,
            batch_size: Optional[int] = 32,
            clip_rewards: Optional[bool] = False,
            show_time: Optional[bool] = False,
            delay_decay: Optional[bool] = False,
            log_path: Optional[str] = None,
            verbose: Optional[bool] = False,
    ):
```

Figure 3.5: Agent

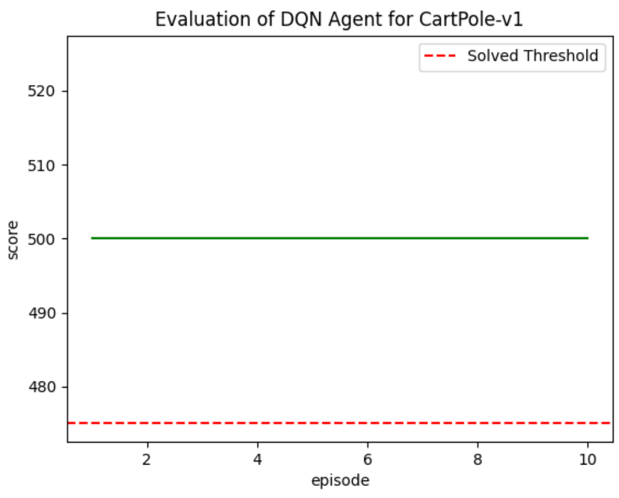


Figure 3.6: Evaluation Curve

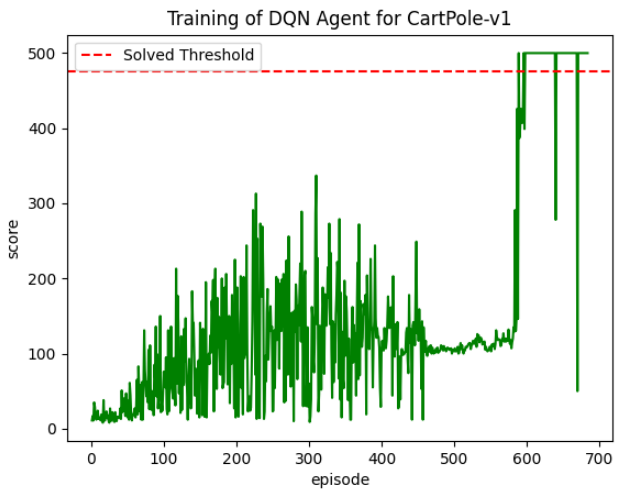


Figure 3.7: Training Curve

Chapter 4

4.1 CONCLUSION

In conclusion, the incorporation of Deep Q-Network (DQN) into the Cartpole problem environment signifies a substantial progression in the realm of reinforcement learning. Through our investigation of training an autonomous agent to balance the pole using DQN, we have witnessed the extraordinary capacity of this algorithm to acquire optimal strategies through interaction with the problem environment. Despite inherent challenges such as balancing exploration and exploitation and considerations of training time, our experiments have shown the effectiveness of DQN in maintaining balance for extended periods and mastering the problem.

The success of DQN in the Cartpole problem highlights its potential for wider applications in problem-solving and beyond. By employing reinforcement learning techniques, DQN-based methods can improve problem-solving experiences, enable autonomous agents in navigating complex problem spaces, and contribute to the progress of artificial intelligence research.

Looking forward, further research can delve into improvements to the DQN algorithm, such as integrating prioritized experience replay, double Q-learning, or other enhancements to speed up learning and enhance performance. Moreover, extending the application of DQN to other classic problems and real-world scenarios could reveal new understandings and uses in reinforcement learning.

4.2 Future Directions

1. **Advanced Deep Reinforcement Learning Techniques**: Future research can delve into advanced deep reinforcement learning methods like Double DQN, Dueling DQN, or Rainbow DQN for training agents in the Cart Pole game. These techniques could offer improved stability, faster convergence, and better performance in balancing the pole.

2. **Hierarchical Reinforcement Learning (HRL)**: Incorporating HRL into the Cart Pole game could enable agents to learn hierarchical structures of tasks, breaking down the balancing task into sub-tasks and learning policies for each level of abstraction. This could lead to more efficient learning and decision-making processes.

3. **Multi-Agent Reinforcement Learning (MARL)**: Exploring MARL approaches in the context of the Cart Pole game could involve training multiple agents to balance poles simultaneously or compete against each other. This could lead to emergent behaviors and strategies that enhance the understanding of cooperative or competitive dynamics.

4. **Transfer Learning and Generalization**: Investigating transfer learning techniques for transferring knowledge from the Cart Pole game to related tasks or domains could enhance agents' adaptability and generalization capabilities. This could involve adapting learned balancing strategies to other control problems or physical systems.

5. **Real-World Applications**: Extending the application of Cart Pole-playing agents to real-world scenarios, such as robotics control or physical rehabilitation, could unlock practical applications. Agents could be trained to balance real-world objects or assist in maintaining stability in various physical systems.

6. **Human-AI Collaboration**: Exploring methods for facilitating collaboration between human operators and AI agents in the Cart Pole game could enhance training experiences and provide insights into human-AI interaction. Developing AI companions with adjustable difficulty levels could cater to different skill levels and preferences.

7. **Ethical and Societal Implications**: Considering the ethical and societal implications of deploying AI agents trained on the Cart Pole game is crucial. Future research should address concerns such as algorithmic bias, fairness, transparency, and accountability to ensure responsible development and deployment of AI technologies in real-world applications.

By pursuing these future directions, researchers can continue to advance the state-of-the-art in reinforcement learning, gaming AI, and AI ethics, ultimately paving the way for innovative applications and societal impact.

References

[1] V. Mnih et al. *'Human-level control through deep reinforcement learning'* International Journal on Digital Libraries,Vol 6, Issue 2, pp 219-232. (April 2006).

[2] D. Silver, A. Huang, and C. J. Maddison *'Mastering the game of Go with deep neural networks and tree search'* Nature, vol. 529, no. 7587, pp. 484-489, 2016.

[3] T. P. Lillicrap et al. *Continuous control with deep reinforcement learning* arXiv preprint arXiv:1509.02971, 2015.

[4] R. S. Sutton and A. G. Barto *'Reinforcement Learning: An Introduction'* 2nd ed. MIT Press, 2018.

[5] C. J. C. H. Watkins *'Learning from delayed rewards'* PhD thesis, King's College, Cambridge, 1989.

[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] D. P. Bertsekas and J. N. Tsitsiklis *'Neuro-dynamic programming'* Athena Scientific, 1996.

[8] C. Szepesvári *'Algorithms for reinforcement learning'* Synthesis Lectures on Artificial Intelligence and Machine Learning, vol. 4, no. 1, pp. 1-103, 2010.