TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

USING DQN TO BALANCE CART ON A POLE

By:
Anubhav Khanal (076/BCT/009)
Mahima Dhakal (076/BCT/034)
Nadika Paudel (076/BCT/038)
Pramish Paudel (076/BCT/047)

A PROJECT PROPOSAL TO THE DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE BACHELOR'S DEGREE IN ELECTRONICS AND COMMUNICATION ENGINEERING

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
LALITPUR, NEPAL

March, 2023

ABSTRACT

This project explores the use of Deep Q-Networks (DQN) for solving the cartpole problem, which involves balancing a pole on a cart by controlling the cart's movement. The DQN algorithm is a popular method in reinforcement learning that uses a neural network to estimate the Q-value function, which determines the expected future reward for taking a specific action in a given state. The implementation of the DQN algorithm was done using the PyTorch machine learning framework and tested on the OpenAI Gym environment for the cartpole problem. The results show that the DQN algorithm was able to successfully balance the pole on the cart and achieve high scores in the environment. Additionally, the project discusses the various components of the DQN algorithm, including the neural network architecture, experience replay, and target network, and evaluates their impact on the performance of the algorithm. Overall, this project demonstrates the effectiveness of the DQN algorithm for solving complex reinforcement learning problems and highlights the potential of deep learning techniques for developing intelligent agents.

Keywords: dqn, cartpole, gym, reinforment learing

TABLE OF CONTENTS

List of Figures

1. INTRODUCTION

The Cartpole problem is a classic benchmark problem in the field of reinforcement learning. The problem involves balancing a pole on a cart that moves along a track, with the goal of keeping the pole upright for as long as possible. The problem is challenging because the cart must move in a way that balances the pole, and the pole can quickly become unstable due to small perturbations.

In recent years, deep reinforcement learning techniques such as Deep Q-Networks (DQNs) have been used to solve the Cartpole problem with impressive results. DQNs are a type of neural network that can learn to approximate the Q-value function for a given state-action pair, which can be used to select actions and update the agent's policy. The goal of this project is to implement a DQN to solve the Cartpole problem, and evaluate its performance in terms of the average episode length and maximum episode length achieved. This project builds on previous work in this area. The project has important implications for the field of reinforcement learning and could lead to advancements in solving other complex control problems.

1.1. Background

Reinforcement learning (RL) is a subfield of machine learning that involves learning how to take actions in an environment to maximize a reward signal. The goal of RL is to learn a policy that maps states to actions, which can then be used to achieve some desired goal. The Cartpole problem is a well-known RL problem that involves balancing a pole on a cart, where the goal is to keep the pole balanced for as long as possible. Deep Q-Networks (DQNs) are a type of neural network that have been used to solve the Cartpole problem with impressive results. DQNs were introduced in 2013 by Mnih et al., and have since been applied to a wide range of RL problems. A DQN is a Q-learning algorithm that uses a neural network to approximate the Q-value function, which is a measure of the expected future reward for a given state-action pair.

$$R_{t_0} = \sum_{t=t_0}^{\infty} \gamma^{t-t_0} r_t$$

The Cartpole problem presents specific challenges that make it an interesting problem to study. The problem involves a continuous state space, which makes it difficult to represent the state in a way that is suitable for RL algorithms. Additionally, the problem can become

unstable quickly due to small perturbations, which requires the agent to learn how to quickly react to changes in the environment. Previous work on the Cartpole problem has focused on using Deep Q-Networks (DQNs) to learn a policy that balances the pole on the cart. While much of this work has yielded impressive results, the goal of this project is to gain a deeper understanding of the DQN algorithm by implementing it from scratch.

1.2. Problem statement

The problem can be framed as a reinforcement learning problem, where an agent learns to balance the pole by interacting with the environment and receiving rewards or penalties based on its actions. The agent's goal is to learn a policy that maximizes the total cumulative reward over a certain number of time steps.The main challenge of this problem is the high-dimensional and continuous state space, which requires sophisticated learning algorithms and exploration strategies. In addition, the problem has a non-linear and non-stationary nature, which makes it even more challenging to solve.

Therefore, the problem statement for the cartpole balance problem using reinforcement learning is to train an agent to balance a pole on top of a cart by learning a policy that maximizes the cumulative reward over a certain number of time steps, while facing the challenges of a high-dimensional and continuous state space, as well as non-linear and non-stationary dynamics.

1.3. Motivation

The cartpole balancing problem is a classic control problem that has been extensively studied in the field of robotics and control theory. This problem is a canonical example of a system that is inherently unstable and requires a feedback control mechanism to maintain stability.

The motivation of the project is the cartpole balancing problem being an important problem to study as it has many real-world applications. For example, the control strategies developed for this problem can be applied to the stabilization of inverted pendulums, which are commonly used in robotics and industrial automation. Additionally, the control techniques used for the cartpole balancing problem can be extended to a variety of other systems, such as spacecraft and satellites.

Furthermore, the cartpole balancing problem provides an excellent testbed for evaluating control algorithms. Because the problem is well-defined and relatively simple, it is possible to evaluate the performance of different control strategies using a variety of metrics. This

makes the problem an ideal platform for comparing the performance of different algorithms and for developing new control strategies.

Thus, the cartpole balancing problem is a fascinating and challenging problem that has broad applications in the fields of robotics and control theory. By studying this problem, we can gain insights into the design of feedback control systems, and develop new techniques for stabilizing unstable systems.

1.4. Objectives

The main objective of using reinforcement learning for the cartpole balancing problem is to learn reinforcement learning from scratch and develop a control policy that can balance the pole on the cart with high accuracy and efficiency, while also being robust to disturbances and adaptable to changes in the environment. This will enable us to apply the technique to a wide range of real-world control problems, and improve the performance and efficiency of many different types of systems. The major objectives are listed below:

1. To learn and implement reinforcement learning from scratch.
2. To develop a control policy that can balance the pole on the cart for an extended period of time.
3. To minimize the time it takes for the controller to learn to balance the pole.
4. To improve the stability and robustness of the control policy against disturbances and perturbations.
5. To develop a control policy that can adapt to changes in the environment, such as changes in the mass or length of the pole.
6. To compare the performance of different reinforcement learning algorithms and identify the most effective approach for solving the cartpole balancing problem.
7. To gain insights into the application of reinforcement learning to real-world control problems, and identify areas where the technique can be applied to improve system performance.

1.5. Scope of Project

The scope of the project is to implement and train a Deep Q-Network (DQN) algorithm to solve the cart-pole balancing problem. The project will involve the following steps:

1. Research and understand the cart-pole balancing problem and the DQN algorithm.
2. Implement the DQN algorithm in Python using a deep neural network as a function approximator.
3. Train the DQN algorithm using the simulation environment and evaluate its performance.
4. Conduct experiments to explore the effect of different hyperparameters and network architectures on the performance of the DQN algorithm.

2. LITERATURE REVIEW

The cart-pole balancing problem has been extensively studied in the field of reinforcement learning, and a variety of algorithms have been proposed to solve it. Among these algorithms, the Deep Q-Network (DQN) algorithm has shown promising results and has become one of the most popular algorithms for solving the cart-pole problem.

Sutton and Barto (1998) introduced the cart-pole problem as a benchmark problem for reinforcement learning algorithms. They presented a simple control algorithm based on the principle of ”reinforcement comparison” and showed that it could achieve near-optimal performance on the problem.

Mnih et al. (2013) proposed the DQN algorithm, which uses a deep neural network as a function approximator to estimate the Q-values in a Q-learning algorithm. They showed that the DQN algorithm could achieve state-of-the-art performance on a range of Atari games, including the game of Pong.

Van Hasselt et al. (2016) proposed several improvements to the original DQN algorithm, including the use of Double Q-learning and prioritized experience replay. They showed that these improvements could significantly improve the performance of the algorithm on the Atari games.

Wawrzyński and Tabor (2017) proposed a modified version of the DQN algorithm for the cart-pole balancing problem, which they called the ”DQN-SP” algorithm. The algorithm used a state-prediction module to predict the next state of the system and included a novel reward shaping scheme. They showed that the DQN-SP algorithm could achieve better performance than the standard DQN algorithm on the cart-pole problem.

Zhang and Chen (2019) proposed a hybrid algorithm that combines the DQN algorithm with a proportional-integral-derivative (PID) controller. They showed that the hybrid algorithm could achieve better performance than either the DQN algorithm or the PID controller alone on the cart-pole problem.

In summary, the cart-pole balancing problem has been extensively studied in the literature, and the DQN algorithm has shown promising results for solving the problem. Modifications and improvements to the DQN algorithm, such as Double Q-learning, prioritized experience replay, state prediction, and reward shaping, have been proposed and have the potential to improve the performance of the algorithm on the cart-pole problem.

3. THEORETICAL BACKGROUND

3.1. General Synopsis

The cartpole problem is a classic example of a control problem in which the goal is to balance a pole on a cart by controlling the movement of the cart. It is a challenging problem for traditional control methods due to the non-linear dynamics and high-dimensional state space of the system. Reinforcement learning is a powerful approach for solving control problems, as it enables an agent to learn optimal actions through trial-and-error interactions with the environment.

Deep Q-Networks (DQN) is a popular method in reinforcement learning that uses a neural network to estimate the Q-value function, which determines the expected future reward for taking a specific action in a given state. The Q-value function is defined as:

$$Q(s,a) = r(s,a) + \gamma \max_{a'} Q(s',a')$$

where s is the current state of the environment, a is the action taken by the agent, $r(s,a)$ is the immediate reward received by the agent for taking action a in state s, s' is the next state of the environment, γ is a discount factor that determines the importance of future rewards, and $\max_{a'} Q(s',a')$ is the maximum Q-value for the next state s'.

The DQN algorithm uses a neural network to approximate the Q-value function. The network takes the current state of the environment as input and produces Q-values for each possible action as output. The agent selects the action with the highest Q-value and receives the corresponding reward. The neural network is trained using a variant of stochastic gradient descent known as Q-learning. The algorithm updates the Q-values in the neural network using the Bellman equation, which states that the optimal Q-value for a given state-action pair is equal to the immediate reward plus the maximum Q-value for the next state:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

where α is the learning rate, which determines the rate at which the Q-values are updated.

To improve the stability and convergence of the DQN algorithm, several modifications have

been proposed, including experience replay and target network. Experience replay involves storing the agent's experiences (state, action, reward, next state) in a replay buffer and sampling random batches of experiences for training the neural network. Target network involves using a separate neural network to generate the target Q-values used in the Bellman equation, which reduces the impact of changes in the Q-values during training.

4. METHODOLOGY

1. Environment Setup: The DQN is operated in an environment, here to simulate the cartpole game environment we have used OpenAI gym.

2. State and action space: The state space for a cart pole is represented using four variables: the position and velocity of the cart and the angle and angular velocity of the pole. Similarly, the action space for the game is the set of possible actions that the agent can take in the game. The action space consists of two actions: pushing the cart to the left or to the right.

3. Experience Replay: Some of the experiences from the agent's interaction with the environment during the training are stored and replayed. This allows the agent to learn from past experiences, reducing overfitting and improving efficiency. By breaking the correlation between consecutive samples, experience replay can reduce variance and improve stability in the training process. It also enables efficient use of hardware resources by allowing the agent to learn from a batch of experiences instead of interacting with the environment at every step.

4. Q-Network: A neural network was used to approximate the Q-value of state-action pairs. The Q-value represents the expected future reward of taking an action in a given state.

5. Target Q-Network: Another neural network was used to estimate the target Q-value. The target Q-value is used to calculate the loss function during training.

6. Neural Network Architecture: Here a neural network is used to approximate the Q-value function, which estimates the expected future reward for taking a specific action in a given state. The neural network takes the current state of the environment as input and produces Q-values for each possible action as output. The Q-values are then used by the agent to select the action that maximizes its expected future reward.

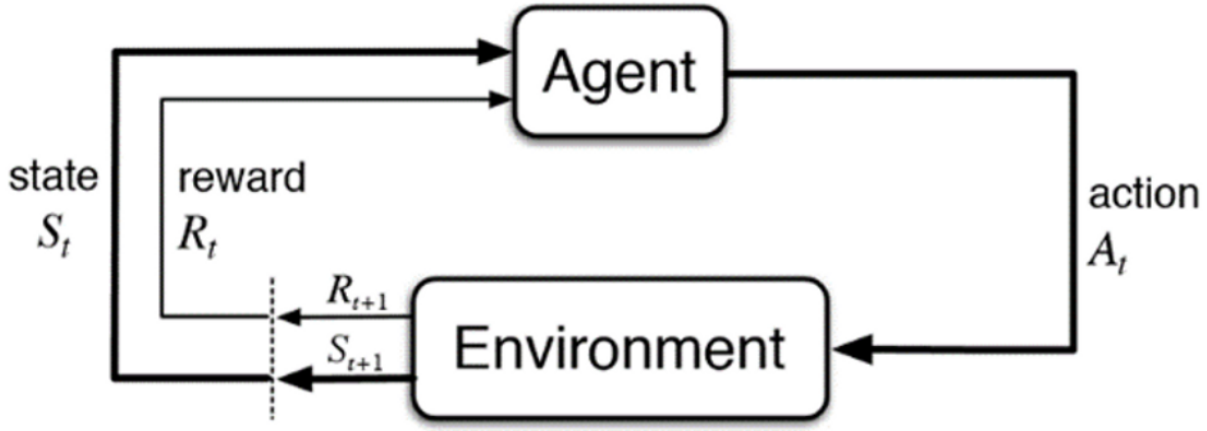


Figure 4.1: Neural Network Architecture

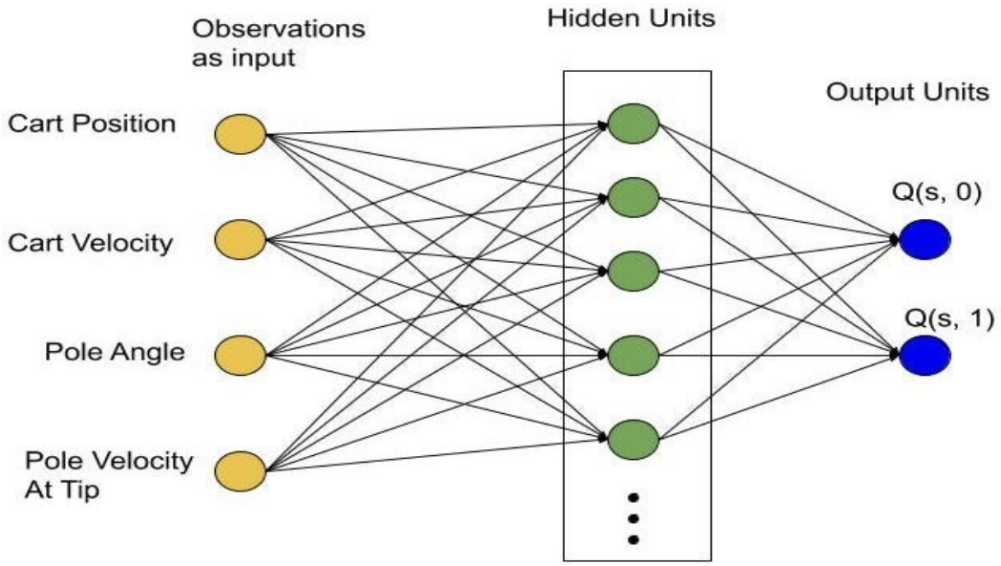


Figure 4.2: DQN Neural Network

The network consists of three fully connected layers. The first layer takes the state size as input and has 64 output neurons. The second layer has 64 input and output neurons. The third layer takes the output from the second layer as input and has the same number of output neurons as the action size.

The activation function used in the hidden layers is Rectified Linear Unit (ReLU),

which introduces non-linearity to the network and helps it to learn complex mappings between inputs and outputs.

7. Loss Function: Here a mean squared error (MSE) loss function is used to minimize the difference between the predicted Q-value and the target Q-value.

8. Gradient Descent: Gradient descent is used to update the weights of the Q-Network. The weights are updated based on the gradients of the loss function with respect to the weights.

9. Exploration vs Exploitation: A hyperparameter epsilon is used to calibrate the exploration vs the exploitation factor. Epsilon is set so that at the start of the training exploration is prioritized over exploitation. As the training goes on exploration gets more encouraged. This is due to the gradual decay in epsilon till it reaches its minimum value.

10. Changing Hyperparameters: Hyperparameters are important parameters that can have a significant impact on the performance of a DQN agent. We change these parameters to obtain the optimal values for this particular solution. As adjusting hyperparameters help the agent learn faster, generalize better to new states, and avoid overfitting. Examples of hyperparameters include learning rate, discount factor, exploration vs. exploitation trade-off, replay buffer size, and target network update frequency. Increasing the exploration rate can help the agent discover new actions, but it can also result in inefficiency. A larger replay buffer size can help the agent learn from a larger set of experiences, but it can increase computational costs. Updating the target network less frequently can lead to faster training but may result in unstable learning.

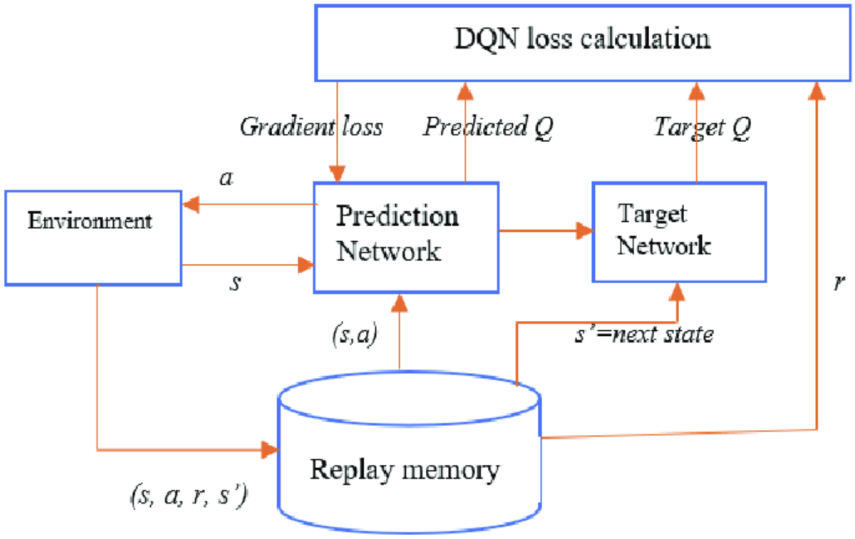


Figure 4.3: DQN Architecture

5. TECHNOLOGIES USED

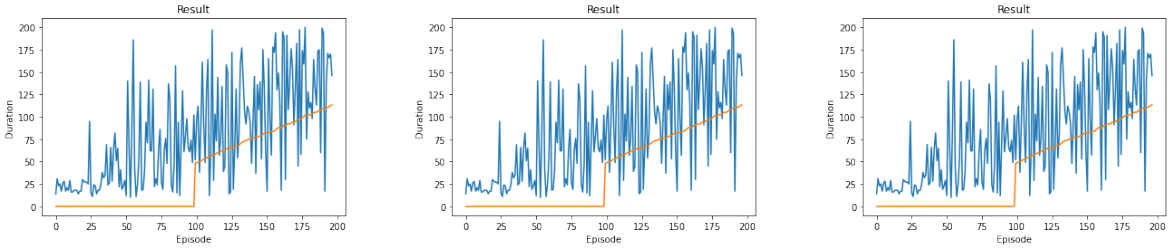
The implementation of the cartpole problem using DQN in this project was achieved using several technologies, including:

- **Gym**: Gym is an open-source toolkit for developing and comparing reinforcement learning algorithms. It provides a set of environments, including the cartpole problem, that can be used for testing and evaluating reinforcement learning algorithms.
- **PyTorch**: PyTorch is a popular open-source machine learning framework that is widely used for deep learning applications. It provides a simple and flexible API for building and training neural networks, making it an ideal choice for implementing the DQN algorithm.
- **Python**: Python is a high-level programming language that is widely used in scientific computing and machine learning. It was used in this project for implementing the DQN algorithm, as well as for writing scripts to preprocess the data and analyze the results.
- **NumPy**: NumPy is a popular library for numerical computing in Python. It provides a set of high-level mathematical functions that are optimized for performance, making it ideal for handling large arrays of data in machine learning applications.
- **Matplotlib**: Matplotlib is a data visualization library in Python that provides a set of tools for creating a wide range of plots and charts. It was used in this project for visualizing the training process and analyzing the results.

In summary, the combination of Gym and PyTorch, along with Python, NumPy, and Matplotlib, provided a powerful set of tools for implementing and evaluating the DQN algorithm for solving the cartpole problem.

6. RESULTS AND DISCUSSIONS

The DQN algorithm was trained and tested on the cart pole problem with different combinations of hyperparameters. The variations included changes to the replay capacity, batch size, epsilon, epsilon decay, gamma, and learning rate to determine the optimal hyperparameters for the problem.



(a) Variation 1 (b) Variation 2 (c) Variation 3

Figure 6.1: Variation of hyperparameters and their effect in DQN training

The first variation had a replay capacity of 10,000, a batch size of 32, an epsilon of 0.1, an epsilon decay of 0.995, a gamma of 0.95, and a learning rate of 0.001. The second variation had a replay capacity of 1,000, a batch size of 128, an epsilon of 0.01, an epsilon decay of 0.999, a gamma of 0.99, and a learning rate of 0.0005. The third variation had a replay capacity of 5,000, a batch size of 16, an epsilon of 0.5, an epsilon decay of 0.9, a gamma of 0.5, and a learning rate of 0.1.

The results showed that the optimal combination of hyperparameters depended on the specific environment and implementation of the DQN. In particular, a replay capacity of 10,000, batch size of 32, epsilon of 0.1, epsilon decay of 0.995, gamma of 0.95, and learning rate of 0.001 produced the best learning performance in the cart pole problem. However, the performance of the agent was found to be sensitive to changes in the hyperparameters, and some combinations of hyperparameters led to suboptimal learning or instability.

In particular, reducing the replay capacity to 1000 or 5000 led to decreased learning performance, as the agent had less experience to learn from and did not generalize as well to new situations. Increasing the batch size to 128 or decreasing it to 16 led to suboptimal performance, as larger batch sizes increased the noise and instability of gradient estimates, while smaller batch sizes led to slower learning due to less efficient use of data. Adjusting the epsilon and epsilon decay parameters also had a significant impact on the learning performance, as a low epsilon value with a high decay rate led to premature convergence to a suboptimal policy, while a high epsilon value with a low decay rate led to slower learning

and exploration.

Overall, the results indicate that careful tuning of hyperparameters is critical for achieving optimal learning performance in the DQN algorithm. The optimal combination of hyperparameters can depend on the specific environment and implementation of the algorithm, and it is important to test and evaluate different combinations to find the best set of parameters. The results of this study provide insights into the sensitivity of the DQN algorithm to different hyperparameters and can guide future research in optimizing hyperparameters for other reinforcement learning problems.

7. CONCLUSION

In this project

[1] [2] [**?**] [3] [4]

References

[1] R. Sutton and A. Barto, "Reinforcement learning: An introduction vol. 1 mit press," *Google Scholar Google Scholar Digital Library Digital Library*, 1998.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[3] M. Roderick, J. MacGlashan, and S. Tellex, "Implementing the deep q-network," *arXiv preprint arXiv:1711.07478*, 2017.

[4] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.