DEEP Q-NETWORK BASED AUTONOMOUS NAVIGATION MOBILE ROBOT

-by Kumar Prasenjeet Sarkar

Abstract

This document discusses the details of the development of a Autonomous Navigation Robot using Reinforcement Learning. As a proof of concept, the robot is made to learn 'Line Following' task using a Deep Q-Network (DQN), an advanced reinforcement learning algorithm. The algorithm combines Q-Learning with deep neural networks, allowing the robot, initially an e-puck in the Webots simulation, to make informed decisions from sensory inputs. This model was adapted to a physical Firebird V robot, showcasing cross-platform adaptability. We achieved sensor-actuator value independence by converting sensory inputs to binary states and actuator outputs to numerical values, making the DQN model versatile across different robotic systems.

.

1. Introduction

Robotics is increasingly integrating machine learning to enhance robot autonomy and intelligence. Reinforcement learning (RL), where an agent learns to perform actions to maximize rewards, is suited for undefined policy tasks. Deep Q-Networks (DQN) merge RL with deep neural networks for advanced decision-making. This project developed a DQN to direct a line follower robot, traditionally following a line on the floor. The project's stages included DQN model training in a simulated environment with the e-puck robot and adaptation to the physical Firebird V robot.

.

2. Background on Reinforcement Learning and DQN

Reinforcement Learning (RL) is a type of machine learning where an agent learns to make decisions by performing actions in an environment to achieve a goal. A DQN is an RL algorithm that uses a deep neural network to approximate the optimal action-value function, which describes the expected cumulative reward of taking an action in a given state.

DQN Model Architecture

The architecture comprises a policy network, target network, and replay buffer. The policy network, a neural network, estimates Q-values for actions based on the current state, driving the agent's decision-making. The target network, mirroring the policy network, stabilizes training by providing target Q-values. The replay buffer stores experiences, enabling learning from various transitions and preventing catastrophic forgetting.

Reward Function

The reward function is the driving force in reinforcement learning (RL), dictating the agent's actions by offering incentives for desirable behaviors and penalties for undesirable ones. For our Deep Q-Network (DQN) based line follower robot, the reward function was meticulously crafted to steer the robot towards following the line consistently and precisely.

In the model training code for the e-puck robot in the Webots simulation environment, the reward mechanism was directly tied to the line-following accuracy, as indicated by the robot's ground sensors. The sensors' readings were binarized to reflect whether the robot was on the line (a binary state of 0) or off it (a binary state of 1). The algorithmic structure of the reward function was as follows:

- **For Perfect Alignment (All Sensors On the Line):**
    - The robot receives a high positive reward (e.g., +50), reinforcing the ideal behavior of staying on the line.
- **For Minor Deviation (One Sensor Off the Line):**
    - A moderate positive reward (e.g., +20) is given, indicating that the robot is close to the line but needs to adjust its trajectory slightly.
- **For Significant Deviation (Two Sensors Off the Line):**
    - A smaller positive reward (e.g., +10) is provided, signaling that the robot is veering off course and needs to correct its path more substantially.
- **For Major Error (All Sensors Off the Line):**
    - A negative reward (e.g., -10) is administered, denoting a critical mistake that necessitates an immediate and significant course correction.

To counteract the robot's potential to engage in suboptimal behavior, such as getting stuck in a corner or continually circling without proper line tracking, the reward function included a penalty mechanism. If the robot remained off the line for an extended duration (as measured by a counter multiplied by the time step), it was deemed to have failed the episode, triggering a reset to the initial state. This reset procedure was crucial in preventing the learning process from stalling and ensuring that the robot did not develop persistent, ineffective behavior patterns.

By adjusting the magnitude of the rewards and penalties during hyperparameter optimization, we were able to fine-tune the model's behavior. This careful calibration ensured that the robot would prioritize staying on the line over all other actions, thereby leading to a more effective and reliable line-following algorithm.

Sensor-Actuator Value Independence

Our DQN model's distinctive feature is its independence from specific sensor-actuator values. Instead of raw sensor data, the DQN receives binary states, and its outputs are mapped to actuator functions. This abstraction allows the model to be transferred to other robots by simply converting new sensor data to the binary state space and adapting the output to the actuators.

Hyperparameter Optimization

Hyperparameters were fine-tuned for optimal performance, including replay buffer size, time-step duration, reward values, learning rate, and episode reset time. These adjustments were crucial for the robot's efficient and accurate performance.

3. Implementation

3.1. Initial Acquisition and Familiarization

3.1.1. Issue of Firebird V Robot from Lab

The project began with obtaining the Firebird V robot from our laboratory, which served as the physical platform for testing and implementation.

3.1.2. Acquaintance with Firebird V Hardware

Understanding the robot's sensor layouts, motor specifications, and onboard processing capabilities was crucial for the subsequent development stages.

3.1.3. Acquaintance with Wrapper Functions for Firebird V

Pre-existing wrapper functions were used to interface with the Firebird V hardware, facilitating an abstraction layer between the hardware commands and our high-level programming logic.

3.2. Initial Testing and Conceptualization

3.2.1. Testing of Simple IF-ELSE Logic Line Follower Controller on Firebird V

A basic line follower algorithm using IF-ELSE logic was implemented to establish a baseline for the robot's line-following capabilities.

3.2.2. Research on Reinforcement Learning Framework PyTorch

An extensive research phase on PyTorch was conducted to leverage its capabilities for our reinforcement learning model.

3.2.3. Conceptualization of DQN Line Follower Logic

The DQN line follower logic was conceptualized, framing the line following task as a reinforcement learning problem.

3.3. Development and Training in Simulation

3.3.1. Coding of First Iteration of DQN Line Follower Logic for Firebird V

The initial version of the DQN logic was developed with the Firebird V's specifications in mind.

3.3.2. Modification of Code for Implementation on e-puck Robot for Webots Simulation

The code was adapted to the e-puck robot within the Webots simulation environment to benefit from the controlled setting for learning and debugging.

3.3.3. Multiple Cycles of Code Iterations and Debugging

The model underwent iterative refinement through coding and debugging cycles to improve its stability and performance.

3.3.4. Initial DQN Line Follower Model on e-puck Made

A working DQN line follower model for the e-puck robot was developed as a precursor to the real-world implementation.

3.4. Overcoming Challenges and Optimization

3.4.1. Model Found Stuck in Suboptimal Behavior

The initial model displayed suboptimal behavior due to inaccurate sensor readings.

3.4.2. Explored Flushing of Sensor Values

Sensor value flushing was explored to address the issue of incorrect readings, but the same was not found to be a viable solution.

3.4.3. Identified Cause of False Sensor Values

The false sensor readings were traced back to issues with the simulation physics not accurately representing the robot's movements.

3.4.4. Hyperparameter Optimization to Elicit Optimal Behaviour

Systematic hyperparameter adjustments were made to encourage the emergence of optimal behaviour, which included

- Increasing Replay Buffer Size such that more samples of the environment are present at any given time for the network to learn from
- Reducing the Time Step such that exploration is carried out at a slower pace
- Increasing the absolute values of Rewards such that there is more contrast between the rewards.

3.5. Final Hyperparameter Adjustments and Model Convergence

Hyperparameters were finely tuned to achieve stable and consistent behaviour from the model.

3.5.1. Reducing the Learning Rate

The learning rate was reduced to ensure that the model updated its policy incrementally.

3.5.2. Fine Tuning Episode Reset Time

The reset time for each training episode was optimized to balance exploration with efficient learning.

3.5.3. Limit Overfitting; Time Bounded Max Episode Duration

A maximum episode duration was set to prevent the model from overfitting to the simulation environment.

3.6. Final Training and Adaptation

3.5.1. Successfully Trained Model for Simple Circular Track

The model was successfully trained on a simple circular track, demonstrating its fundamental line-following ability.

3.6.2. Successfully Trained Model on More Complex Tracks

The training was extended to more complex tracks, testing the model's robustness and adaptability.

3.6.3. Conceptualized Architecture to Migrate onto Firebird V

An architecture to transition the trained model onto the Firebird V platform was developed.

3.6.4. Coding of Script to Run the Trained Model on Firebird V

A script was developed to deploy the trained DQN model on the Firebird V robot.

3.6.5. Testing and Debugging on Real Firebird V Robot

The necessary codes and models developed were converted to ONXX, TensorFLow and TensorFLowLite, so that it works on the Raspberry Pi integrated with FireBird V Robot. However, none of the modules i.e Pytorch, TensorFLow or TensorFLowLite could be installed on Raspberry Pi. Due to this the codes and models developed could not be tested on the physical robot.

4. Conclusion

The development of the Deep Q-Network based line follower robot on the Firebird V platform marks a significant achievement in the application of reinforcement learning to physical robotics systems. This project encapsulated a full spectrum of development phases, from conceptualization to real-world application, with a strong emphasis on iterative learning and improvement.

Through the course of this project, approximately 25% of our collective effort was dedicated to mastering the tools and technologies required for its completion. This involved a deep dive into the intricacies of the PyTorch framework, the mechanics of reinforcement learning, and the specifics of the DQN algorithm. An equal amount of effort, another 25%, was allocated to the coding and development of the models. This phase was characterized by rigorous testing and debugging, with each iteration bringing us closer to a viable model.

Hyperparameter optimization was pivotal and consumed another 25% of our resources. It was during this phase that the true potential of the model was unlocked. Through careful tuning of the replay buffer size, time-step duration, and reward values, we were able to refine the model to the point of stable behavior convergence. Adjustments to the learning rate and episode reset time further enhanced the model's performance, preventing overfitting and ensuring that the learned behaviors were both effective and generalizable.

The remaining 25% of our effort was focused on testing the model with the real Firebird V robot. However, the limited capabilities of Raspberry Pi proved to be a hindrance in the installation of necessary modules, for the code to run.00

The project's success can be attributed to the robust architecture of the DQN model, which proved to be adaptable and sensor-actuator value independent. By translating sensor readings into a binary state space and generating actions as real numbers, the model achieved a high degree of robot-agnosticism. This allowed for the trained model, initially developed in a simulation environment with the e-puck robot, to be effectively converted for Firebird V robot.

In conclusion, the project not only achieved its goal of developing a line follower robot using advanced machine learning techniques but also contributed valuable insights into the field of robotics. The knowledge gained from this project regarding the adaptability and transferability of machine learning models holds promising implications for future research and applications in various domains.