Navigating Complex Highway Scenarios with Advanced RL techniques in Highway-Env

Swapneel Wagholikar
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, United States of America
swagholikar@wpi.edu

Bhaavin Jogeshwar
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, United States of America
bjogeshwar@wpi.edu

Chinmayee Prabhakar
Department of Robotics Engineering
Worcester Polytechnic Institute
Worcester, United States of America
cprabhakar@wpi.edu

***Abstract*—This research compares a variety of architectures that use Deep Q-Learning (DQN), Deep Q-Learning Memory Replay (DQN-MR), and Deep Q-Learning Prioritized Experience Replay (DQN-PER), all of which were carried out in discrete Action Space. An autonomous vehicle decision-making environment, highway-v0, is used for this study. This study shows that an algorithm's performance depends on the kind of environmental observation retrieved. The discrete action space algorithms were evaluated for performance, and DQN-PER was on top. Several challenges were faced with the reward distribution and environmental setup, which were soon resolved, and we arrived at satisfactory results.**

***Index Terms*—DQN, highway env, discrete action space, RL**

I. Introduction

Combining Deep Learning with Reinforcement Learning techniques has created a practical framework for learning complicated decision-making procedures in high-dimensional environments. Reinforcement learning addresses the issue of a learning agent being put in a setting to accomplish a task. Autonomous driving is a technology that has the potential to improve road safety, reduce traffic congestion, use less fuel, and free up human drivers. Making decisions is crucial to the autonomous driving architecture required to achieve autonomy. In order to drive safely and correctly in an urban setting without colliding with other agents, autonomous automobiles must make the right judgments. Rule-based and imitation-based algorithms make up the autonomous driving decision-making framework. Decision-making for autonomous driving also uses learning-based approaches like reinforcement learning and machine learning. We investigated Deep Reinforcement learning for autonomous vehicle decision-making and tested our systems using the OpenAI Gym-based Highway Environment [1]. The following algorithms were chosen for implementation:

- Deep Q-Learning (DQN) in discrete action space
- Deep Q-Learning Memory Replay (DQN-MR) in discrete action space
- Deep Q-Learning Prioritized Experience Replay (DQN-PER) in discrete action space

II. Related Work

Mnih et al. (2013) proposed the original DQN algorithm, which combined Q-learning with deep neural networks to learn a control policy for playing Atari games. The authors used a simple experience replay mechanism to store and sample transitions from the agent's experience.

Schaul et al. (2016) introduced DQN-MR, which improved upon the original DQN algorithm by incorporating a more efficient and effective memory replay strategy. The authors proposed a prioritized sampling scheme that used a rank-based prioritization method to sample experiences from the replay buffer.

Schaul et al. (2016) also introduced DQN-PER, which further improved upon DQN-MR by introducing a more sophisticated prioritization scheme based on the magnitude of the temporal difference error. The authors proposed a power-based prioritization scheme that assigns a higher priority to experiences with a higher impact on the Q-values.

Wang et al. (2016) proposed Dueling Network Architectures for Deep Reinforcement Learning, which introduced a novel network architecture that separates the estimation of the state value and the advantage for each action. The authors showed that this architecture improved the learning efficiency of the DQN algorithm.

Hessel et al. (2018) proposed Rainbow, which combines several extensions to the DQN algorithm, including DQN-MR, DQN-PER, dueling networks, multi-step returns, distributional RL, and noisy networks. The authors showed that Rainbow achieved state-of-the-art performance on several Atari games.

Van Hasselt et al. (2015) proposed Double Q-learning, which addresses the overestimation bias of the Q-learning algorithm by using two Q-functions instead of one. The authors showed that Double Q-learning improved the performance of the DQN algorithm on several Atari games.

Schaul et al. (2015) proposed Prioritized Experience Replay for Deep Reinforcement Learning, which introduced prioritized experience replay as a general technique for improving the efficiency of learning in deep reinforcement learning. The authors showed that prioritized experience replay improved the performance of the DQN algorithm on several Atari games.

III. Methodology

A. Deep Q-Network (DQN)

DQN (Deep Q-Network) is a variant of Q-Learning, a reinforcement learning algorithm that uses Q-values to determine the optimal policy for an agent. DQN improves upon Q-Learning by using a deep neural network to estimate Q-values, allowing the algorithm to learn from high-dimensional inputs such as images from a camera in an autonomous vehicle.

Lane switching is an important task for autonomous vehicles that requires the vehicle to make a decision on when and how to switch lanes safely and efficiently. DQN can be used to train an agent to make these decisions by defining the state of the agent as the current lane, the speed of the vehicle, the distance to the vehicle in front and behind, and other relevant information. The actions available to the agent can be defined as switching to the left or right lane, staying in the current lane, and adjusting the speed. The reward function can be designed to encourage safe and efficient lane switching.

During training, the agent interacts with the environment, observes the current state, and selects an action based on the Q-values estimated by the neural network. The agent receives a reward based on the action taken and the resulting state, and the experience tuple (state, action, reward, next state) is stored in the replay buffer.

$$\begin{aligned} target\ Q - value = &\ reward \\ &+ \gamma * max(Q(next\ state, all\ actions)) \end{aligned}$$

The Q-network is updated using gradient descent to minimize the mean squared error between the predicted Q-value and the target Q-value.

Once the Q-network is trained, the agent can use it to make decisions on when and how to switch lanes based on the current state. Overall, DQN can be an effective approach for training autonomous vehicles to perform lane switching safely and efficiently.

B. Deep Q-Network Memory Replay (DQN-MR)

We have implemented this RL algorithm using PyTorch and the Highway-Env environment.

The essential libraries first imported into the code are the gym, PyTorch, and the highway env module. Then, a class called Replay Memory is developed to store transitions. Later, these transitions are repeated to compute loss on batches without linked states.

The difference between the current and previous screen patches is used as the input by a DQN class, which outputs various actions. Three convolutional layers are used in this class's forward method, followed by a fully connected layer.

The duration of each episode is plotted as a function of episode number using the plot durations function. Additionally, a get screen function is defined, returning a screen requested by the gym as a numpy array.

To pick an action, use the select action function. The action with the highest expected reward is selected, using the epsilon-greedy method, with probability (1-EPS START), while a random action is selected with probability EPS START.

The loss is calculated, and the model weights are optimized using the optimize model function. The function exits if the memory buffer has fewer transitions than BATCH SIZE. If there are enough transitions in the buffer, a batch is sampled. The batch items are concatenated with the non-final state mask once calculated. The mean squared error between the actual and expected Q-values is then used to determine the loss. The Adam optimizer is then used to improve the model weights.

C. Deep Q-Network Prioritized Experience Replay (DQN-PER)

Using PyTorch and the highway-v0 environment, we have put the Prioritized Experience Replay (PER) algorithm into practice. In the highway-v0 environment, a simulation of a vehicle race, CNN models can complete tasks just by observing the surroundings. A portion of the screen is used by the algorithm as an input for the CNN models. A memory buffer replay technique that prioritizes experiences is called the PER algorithm. It is used to calculate the loss on batches with prioritized experiences and no associated states. The DQN algorithm is set up to use the distinction between the most recent and earlier screen patches as input and produce various results.

The PER class is intended to build a replay from a memory buffer that prioritizes experiences. To store and retrieve the transitions with priority, it includes push and sample methods. The transitions' priorities are updated using the update priorities function.

Three convolutional layers and one linear layer make up the CNN model that the DQN class is defined to produce. 16, 32, and 32 filters are present in the three convolutional layers, correspondingly. The action values are generated by the linear layer using the convolutional layers' output. The difference between the patches on the current and previous screens serves as the input.

The beta value for PER is calculated using the beta by frame technique, which is specified. The frame index is inputted, and the beta value is output. The beginning and final beta values are determined by the variables beta start and beta frames.

IV. Environment

The OpenAI Gym third-party environment highway-env (1) is utilized to implement the autonomous driving decision-making challenge. This environment is made up of several scenarios, each with its own special circumstances and continuous or discrete action space. The highway-v0 environment that we employed is a straight highway scenario where maintaining speed and continuously changing lanes to pass other vehicles are the major goals.

Observation: The environment exposes 4 observation types:

- Kinematics: This observation provides information on all of the vehicles in the surroundings, including position, velocity, and heading. The features parameter allows the type of features to be set.
- Gray Scale: This observation creates a grayscale picture of the surrounding area using the weights parameter in the RGB to grayscale conversion.

- Occupancy Grid: For constructing an occupancy grid around our agent, this kind of observation produces the properties needed for the grid's participating cars.
- Time to Collision: An array representing the expected time to collision of observed cars traveling on the same route as the agent is provided by this observation.

Each observation has its own benefits and drawbacks. We came to the conclusion after investigating and testing that value-based algorithms perform better with Gray Scale observation types while policy-based algorithms perform better with Kinematics observation types. It should be noted that while a multi-layered neural network may be utilized for all other observations, Gray Scale observations require a convolutional neural network for the agent to understand the observation.

The agent is given the observation after it has been obtained, and the environment provides discrete action space.

Actions: When DiscreteMetaAction is used the actions available are:

- 0 : "Change the lane to one to the left"
- 1 : "Stay idle"
- 2 : "Change the lane to one to the right"
- 3 : "Go Faster"
- 4 : "Go slower"

Reward: The objective is to have the agent concentrate on two jobs at once. The agent should move rapidly and advance while avoiding collisions. Thus, the reward function includes both of the following:

$$R(s,a) = a\frac{v - v_{min}}{v_{max} - v_{min}} - b_{collision}$$

where $v, v_{max}, and\ v_{min}$ represent the ego-vehicle's current, minimum, and maximum speeds, respectively, and are two coefficients.

V. Evaluation Metrics

Deep reinforcement learning algorithms are considerably sensitive to implementation details, hyper-parameters, choice of environments, and even random seeds. The variability in the execution can put reproducibility at stake. So it is necessary to have common evaluation metrics across all the algorithms. In our case, we have used the base default evaluation metrics of highway-env. To evaluate different algorithms, we trained the various algorithms with 1000 episodes and compared the plots of mean reward versus the number of episodes. The plot with the highest average reward and faster learning would be the efficient algorithm.

VI. Results

The graphs in figures 1, 2, and 3 indicate that after training for 1000 episodes, the DQN agent had a mean score of 18.53. On the other hand, the DQN-MR agent had a mean score of 35.79, and the DQN-PER agent had a mean score of 57.76 after training for the same number of episodes.

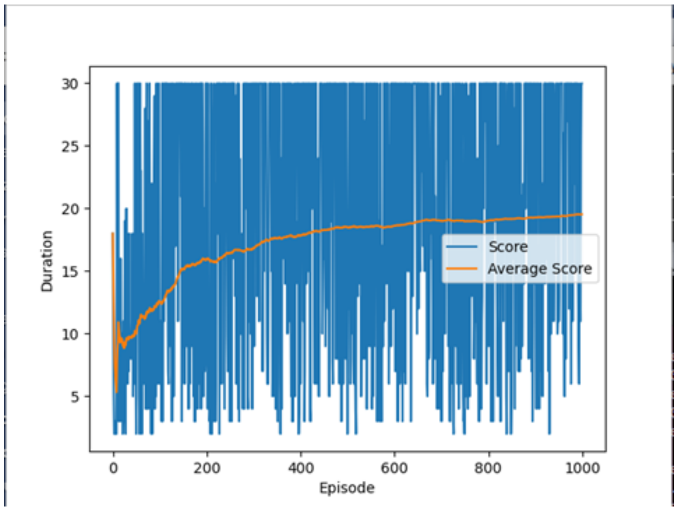

Fig. 1. DQN rewards vs number of episodes

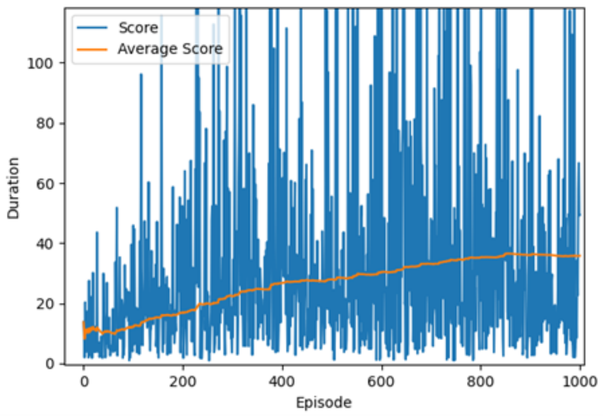

Fig. 2. DQN-MR rewards vs number of episodes

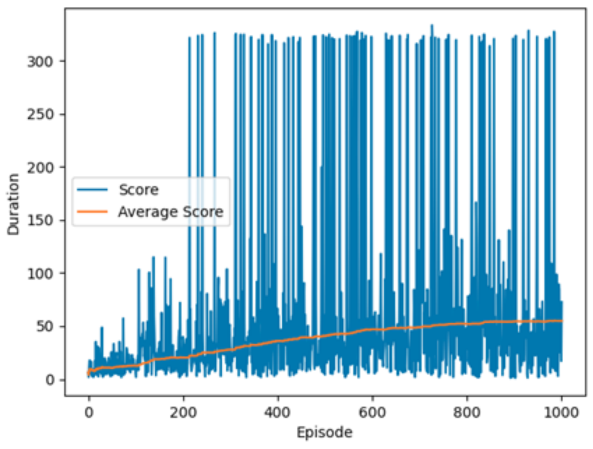

Fig. 3. DQN-PER rewards vs number of episodes

VII. Conclusion

After running simulations with different value-based Deep Reinforcement Learning algorithms such as DQN, DQN-MR, and DQN-PER in a highway setting, we found that DQN-PER with a discrete action space provided higher rewards and better testing performance. To further enhance an agent's decision-making abilities in a discrete action space, we plan to investigate additional methods. Additionally, we aim to explore these algorithms in continuous action space for possible improvements.

References

[1] Mnih, Volodymyr, et al. ”Playing atari with deep reinforcement learning.” arXiv preprint arXiv:1312.5602 (2013).

[2] Wang, Ziyu, et al. ”Dueling network architectures for deep reinforcement learning.” International conference on machine learning. PMLR, 2016.

[3] Liu, Jing, and Yuncheol Kang. ”Automated Cryptocurrency Trading Approach Using Ensemble Deep Reinforcement Learning: Learn to Understand Candlesticks.” Available at SSRN 4348791 (2023).

[4] D'Arcy, Laura, Padraig Corcoran, and Alun Preece. ”Deep Q-Learning for directed acyclic graph generation.” arXiv preprint arXiv:1906.02280 (2019).

[5] Hessel, Matteo, et al. ”Rainbow: Combining improvements in deep reinforcement learning.” Proceedings of the AAAI conference on artificial intelligence. Vol. 32. No. 1. 2018.

[6] Dulac-Arnold, Gabriel, et al. ”Deep reinforcement learning in large discrete action spaces.” arXiv preprint arXiv:1512.07679 (2015).

[7] Schaul, Tom, et al. ”Prioritized experience replay.” arXiv preprint arXiv:1511.05952 (2015).