Q-Learning versus Actor-Critic Performance in 2D Highway Environment

Elida Met-Hoxha
(em744)

Karan Baijal
(kb553)

Abstract

Reinforcement Learning is often used to train self-driving cars to operate under different scenarios. In this paper, we consider the OpenAI gym highway environment and train with both off and on policy algorithms, namely Q-learning and Actor-Critic. We implement and assess the performance, run-time, and stability of both algorithms in the environment. We find that the Actor-Critic Advantage performs achieves higher rewards than the Q-Learning algorithm.

1 Introduction

Autonomous driving is one of the most widely researched areas, with rapid advancements being made every year. To develop and train algorithms that work on the There exists a large number of open-source driving simulators that are often used in this area of research, including programs like CARLA, BARK-ML, DonkeyCar, etc. These are robust simulators which can be used to tackle many areas of autonomous driving research, such as handling a busy intersection or maneuvering around pedestrians. These are complicated issues which often times have very large state spaces. Powerful and effective algorithms are required for such tasks.

Platforms like Gym-AI offer simpler 2D environments which allow for a simplified state space and more manageable and tractable tasks. In this paper, we explore the gym environment *highway-fast-v0* [4] and compare a vehicle's performance when trained on a two distinct Reinforcement Learning (RL) algorithms: Q-learning and Actor-Critic.

RL encompasses a broad spectrum of algorithms designed for decision-making in environments where an agent learns to perform actions through trial and error. On-policy and off-policy are two fundamental approaches within this domain. On-policy algorithms directly learn from the actions taken based on the current policy, thus maintaining a consistency between the policy used for selecting actions and the policy being improved. In contrast, off-policy methods decouple the policy used for action selection from the policy being optimized.

First, we define these algorithms and subsequently explain how they were implemented in this 2D highway environment. Finally, we assess their performance and highlight differences. This comparison is crucial as it provides insights that could influence the selection and application of RL algorithms in not only simulated vehicular environments but also in real-world autonomous driving applications where decision-making in real-time is critical.

2 Problem

We aim to study how on-policy and off-policy compare with each other in the context of self-driving car, specifically driving on a highway. Driving a car on a highway is a dangerous and difficult task consisting of multi-agent interactions, requiring strategic decision-making under varying conditions. In order to test our algorithms, we use the 'Highway-v0' environment developed by the Farama Foundation [4]. The goal of the 'Highway-v0' environment is to drive a care at a high speed without colliding with nearby vehicles. Additionally, driving on the right side of the road is rewarded to drive freely at high speeds. The intricacies of the environment make it an ideal candidate to compare the efficacy of different RL algorithms. We use the 'Highway-fast-v0' environment for training as it has the same structure as 'Highway-v0' but the agents move faster, leading to faster training times and is less computationally-intensive.

Specifically, we compare off-policy Q-learning algorithm with on-policy Advantage Actor Critic (A2C) to investigate which performs optimally in terms of learning speed, stability, and ability to handle complex scenarios presented by the Highway-v0 environment. We aim to determine not only which method yields the highest rewards but also which exhibits robustness against environmental perturbations and adaptability to evolving scenarios. We hypothesise that because of its flexibility and ability to generalize, the Q-learning algorithm will perform better.

3 Approach

Training for the following algorithms is conducted in the *highway-fast-v0* Gym environment. This environment is comprised of a continuous 5*5 state space consisting of the position, orientation, and velocity of each agent on the highway. Additionally, the environment includes a discrete action space, where the agent can perform five different actions: switch to right lane, switch to left lane, stay idle, slow down, and speed up.

3.1 Q-Learning Algorithm

First, recall the Bellman Optimality equation:

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma max_{a'} q_*(s',a')] \tag{1}$$

Here:

- $q_*(s,a)$ represents the optimal action-value function, or Q-function, indicating the expected return for taking action a in state s and then following the optimal policy.
- R_{t+1} is the reward acquired after transitioning to a new state s'.
- γ is the discount factor.
- s' represents the next state
- a' represents any possible action taken from the state s'.
- The expectation averages over all possible next states and rewards, assuming an optimal policy is followed.

Q-learning is an off-policy reinforcement algorithm that utilizes the Bellman optimality equation to estimate the values of the action-value function. The algorithm's primary objective is to ultimately learn a policy that maximizes rewards, or returns. The Q-values are iterated as follows [1]

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)\right] \tag{2}$$

where α is the learning rate, adjusting how much new information impacts the existing Q-value estimates.

When a policy is derived from the Q-values, an action is commonly taken using an ϵ-greedy policy, where ϵ is the probability a random action is taken and $1-\epsilon$ is the probability an action corresponding to a maximum Q-value is taken.

While quite effective for discrete and robust, Q-learning can become computationally expensive for larger or continuous spaces. To address this, function approximation methods like deep learning often have to be integrated with Q-learning. Figure 1 is a version of such a method which trains a neural network to learn the Q-values for state-action pairs.

3.1.1 Model Architecture

This model is based on already existing implementations of DQN in Gym environments as seen in [3] and [4] and [5]. A PyTorch framework is used to define a fully connected neural network for

Algorithm 1 Deep Q-learning with Experience Replay

```
Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize action-value function $Q$ with random weights
for episode $= 1, M$ do
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ do
        With probability $\epsilon$ select a random action $a_t$
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
    end for
end for
```

Figure 1: Pseudocode for Deep Q-Learning Network [5]

the DQN model. This model takes as input the state of the environment and outputs a value for *each* possible action, representing the expected cumulative future reward if said action is taken. An epsilon-greedy strategy is employed for action selection, and it is important to note the probability of choosing a random action decays exponentially from 1.0 to 0.005 over the first half of the training steps. This allows for the agent to strike a balance between *exploration* and *exploitation*, making use of knowledge it has acquired to make more informed decisions as training progresses.

To reduce any correlation in consecutive training samples, a replay memory is implemented, much like the one mentioned in figure 1. Every few steps, a batch of experiences is sampled from the replay memory, which stores some fixed number of recent experiences. Each experience is represented by a tuple containing the current state, action taken, subsequent reward received, the next state, and a boolean indicating whether the episode has ended. The DQN model is updated using this batch, ie. current Q-values from the DQN model are computed as well as loss is calculated. This loss is minimized using the Adam optimizer, which adjusts the model weights through backpropagation.

3.2 Advantage Actor-Critic

To study on-policy performance, we implemented the Advantage Actor-Critic (A2C) algorithm for the 'Highway-v0' environment. We built the Advantage Actor-Critic from the groundup using code inspired by the CS4756 assignment [2].

3.2.1 Model Architecture

Actor-Critic works on the play-off between two policies: the Actor and the Critic. Both components are implemented using neural networks with the following architecture:

- **Actor**: Responsible for policy estimation and improvement of the current policy. The actor network outputs a distribution over possible actions given the current state. It comprises three layers of fully connected networks with 32 hidden units each and ReLU activation functions, followed by a final linear layer that maps to the action space.
- **Critic**: Provides a value estimate of the current state and is used to calculate the advantage function. The critic network mirrors the actor in architecture but has a single output unit from a final softmax function that represents the state value.

Both networks are optimized using the RMSprop algorithm, with separate learning rates for the actor and the critic to improve learning dynamics. The main algorithm of the A2C algorithm is shown in Figure 2. The A2C uses Advantage Estimation to compute the advantages, which help in reducing the variance of the policy gradient estimates.

```
Start with an arbitrary initial policy $\pi_\theta(a \mid s)$
while not converged do
    Roll-out $\pi_\theta(a \mid s)$ to collect trajectories $D = \{s^i, a^i, r^i, s_+^i\}_{i=1}^N$
    Fit value function $\hat{V}^{\pi_\theta}(s^i)$ using TD, i.e. minimize $(r^i + \gamma \hat{V}^{\pi_\theta}(s_+^i) - \hat{V}^{\pi_\theta}(s^i))^2$
    Compute advantage $\hat{A}^{\pi_\theta}(s^i, a^i) = r(s^i, a^i) + \gamma \hat{V}^{\pi_\theta}(s_+^i) - \hat{V}^{\pi_\theta}(s^i)$
    Compute gradient
        $\nabla_\theta J(\theta) = \frac{1}{N}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t^i \mid s_t^i)\, \hat{A}^{\pi_\theta}(s^i, a^i)\right]$ s.t. $KL(\pi(\theta + \Delta\theta) \mid\mid \pi(\theta)) \leq \epsilon$
    Update parameters    $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
```

Figure 2: Pseudocode for Advantage Actor-Critic algorithm [2]

3.2.2 Training Procedure

The reward is discounted at each time step with $gamma$ being 4. The actor learning rate is fixed at 0.0005 and the critic learning rate is set at 0.0025.

The objective is to minimize these losses through gradient descent, effectively training the actor to make more accurate predictions of beneficial actions while training the critic to make more accurate predictions of value estimates. Entropy is added to the actor loss to encourage exploration and improve on our reward values.

4 Results

We train the DQN and A2C model over 35000 time steps. The returns and losses for DQN and A2C are shown in Figure 3 and Figure 4 respectively. In the case of DQN, while the losses appear volatile, they remain within a specific range throughout. In comparison, the Actor-Critic losses are much more stable. The losses rise initially but begin to decay over the number of episodes.

The rewards for DQN and A2C feature a sudden increase over the first couple hundred time-steps and then stabilize. The DQN rewards stabilize at 18 whereas the A2C rewards stabilize around 20. However, the DQN algorithm reaches a stable state sooner than the A2C algorithm. This could be due to the additional entropy term added to the reward function. Noticeably, the Actor-Critic achieves and maintains a higher reward than Q-learning, thus disproving our hypothesis. Additionally, on evaluating the code on the highway-fast-v0 environment, we observed that the A2C algorithm led to better lane changing and seemed to reach higher speeds than the Q-learning implementation.

5 Acknowledgements

The paper was written collectively by both authors. Elida focused on developing the Q-Learning Algorithm while Karan developed the A2C algorithm. We would also like to acknowledge Prof. Sanjiban Choudhury and the teaching assistants of CS4756 for their assistance throughout the semester.

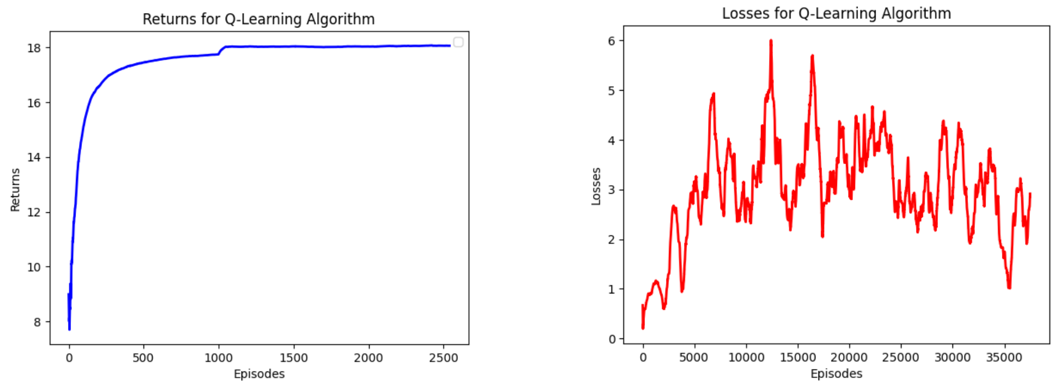

Figure 3: Q-Learning Results

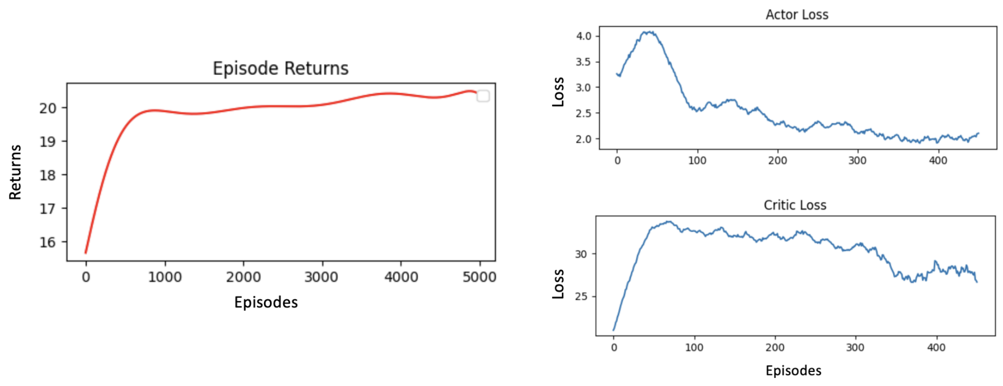

Figure 4: A2C Results

References

[1] Amrani Amine. Q-learning algorithm: From explanation to implementation. Dec 2020.

[2] S. Choudhary. Robot learning. `https://www.cs.cornell.edu/courses/cs4756/2024sp/`, 2024. Accessed: 2024-05-11.

[3] Pradeep Gopal. Implementation of decision making algorithms in openai parking environment, Nov 2020.

[4] Edouard Leurent. An environment for autonomous driving decision-making. `https://github.com/eleurent/highway-env`, 2018.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.