Racing Game with Deep RL

Nicholas Roger ElHabr – 001080240
CS 5180 – Prof. Amato
Monday, December 12, 2022

Abstract:

Our goal is to see whether a lap could be successfully completed by the agent or not, without crashing into obstacles or performing any sudden/time wasting actions while driving, or training/testing losses, and accuracy during training the neural networks in the algorithm. By developing a convolutional neural network as the feature extractor to capture the environment's state, the input image will be divided into several parts, which can capture the environment's different details like road, traffic signs, and land. Then I designed a reinforcement learning algorithm that can learn from the extracted feature, including an actor-critic network which will be used to explore the environment and take the appropriate action. An experience replay buffer stores the previous experiences and uses them for training the neural network which helps the agent remember past experiences and optimize its decision-making process for future actions. The reward system incentivizes the agent to take the right actions to reach the final goal. I trained the model using reinforcement learning to help the agent learn the basics of navigation and improve the decision-making ability. I tested the trained model in an environment and achieved for successful navigation. Our goal is to understand if the reinforcement learning technique chosen is a viable enough algorithm for self-driving cars through the use of the Outrun simulator.

Deep Reinforcement Learning (DRL) is a type of machine learning that combines reinforcement learning and deep learning to allow computers to learn from their experience in an environment. It is an area of artificial intelligence that has been gaining traction in recent years due to its potential to develop autonomous agents that can act intelligently within complex environments. One application of DRL is its use in racing games. In a racing game, the goal is to drive a car around a track as quickly as possible with as few crashes as possible. In order to do this, the agent must learn how to drive in a way that optimizes the car's speed and performance. This requires the agent to be able to make decisions in response to changing environmental conditions, such as the position of other cars on the track and the current track layout. The DRL agent can be taught to play a racing game by providing it with certain rewards for good performance and punishments for bad performance. In this way, the agent can learn over time how to better drive the car and improve its performance. The agent can also be trained to react to the environment, such as maintaining a safe distance from other cars and avoiding collisions. This type of learning can be used to create an agent that can drive a car safely and quickly around a track without human intervention. DRL can also be used to create a racing game that is more challenging for humans. By training the agent to react to the environment in a way that is more sophisticated than a human, the game can become more difficult and require a higher level of skill from the player. Overall, DRL is a powerful tool that can be used to create intelligent agents that can act autonomously in complex environments. Its application in a racing game can help create more challenging and engaging games for players, as well as agents that can drive safely and quickly without human intervention. The development of reinforcement learning has truly changed countless applications of important stature in different domains. From medical to financial based cases, or real estate dip predictions, RL has as an effective way of solving problems, and

it is now clear that these problems are closely related to those formulated as Markov-Decision Processes, and must consider the solution methods of dynamic programming as one of the reinforcement learning methods. Dynamic programming methods are iterative as they increment with each sequence as they gradually reach the correct answer through sequential approximations. I was always interested in playing mind scratching games such as chess, hangman, tic tac toe. I have chosen for my project to see if a Deep RL algorithm can be trained to successfully play the Out Run arcade game. The aim of this project is to apply a Deep Reinforcement Learning technique to a racing game and investigate the performance on autonomous driving tasks, and in return, improve upon our policy implementation. We can judge the result of each action choice based on observing the reward which in our case would be the progression of different time completions. The agent learns optimal behavior by interacting with the environment and attempting to maximize the total reward. actions, *A*, of the agent using TensorFlow, Keras, and an environment simulator for the driving/racing game. The choice of environment is crucially important for this project. OpenAI gym's driving environment which shows a simplistic top-down view of a track was used as a test. Since the Actor-critic methods implement a generalized policy iteration and work toward evaluation and improvement, the actor aims to improve the current policy, while the critic evaluates the policy and we end up seeing better performances than critic-only policies or actor-only policies because the critic, approximates and updates a value function using samples which updates the actor's policy improving in turn, performance, while ensuring convergence.

Algorithm 1 Deep Deterministic Policy Gradient Algorithm

Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ

Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$

Initialize replay buffer *R*

for *episode=1:M* **do**

Initialize a random process $\aleph$ for action exploration

Receive initial observation state s_1

for *t=1:T* **do**

Select action $a_t = \mu(s_t|\theta^\mu) + \aleph_t$ according to the current policy and exploration noise

Execute action a_t and observe reward r_t and observe new state s_{t+1}

Store transition (s_t, a_t, r_t, s_{t+1}) in *R*

Sample a random minibatch of *N* transitions (s_i, a_i, r_i, s_{i+1}) from *R*

Set $y_i = r_i +' (s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$

Update critic by minimizing the loss:

$$L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$$

Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{(s_i,\mu(s_i))} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{(s_i)}$$

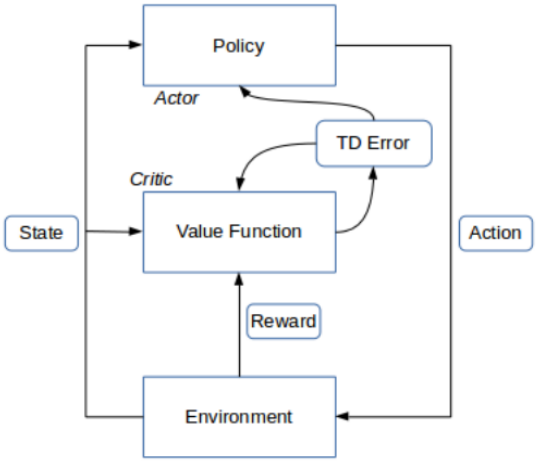
Update the target networks:

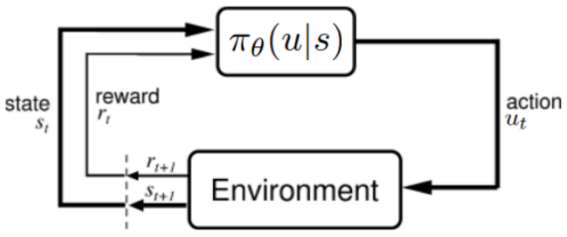
$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

end

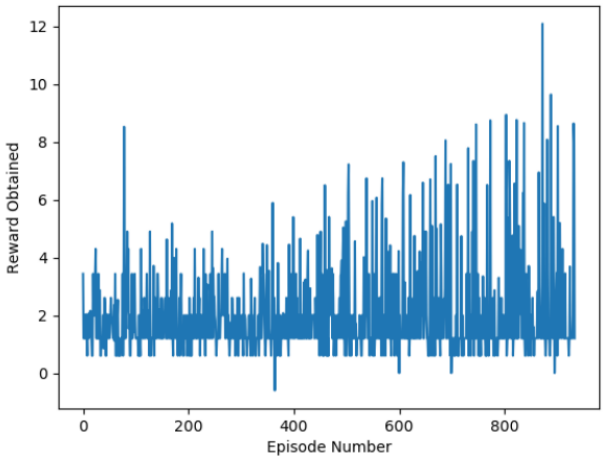
end

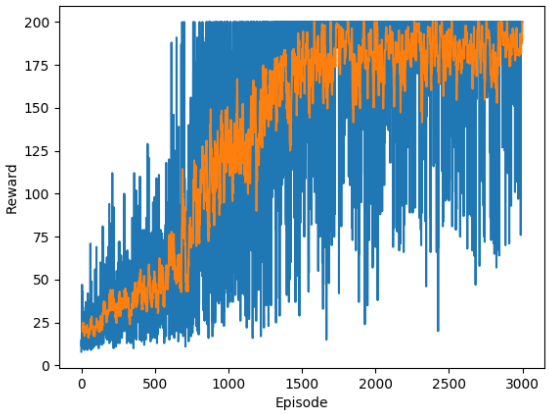


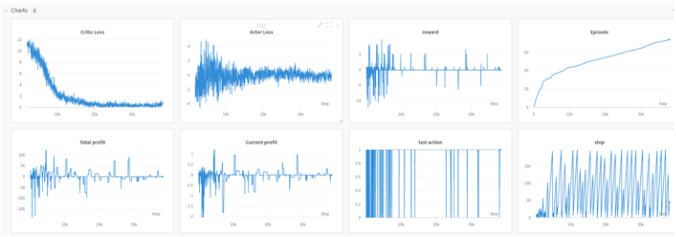


The main challenges of implementing a DRL algorithm on a simulator would be training it on a GPU until the normalized reward vs. steps/episodes converges. Furthermore, choosing a proper actor-network, a critic-network, as well as a proper reward function are the natural challenges of this project. It involves having to properly understand the observations that can be utilized, as well as understanding the scenario. There are few immediate benefits from solving this problem, however, it will show that as a proof of concept, that reinforcement learning algorithms can be utilized for solving simulation games for driving behavior. With proper tuning of reward functions, perhaps, the same approach could be used as one possible algorithm (among an ensemble of algorithms) in actual hardware. Choosing Deep Deterministic Policy Gradient algorithm due to the actor-critic algorithms generally outperforming value-based algorithms, both in terms of the time and the resources required for training the algorithm. DDPG algorithm is an off-policy, model-free technique that builds Deterministic Policy Gradients. It is a policy gradient algorithm that uses a stochastic behavior policy for exploration and estimates a deterministic target policy. It uses two neural networks for the actor and the critic, predicting actions for the current state and generating temporal-difference error signals at each step. The implementation uses a stochastic exploration policy to implement a deterministic target policy. The policy-gradient optimizes a policy end-to-end by using noisy estimates of the gradients of the expected end reward in a policy to update the policy in the direction of the gradient, which in turn updates weights of the actor neural network. The algorithm is evaluated based on the improvements in the rewards received during the training period. A reward function and the appropriate environment structure is implemented from scratch to enable the deep reinforcement algorithm to learn to play the game, and at each step a positive reward equal to the log speed of the car is to be awarded to the algorithm. Deep Neural Networks have been integrated with Reinforcement Learning where the value function, model, or policy can be replaced with a deep neural net. The aim, in this project, is to see if a Deep RL algorithm can be trained to successfully play the Out Run arcade game. By creating an autonomous agent using recognition to deal with identifying components in the environment, which will be the surrounding cars in our case. Prediction for information obtained to predict the future state of the environment. Planning for the creation of an efficient model to plan a series of driving actions for successful navigation which will help the agent achieve human-level control. There are multiple ways to evaluate the algorithm, depending on the exact environment of the game. If it's a racing game, the metric can be the total time taken by the agent to drive around the track or the average speed of the car throughout the lap, as well as whether the agent wins the first place. The measurement for a driving game will be whether a lap could be successfully completed by the agent or not, without crashing into obstacles or performing any dangerous driving actions. Another possible measurement would be training-testing losses, and accuracy during training the neural networks within DRL. A penalty of -0.04 was given for states in which the car was not moving, a penalty of -0.6 when the car went off the track, and -2 for crashes. An end-to-end Convolutional Neural network was trained to map pixels to steering information. It learns to implicitly detect road outlines etc, and also to output an appropriate steering angle given an input road image. However, most algorithms used in real-world driving cars utilize just deep learning. This end-to-end network could be implemented on a expert-driven policy. In the first few episodes of training the rewards obtained by the algorithm are around a total of 2 on average. As the training progresses, and the number of episodes increases, it can be seen that the rewards accumulated by the algorithm at the end of each episode range from about 0 to 12. The

average reward increases to about 6 per episode. Due to time and computing power constraints, I could not train the algorithm for more than 3000 episodes. The game is open-ended with different tracks leading to different environments opposed to just racing around a similar lap which would mean that the algorithm needs to be continuously trained.







We expect the agent to find a policy to maximize expected rewards increases over episodes by making the car go for long distances without crashing into obstacles therefore obtaining a higher score. Implementation of the algorithm in the real world is still a work in progress as the models currently operate in simulated environments. In an algorithm where we took ddpg's trained policy in a dnn, took in images and outputted actions, these actions will take place if and only if an agreement takes place between both the aforementioned parties. In conclusion, we see that Deep Reinforcement Learning techniques can help us curate methods to control and manage automated driving. Some of the difficulties encountered while selecting the algorithm was choosing an appropriate actor critic network given the initial training states, testing observation inputs, and reward functions. By comparing different algorithms such as Asynchronous Actor Critic as well as random policies, the mean, or completion time, we can evaluate better models.

References:

- Kiran BR, Sobh I, Talpaert V, et al (2021) Deep Reinforcement Learning for Autonomous Driving: A Survey. In: arXiv.org. https://arxiv.org/abs/2002.00444. Accessed 12 Dec 2022
- Benam B (2021) Applying of reinforcement learning for self-driving cars. In: Medium. https://towardsdatascience.com/applying-of-reinforcement-learning-for-self-driving-cars-8fd87b255b81. Accessed 12 Dec 2022
- Ho@ng (2020) Deep Reinforcement Learning for Autonomous Vehicles with openai gym, Keras-RL in AirSim Simulator. In: Medium. https://medium.com/analytics-vidhya/deep-reinforcement-learning-for-autonomous-vehicles-with-openai-gym-keras-rl-in-airsim-simulator 196b51f148e4. Accessed 12 Dec 2022
- Cohen J (2021) Deep reinforcement learning for self-driving cars - an intro. In: Medium. https://thinkautonomous.medium.com/de

ep-reinforcement-learning-for-self-driving-cars-an-intro-4c8c08e6d06b . Accessed 12 Dec 2022

- (2018) Reinforcement learning: From grid world to self-driving cars. In: Exxact. https://www.exxactcorp.com/blog/Deep-Learning/reinforcement-learning-from-grid-world-to-self-driving-cars. Accessed 12 Dec 2022
- Barla N (2022) Self-driving cars with Convolutional Neural Networks (CNN). In: neptune.ai. https://neptune.ai/blog/self-driving-cars-with-convolutional-neural-networks-cnn Accessed 12 Dec 2022