Racing Cars

Team Members:

1. Amina Tabassum (NUID: 002190127)
2. Siddhartha Dhar Choudhury (NUID: 001020111)

1. Problem Description

1.1. Problem Statement

Self-driving cars are a result of great technology, and deep reinforcement learning has played a significant role in their success. In this project, we aim to train agents to learn to drive and compete against each other in virtual racetracks. The agents will be presented with several obstacles and have to learn to evade them and other cars in the race to win the race. We also aim to generalize the trained agents (in a finite number of tracks) to virtually any possible circuit that can be created and test the learning algorithms' generalization abilities.

1.2. Environment

We are using the Highway-Env environment. This contains several tasks that can be helpful to train a self-driving car. Our focus will be on the racetrack environments for this project. At each step, the state (observation) that the agent will receive is - the position in the cartesian coordinate system (x, y) and the velocity for each axis (vx, vy). The agent can take three actions at each step: IDLE (no movement), FASTER (increase velocity), and SLOWER (decrease velocity). The reward function is based on the agent's velocity and the chances of collisions and is described by the following function:

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

Where v, v_min, and v_max are the current, minimum, and maximum speed of the vehicle respectively, and a, and b are two coefficients.

1.3. Why is this problem interesting?

This problem is interesting because the promise of self-driving cars is something that has fascinated us for a long time. With the help of deep neural networks and reinforcement learning algorithms, people have made significant progress in this field. This problem is also inspired by Amazon's Deepracer challenge which is an online platform where people can put their RL agents to race against other models (made by other people) in complex circuits. The idea that our algorithms compete against each other and learn without explicit knowledge about driving is fascinating.

1.4. Related work

The highway-env environment is a popular environment and has been used in several projects, here are a few of them:

1. Schott, Lucas, et al. 'Improving Robustness of Deep Reinforcement Learning Agents: Environment Attack Based on the Critic Network'. *2022 International Joint Conference on Neural Networks (IJCNN)*, 2022, pp. 1–8. *IEEE Xplore*, https://doi.org/10.1109/IJCNN55064.2022.9892901.
2. Eysenbach, Benjamin, et al. *Robust Predictable Control*. Sept. 2021. *arxiv.org*, https://doi.org/10.48550/arXiv.2109.03214.
3. Brito, Bruno, et al. *Learning Interaction-Aware Guidance Policies for Motion Planning in Dense Traffic Scenarios*. July 2021. *arxiv.org*, https://doi.org/10.48550/arXiv.2107.04538.
4. Xu, Mengdi, et al. *Scalable Safety-Critical Policy Evaluation with Accelerated Rare Event Sampling*. June 2021. *arxiv.org*, https://doi.org/10.48550/arXiv.2106.10566.
5. Chen, Dong, et al. *Deep Multi-Agent Reinforcement Learning for Highway On-Ramp Merging in Mixed Traffic*. May 2021. *arxiv.org*, https://doi.org/10.48550/arXiv.2105.05701.
6. Zhang, Songan, et al. *Quick Learner Automated Vehicle Adapting Its Roadmanship to Varying Traffic Cultures with Meta Reinforcement Learning*. Apr. 2021. *arxiv.org*, https://doi.org/10.48550/arXiv.2104.08876.
7. Rana, Ashish, and Avleen Malhi. *Building Safer Autonomous Agents by Leveraging Risky Driving Behavior Knowledge*. Mar. 2021. *arxiv.org*, https://doi.org/10.1109/CCCI52664.2021.9583209.

8. Feng, Shuo, et al. ‘Intelligent Driving Intelligence Test for Autonomous Vehicles with Naturalistic and Adversarial Environment’. *Nature Communications*, vol. 12, no. 1, Feb. 2021, p. 748. *www.nature.com*, https://doi.org/10.1038/s41467-021-21007-8.

2. Algorithms

We will perform our experimentation with the following algorithms:

2.1. Proximal Policy Optimization (PPO)

It is a policy gradient method based on online policy and used for both discrete and continuous action spaces. We are using it because it is very robust and hence stability is not affected by the change in data distribution. It is computationally efficient as compared to other algorithms and hence hyperparameters can be easily tuned. It can be adapted to be asynchronous and hence supports handling multiple parallel environments to improve convergence by reducing the correlation between samples. Several applications of this algorithm are Atari games, CartPole, Car Racing, etc.

Although the car racing problem deals with continuous action space, we will implement PPO with discrete action space to make it computationally efficient and optimize it. CNN or fully connected neural networks will be used to train models and finally, hyper-parameters will be tuned for optimal results. Previously, this problem has been solved using fully connected neural networks to train actor-critic models. Discrete PPO was implemented by discretizing action spaces. Two different actor-critic models were trained using CNN and ResNet-18. But, these models suffered from several problems such as hyperparameter tuning, computational efficiency, and several environmental features such as speed, steering angle, and distance from a certain point.

2.2. Deep Q Networks (DQN)

Deep Q Networks is a deep neural network variant of Q Learning. Q Learning is a Temporal Difference method that estimates the Q function based on the current reward and the maximum Q estimate for the next state for all actions. When the number of states and actions increases it becomes infeasible to estimate Q values using the tabular method (TD method) hence the updates are done by a multi-layered perceptron that can estimate the function that maps each state to a particular Q estimate. DQN was originally developed to learn to play Atari games and made a breakthrough in the field of RL by showing promise in the idea of combining deep neural networks as function approximators for estimating value functions in RL. It has been used in several other games like chess, and backgammon and has been applied to high-risk areas like portfolio management.

DQN can be applied by either passing the states in form of the current position and velocity and training a fully-connected network to learn the Q function or passing the current state as an image and training a convolutional neural network (either from scratch or using a pre-trained network). DQN has been applied to gym's race environment but has not been applied to gym's racetrack environment and has achieved great results in a few epochs of training, but it has not been applied to more complex environments like highway env.

Our experimentation can optionally be extended with the following algorithms (since these won't be our primary focus, we omitted the details of usage from the following descriptions):

2.3. REINFORCE

It is one of the policy gradient algorithms based on offline policy. Since this is a Monte-Carlo variant of policy gradients, the trajectory roll-out is performed using the current policy, discounted computed reward is computed and this policy is updated at the end of the episode. Several applications of this algorithm are cart pole, Lunar Lander, and Pong environments. Although, it is not a computationally efficient solution for the car racing problem and it takes very long even for simpler algorithms to converge. We just want to see if we can modify it on several updates to make the algorithm converge faster.

2.4. Advantage Actor Critic (A2C)

It is a hybrid algorithm combining both value-based learning and policy gradients. Algorithms such as REINFORCE use Monte Carlo Updates to update policy parameters and hence introduce high variance and stability problems in update steps. In order to improve the policy gradients i.e, reduce variance, we introduce baseline functions. Advantage actor-critic is one of the baseline functions. So, in this method, the critic estimates the value function, and the actor updates the policy in the direction specified by the critic hence these two functions are parametrized with neural networks. Advantage actor-critic A2C is similar to A3C except for asynchronous implementation.

2.5. Asynchronous Advantage Actor Critic (A3C)

Asynchronous Advantage Actor-Critic is an asynchronous version of A2C (Advantage Actor-Critic) and is an actor-critic technique that involves training two different neural networks. In this algorithm, multiple agents learn their own version of the Q function asynchronously (i.e. they do not depend on each other). Each agent is controlled by a global network. Each agent contributes to this global pool of knowledge, which is then shared among all the agents thus being able to experience and learn from a variety of situations in a short span of time. A3C has been successfully applied to self-driving racing car environments with complex turns and tough obstacles.

2.6. Deep Deterministic Policy Gradients (DDPG)

Deep Deterministic Policy Gradient as the name suggests is a policy gradient method based on a deep neural network that learns both the Q function and a policy. This makes use of off-policy data and updates the Q function using Bellman equations. This interleaves the problem of learning the Q function with that of learning an approximation of the action directly. This uses a separate target network which is updated on regular time steps based on the behavior network. DDPG has been successfully applied to gym’s car racing environment, and people have also applied it to more complex environments with difficult tracks. Although it is primarily used for continuous control problems, it can be extended to be used with problems that have discrete time steps.

3. Results

Our initial goal of this project is to make agents learn to drive and compete against each other in a virtual highway environment evading several obstacles introduced on the way. We, initially, plan to use Proximal Policy Optimization and Deep Q Networks. Other than this, we plan to implement REINFORCE, A2C, A3C, and DDPG, compare the performance of these algorithms for car racing and select the best one. The comparison parameters will be convergence rate, computational efficiency, and accuracy of results. In our case, the agent (racing car) should follow the right direction without slipping from the racetrack and it should be able to recover to track after slipping instead of going in any random direction. One of the challenges can be to ensure it always demonstrates safe behavior (never get off the track and recover to the original position if it slips off). Additional challenges can be agents unable to learn models at complex points such as curves. In this case, we will optimize the algorithm by tuning hyperparameters such as step size in PPO. To improve model learning, we will consider environmental simplicity or improve the number of hidden layers of CNN so that it can extract features such as steering angle and speed or distance from the point it gets off the track. Computational efficiency can also be one of the limiting factors that we will have to deal with.