Autonomous Driving using Reinforcement Learning

Pascu Alexandru, Stan Cătălin, Vișan Alexandru

January 22, 2024

1 Introduction

Autonomous driving is a growing topic among scientists and mass population alike. It describes the idea of vehicles that can navigate autonomously, aided by sensors and artificial intelligence.

In this project, we explore the application of Reinforcement Learning (RL) in the context of autonomous driving. Our objective is to compare the performance of multiple reinforcement learning algorithms in an environment that requires computer-vision techniques.

2 Environment

We will use gymnasium's Car Racing environment [1]. Each observation is a top-down 96x96 RGB image of a car and race track. The observation includes some indicators shown at the bottom of the window.

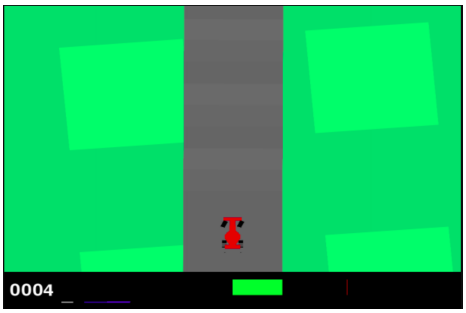

Figure 1: Observation

For every time step, we are penalized with a constant amount of points. We can earn points if we visit new tile tracks. This encourages the model to try to avoid wasting time while visiting as much of the track as possible.

For this project, we chose to solve the "Discrete" action space. This means that there are only 5 possible actions: "do nothing", "steer left", "steer right", "gas", and "brake".

3 Feature Extraction

Our first idea was to manually extract features from the given image at each step. This approach would make it possible to use algorithms such as Q-Learning and Deep Q-Learning on a small feature set before trying to use more complex feature extraction solutions like Convolutional Neural Networks.

Some indicators are rendered at the bottom of the window. We can cheaply extract their values from the image.

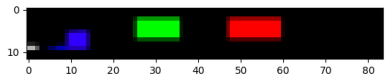

Figure 2: Indicator Bar

After some experimentation, we observed that some of those were not so useful for our task. For example, we can ignore the "Steering" bar, since we control the steering. Also, "ABS" sensors seemed to be strongly correlated with the "Gyroscope" meter, so we ignored them as well.

The indicator bars give us some information about the state of the vehicle, but not its surroundings. We also need to extract information from the actual image of the track.

One such useful information is the distance to the edges of the track. The intuition behind this is that the model will learn to maximize the distance to the left and right so that the car will always follow the center of the road. Also, the model needs to know when to turn, so we also get the distance to the next curve, by casting a ray in front of a car.

Another feature that we can cheaply extract is the vector to the center of mass (COM) of the road. It will be useful in case the car exits the track: we need some way for it to know how to get back.

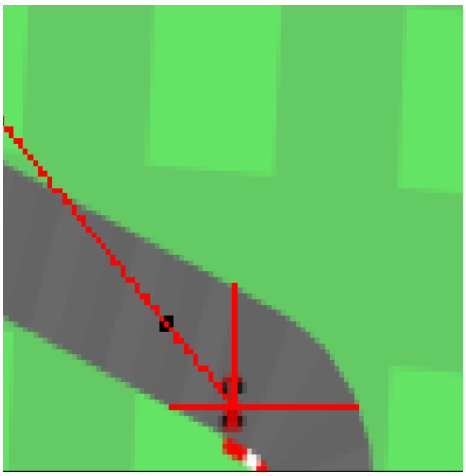

Figure 3: Raycasting

4 Basic Q Learning

We first decided to experiment with the basic Q-learning algorithm, without any kind of function approximation. To do this, we need to convert our feature space to a discrete one. Because we know the interval of the value of each feature, we can simply partition it into a finite set of intervals. Increasing the level of refinement provides greater precision, but grows the size of the Q-table, which may lead to slower convergence.

We have found that a factor $\mu = 10$ works best in our case.

For training the model, we ran 800 episodes with a learning rate of $\alpha = 0.1$ until convergence. We ignored the Gyroscope and COM features because the model would not converge otherwise (keeping those would have grown the size of the Q-table by a factor of 100). The results can be found in 4.

5 Deep Q Learning

There is much room for improvement in the Q-table learning approach. The main problem is that some states rarely get updated during simulations. We need some way of approximating the Q-function so that "meaning" is not lost when converting the feature space to a discrete one (the model should extrapolate!).

We can use neural networks as a way to improve our old approach. It also allows us to use real values (better precision) while keeping memory usage low.

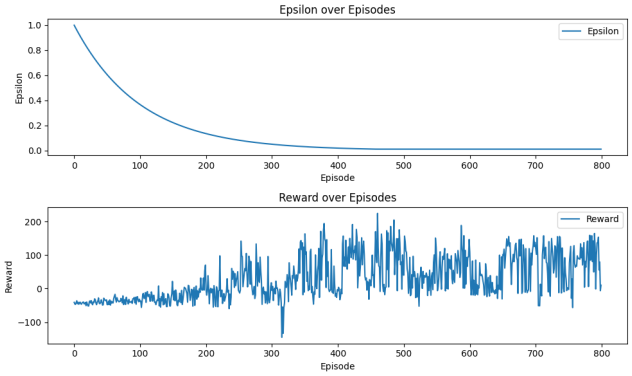

Figure 4: QLearning using manually extracted features

5.1 Using Manually Extracted Features

We chose to use a small neural network with 1 hidden layer, passing our extracted features as an input vector. We trained the model for 230 episodes and noticed that it converged only towards the end, whereas the reward grew significantly on average. The results can be found in 5.

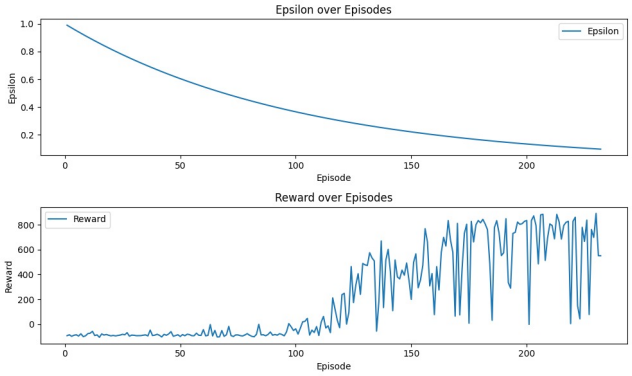

Figure 5: DQN using manually extracted features

5.2 Using Convolutional Networks

Training Deep Q learning takes a lot of time, especially when the observations consist of game-state RGB images, a case that requires convolutional networks that significantly increase the model's parameter count. For this reason, we choose to take advantage of Stable Baselines3 vectorized environments[2]. In a basic environment the agent takes 1 step at a time, with the help of vectorized environments we can train our agent in multiple environments at a time, each one in a separate process. Using this we also improve the exploration of our agent. After training it for $2,000,000$ steps, with a replay memory size of $100,000$ and 16 parallel environments the agent managed to converge to optimal and solve any track.

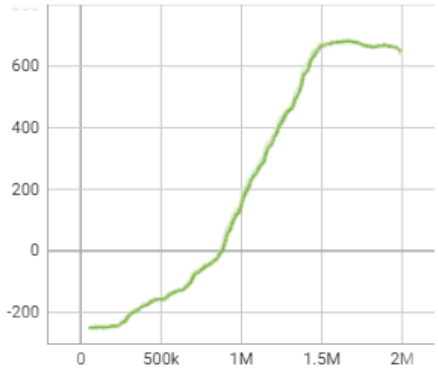

Figure 6: Mean episodes reward

6 Results

CNN RL agent.

References

[1] Gymnasium Car Racing Environment. *https://gymnasium.farama.org/environments/box2d/car_racing/*

[2] Vectorized Environments *https://stable-baselines3.readthedocs.io/en/master/guide/vec_envs.html*