CS230

Race Track Driving Using Deep Reinforcement Learning With Demonstration

Lear Du (06027896), Yi-Ju Chen (06522324), Jou-An Pan (06440768)

Abstract

The purpose of our project is to optimize the action-value function on car racing simulator provided by OpenAI gym using deep reinforcement learning. We model the problem by training the car agent to predict estimated Q value rewards of 12 discrete actions using convolutional neural network. The data was generated by allowing the agent to explore the race track environment. We improved the training data by using expert supported dataset. More specifically we introduced imitation learning techniques such as n-step reward, pretraining and margin loss. Our model with expert demonstration using pretraining was able to improve the baseline in terms of the rewards and success rates. Our results indicate that the expert supported data set is able to achieve 1.75 times higher average rewards after 200 iterations compared to the normal data set. We further demonstrate that revising the Q value loss function to include n-step reward and margin cost helped the agent achieve average rewards of up to 6 times higher compared to the baseline in the first 200 iterations.

1 Introduction

Self driving cars have been an active area of exploration in domains such as transportation as a service and food delivery networks. In this project we explore applications of self driving in autonomous racing using deep reinforcement learning. To simplify the problem space, we selected a OpenAI gym simulator to mimic the track and race car. The input to our algorithm is a $96 \times 96 \times 5$ sequence of grey scaled images of the race track and the output is the estimated reward values for 12 discrete actions the car can perform at a given frame. Because this is posed as a reinforcement learning problem, the car will have to self generate the input data by performing a mixture of random actions and selecting the action that obtain the highest future discounted reward. We developed a baseline model to approximate the reward function using a CNN network. In addition, we experimented with generating expert demonstrated datasets by adding support for user keyboard control in the simulator. The importance of this study is to investigate whether boosting agent self play with expert demonstrations datasets can improve the learning efficiency.

2 Related work

There has been many research work related to path and motion planning for autonomous vehicles. Some rule based algorithms performed a modified A* search algorithm to generate a kinematically feasible path trajectory and then smooth the trajectory using nonlinear optimization. [3]. Supervised learning methods included training a CNN model to generate driving controls by feeding in virtual front view images of human driving [2]. One downside of traditional supervised learning methods is the need for humans to label large amount of training data. Reinforcement learning approaches have been introduced to allow agents to generate new data automatically and receive feedback based on some reward function. For example, some continuous off policy Q learning approach such as deep deterministic policy gradients (DDPG) was used to train race car agents in the TORCS simulator [7]. More recent approaches such as deep imitative reinforcement learning (DIRL) trained agents to drive a simple race track using a mixture of supervised imitation learning and reinforcement learning [1]. Introducing expert demonstration have been shown to speed up the learning process of some deep networks. [4].

3 Dataset and Features

Because the problem was approached using reinforcement learning rather than a traditional supervised learning model, there is no input dataset that can be separated into training, test and validation splits. The input dataset to the reinforcement learning model was obtained through self play and observation of the Car Racing V1 environment provided by OpenAI Gym [6]. Car Racing V1 consists of a race track and a race car that can perform continuous actions in $\mathbb{R}^3$ consisting of $[S, G, B]$. $S \in [-1, 1]$ is the steering angle of the front axis wheels, such that the car is steering left when $S < 0$, steering right when $S > 0$ and going straight when $S = 0$. $G \in [0, 1]$ gives the acceleration value and $B \in [0, 1]$ gives the brake value. During the self exploration phase, the action space was discretized to be $S \in \{-1, 0, 1\}$, $G \in \{0, 1\}$, and $B \in \{0, 0.2\}$. Thus there are 12 total actions in total. Note that since the acceleration the brake actions can cancel each other out, the max brake value was set to be 0.2. This allows the race car to be able to accelerate at a more fine grained value of 0.8.

Each frame of the environment consists of $96 \times 96 \times 3$ RGB image of the racetrack. The images were preprocessed by converting it to greyscale and the pixel values were scaled to be between 0 and 1. Finally, five consecutive frames of the environment were stacked together to form a $96 \times 96 \times 5$ 3D image that was fed into the reinforcement learning model.

As for the reward structure, the race car receives $+1000.0/\text{num\_tiles}$ points every time it overlaps with a new tile. So if the track generated 300 tiles at the initialization of each game the positive rewards are $+3.3$. In addition, we give an extra $1.5\times$ reward multiplier if the racecar gives an acceleration action of 1. Finally, after each passing of a frame the race car loses -0.1 points. This will encourage the agent to move faster through the track and not veer off coarse.

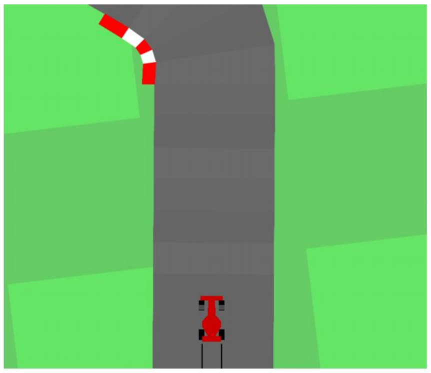

Figure 1: One Frame of CarRacing-V1 environment

4 Methods

Our baseline method uses a standard deep Q learning algorithm as presented in "Playing Atari with Deep Reinforcement Learning" [5]. The main objective of Q learning is to learn the optimal action-value function $Q_{opt}(s, a)$, which denotes the maximum expected return achieved after taking action a at state s and satisfies the Bellman equation:

$$Q(s, a) = \mathbb{E}_{s' \sim \varepsilon}\left[r + \gamma \max_{a'} Q(s', a') | s, a\right]$$

We train the Q function by using functional approximation of the Q function parametrized by ω as $Q_{opt}(s, a; \omega)$, and minimizing the empirical squared loss function as follows

$$J_{DQ}(Q) = \min_{\omega} \sum_{s,a,r,s'} \left[Q(s, a; \omega) - \left(r + \gamma \max_{a' \in actions(s')} Q(s', a'; \omega) \right) \right]^2 ,$$

After each iteration, agent replay will minimize the sum of losses over a randomly sampled batch of s, a, r, s' datapoints from experience replay. This method is model free since it solves the problem without explicitly constructing an estimate of the state distribution ε but rather uses empirical distribution of sample paths. Additionally, the algorithm utilizes an off policy approach in which a random action is execuated with probability ϵ, that starts at 1.0 and decays gradually with a decay rate of 0.9995 as the model learns the environment. Finally for backpropogation, we employed a target network for the second Q variable that updates itself to match the primary network every 5 frames. This will ensure the target reward does not fluctuate too much and stabilizes the gradient descent process.

A $96 \times 96 \times 5$ sequence of images is fed into the Q function approximator. It is then passed through a Conv2d layer with 12 filters, kernel size (7×7), stride 3, and relu activation. It is then passed through a max pooling layer of size 2. This is repeated again with same Conv2d and max pooling layer, before it is flattened and passed to a dense layer of size 216 and relu activation. Finally, it is passed through through dense layer of size 12 which represents the Q values for each combination of actions.

As presented in "Deep Q-Learning from Demonstrations" [4], the imitation learning model will first initialize the memory replay buffer with the expert demonstration datasets. The existing CNN model will then randomly sample a mini-batch of n transitions from the replay buffer to perform gradient descent on. This is repeated several epochs before the agent begins self play on the environment. New transitions from the agent exploration are mixed with the expert demonstrations in the replay buffer and gradient descent is further performed in each iteration. The demonstration dataset is never removed from the replay buffer and is sampled at a higher frequency by adding sampling priority weights. One thing to note is the cost function is modified to ensure that the agent learns the expert action a_E faster if the sampled transition is from the demonstration dataset. To achieve this a large margin classification loss is added to J_{DQ}:

$$J_E(Q) = \max_a [Q(s, a) + l(a_E, a)] - Q(s, a_E)$$

Here $l(a_E, a)$ is a margin function such that when $a = a_E$ the margin is 0 and some large positive number otherwise. This loss will force the learned Q value of the expert action to be at least a margin value higher than all other actions for the current state. If the input dataset was obtained through agent self play, then the J_E loss is set to zero.

The last feature added was saving the n-step returned reward to the expert replay memory instead of the immediate reward plus the max Q value of the next state as presented in $J_{DQ}(Q)$. The n-step reward is computed by summing the discounted rewards over the next 10 states:

$$Q_n(s, a) = r_t + \gamma r_{t+1} + \cdots + \gamma^{n-1} r_{t+n-1} + \max_a \gamma^n Q(s_{t+n}, a)$$

This will help propagate the values of the expert's trajectory to all the earlier states, allowing the agent estimate a better Q value for the expert trajectory.

5 Experiments/Results/Discussion

For the baseline model, we found a learning rate of 0.001 and a mini-batch size of 16 achieved the highest average rewards. The exploration rate ϵ was set to 1.0 (fully random action) and decayed at a rate of 0.9995 after each step in the environment. Figure 2 shows the moving average (window size 20 iterations) of the training reward and success rate over 750 iterations. Success rate defines the percentage of trials in which the racecar fully completes the track and touches all the tiles in sequential order. The reward curve increased gradually until it peaked around 500 points in 400 iterations. The peak success rate obtained was about 0.37. For comparison a full completion of the track can typically obtain rewards of above 1000.

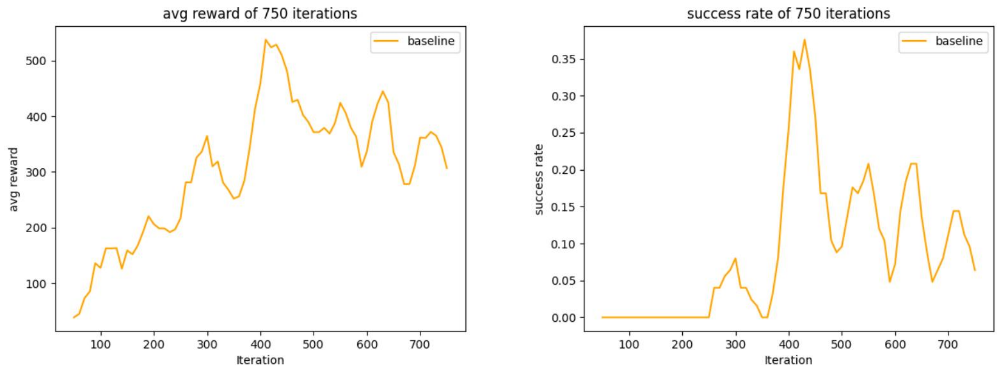

Figure 2: Baseline Average Reward and Success Rate

Next, we experimented with encompassing expert demonstration dataset into our training process. The expert demonstration data included approximately 6000 replay instances (consisting of state, action, reward, and next state tuples) that were created by letting a user input commands to the racecar using a keyboard. The expert demonstration dataset only included successful runs and excluded all instances where the expert made a mistake (e.g. ran off the track). For the first version of the model, a pretraining step using 200 iterations of mini batch size 16 was introduced but the loss function did not include the margin loss $J_E(Q)$ or n-step replay. The expert demonstration data was sampled with a sampling rate of 0.05 (probability of selecting expert memory). Figure 3 shows the average reward and success rate obtained for the expert (orange) and the baseline (red). We observed that expert demonstration model was able to obtain higher average reward (about $1.75\times$ higher after 200 iterations) and success rate early in the training process but the two models eventually converged to similar performance.

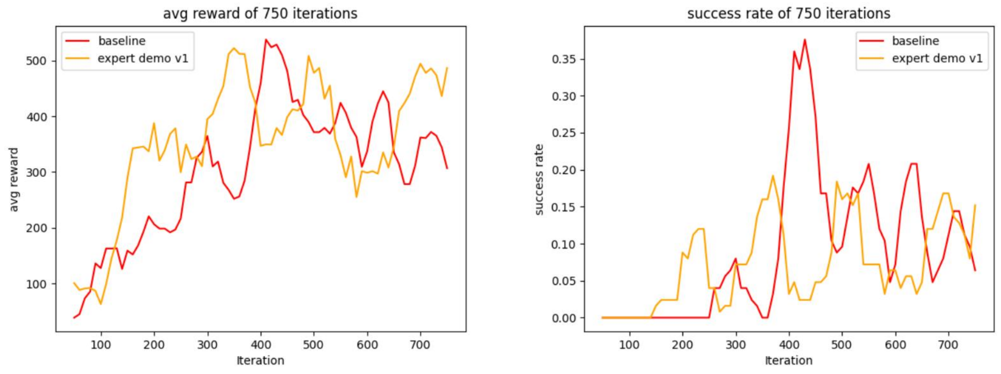

Figure 3: Expert Demonstration Learning First Version

The final model incorporated various methods presented in the paper including n-step reward of length 10, margin cost function of margin size 10.0, and pretraining using 200 iterations of mini batch size 16 using the expert dataset. Additionally the sampling rate of expert dataset was incremented over the first 100 iterations of self play (from 0.05 to 0.15). As shown in figure 4 the expert demonstration dataset achieved an average reward $6\times$ higher than that of the baseline model in the first 50 iterations. The model trained on the expert demonstration also achieved maximum reward that was $1.3\times$ higher than that of the baseline and slightly higher (0.42 vs 0.38) success rate.

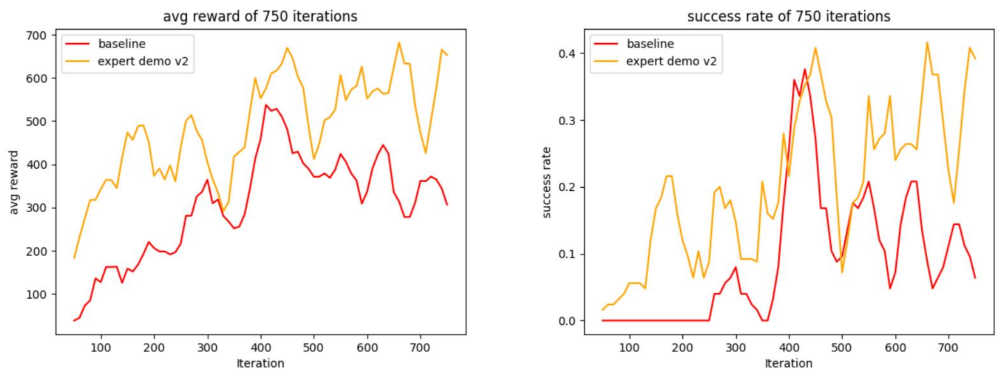

Figure 4: Expert Demonstration Learning Second Version

The most common failure cases for both the expert and the baseline models are sharp corners as shown in figure 5. The race car will tend too overshoot the curves and veer off track. One small difference was the model with expert dataset will approach sharp curves at a lower speed compared with the baseline model. Another observation was the expert action will have more jerky motion compared to the baseline model. For example, when going through a bend the expert outputted action will flip between left steer and right steer for several consecutive frames.

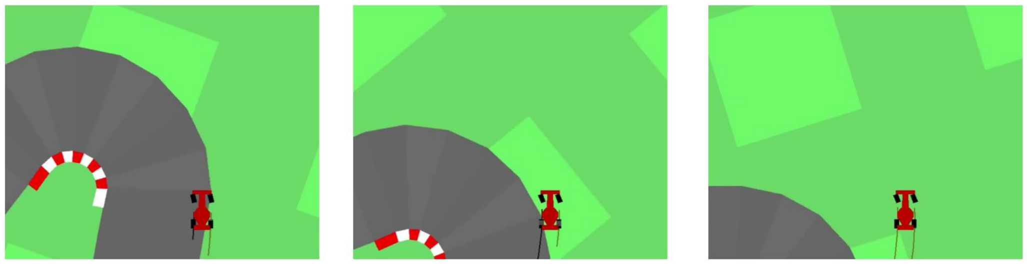

Figure 5: Sharp Corner Scenario

6 Conclusion/Future Work

In conclusion, enhancing agent self play by adding expert demonstration data to the the Car Racing Gym environment boosted the reward score of the agent by up to 6 times in the early parts of the training process. Additionally, the max reward gained and success rate of the agent can also be improved slightly. One downside to adding expert demonstration data was the trajectory planned by the car tended to be less smooth. Additionally expert demonstrated datasets did not help with maneuvering more complicated scenarios such as sharp V corners. Future work could be to introduce a new reward structure that penalizes the agent for changing actions too frequently (to help with smoothness). Another improvement to the model would be to introduce regularization to ensure the performance does not fluctuate too much during the training process.

7 Contributions

Lear Du implemented and trained the model, in additional to contributing to the writing the final report. Yi-Ju Chen performed the metric and error analysis. Jou-An Pan wrote the final report, prepared the final presentation with the video demonstrations.

References

[1] Peide Cai, Hengli Wang, Huaiyang Huang, Yuxuan Liu, and Ming Liu. Vision-based autonomous car racing using deep imitative reinforcement learning. *IEEE Robotics and Automation Letters*, 6(4):7262–7269, 2021.

[2] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE international conference on computer vision*, pages 2722–2730, 2015.

[3] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Practical search techniques in path planning for autonomous driving. *Ann Arbor*, 1001(48105):18–80, 2008.

[4] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

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

[6] Pierre. Car racing v1. `https://github.com/openai/gym/blob/master/gym/envs/box2d/car_racing.py`, 2022.

[7] Sen Wang, Daoyuan Jia, and Xinshuo Weng. Deep reinforcement learning for autonomous driving. *arXiv preprint arXiv:1811.11329*, 2018.