Learning How To Race with Deep Q-Learning

Sean Ye, Patrick Grady

CS 8803 Adaptive Control

1 Problem Statement

We attempt to solve the OpenAI Gym CarRacing-v0 challenge [1]. In this game, the agent must drive a car around a racetrack without crashing. The goal of the game is to complete the track by driving over every tile in the least amount of time possible. For every new tile visited, the agent receives +1000/N reward, where N is the total number of tiles in the track (300). For every frame that passes, the agent receives -0.1 reward. Thus, the agent is incentivized to traverse the course quickly.

Fig. 1 shows an image of the environment. The action space of the environment includes steering, gas, and brake. Each of these actions can accept an analog value at each frame between -1 and 1.

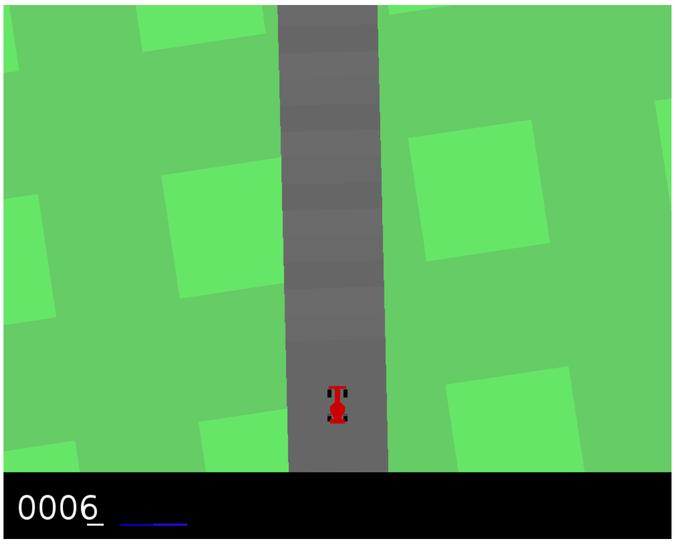


Figure 1: Car Racing OpenAI environment

The simulation features a detailed physics model, which calculates per-wheel slip angle and slip ratio. Each tire's lateral and longitudinal slip is modelled, creating dynamics similar to a real vehicle at high speed. If the vehicle turns when travelling quickly, it can understeer or oversteer and spin. Additionally, the coefficient of friction on the grass is significantly lower than on the track. These dynamics, combined with the time penalty, means the agent's task is not only steering to follow the track, but to control the car's complex dynamics and learn a vehicle feedback loop. Fig. 2 shows an inexpertly controlled vehicle that has spun due to excessive throttle and turning inputs applied at the same time. Achieving a score past 800 requires mastering the dynamics of the vehicle.

2 Deep Q Learning

It is theoretically possible to use traditional Q-learning, and build a Q table mapping states and actions to solve this problem. However, the state space (each pixel) becomes so large that this is intractable.

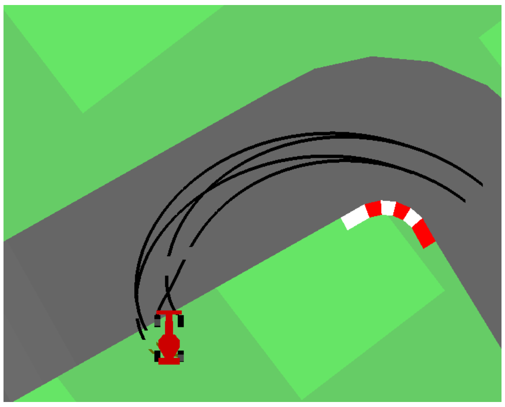


Figure 2: Consequences of driving too fast! Oversteer leading to spin

We instead design an agent based on the seminal Deep Q Learning paper from 2013 [2]. In this work, DeepMind solved Atari games from only high-dimensional pixel input. Their Deep Q Network uses a convolutional neural network to approximate this Q function. CNNs have been demonstrated with good performance on visual tasks such as classification and detection. They leverage this strength with a small CNN to learn the Q value of each action given the current pixels as input.

Because each observation is highly correlated, they also implement an experience replay buffer. This buffer stores a large number of frames from previous rollouts. To train the network, a minibatch of samples is selected from this buffer. Sampling from a long history of policies adds variance and avoids catastrophic forgetting.

In this section, we will describe the state representation, reward function, Q-learning implementation, and general algorithm implemented in our agent.

2.1 State representation

Before the image is sent to the CNN, two steps of preprocessing occur.

First, the image is compressed from a 3-channel color image to grayscale. In this problem, color information is not critical, and this trick reduces the state space by a factor of 3.

Second, the last four frames from the game are concatenated together. This state representation allows the CNN to perceive movement in frames, and thus the speed of the vehicle. Fig. 3 shows an example state representation, with four grayscale images side by side.

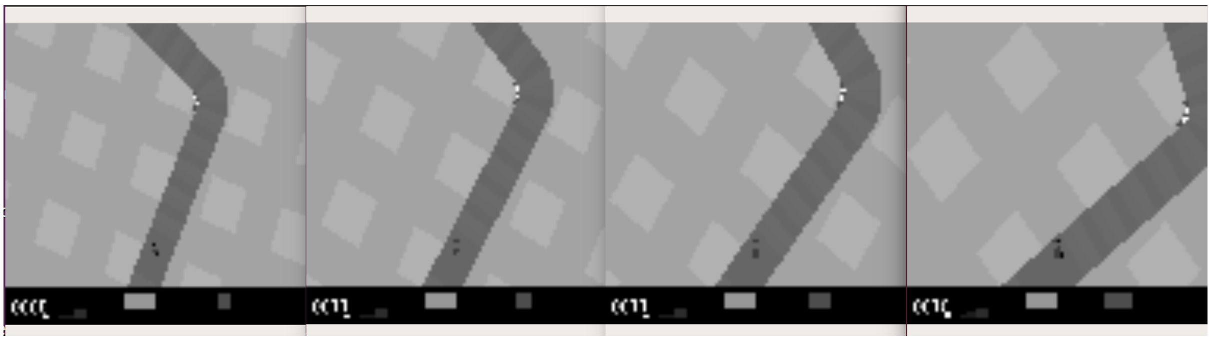


Figure 3: Input Images of State

2.2 Neural Network

A small neural network is used to estimate the Q value at every state. The input to the network is the four sequential 96 x 96 pixel images generated in the previous step. The network has three convolutional layers and two fully-connected layers. This network is significantly smaller than state of the art networks used in computer vision, however this contributes to its ease of training.

The output of the network is five nodes, each representing the Q-value for that action. Fig. 4 shows the architecture of the neural network we used.

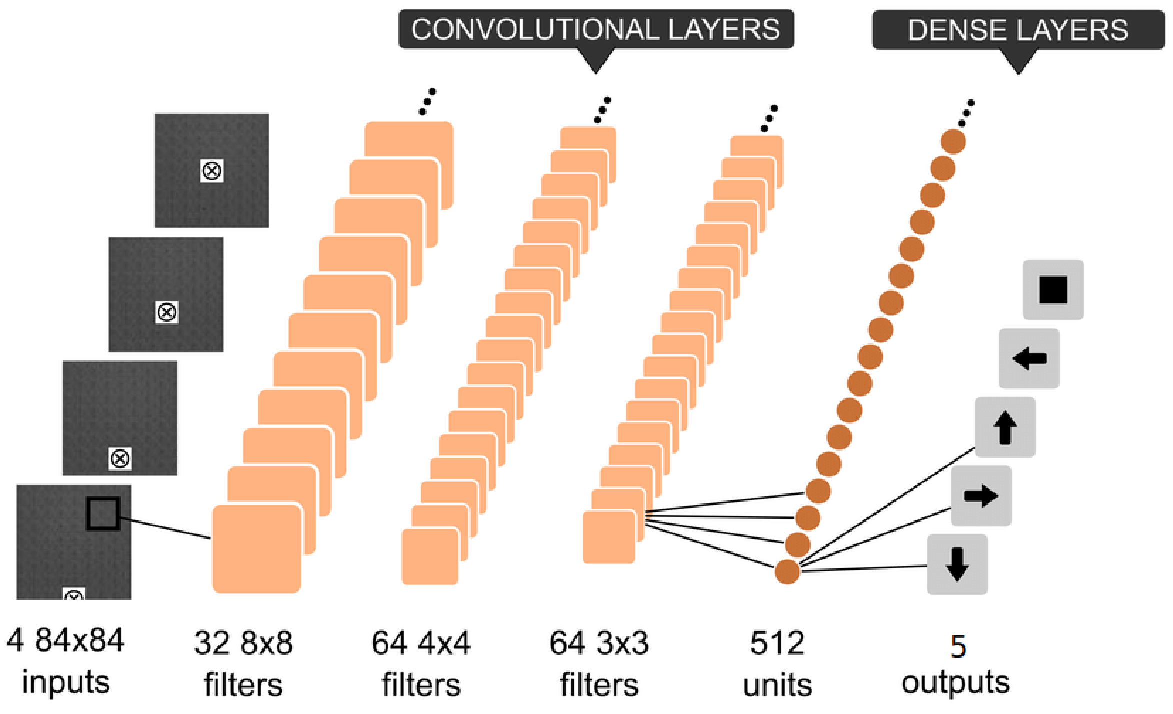


Figure 4: DQN Network Architecture

2.3 Action space

We discretized the action space to five actions: steer left, steer right, gas, brake, and do nothing. This mimics how a human plays the game with binary keyboard inputs. Additionally, this discretization simplifies the learning process and broadens the class of algorithms we can apply.

2.4 Double Deep Q-learning Algorithm with Experience Replay

We implemented the Deep Q-learning algorithm from Deep Mind's Atari paper [2] combined with the Double Deep Q-learning strategy from [4]. The algorithm is reproduced below:

Algorithm 1 Double Deep Q-Learning Algorithm with Experience Replay

```
Initialize action-value function $Q$ with random weights
for episode = 1 to $M$ do
    while Car environment not done do
        Every C steps set $\theta^- = \theta$
        With probability $\epsilon$ select random action $a_t$
        otherwise select $a_t = max_a Q(x_t, a; \theta)$
        Perform action $a_t$ and observe reward $r_t$ and next state $x_{t+1}$
        Store transition $(x_t, a_t, r_t, x_{t+1})$ in replay memory
        Sample random minibatch of transitions $(x_j, a_j, r_j, x_{j+1})$ from replay memory
        Set $y_j = \begin{cases} r_{j+1} + \gamma\, Q(x_{j+1}, \text{argmax}_a\, Q(x_{j+1}, a; \theta); \theta^-) & \text{if } x_{j+1} \text{ is not terminal} \\ r_{j+1}, & \text{otherwise} \end{cases}$
        Perform gradient descent on $(y_j - Q(x_j, a_j; \theta))^2$
    end while
end for
```

The weights of the online network are represented by θ and the weights of the target network are represented by θ^-. The separation of the online and target network allows the learning process to have a fixed target for certain durations. The algorithm also chooses an argmax action of the online network to evaluate in the target q-network. This separates action selection from action evaluation and prevents overestimation of the Q-value. Finally, the experience replay is crucial in saving past experiences to ensure the network doesn't “forget” previous state, action, reward pairs.

3 Accelerating Learning

While the Double Deep Q Network learns a successful policy, it requires a large number of samples. To achieve an average score of 800 (human level), the DDQN requires 1500 training episodes, taking roughly 6 hours to train. This is not prohibitive as the world is simulated, and the only cost of additional episodes is a longer training time. However, if the DDQN were implemented on a real robot, acquiring such a large dataset is impossible. Not only would it take a prohibitive amount of time, but a single crash could destroy the robot.

Thus, we are strongly motivated to increase the “sample-efficiency” of our RL approach. This is one of the primary frontiers in RL research. Real-world adoption is often limited by the number of training examples required.

3.1 Online Imitation Learning

What if we could leverage the experience of an expert to bootstrap our agent's behavior? If an expert is already available (human player or oracle algorithm), their knowledge can be used to teach our agent and speed up learning. The most obvious approach is to directly rollout the expert policy and train the DQN from these examples. However, this results in poor performance as the testing distribution diverges from the training distribution at test time. The training dataset is very narrow, having only explored the expert's actions. When faced with a new state, the agent fails to generalize and crashes.

We instead interleave the expert and the learning policy to learn more robustly. We implement the online imitation learning in an approach very similar to [3]. This approach, based on DAgger [7], executes the expert's policy to drive around the track successfully, but mixes in the current policy's actions. This mixing of policies explores a much broader distribution of states, as the learning agent initially performs random actions. More importantly, the expert policy can "teach" the learner how to recover once the state is disturbed from optimal.

```
Initialize $\mathcal{D} \leftarrow \emptyset$.
Initialize $\hat{\pi}_1$ to any policy in $\Pi$.
**for** $i = 1$ **to** $N$ **do**
    Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$.
    Sample $T$-step trajectories using $\pi_i$.
    Get dataset $\mathcal{D}_i = \{(s, \pi^*(s))\}$ of visited states by $\pi_i$
    and actions given by expert.
    Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \bigcup \mathcal{D}_i$.
    Train classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$.
**end for**
**Return** best $\hat{\pi}_i$ on validation.
```

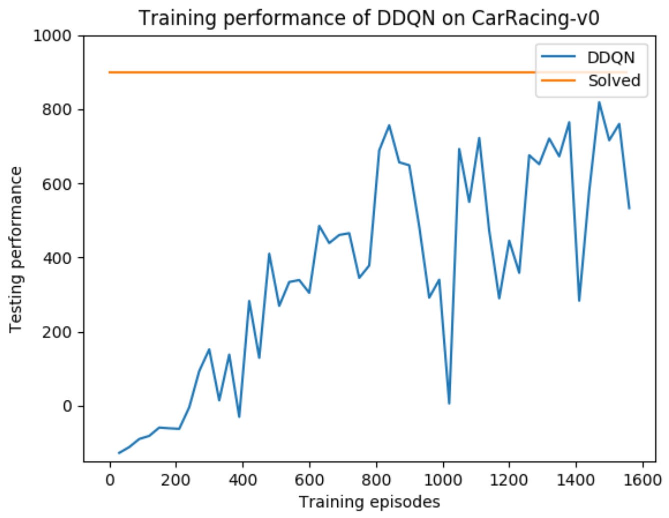
Figure 5: DAgger algorithm

The DAgger algorithm (Figure 5) features a hyper-parameter β, which controls the probability an expert action or learner action is executed. $\beta = 1$ would result in rollouts of only the expert policy, and $\beta = 0.5$ would result in actions that are sampled from the expert and learner each with 0.5 probability. During training, β is slowly reduced to zero.

The expert policy is implemented by reading from the game's state. It implements a pure pursuit controller, driving towards the nearest unvisited tile.

4 Results

We evaluate the performance of the trained models on randomly generated tracks. We find that the agent reaches a maximum score of 817 after 1470 iterations. This surpasses average human performance of roughly 800 points, but falls below OpenAI's threshold of "solved" at 900 points.

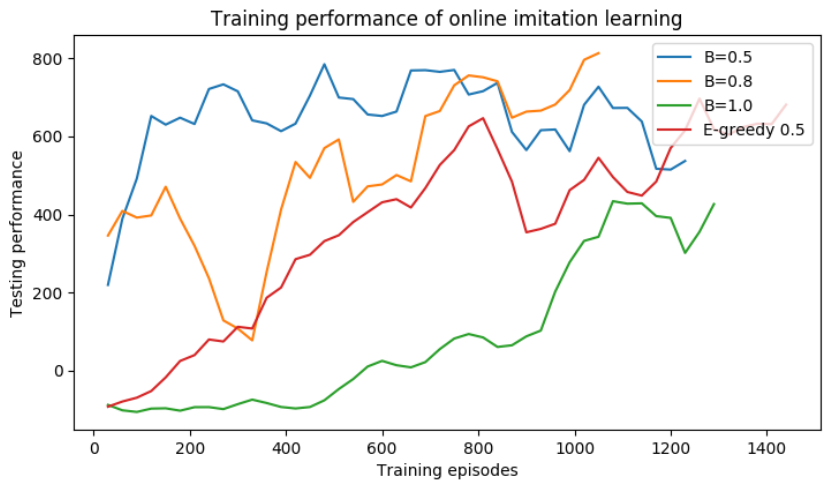


Further, we verify the effectiveness of online imitation learning. We experiment with three

β values, 0.5, 0.8, and 1.0. Additionally, the performance of the baseline DDQN with epsilon-greedy exploration is plotted.

The best imitation learner performs much better than baseline DDQN algorithm. The $\beta = 0.5$ trial reaches a score of 600 after 100 episodes, while the DDQN learner reaches the same performance after 800 episodes. This represents a massive learning rate increase of more than 5x.

Finally, the trial with $\beta = 1.0$ performs poorly, as expected. This agent is trained with only expert actions, and the learner never executes its own actions. Thus, the learner only sees perfect states, and has a very narrow training distribution. It can never learn to recover from a state that the oracle hasn't visited.



5 Conclusion and Future Work

In our work, we implemented double deep q-learning on the car-racing OpenAI Gym environment and achieved scores ranging from 800-900 points. We also implemented an imitation learning technique which drastically improved the training time required for the network to learn the optimal policy. Future work can be done on generalizing the results to different car domains and perhaps using meta-learning strategies so the agent is able to adapt quickly to new environments.

References

[1] https://gym.openai.com/envs/CarRacing-v0/

[2] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari With Deep Reinforcement Learning. In NIPS Deep Learning Workshop.

[3] Pan, Y., Cheng, C. A., Saigol, K., Lee, K., Yan, X., Theodorou, E., & Boots, B. (2018). Agile autonomous driving using end-to-end deep imitation learning. Proceedings of Robotics: Science and Systems. Pittsburgh, Pennsylvania.

[4] Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double Q-Learning. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI'16). AAAI Press 2094-2100.

[5] Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J., & Quillen, D. (2018). Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. The International Journal of Robotics Research, 37(4-5), 421-436.

[6] https://openai.com/five/

[7] Ross, S., Gordon, G., & Bagnell, D. (2011, June). A reduction of imitation learning and structured prediction to no-regret online learning. In Proceedings of the fourteenth international conference on artificial intelligence and statistics (pp. 627-635).