# Deep Learning for Artificial Intelligence in Car Racing Games

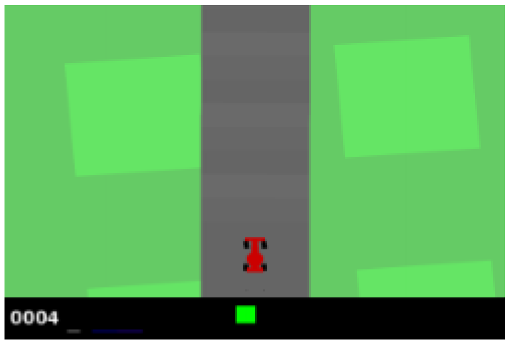
Nicholas Payson, Abhishek Rao, Meng-Hsien (Joey) Tsai, Zakir Makhani

## I. Introduction

This project adapts the TD3 algorithm for a racing-style video game, traditionally controlled via continuous WASD-style inputs, to evaluate its performance in a modified control environment. The agent will have the goal to drive around a track and be rewarded for each portion of the track it reaches. There are three main tasks to complete (creating the environment, creating the model, and integration). This paper describes our approach to setting up the game environment, adapting and integrating the TD3 model for our specific needs, and the collaborative, parallel workflow we adopted for development. By leveraging the strengths of TD3 and modifying it to function within a continuous action framework, we explore new ground in the application of continuous control algorithms to traditionally discrete control tasks, thereby broadening the potential uses of these advanced RL techniques in gaming and simulation. The project will focus on research and the technical aspects of artificial intelligence and we intend for everyone to work on all of the tasks. We believe splitting tasks would be less efficient as we'd be working in sequence rather than in parallel and it'd be better to have input from both those who are focused on game design and those who are focused on artificial intelligence simultaneously.

## II. Environment

Although we initially planned to create our own environment from scratch, we ended up making use of a public environment from gymnasium, "CarRacing-v2", and adding our own modifications on top of it. This environment was convenient for our needs and didn't require a change to our project goals at all, so we used it to save time. The environment has two modes: continuous input where values are used for steering, accelerating, and braking at the same time, and discrete inputs where there are five individual actions available to the agent: accelerate, turn left, turn right, brake, and do nothing. We trained the TD3 model in the continuous environment and set up the player version to run in the discrete environment because driving with WASD is naturally a discrete set of inputs. Other than the ability to turn at different angles rather than full left / full right, the discrete player environment and continuous AI environment are identical. The environment can be rendered in multiple modes, but the easiest for human observation is a top-down RGB picture that centers on the car as it moves around the

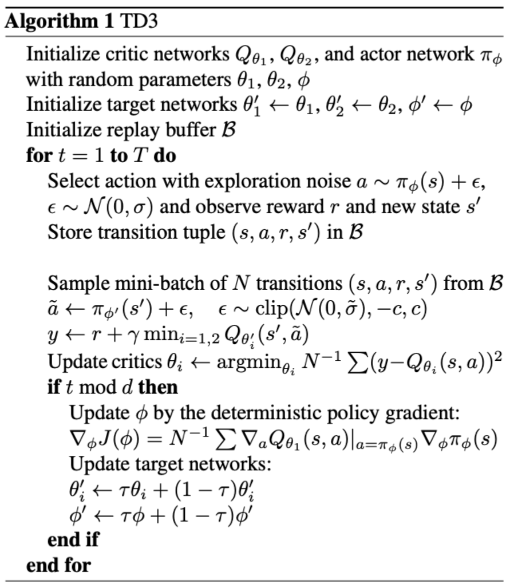


track, pictured left. The AI agent we trained perceives the environment based on the attributes of the car and the pixels around it. Such attributes include: distance to grass tiles, road segment angles ahead, speed, and wheel angles. Such a state contrasts with the typical observation space of a 96x96 RGB image (so typically a state size of 96x96x3). Some of this information is displayed graphically through the indicator bars at the bottom of the window for true speed, ABS sensors, steering wheel position, and a simulated gyroscope. The track layout is always a closed circuit, but the exact dimensions and corners are randomly generated for each run of the environment. The environment calculates reward according to the formula 1000/N (for all tiles visited) - 0.1(#frames), where N is the total number of tiles visited in the track. If a training episode finishes in 732 frames, the calculated reward is 1000 - 0.1*732 = 926.8 points. There is also a -100 penalty for driving out of bounds (the black boundary past the grass). Thus, the agent is incentivized to drive around the track quickly (to minimize loss through frames) and to drive the whole track (to finish the episode, visiting all the tiles). There were some modifications to the step function of the environment where the reward while being on the grass jumps from -0.1 to -0.5, and we terminate on after a threshold of negative reward. Additionally, we added nondeterministic properties to the environments to make the gameplay more interesting for both the AI and a human player. For any given timestep, the car has a chance to turn the opposite direction as intended, as well as a chance of brake failure (note these changes are exclusive so only one or the other could happen).

## III. Twin-Delayed DDPG

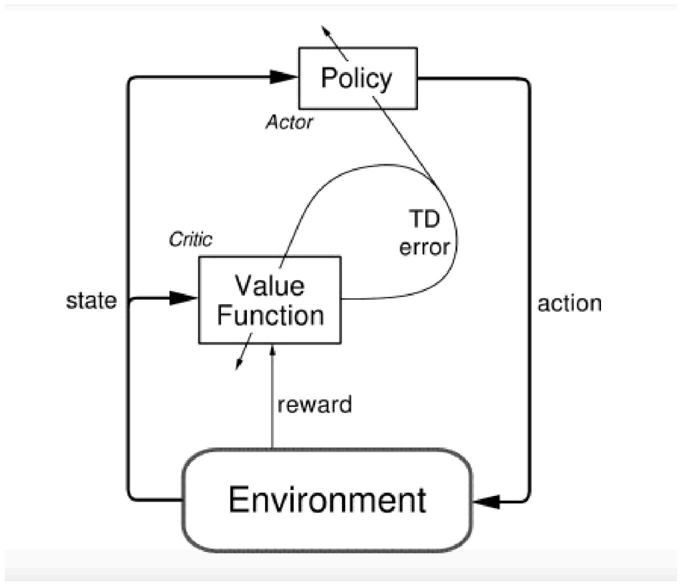
Originally the project was intended to be done using a Deep Q Network (DQN). DQN's almost strictly support discrete action spaces (unless continuous values are encoded), already limiting the ceiling of our model. Furthermore, implementations of our model failed in a peculiar way, only driving the beginning stretch and then immediately going out of bounds. One possible explanation is the overestimation bias prevalent in DQN's. Such bias, compounded with per update error in temporal difference (TD) learning, results in function approximation error in both policy and value updates. The accumulation of these errors ultimately concludes in either a suboptimal policy or (which resembles the case of running straight into the boundary) unintended behavior. For these reasons, we decided to change the implementation of the agent.

Twin Delayed Deep Deterministic Policy Gradient (TD3), provides a significant number of features that mitigate the problems of the DQN approach. TD3 implements a clipped double Q-learning approach, which limits the overestimation bias of as we use the smaller Q estimate when updating the critic. Although this may introduce an underestimation bias, policy updates will not be explicitly affected as opposed to with overestimation bias. In addition the Actor-Critic approach allows for continuous input, providing the agent with more degrees of freedom in its actions. Similar to DQN's, TD3 implements a target network to provide an objective to the agent when learning, also making it more stable. However, Actor-Critic struggles in the event of high-variance policy updates as the evaluation of policy and value are dependent on both being consistently accurate. Thus, to minimize error before a policy update the policy is updated less often than the value network. Finally, TD3 uses target policy smoothing which adds another value to the action selection ($a \sim \pi(s) + \epsilon$). This allows for similar



```
Algorithm 1 TD3
  Initialize critic networks $Q_{\theta_1}, Q_{\theta_2}$, and actor network $\pi_\phi$
  with random parameters $\theta_1, \theta_2, \phi$
  Initialize target networks $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \phi' \leftarrow \phi$
  Initialize replay buffer $\mathcal{B}$
  for $t = 1$ to $T$ do
    Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$,
    $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward $r$ and new state $s'$
    Store transition tuple $(s, a, r, s')$ in $\mathcal{B}$

    Sample mini-batch of $N$ transitions $(s, a, r, s')$ from $\mathcal{B}$
    $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
    $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
    Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
    if $t$ mod $d$ then
      Update $\phi$ by the deterministic policy gradient:
      $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
      Update target networks:
      $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
      $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
    end if
  end for
```



yet different actions to be valued similarly, as well as preventing the agent from exploiting small changes in Q-values (which may result in a suboptimal long-term policy). All of these changes combined allow for significant improvements compared to the DQN implementation.

Our current implementation uses a symmetric architecture for both the actor and the critic, although we have implemented an asymmetrical implementation as well. We also use a replay memory which uses previous experiences the agent has in its runs. Initially, we start with a random policy before letting the agent select actions for itself. We save the agent once it reaches a certain reward threshold (900) when evaluated. The agent was trained on a specific track (seed 123) without the nondeterministic features. Due to time constraints, the current implementation was trained on only about 80000 steps before training was terminated manually. Despite being in a fixed environment, the agent was able to perform well on tracks with different seeds and/or nondeterministic features on.

## IV. Results

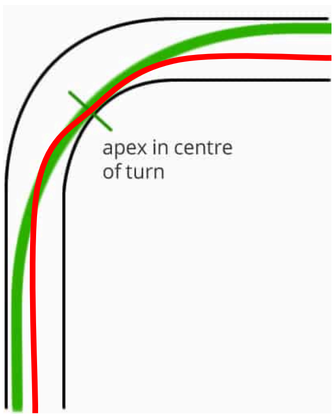
A short recording of the TD3 agent driving a track can be found here. Clearly, the model did an excellent job of learning the environment, as the car drives around the track very quickly.

The agent's driving technique has several interesting quirks when compared to a more human-like style, one of which is a constant jittering of steering inputs on long straights. This behavior is particularly visible from 0:02 - 0:04 in the recording. Our best hypothesis is that the agent has learned that it should keep to the center of the track as nondeterministic steering near the edges of the track increases the risk of touching grass and spinning out. Where humans would drive a straight line down the middle of the road, the agent can't quite nail the required inputs to do that, so it oscillates between two states:

- The agent is left of center (requiring a right steering adjustment)
- The agent is right of center (requiring a left steering adjustment)

In each state, the agent takes the appropriate action to bring itself toward the centerline, but overshoots and enters the other state. The reason the jittering is so fast relative to human course corrections is because the agent can react to every single frame, as opposed to a human which obviously cannot.

The second interesting behavior the agent exhibits is a choice of racing line through the corners of the track. Normally, human drivers take a path similar to the one shown in green in the image below through a corner. This path is called the 'ideal racing line', and takes the shortest path through the turn while maximizing the car's speed, resulting in the car taking the minimum amount of time necessary to make the turn. A human racing driver would take this sort of path, but our agent does not. Instead, our agent takes something closer to the red path through most turns, approaching the turn from the centerline rather than the outside of the curve, turning in later to meet the apex, and only swinging as wide as the centerline on the exit of the turn. We believe the reason for this is that the agent can



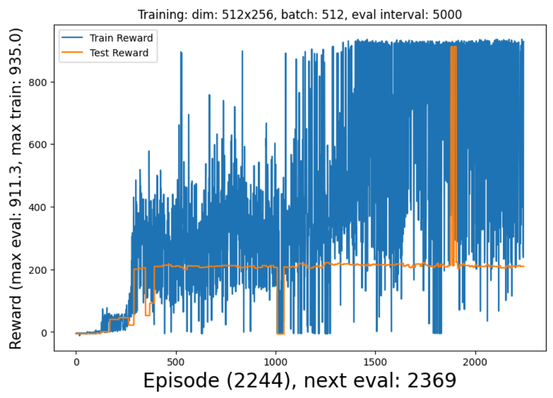
only see a certain distance ahead on the track, so it can’t see the whole corner and therefore know the optimal course through the corner until just before the point at which it starts to turn. Since the agent likes to stay on the centerline on the straights, by the time it knows the exact dimensions of the corner, it is too late to move to the outside of the track to take the optimal racing line. The agent could still use more track on the exit of the corners, but prefers taking the centerline on exit likely because, again, it can’t know exactly what kind of turn may lie just ahead and the center is the best position to be in to react to an unknown turn. The value of this choice can be seen between 0:11 and 0:13 in the recording, when the agent returns to the centerline to quickly prepare for the next upcoming turn.

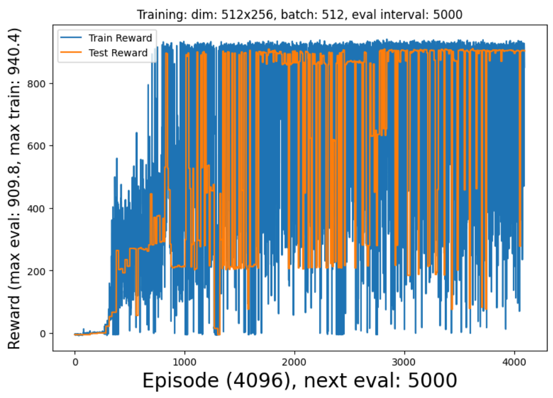
To compare the agent’s performance against a human driver, we compared the agent’s reward score against our own driving on a set track. The agent drove in its continuous environment, while the players drove in the discrete one using WASD.

Agent vs Human Performance Comparison (3 run averages)

| Agent score | Nicholas’ score | Abhishek’s score | Meng-Hsien (Joey)’s score |
|---|---|---|---|
| 911 | 891 | 888 | 898 |

While the reward scores were close, the agent managed to beat three human drivers on the same track, and performed remarkably consistently across its three runs. The following images below are the rewards for the symmetric and asymmetric networks (respectively) during training and evaluation. The agent’s cautious approach, staying in the middle of the track when possible, proved to be an excellent strategy for driving in this nondeterministic environment.





## V. Conclusion

The TD3 algorithm emerges as a formidable reinforcement learning framework, demonstrating remarkable efficacy and accuracy in assessing agent performance while navigating complex environments. Throughout our project, we adapted the TD3 algorithm to train an agent for a racing-style video game, showcasing its potential to excel in both virtual and real-world applications. Our TD3 agent exhibits promising potential in maneuvering autonomous vehicles within the game environment, providing immersive virtual simulations and gaming experiences while also developing safe and cost-effective innovative solutions for non-virtual scenarios. By leveraging the strengths of TD3 and modifying it to function within a continuous action framework, we attempt to broaden the potential uses of advanced reinforcement learning techniques in gaming and simulation.

In terms of applications, we can utilize our model and optimize it to be implemented in other potential use cases that extend far beyond car racing games. From autonomous vehicle navigation and traffic management systems to robotics and logistics optimization, our agent can be adapted to provide insights on various different industries and domains. Other possible implementations for our models could be energy optimization distribution networks or enhancing healthcare operations where through replayability and continuous learning, our TD3 agent can adapt to each dynamic environment to improve decision-making processes. By continuously refining our TD3 agent through extending and iterating current strategies to fit the current needs, we could create more efficient systems that could not impact car racing-games but real-world scenarios.

Reflecting on our findings, we observed interesting behaviors exhibited by the TD3 agent, including its tendency to jitter steering inputs on long straights, where these insights provide valuable results to improve learning and decision-making processes of our TD3 agents. This means that future tunings and iterations of our TD3 agent could focus on refining our current navigational strategies to improve sample efficiency and reliability, while also integrating capabilities to optimize decision-making processes in dynamic environments. For instance, providing the capability for our agents to take the 'ideal racing line' could lead to faster times and more competitive performances within car racing games. Additionally, exploring novel exploration techniques and model architectures could further enhance the adaptability and robustness of our TD3 agent across diverse tasks and environments. By continuously refining our TD3 agent through extending and iterating current strategies, we could create more efficient autonomous vehicle systems not only in the car racing games but for transformative advancements in various different industries and applications.

## VI. Appendix

Fujimoto, Scott et. al. “Addressing Function Approximation Error in Actor-Critic Methods”, Herke van Hoof, David Meger. v3, 2018. https://arxiv.org/pdf/1802.09477.pdf.