Using Deep Reinforcement Learning Models to Learn Autonomous Driving Policies

Eugene Saghi
University of Colorado, Colorado Springs
1420 Austin Bluffs Pkwy
Colorado Springs, CO 80918
esaghi@uccs.edu

Abstract

The development of an Autonomous Driving System (ADS) is an area of research that has received much attention in the past 30 years due to the many benefits that the development of such a system could bring. There are many approaches to creating an ADS which have been explored but the use of Deep Reinforcement Learning (DRL) is a common thread in many of these approaches. This project evaluated the effectiveness of 3 different DRL models to learn optimal driving policies for an agent in the open-source CARLA urban driving simulator. The models studied were a Deep Q-Network (DQN), a Soft Actor-Critic (SAC) network, and a Deterministic Actor-Critic (DAC) network. Of the 3 models the DQN was only model that learned a meaningful policy, although this policy favored only one action regardless of the input state. The SAC and DAC models failed to learn a policy which maximized rewards. Further experimentation is needed to stabilize these actor-critic models and cause them to learn a policy which converges.

Introduction

Autonomous driving is a field of research which has garnered considerable interest over the past 30 years thanks to the many potential benefits (including financial, environmental, and safety benefits) that the development of an ADS could bring. Any successful ADS will need to process a multitude of complex input data, extract meaningful features from this data, and then use these features to make decisions and perform actions. While the number of actions an ADS has available to it are relatively limited, the number of states such a system could find itself in are nearly infinite. This challenge of learning an optimal control policy over a nearly infinite set of possible states makes DRL an ideal paradigm for the development of an ADS.

Background

Automobiles are ubiquitous in everyday life and an integral part of modern cities, with the U.S. Department of Transportation Federal Highway Administration reporting a total of 283,400,986 vehicles registered in the U.S. in 2022[1]. The drawbacks of having so many vehicles so closely integrated in the daily lives are Americans are manifold and include traffic congestion, air pollution, fossil fuel consumption, and of course traffic fatalities. A technical report published by the National Highway Traffic Safety Administration reported that 94% of road accidents are caused by human errors (Yurtsever et al. 2020), and it's probable that in many instances traffic congestion (and the resulting increase in air pollution and fossil fuel consumption it causes) can be directly attributed to such accidents. It is in light of these facts that a great deal of research has gone into the development of ADSs. This research began as early as the 1980s (Yurtsever et al. 2020) and has only intensified as the capabilities of both hardware platforms and software systems (particularly deep learning techniques) have matured.

Broadly speaking, current approaches to ADSs can be broken into two categories: standalone, ego-only approaches and connected multi-agent approaches (Yurtsever et al. 2020). Most research today is being conducted on ego-only approaches for a number of practical reasons, and while multi-agent approaches are still being explored there are currently no functional ADSs that use this modality. ADSs can be categorized along a different axis into either modular systems or end-to-end systems (Yurtsever et al. 2020). In modular systems separate modules handle the tasks of object detection and tracking, behavior prediction, planning/decision making, and vehicle control. In end-to-end systems Deep Learning (DL) is used to directly process input sensor data and produce control signals for the vehicle. DRL has shown great promise for end-to-end ADSs and is even appropriate for some tasks in modular ADSs (such as controller optimization, path planning and trajectory optimization, and dynamic path planning) (Kiran et al. 2021). This project explores the use of a DQN (a type of DRL neural network) to create an end-to-end ego-only ADS in a simulated environment.

Related Work

Because of the promise of an ADS truly independent from human intervention there is a great deal of ongoing research into various applications of DL in the context of ADSs. The following sections provide greater detail about a number of recent research projects that use DL to construct or improve ADSs.

[1] https://www.fhwa.dot.gov/policyinformation/statistics/2022/

Imitation Learning

Imitation learning is a type of DL which in the context of ADSs "maps sensor inputs to vehicle control commands via supervised training on large amounts of human driving data" (Liang et al. 2018). In other words, unlike DRL imitation learning is a form of supervised learning where the optimal policy is learned by observing the actions taken by an "expert" (in this case a human driver) given a particular state. When compared to DRL this approach has both advantages and disadvantages. The primary advantage is the ease of obtaining training data at scale (Codevilla et al. 2018). However, a major drawback to this approach is the fact that long-term driving goals (such as reaching a particular destination) cannot be learned from sensory input alone but instead require a degree of knowledge which is not present in labelled training data. Furthermore it has been found that imitation learning systems generalize poorly to unseen scenarios (Liang et al. 2018).

To overcome these limitations a group of researchers in 2018 proposed using conditional imitation learning for autonomous driving, in which an imitation learning network was given the intentions of the driver during training in addition to the sensory inputs and control commands (Codevilla et al. 2018). During testing these driver intentions were supplemented with similar intentions specific to the testing environment. While this approach showed promise in a virtual environment, the researchers concluded that there remains "significant room for progress" in the development of such an approach (Codevilla et al. 2018). A separate group of researchers in 2018 attempted to combine imitation learning with DRL to overcome the shortcomings of both in an approach they called Controllable Imitative Reinforcement Learning (CIRL). For their experiments an imitation learning network was trained first and then transfer learning was applied to transfer the weights learned during imitation learning to a DRL actor network (Liang et al. 2018). A Deep Deterministic Policy Gradient (DDPG) algorithm was then used in conjunction with a critic network to optimize the policy learned by the actor network. This CIRL network achieved state-of-the-art driving performance in a simulated environment (Liang et al. 2018).

Computer Vision Tasks

Another area of active research in the development of ADSs is designing or improving Computer Vision (CV) systems specifically for autonomous driving tasks. In one such approach, a group of researches proposed a a novel architecture and data generation procedure for enhancing low-light images in an effort to improve the safety of ADSs navigating at night, particularly in rural areas (Li et al. 2021). For this approach the researchers generated training data by artificially darkening daytime images in a 3-step process: a gamma transform, a contrast adjustment, and a histogram matching transformation. This novel data pipeline allowed the researchers to train their network on daytime images from the BDD100K dataset, using the daytime images as ground truth and the transformed images as training data (Li et al. 2021). In addition to this training pipeline the researchers also proposed a novel architecture, which they named LE-net. This architecture was based on an encoder-decoder framework and combined depthwise separable convolution modules, linear bottlenecks, and a feature pyramid module to enhance input images. This model was tested on images captured at night which were then enhanced by the LE-net and it was found that the LE-net achieved state-of-the-art results when evaluated using Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity (SSIM) as metrics (Li et al. 2021).

Object detection is another CV task which is highly applicable to ADSs. In 2020 a group of researchers proposed a framework which combined a modified YOLOv4 (You Only Look Once) object detection model with an intention recognition network, and a risk assessment network (Li et al. 2020). The overall framework uses the modified YOLOv4 model to detect and classify objects into vehicles, pedestrians, and traffic lights. Pose estimation and intention recognition are carried out by convolutional neural networks (CNNs) on any pedestrians detected, while vehicles detected are classified as either dangerous or not dangerous by another CNN. Finally, traffic lights detected were also classified by a CNN to extract the information they were signaling (such as green light, red light, green arrow, etc.). While the researchers were able to obtain good performance results for all of the individual tasks, they concluded that the overall framework required too much processing time per image to be useful in real-time applications (Li et al. 2020).

Methodology

This project compared the results of using a DQN, a SAC network, and a DAC network to learn the optimal policy for an ADS agent to move within a simulated environment without crashing into environmental obstacles. The following sections provide more detail about the architectures used, the simulation environment, and the sets of states, actions, and rewards available to the agent.

Network Design

For this project 3 separate models were built and evaluated: a DQN, a SAC network, and a DAC network. All 3 of the models used the same convolutional base to process input images from the CARLA simulator. This convolutional base contained 3 convolutional layers, each with 64 filters, a kernel size of 3, and a ReLU activation function. After each convolutional layer is an average pooling layer with a pool size of 5 and a stride of 3. The outputs of the final pooling layer were then flattened before being passed to the rest of each model. The following subsections give more detail regarding the DQN, SAC network, and DAC network used for this project.

DQN The first model architecture used for this project was based on the DQN architecture proposed by a group of researchers in 2015. The development of the DQN was motivated by a desire to leverage new developments in deep neural networks to build an agent capable of end-to-end reinforcement learning using only high-dimensional sensory inputs (Mnih et al. 2015). The researchers tested the DQN

on 49 different Atari 2600 games using 4 frames of 84x84 images as input (which were obtained by preprocessing the 210x160 60Hz video output of the games), and found that it outperformed state-of-the-art reinforcement learning models and even achieved comparable results to professional human players across a majority of the games the agent was trained on (Mnih et al. 2015). The DQN consists of 3 convolutional layers, a hidden fully-connected layer, and output layer which uses a softmax activation function to select the appropriate action from the set of all actions available to the agent.

The DQN built for this project used 640x480 3-channel images as inputs to the convolutional base described above. The outputs of this convolutional base were passed to a hidden dense layer containing 256 neurons and using a ReLU activation function, and the output layer had 9 neurons (corresponding to the 9 actions available to the agent, detailed in a later section) and used a simple linear activation function. As was the case with the original DQN proposed in 2015, 2 models were maintained during training: a model whose weights were updated with every episode of training and a target model which reflected the current policy and whose weights were updated to match the first model every 10 episodes. Using a second, target model to calculate the maximum future reward (based on the current policy) for the Q-learning update of a main model increases the stability of training (Mnih et al. 2015).

SAC The SAC algorithm differs from the Q-learning algorithm used by the DQN in 2 significant ways. Firstly, the SAC algorithm is based on the maximum entropy RL framework, meaning that it seeks to maximize both entropy and the rewards earned by the agent (Haarnoja et al. 2018). Maximizing entropy causes the agent to strike a balance between exploration and exploitation without the need for an ϵ-greedy policy. The second major difference is that unlike the DQN, actor-critic networks are suitable for continuous action spaces. This is of particular interest for this project since ADSs in the real world do operate in a continuous action space.

The SAC network built for this project used one actor (or policy) network and 2 critic networks. The actor network used the same convolutional base as the DQN described above. After the flatten layer 2 dense layers were added, both with 256 nodes and a ReLU activation function. The actor network had 2 outputs, one which calculated the mean of the probability distribution of actions given the input state and the other which calculated the standard deviation. Both outputs were produced by a dense layer with 2 nodes and a linear activation function.

The critic networks were identical and used the same convolutional base to process input images. The outputs of the flatten layer of this convolutional base were concatenated with the action inputs (a vector containing 3 values) before being passed to 2 dense layers with 256 nodes each and both of which used a ReLU activation function. The output layer of the critic networks contained only a single node, as these networks produce only a single $Q(s, a)$ value based on the input values of s and a. As was the case with the DQN, both of the critic networks had their own target networks which were used as an estimator for the $Q(s, a)$ values. Figure 1 is a diagram of the architectures for the actor and critic networks used by the SAC model.

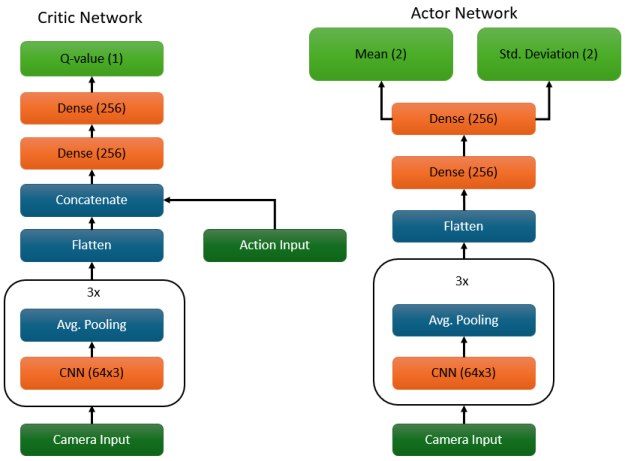

Figure 1: Architecture detail of the actor and critic networks

DAC The DAC model used for this project was very similar to the SAC model. The only major difference is that this model learned a deterministic policy rather than a stochastic policy, so rather than outputting a mean and standard deviation for a probability distribution the output model of this layer contained 2 nodes which directly correlated with the action space available to the agent. The activation function for the output layer was also changed and a tanh activation function was used. This activation function was chosen to bound the action space between -1 and 1, which are the same bounds used by the agent in the CARLA simulator. Because the DAC learned a deterministic policy it also did not seek to maximize entropy, so to solve the exploration vs. exploitation trade-off an ϵ-greedy policy was used. In all other regards the DAC model was identical to the SAC model.

Environment

The environment in which the ADS agent operated for this project was the Car Learning to Act (CARLA) simulator. CARLA is an open-source simulator that was developed with the goal of supporting the training, prototyping, and validation of ADSs (Dosovitskiy et al. 2017). It is highly sophisticated and contains a suite of urban maps with assets including buildings, vegetation, infrastructure, street signs, vehicles, pedestrians, and even a variety of weather conditions. Environmental settings (such as weather conditions, number of non-agent vehicles, number of pedestrians, and vehicle spawn points) are highly customizable. CARLA also simulates a wide variety of sensor inputs for the agent vehicle which are based on sensors used in real ADSs. The following sections define the state space, action space, and rewards that were used to train the agent for the project.

State Space As is the case with ADSs that exist outside the virtual space, the state space for the ADS agent in this

project comprised of sensor inputs that describe the relationship between the agent and the environment. The primary input to the agent was a simple 3-channel (RGB) camera sensor simulated by the CARLA platform and anchored above the agent to capture the view of the simulated road in front of the agent. A sensor that detects and measures the impact of collisions was also used to describe the state space. However, this sensor was not used as an input to the DQN but was instead used to determine if the agent collided with anything in the virtual environment for the purpose of assigning rewards. For the actor-critic models a GNSS sensor was also used. This sensor provided geolocation data analogous to a GPS and was used to calculate the total distance traveled by the agent during an episode of training. This distance metric was used to calculate rewards but was not provided to any of the networks as an input describing the state.

Action Space Agents in the CARLA simulator have an action space described by a 3-dimensional vector: throttle, steering, and braking. The throttle and braking values range from 0 to 1 and describe how much acceleration or deceleration to apply, while the steering value ranges from -1 (fully to the left) to 1 (fully to the right). However, the DQN required a discrete action space (with each action corresponding to a node in the output layer) while the action space available to agents in the CARLA environment is continuous. Therefore this action space was discretized to make it compatible with a DQN. The action space was divided into 9 possible actions: forward (1, 0, 0), left (0, -1, 0), right (0, 1, 0), forward-left (1, -1, 0), forward-right (1, 1, 0), brake (0, 0, 1), brake-left (0, -1, 1), brake-right (0, 1, 1), and no action (0, 0, 0). Actions were chosen using an ϵ-greedy policy with an initial ϵ value of 1 which decayed by 0.999 every episode as training progressed (until reaching a minimum value of 0.1).

The SAC and DAC models were able to take advantage of the continuous action space available to the agent, and therefore they did not discretize the action space in the same way as the DQN model. In early training experiments it was discovered that using an action space of $a = [throttle, steering, brake]$ allowed the agent to use both the throttle and the brake at the same time, something which is almost never done in real world driving situations. Therefore, the outputs of the actor networks for both the the SAC and DAC models was changed to be a vector with only 2 dimensions, acceleration and steering. If the acceleration was greater than 0, the acceleration value was given as the throttle value to the agent while the brake value was given as 0. Conversely, if the acceleration was ¡ 0 then the absolute value of the acceleration was given as the brake value while the throttle value was set to 0. This prevented the agent from using both the throttle and brake simultaneously and resulted in better training results.

Rewards Many reward schemes were experimented with for this project in an attempt to improve the policies learned by the agent. This section describes only the final reward schemes used, which were the schemes used to train the agents whose results are given in a later section.

The DQN agent in this project was only rewarded for 2 behaviors: colliding with something in the environment and the speed at which it drove. Whenever the agent collided with something in the environment it was given an immediate reward of -1 and the episode of training ended. The reward given for the speed of the agent was calculated in a linear fashion: $Reward = \frac{speed}{50} - 1$ where $speed$ is the speed of the agent vehicle in kilometers per hour (kph). This meant that the agent received a positive reward for driving over 50 kph and a negative reward for driving below 50 kph. This was done so that the agent would learn to drive at or above 50 kph. Without this reward the agent could learn a policy that involved not moving at all to avoid collisions.

The SAC and DAC agents used the same set of rewards, which were more complex than those used given to the DQN agent. The agent still received a negative reward for colliding with an object in the environment, but this negative reward was lowered to -0.1. However, training still ended after a collision occurred, precluding the agent from earning any more positive rewards. The agent was given a positive reward based on the speed it was going, calculated according to the following equation: $Reward = \sqrt{\frac{speed}{50}}$. An additional positive reward was given for the total distance the agent traveled. This distance reward was calculated according to the formula: $Reward = max(\sqrt[4]{distance}, distance)$. The distance and speed rewards were added together to give an immediate reward for every action the agent performed. Combining these rewards, along with lowering the penalty for collisions, was done to encourage the SAC and DAC agents to move more as this was a recurring problem during training.

Results

The results obtained by the 3 models are presented below. The DQN model was the first model trained and evaluated, and it was discovered that the agent learned to only drive in circles to the left (meaning it heavily favored the discrete action "forward-left" defined above). This makes sense: doing so allowed the agent to minimize the negative reward it received for driving below 50kph while also avoiding most collisions. This discovery is what motivated the creation and evaluation of the actor-critic models, as it was determined that expanding the action space could resolve this issue.

DQN

The DQN agent was trained for 5000 episodes in total. Each episode lasted until the agent collided with something in the environment. If there was no collision within 10 seconds the episode also terminated. Every 10 episodes of training the maximum reward, minimum reward, and average reward earned by the agent were tracked, along with the percentage of episodes that ended in collisions, the average speed the agent traveled at, and the maximum speed reached by the agent. Figure 2 shows the average, minimum, and maximum rewards earned by the agent during training. This plot clearly shows that the agent is learning, as the value of the rewards earned grows quickly at first and then continues to grow throughout training. This growth appears to plateau some-

where around episode 2000, which correlates with when the value of ϵ reaches its minimum value (0.1).

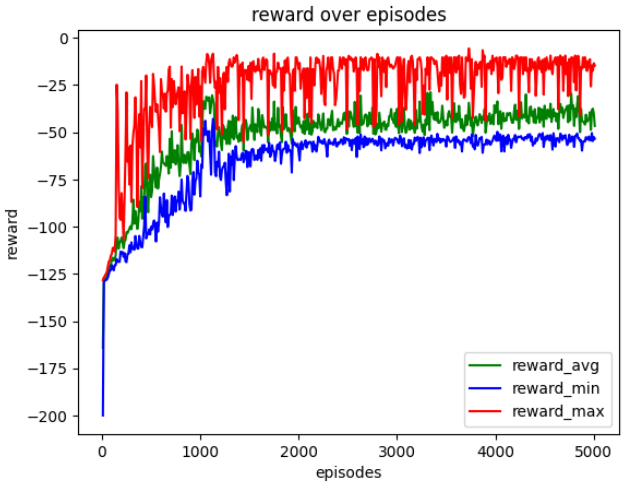

Figure 2: Rewards earned during training (DQN)

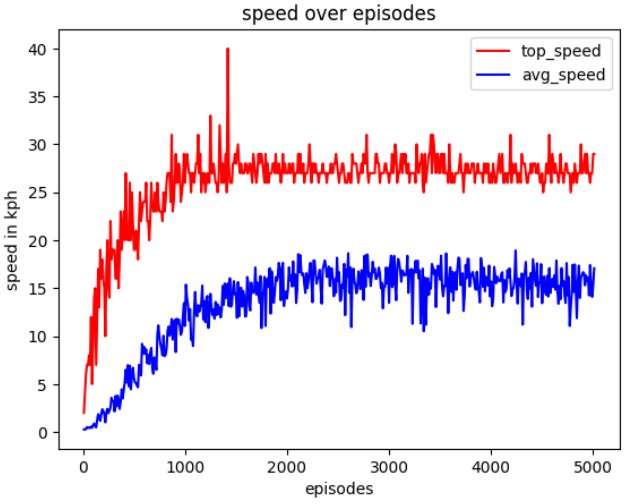

Figure 3: Speeds reached during training (DQN)

Figure 3 shows the average and top speeds achieved by the agent during training. As was the case with the rewards earned by the agent, the average and top speeds increased quickly at first, but this rate of increase slowed around the 2000th episode of training. By the end of training the agent was averaging between 10 and 15 kph, but reaching top speeds between 25 and 30 kph.

Figure 4 is a scatterplot of the percentage of episodes that ended in collisions during the course of training. A scatterplot was chosen because even towards the end of training there was a fair amount of fluctuation in the percentage of episodes that were ending in collisions. Early in training (prior to the 1000th episode) the range of episodes that ended in collisions was between 0-80%, but as training progressed this range narrowed to between 10-50%. While this is an improvement it's still a lot of variation, and a relatively high number of episodes continue to end in collisions. One possible explanation for this observed variation is that the agent starts in a random location in the environment with every episode. Therefore, in some episodes it will start closer to obstacles than in others. It is likely that if the agent's starting location was controlled fewer episodes would end in collisions as training progressed. However, this would come with the trade-off that the agent would not generalize as well and would only learn to drive starting from one specific location in the environment.

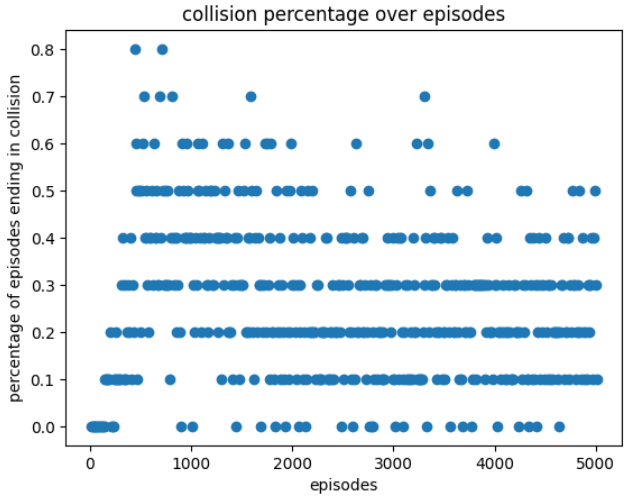

Figure 4: Collision percentage during training (DQN)

Actor-Critic Models

Both the SAC and DAC models exhibited similar behavior during training. Figures 5 and 6 both show the rewards received by the actor-critic agents during training. Neither actor appears to be learning a policy which maximizes these rewards; instead the rewards slowly decrease over time. It's unclear why this is the case, despite many iterations of reward schemes this trend never improved. It could be the case that further hyperparameter tuning is needed to produce a model which converges to improve the rewards over time. It could also be a problem with the target network update frequency or the replay memory implementation. Regardless, expanding the action space to be continuous complicated the model enough that no good results were obtained using either actor-critic model.

Figures 7 and 8 show the maximum and average speeds reached by the actor-critic agents during training. For both models the maximum speed reached far exceeds the average speed reached, suggesting that on average the agents moved very little but their average speeds were brought up by these outlier values. This is why the negative penalty for traveling below 50kph was dropped and an added reward for distance traveled was added, but neither of these changes to the reward scheme caused the agent to learn to move more. Interestingly, the SAC model appears to achieved much higher

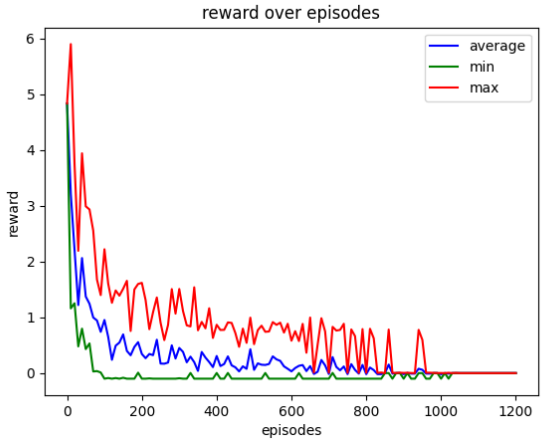

Figure 5: Rewards earned during training (SAC)

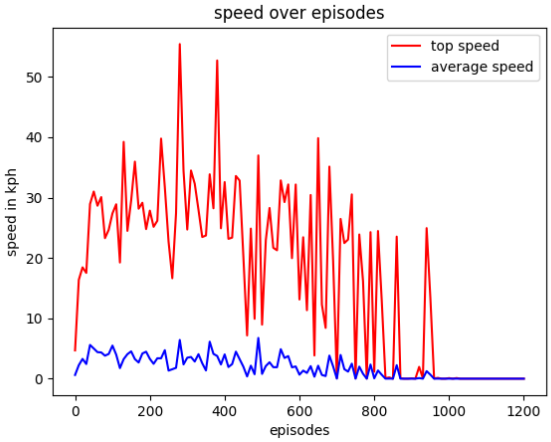

Figure 7: Speeds reached during training (SAC)

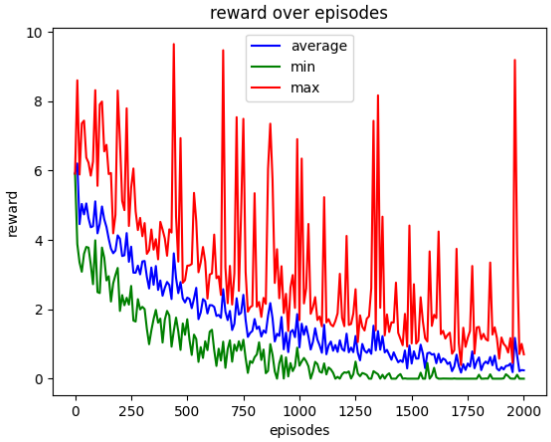

Figure 6: Rewards earned during training (DAC)

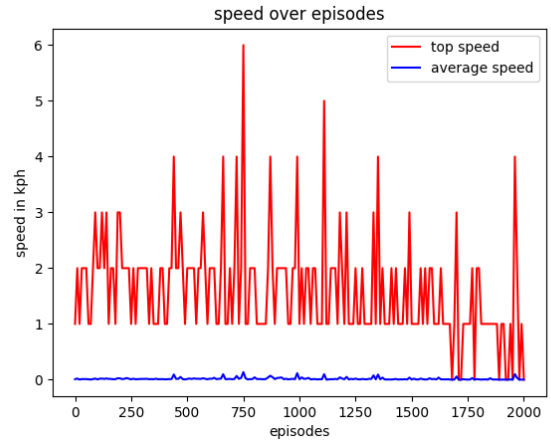

Figure 8: Speeds reached during training (DAC)

maximum speeds than did the DAC model, with values that actually exceeded the target of 50kph. This can likely be attributed to the use of entropy maximization to encourage exploration.

Figures 9 and 10 show the percentage of episodes that ended in collisions for the 2 actor-critic models. The percentage of collisions for the SAC model roughly mirrors the plot for the speeds reached by the model, and this correlation makes sense: it's logical that the agent would be more likely to crash if it's moving faster. The DAC agent never crashed, which is also not surprising since it never reached the same speeds the SAC agent did.

Future Work

The relatively poor performance of all 3 models leaves a lot of room for improvement. The changes which could be made can be broadly broken into 3 categories: making changes to the model used for learning, making changes to the state/action space, and making changes to the rewards given to the agent during training. Each of these avenues of improving the various models will be explored in the following sections.

Model Changes

The DQN is a versatile and powerful model, but not one particularly well-suited to the task of autonomous driving since it requires a discrete action space. This was demonstrated by the fact that the model trained for this project learned to heavily favore one action which caused it to drive in circles. While the discrete action space for this model could be expanded this will always be a limitation of the model.

While the actor-critic models do not share this limitation they did not meaningfully improve on the results achieved by the DQN. It is likely that further experimentation with hyperparameter values for these models will be needed so that training can stabilize and begin to converge to meaningful policies. Once these models are able to learn policies which maximize rewards further experiments could be con-

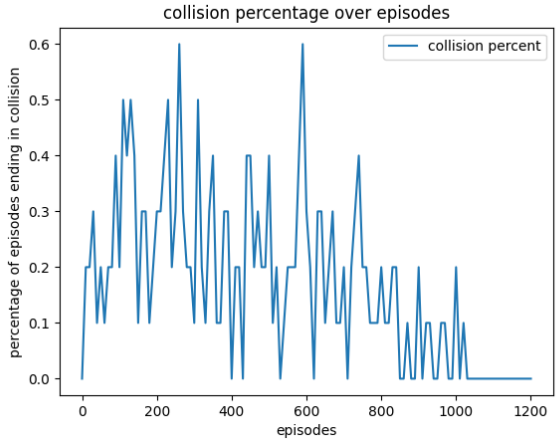

Figure 9: Collision percentage during training (SAC)

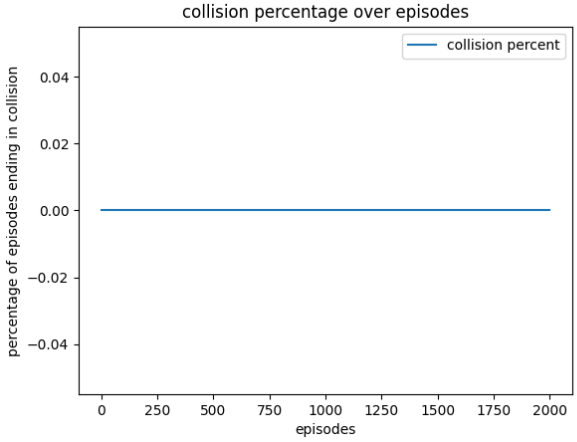

Figure 10: Collision percentage during training (DAC)

ducted to change the components of the models (such as the number of convolutional layers or the number of nodes in dense layers) in order to further optimize them to learn driving policies for an ADS agent.

State Space Changes

There are many ways in which the environment and state space used in this project could be altered to increase the amount of information available to the ADS agent and also increase the ability of the agent to generalize. All training was conducted on a single map in the CARLA simulator, with no alterations made to weather patterns. Changing both the map and the weather conditions would make the model generalize better. Additional vehicles could also be added to the map to simulate traffic (which may not improve the results of the model, but would better simulate driving in a real-life urban environment). Finally, the CARLA simulator can simulate a number of sensors that were not utilized for this project including LIDAR, speed, and a depth camera (Dosovitskiy et al. 2017). Incorporating one or all of these sensors would increase the input available to the agent and could improve the performance of the agent overall.

Reward Changes

Perhaps the greatest tool for improving the performance of the agent is altering the rewards it receives during training. For this project efforts were made to incentivize the actor-critic models to move more and achieve greater speeds. It is likely that the negative reward for collisions would need to be increased for these models once collisions become a problem. Future experiments could add rewards for the agent staying within its lane (facilitated by a lane invasion sensor which CARLA can simulate) and alter the target speed of the agent (currently 50 kph). Additionally, rather than spawning the agent in a random location on the map and training it for 10 seconds (or until a collision occurs) it would be possible to set a goal destination (another spawn point chosen at random) that the agent must reach, with diminishing rewards given for the amount of time it takes the agent to reach its destination.

Conclusion

For this project a DQN, SAC network, and DAC network were constructed and trained in the CARLA simulator to learn optimal driving policies. The agents were rewarded for driving at or above 50 kph without colliding with any environmental obstacles. While the value of the rewards received and speeds reached by the DQN model increased as training progressed, the average episode length and number of episodes which ended in collisions showed a high degree of variation which did not noticeably decrease as training progressed. The SAC and DAC agents did not learn policies which maximized rewards, so while they experimented with moving in the environment both agents ultimately learned to simply stay still or move at very low speeds. To improve the results obtained by these agents future experiments will alter the model hyperparameters used, the state space the agents operate in, and the rewards they receives during training.

References

Codevilla, F.; Müller, M.; López, A.; Koltun, V.; and Dosovitskiy, A. 2018. End-to-end driving via conditional imitation learning. In *2018 IEEE international conference on robotics and automation (ICRA)*, 4693–4700. IEEE.

Dosovitskiy, A.; Ros, G.; Codevilla, F.; Lopez, A.; and Koltun, V. 2017. CARLA: An open urban driving simulator. In *Conference on robot learning*, 1–16. PMLR.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In Dy, J.; and Krause, A., eds., *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, 1861–1870. PMLR.

Kiran, B. R.; Sobh, I.; Talpaert, V.; Mannion, P.; Al Sallab, A. A.; Yogamani, S.; and Pérez, P. 2021. Deep reinforcement learning for autonomous driving: A survey. *IEEE*

Transactions on Intelligent Transportation Systems, 23(6): 4909–4926.

Li, G.; Yang, Y.; Qu, X.; Cao, D.; and Li, K. 2021. A deep learning based image enhancement approach for autonomous driving at night. *Knowledge-Based Systems*, 213: 106617.

Li, Y.; Wang, H.; Dang, L. M.; Nguyen, T. N.; Han, D.; Lee, A.; Jang, I.; and Moon, H. 2020. A deep learning-based hybrid framework for object detection and recognition in autonomous driving. *IEEE Access*, 8: 194228–194239.

Liang, X.; Wang, T.; Yang, L.; and Xing, E. 2018. Cirl: Controllable imitative reinforcement learning for vision-based self-driving. In *Proceedings of the European conference on computer vision (ECCV)*, 584–599.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533.

Yurtsever, E.; Lambert, J.; Carballo, A.; and Takeda, K. 2020. A survey of autonomous driving: Common practices and emerging technologies. *IEEE access*, 8: 58443–58469.