Using a Deep Q-Network 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 explored the use of a Deep Q-Network (DQN) to learn optimal driving policies for an agent in the open-source CARLA urban driving simulator. The agent received negatives rewards for driving below 50 kph and for crashing, and received positive rewards for driving above 50 kph. While it is clear that the agent learned to drive at higher speeds, a high degree of variation in the number of episodes which ended in collisions made it difficult to assess if the agent was learning to avoid collisions in a meaningful way. Future experiments will alter the model architecture, state space, and rewards given to the agent in an attempt to further improve the results obtained by the agent.

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).

Interpretable Learning

Besides the difficulty of developing an ADS which requires no human intervention, the wide adoption of such a system will likely be hindered by public trust in the decisions made by the system. This is due to the fact that neural networks are cryptic and not easily interpretable. To bridge this trust gap and address the interpretability of ADSs a pair of researchers in 2017 proposed the use of an encoder-decoder network to produce visual attention maps which are then further analyzed to determine the saliency of the input regions in the attention maps (Kim and Canny 2017). The encoder portion of this architecture is simply a CNN for feature extraction, while the decoder portion is split into a coarse-grained decoder and a fine-grained decoder. The coarse-grained decoder is a deterministic soft attention mechanism to produce attention heat maps, and the fine-grained decoder applies a clustering algorithm to these heat maps. Clusters are then iteratively masked and the performance of the network is evaluated to determine the visual saliency of each cluster. The researchers found that the incorporation of visual attention did not degrade control accuracy while it did provide meaningful information about what visual data the network used to make decisions (Kim and Canny 2017).

Methodology

This project used a DQN 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 DQN architecture, the simulation environment, and the sets of states, actions, and rewards available to the agent.

Network Design

The 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 consisted of a convolutional base, a hidden fully connected layer, and an output layer. The convolutional base contained 3 convolutional layers, each with 64 filters and a kernel size of 3. After each convolutional layer is an average pooling layer with a pool size of 5 and a stride of 3. Between the convolutional base and the fully connected layers is a flatten layer. The hidden layer has 256 neurons and uses a ReLU activation function, and the output layer has 9 neurons (corresponding to the 9 actions available to the agent, detailed in a later section) and uses 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).

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.

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 requires 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).

Rewards The 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.

Results

The 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 average episode length, the percentage of episodes that ended in collisions, the average speed the agent traveled at, and the maximum speed reached by the agent. Figure 1 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 somewhere around episode 2000, which correlates with when the value of ϵ reaches its minimum value (0.1).

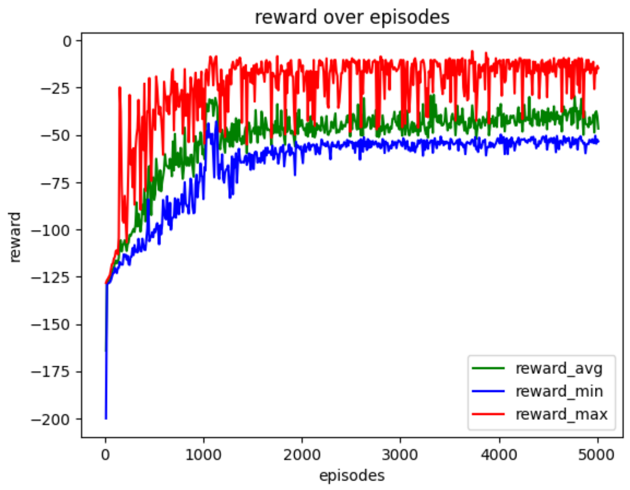

Figure 1: Rewards earned during training

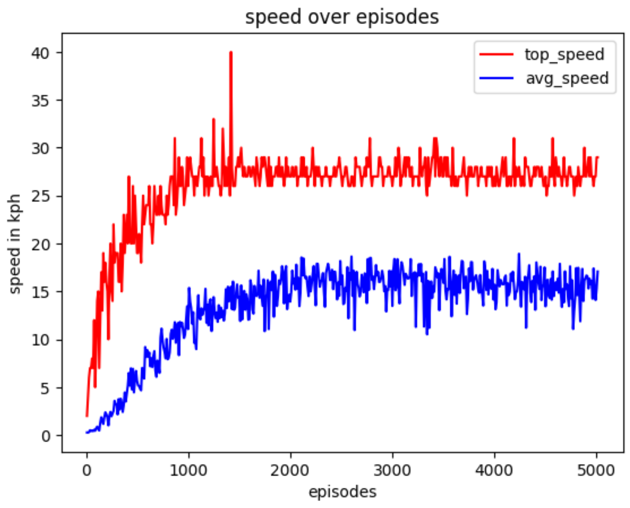

Figure 2: Speeds reached during training

Figure 2 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 3 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%, butas 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.

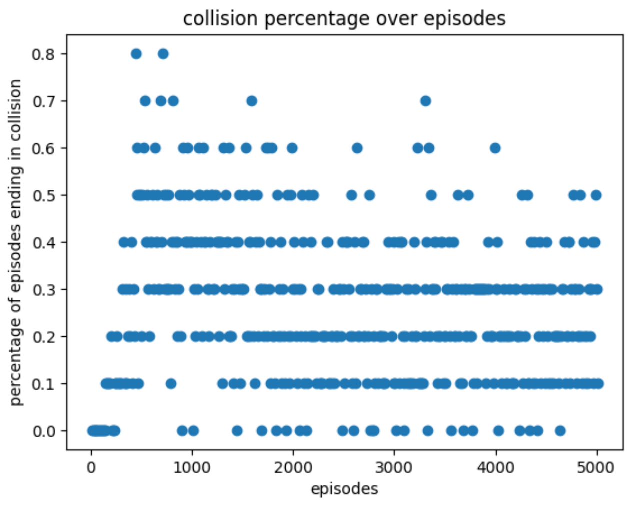

Figure 3: Collision percentage during training

As was the case with the percentage of episodes that ended in collisions, there is a lot of variety and no clear trend in the average length of training episodes over time (as seen in figure 4). This is undoubtedly correlated with the number of episodes which end in collisions since the only way for average episode length to decrease is for a higher percentage of episodes to end in collisions. The ultimate goal would be to see the average episode length drop initially as the agent learns to drive faster (and therefore crashes more often), followed by an increase in episode length as the agent learns to drive at higher speeds without colliding with anything in the environment.

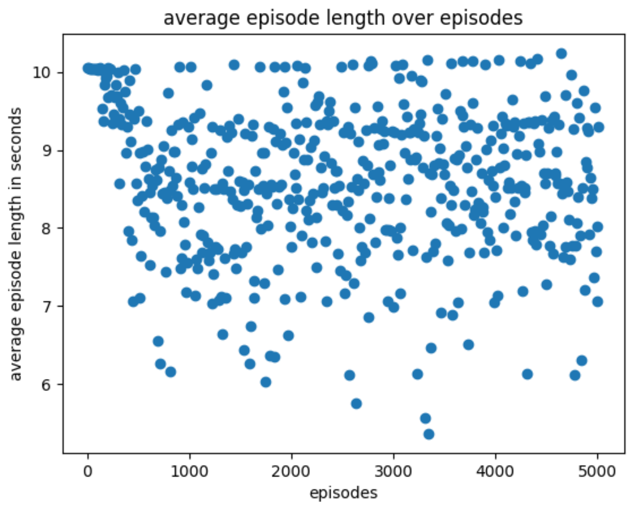

Figure 4: Episode length during training

Future Work

There are many ways in which this project could be altered in attempt to improve the results of the ADS agent. These changes 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 agent 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. The biggest drawback of using a DQN to train an ADS agent is that the action space for a DQN must be discrete, while the action space available to an ADS agent is continuous. In 2017 a group of researches proposed several changes to the DQN architecture to adapt it to the task of autonomous driving. The first of these changes is the integration of an actor-critic learning framework in which the policy function (actor) and value function (critic) are learned by separate networks which work together to learn the optimal policy. This change addresses the discrepancy between the discrete action space a DQN requires and the continuous action space an ADS agent has access to (Sallab et al. 2017). For this project the actor-critic networks used will be based on the Soft Actor-Critic (SAC) algorithm proposed in a 2018 paper in which the researchers found that the SAC algorithm outperformed state-of-the-art model-free DRL methods (Haarnoja et al. 2018). The next change from the original DQN architecture is the inclusion of an attention mechanism to filter the features learned by the CNN layer. This change was motivated by the observation that not all of the features learned by the CNN layer will contribute equally to the final optimization objective (Sallab et al. 2017). The final substantial modification to the DQN is the addition of a LSTM layer between the filtered CNN outputs and the Q-network.

This recurrent layer was added because the researchers observed that an ADS agent will not be learning in an environment that truly obeys the Markov assumption that underlies Markov Decision Processes (MDPs). Instead, the occasional occlusion of objects/vehicles by other vehicles on a roadway mean that an ADS will be operating in an environment that is a partially observable MDP (POMDP). The addition of a LSTM layer adapts the DQN to work for POMDPs (Sallab et al. 2017). These 3 changes will adapt the DQN to be better suited for training an ADS agent, which will hopefully improve the results obtained by the 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 pseudo-sensor which gives information about the position and orientation of the agent in the environment (analogous to a GPS in a non-virtual ADS) (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. Currently the agent is only being rewarded for driving at a reasonable speed without crashing. 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.

Timeline

Table 1 below contains the proposed timeline for the project leading up to the May 8th final.

Date	Objective
4/1 - 4/7	Modify the DQN
4/8 - 4/14	Experiment with the new model
4/15 - 4/21	State space experiments
4/22 - 4/28	Reward space experiments
4/29 - 5/5	Final model training and evaluation

Table 1: Timeline

Conclusion

For this project a DQN was constructed and trained in the CARLA simulator to learn optimal driving policies. The agent was rewarded for driving at or above 50 kph without colliding with any environmental obstacles. The agent was trained for 5000 episodes, and its performance was then evaluated using the rewards the agent received, the speeds it drove at, and the number of training episodes which ended in collisions. While the value of the rewards received and speeds reached 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. To improve the results obtained by the agent, future experiments will alter the model architecture used, the state space the agent operates in, and the rewards it 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.

Kim, J.; and Canny, J. 2017. Interpretable learning for self-driving cars by visualizing causal attention. In *Proceedings of the IEEE international conference on computer vision*, 2942–2950.

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.

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.

Sallab, A. E.; Abdou, M.; Perot, E.; and Yogamani, S. 2017. Deep reinforcement learning framework for autonomous driving. *arXiv preprint arXiv:1704.02532*.

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.