Active Learning for End-to-End Autonomous Driving

Yosuke Higashi YHIGASH1@SWARTHMORE.EDU
Swarthmore College, 500 College Ave., Swarthmore, PA 19081 USA

Michael Song MSONG2@SWARTHMORE.EDU
Swarthmore College, 500 College Ave., Swarthmore, PA 19081 USA

Abstract

A simple way to create an autonomous driving agent is to collect data of a human driving and then train a model using supervised learning. This approach has been shown to be suboptimal due to a mismatch between the states reachable by a human driver and trained agents (Zhang & Cho, 2016). Current approaches to solving this problem are impractical for real world robots and vehicles. We present a simple algorithm that incorporates active learning to train an end-to-end autonomous driving agent that is easily extensible from simulation to the physical world. While a trained agent is driving, a human driver overrides its actions if the agent veers off course or displays other incorrect behavior, and the feedback from the human driver is added to the training data. We test our approach on the popular racing game Mario Kart 64 and empirically show that it effectively counters the data mismatch problem. On both an easy course and a difficult course, the agent trained using our approach shows far superior performance over a naive supervised learning agent.

1. Introduction

End-to-end autonomous driving refers to the task of mapping raw sensory input, such as an image from a front-facing camera, to a vehicle output necessary for driving, such as the steering angle. Convolutional Neural Networks (CNNs) have been shown to excel at this task because of their ability to automatically extract features from the high-dimensional inputs of raw pixel data (Bojarski et al., 2016). The CNN architecture created by Bojarski et al. is particularly well suited to this task (2016). It is able to learn a complex mapping from raw image to control, implicitly extracting features such as the location of lane lines, and is able to achieve approximately 98% autonomy for lane-following on a highway.

Even with a powerful CNN architecture, a naive supervised learning approach, where the learner is trained on data collected by a human driver, is not sufficient for end-to-end autonomous driving. This is because the approach violates the critical assumption of supervised learning that the training data and testing data are drawn independently and identically distributed from the same distribution of data (Ross & Bagnell, 2010). Intuitively, this problem arises because a human driver is often too perfect and does not make any significant errors while driving. An agent trained on this data will inevitably make errors which will lead the agent to error states that do not exist in the training data, causing the errors to compound.

Current state-of-the-art approaches to solving this problem of data mismatch are not easily extensible from simulation to the physical world (Ross & Bagnell, 2010) (Ross et al., 2011) (Hussein et al., 2016). We propose a solution using an active learning approach that would be very simple to implement in real robots and vehicles. When the agent trained on human driving data veers off course, we override the

CPSC 66 Machine Learning Proceedings, Swarthmore College, Fall 2017.

agent and correct its trajectory. The data collected during the override is then added to the training data and the agent is retrained. We test this approach on the popular car racing game Mario Kart 64 as a proof of concept. With just a few iterations of correcting and retraining, the agent is able to learn error correcting behavior and drive much more robustly than an agent trained with naive supervised learning.

2. Background

2.1. DAGGER Algorithm

Data mismatch between the training data and testing data is a key problem in the field of imitation learning, where the goal is to train an agent to mimic an expert. With naive supervised learning, the error compounds and can be shown to grow quadratically in the time horizon of the task (Ross & Bagnell, 2010). Dataset Aggregation (DAGGER) is a state-of-the-art algorithm that solves this problem, providing a guarantee that the error only grows linearly in the time horizon of the task (Ross et al., 2011). In the DAGGER algorithm, an agent is first trained on data collected by an expert policy. Then, the trained agent is run in the environment and the expert policy is queried for the correct output at each state the agent encounters, but the expert never overrides the agent. The data collected is then added to the training set and training produces a new agent. The intuition here is that the agent will explore areas of the state space that the expert would not typically find itself in, meaning that the mismatch between the training data and testing data is reduced.

While DAGGER essentially solves the data mismatch problem, it is highly impractical for application to real robots or autonomous cars. It queries the reference policy for each and every collected state, which is undesirable as the reference policy is often expensive (Zhang & Cho, 2016). DAGGER is also very difficult to implement if a human driver must act as the reference policy, which is almost always the case in the real world. As noted in (Ross et al., 2013), it is very difficult for a human to determine the correct output without actual feedback from the agent.

2.2. Active Learning with a Optimal Agent

(Hussein et al., 2016) presents an alternative to solving the data mismatch problem using active learning. Similar to the DAGGER algorithm, an agent is initially trained on data collected by the expert agent. Then, for each state the agent encounters, the confidence for each prediction is calculated and the expert is queried for the correct action in states with low confidence predictions.

This approach has an advantage over the DAGGER algorithm in that the agent makes far fewer queries to the reference policy. However, it has even more practicality issues than DAGGER because the expert agent must always be alert and respond as soon as the agent signifies that it has low confidence on a prediction. Therefore, a computerized optimal agent is required for this approach.

3. Approximate Active Learning Approach

We propose to train an end-to-end autonomous driving agent by incorporating the idea of active learning as in (Hussein et al., 2016), but in a way that does not require a computerized optimal agent. Specifically, the human driver determines when the agent is not confident about a prediction instead of relying on the agent to self-diagnose.

The first step of our algorithm is to train an agent from data collected by a human driver using supervised learning. Then, the trained agent drives autonomously and is corrected by the human driver whenever the agent veers off course or displays any other kind of error behavior. Data collected while the human driver corrects the agent is added to the training data, and the agent is retrained. The algorithm is summarized in Algorithm 1

This inversion of the party classifying states with low-confidence predictions makes implementation significantly easier. The human driver will not have to respond immediately to react to an agent's queries, since they control when the queries will occur. While this approach is not directly an implementation of active learning, it is very closely related and can be thought of as an approximation. A true active learning agent calculates a confidence score for each of its predictions, and queries the expert when the confidence dips

Algorithm 1 Approximate Active Learning

Input: Base agent A_0 trained on $D = (x, y)$
Initialize $agent = A_0$.
for $i = 1$ **to** $maxIter - 1$ **do**
  repeat
    x = current image frame
    if $manualCorrect = true$ **then**
      y = human driver output
      add (x,y) to D
    else
      y = $agent$ output
    end if
    perform output y
  until end of run
  retrain $agent$ using D
end for

lower than some threshold. In the autonomous driving setting, the agent will likely have low confidence when veering off the road or encountering error states, as they will not be well represented in the training data. Therefore, our inverted active learning approach is in a sense an approximation to active learning, as we approximate the states in which the agent has low confidence.

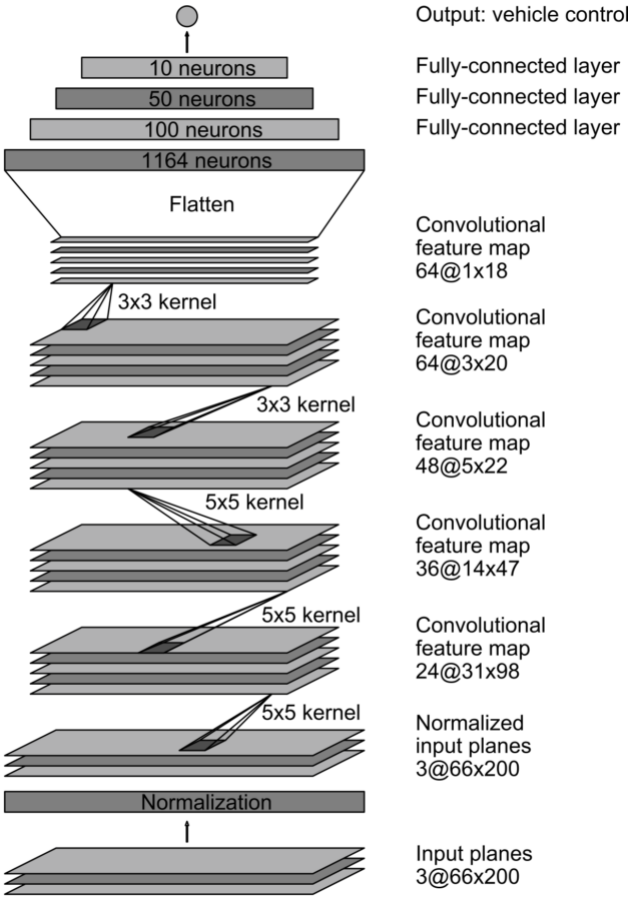

Figure 1. Convolution Neural Network and Fully Connected Neural Network used to train our agents. The CNN contains a total of 5 feature maps, eventually flattened and inputted into our fully connected NN. The output is the predicted output steering value

4. Experimental Methods

4.1. Basic Setup:

We use the popular racing game Mario Kart 64 to test our approximate active learning method. In order to train an agent to drive in the Mario Kart N64 game, we use the Github repository TensorKart by Kevin Hughes (Hughes, 2013). The main emulator program (mupen64plus) runs the Mario Kart N64 ROM, which allows us to freely play the Mario Kart game using a PS4 controller. For collecting data, we use a TensorKart program that allows us to take screenshots of the mario kart game while also recording the the buttons and joystick commands we make. Another program included allows us to run a specified trained agent on any of the mario kart courses. The main environment that this program uses is the gym-mupen64plus environment.

We simplify the driving task by fixing the accelerate button to always be pressed. This causes the steering angle to be the only kart output as the forward velocity is fixed. We made this decision because the accelerate button has only two states (pressed or not pressed) making it not directly indicative of the forward velocity of the kart. The joystick values are split into X and Y values. Moving the stick left to right horizontally corresponds to X values between -80 and 80 respectively; this corresponds directly to the steering angle and is therefore the value we define as the output. We ignore the Y values as they are irrelevant to the steering angle. Besides the joystick and the accelerate button, all other buttons or controls are unused.

4.2. Training the Supervised Learning Agent

We first train a base agent using standard supervised learning. The training data is collected by a human

Figure 2. An example training set image and the correct target output value. If faced with this image, the kart would want to take a slight left, shown by the joystick arrow above.

Agent No.	Course Time	No. of times stuck
IL Agent	4:13	12
Agent 1	3:20	4
Agent 2	2:43	0
Agent 3	2:54	0
Agent 4	2:41	0
Agent 5	2:40	0
Agent 6	2:39	0
Agent 7	2:36	0
Human Player	2:29	N/A

Table 1. The completion times (min:sec) for each agent on Luigi Raceway. The naive supervised agent and Agent 1 fail to complete the course on their own. As a result, the completion time is only possible with human intervention (number of times stuck).

playing optimally on a particular course; the human player always stays on the track and does not hit any walls or obstacles. The data collected consists of input images and the corresponding steering angles that the human takes, as shown in figure 2. Furthermore, we collect the training data without any other computer karts interfering. This data is used to train a Convolutional Neural Network, which maps an input image to an output steering angle 1. The architecture of the CNN is taken from (Bojarski et al., 2016), and has five convolutional layers and four fully connected layers. We also added dropout layers and L2-regularization to the fully connected layers to help prevent overfitting.

4.3. Implementation of Approximate Active Learning

To create our approximate active learning agent, we run the base supervised learning agent on the course and manually override it when we determine that is it in a bad situation. We consider a bad situation to be when the agent is off course, bumping into walls or any other situation that would slow or hinder the agents performance. These bad situation images and their corresponding corrections are then added to the existing training set. After the run, we will have added a new set of images and corresponding steering angles that represent when the agent is unsure or in a tough situation. Using this new dataset, we train another agent, which we call Agent 1, using the process mentioned above. Now, we run Agent 1 on the track and collect more correction data and add it to the existing training data. We repeat these steps to create Agent 2, Agent 3, etc., up until Agent 7.

4.4. Evaluation Metrics:

One way in which we evaluate an agent is by its time to complete a Mario Kart 64 track. However, some agents are unable to complete the track without human intervention, so we also record the number of times a human had to manually help the agent. We also evaluate the performance of each agent on an autonomy metric, which describes how autonomous the agent actually is with regard to how often the human must intervene to sustain optimal driving. Optimal driving in this case is avoiding wall collisions and staying on course. In order to calculate this, we count the number of image frames where the human had to intervene and how many frames the agent successfully drives by itself. We then divide how the number of frames with autonomous driving by the total number of frames. The equation and calculation is listed in equation 1. For each agent, we calculated three autonomy values, one for each lap of the course. We then take the average of these three values to compute the average autonomy value for each agent, shown in equation 2.

$$Autonomy = \frac{Auto}{Auto + Manual} \tag{1}$$

$$AvgAutonomy = \frac{(A_{L1} + A_{L2} + A_{L3})}{3} \tag{2}$$

5. Results

5.1. Simple Track Performance:

Here, we take a look at our results on the simplest track in Mario Kart N64: Luigi Raceway. This track is simple because of its limited and non-sharp turns, which allows us to provide a basis for our supervised learning agent and active learning agents. Looking at table 1, we see that the supervised learning agent performed very poorly, requiring us to intervene 12 times. We only override the agent when it is completely stuck and unable to surpass the wall or obstacle. Even so, this produced an inefficient completion time of 4 minutes and 13 seconds. The next agent performs substantially better, passing through 2 laps before failing on the final third lap. Here, the agent only gets stuck 4 times and has massive improvement of 53 seconds compared to the naive supervised learning agent time. This agent still does not make perfect turns and constantly rams into walls, but is able to slightly correct in these unideal situations. There is even more improvement in Agent 2 with a 37 seconds improvement from Agent 1's time. This agent exhibits new behavior with a few zigzag driving intervals. These zigzags represent incorrect turns, but are corrected immediately due to the correction data we added after the naive supervised learning model. In the times of the next agents, we see a slow and steady decrease of the course time, displaying slight improvement of each model. One exception is Agent 3; however, the agent actually does perform well except only in one spot. Once it learned the appropriate correction for that spot, it performed well as we observe in the later agents. Eventually, however, the agents do not surpass the optimal human player.

5.2. Agent Autonomy on Simple Track:

Here, we look at the average autonomy of each agent starting at the supervised learning agent to the last agent. From figure 3, we see that there is a steady increase of autonomy with the last agent having the best autonomy. The last 4 agents all have very similar autonomies, but the difference between these and the naive supervised learning autonomy is almost over 12%. It is also notable that consecutive agents are not necessarily better than each other, but the trend overall is a steady improvement of autonomy.

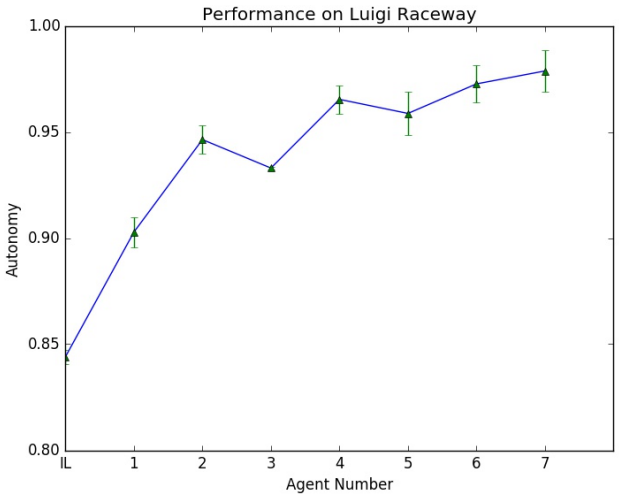

Figure 3. The autonomy value and standard deviation of each agent trained on Luigi Raceway

5.3. Complex Track Performance:

The complex track here is Bowsers Castle. This track is much more complex than Luigi Raceway in that there are multiple moving obstacles, sharp turns, and paths without walls. In Table 2, we see that the many of the agents do not even finish the course or even a lap. Again, we see that the naive supervised learning does the worse, not even passing through the first turn. The next agents do not finish the course, but perform much better than the supervised learning agent. The last agents are actually able to complete the course with a little help from the human controller. The human, however, only intervenes when the agent is completely stuck and is unable to advance in the course. Mistakes such as going off course or crashing into obstacles are left alone and are accounted in the course time completion.

The first viable agent in table 2 is Agent 4. This agent is able to complete Bowsers Castle in 5:28 with help 8 times. We do not account the agents before this one because they are helplessly lost while driving the course and would require numerous human interventions to complete the course. The next agent, Agent 5, does slightly worse than Agent 4, but the latter agents steadily improve. Agent 6 only gets stuck 6 times; while completing the course at a better time of 4:25. The next agent then performs the best by getting stuck 3 times with a completion time of 4:23.

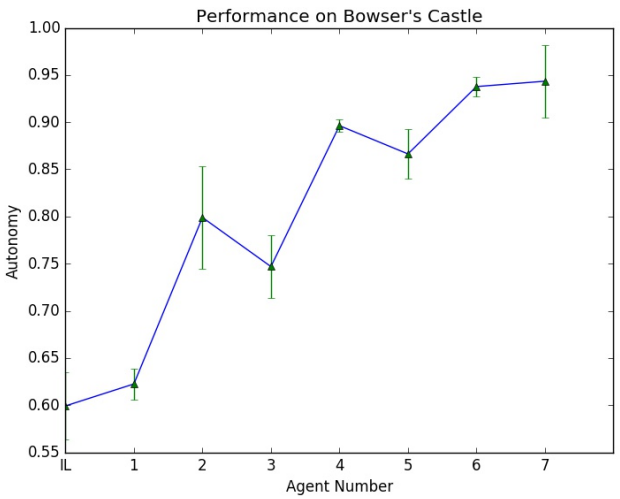

Figure 4. The autonomy value and standard deviation of each agent trained on Bowser's Castle

Agent No.	Course Time	No. of times stuck
IL Agent	DNC	N/A
Agent 1	DNC	N/A
Agent 2	DNC	N/A
Agent 3	DNC	N/A
Agent 4	5:28	8
Agent 5	6:37	10
Agent 6	4:25	6
Agent 7	4:23	3
Human Player	3:04	N/A

Table 2. The completion times (min:sec) for each agent on Bowser's Castle. None of the agents can complete the course without any human intervention. However, Agents 4-7 perform reasonably well to need only a little help. As a result, we record the times for these agents and how much human intervention is needed (number of times stuck).

These agents still exhibit behavior such as zigzag driving, but recovering from mistakes are much more difficult with an intricate course such as Bowser's Castle.

5.4. Agent Autonomy on Complex Track:

Looking at figure 4, we see an overall increase in the autonomy of the agent on Bowsers Castle. With a course so complex, the supervised learning agent requires human intervention almost half of the time. The next agents see a variety of success, but overall a steady increase and an improvement on naive supervised learning. Even with a course so complex, the last agent is able to drive mostly by itself with an autonomy of 95%.

6. Discussion

6.1. Tackling Data Mismatch:

From the results, we can see that the supervised learning agent performed the worst in both courses. This is because of the common data mismatch problem we previously mentioned. The human player is too perfect that the naive supervised learning agent is unsure what to decide in poor situations. In Luigi Raceway, the agent fails on the first turn because it hits a wall and is unable to complete the course because of this. The active learning agents perform much better because the data mismatch is solved by adding correction in these unideal situations. Once the agent hits a wall, it corrects itself appropriately instead of inconclusively deciding where to turn. This then allows the agents to complete the Luigi Raceway course.

6.2. Ability to Correct in Luigi Raceway:

From the results, we see steady improvement from one agent to the next. This is an appropriate illustration that the agents are learning new corrections in each of the agents. We see this in the results because the time to complete the course steadily decreases across agents with Agent 7 having the best time. The zigzags seen in the latter agents illustrate the ability for the agent to correct itself when it marks itself in a bad situation. However, we do not see as much improvement in the latter agents than that of the earlier agents. The earlier agents went from not completing to being able to complete the track at a reasonable time. This is slightly expected because since the track is so simple, there is not much an agent can improve. The latter agents still hit a few walls, but are able to correct from these mistakes more efficiently. Overall, the agents definitely exhibit ability to correct itself and a steady increase in performance, but with such a simple course, the performance will most likely plateau.

6.3. Ability to Correct in Bowsers Castle:

The results for Bowsers Castle are slightly more varied because of the course complexity. Overall, they

show a gradual improvement and the ability to correct itself. This is seen in the decrease number of human interventions in the latter agents as well as the decrease of completion time. The supervised learning agent is unable to complete the course because once it is stuck in a spot, it cannot figure out how to fix itself. The agents afterwards are able to correct mistakes, but we do not get a functionable agent until Agent 4. This is also due to the complexity of the course because the corrections are more complex as well. Bowsers Castle has numerous course sections where the agent can get stuck. Corners, moving obstacles, ledge jumps are just a handful of situations where an agent can struggle to surpass. With so many obstacles, more correction data is needed, leading to more training processes to produce a functionable agent. While none of the agents are able to complete the track by itself, we see the help of the correction data in the decline of human intervention and decrease in completion time in the latter agents.

6.4. Variability in Autonomy:

In figure 4, we see a lot of variability between each of the agents autonomy. Again, this is most likely due to the variability and complexity of Bowsers castle. Moving obstacles are unpredictable and can knock the kart into a wide variety of sections of the course. As a result, a latter agent may not appear better than previous agents; however, this is possibly because of the unpredictably of the course. An agent could be performing optimally, but could be knocked into a suboptimal path that greatly decreases its overall performance.

7. Conclusion

Overall, our results show a clear improvement from the supervised learning agent to the later active learning agents. Our hypothesis is correct in that the latter agents also trained with correction data are able to fix themselves in unideal situations. Analyzing performance in Luigi Raceway, we saw a steady improvement in the latter agents and a significant difference between the supervised learning agent and the latter active learning agents. However, we predict that the improvement will slow because of the lack of complexity, leading to a lesser impact of correction on the completion time.

Thus, we decided to test our agents on a more intricate course: Bowser's Castle. Even though it took more iterations of our algorithm, we produced a viable agent that performs significantly better than the supervised learning agent. With so many unpredictable sections of the track, the results of the latter agents showed some variation, but again overall the performance increases over iterations of our algorithm.

In addition to a better performing agent, our method is more practical and easy to implement. We only needed to correct the kart when we deemed it was in a precarious situation instead of labeling each image with exact values. This makes the method also more realistic and relatable to real-life cars and robots. A human wouldn't need to immediately respond to an agent's queries because he or she controls when these queries occur.

Acknowledgments

Thanks to Professor Lisa Meeden for all of her support and assistance with this project.

References

Bojarski, Mariusz, Del Testa, Davide, Dworakowski, Daniel, Firner, Bernhard, Flepp, Beat, Goyal, Prasoon, Jackel, Lawrence D, Monfort, Mathew, Muller, Urs, Zhang, Jiakai, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

Hughes, Kevin. Tensorkart. https://github.com/kevinhughes27/TensorKart, 2013.

Hussein, Ahmed, Gaber, Mohamed Medhat, and Elyan, Eyad. Deep active learning for autonomous navigation. In *International Conference on Engineering Applications of Neural Networks*, pp. 3–17. Springer, 2016.

Ross, Stéphane and Bagnell, Drew. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 661–668, 2010.

Ross, Stéphane, Gordon, Geoffrey J, and Bagnell, Drew. A reduction of imitation learning and struc-

tured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 627–635, 2011.

Ross, Stéphane, Melik-Barkhudarov, Narek, Shankar, Kumar Shaurya, Wendel, Andreas, Dey, Debadeepta, Bagnell, J Andrew, and Hebert, Martial. Learning monocular reactive uav control in cluttered natural environments. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 1765–1772. IEEE, 2013.

Zhang, Jiakai and Cho, Kyunghyun. Query-efficient imitation learning for end-to-end autonomous driving. *arXiv preprint arXiv:1605.06450*, 2016.