Improving Training for End-to-End Autonomous Driving

Peggy (Yuchun) Wang
Department of Computer Science
Stanford University
peggy.yuchun.wang@cs.stanford.edu

Vinh Nguyen
Department of Electrical Engineering
Stanford University
vnguyen5@stanford.edu

***Abstract*—Autonomous driving has the potential to save thousands of human lives in traffic accidents and is a recent popular area of interest. Recent research has focused on perception and decision making for autonomous driving, including end-to-end learning where steering angle is predicted directly from camera images. In this project we focus on improving end-to-end learning approaches with data augmentation methods and online learning. Our results show that online learning methods such as DAgger improved end-to-end learning with better lap time results, and that data augmentation methods similar to DART has a similar effect as collecting more data. Future work includes experiments with additional generalizations in simulation environments, such as adding obstacles or driving on different tracks.**

I. Introduction

Autonomous driving has the potential to save thousands of human lives in traffic accidents and is a recent popular area of interest. The areas of perception and decision making is integral to the development of a fully autonomous vehicle. Often these systems are large and complex, combining multiple sensor modalities, feature detection, semantic segmentation, and rule-based trajectory planning. End-to-end learning attempts to go directly from raw input (such as image data) to control output, relying on the network to learn appropriate driving behaviors from the data. This minimalist approach could result in smaller and more efficient agents.

Our project focuses on improving behavioral cloning from end-to-end learning methods to generate steering angles. We use a slightly modified version of the Udacity Self Driving Car Simulator[10] to test our algorithm, and experiment with online learning methods inspired by DAgger[7] and data augmentation methods inspired by DART[3]. Using a variant of NVIDIA's end-to-end learning architecture[1] as a baseline, we are able to achieve a similar or better result in terms of lap time around the simulator's "lake track" using these online learning methods in fewer epochs than simply training on a static dataset.

II. Related Work

The DARPA Urban Challenges are the first competitions that sparked an increase in autonomous driving research. Perception and decision making methods for autonomous driving has improved since the 2007 DARPA Urban challenge. The winning car Tartan from CMU[11] used a state machine to generate behaviors such as "Drive-Down-Road" and "HandleIntersection" and used perception features such as lane boundaries to handle following lanes. The second place car Junior from Stanford[4] also uses a similar approach for behavioral generation using state machines and only focused perception on segmenting static and moving obstacles.

More recently, more sophisticated techniques in deep learning and convolutional neural networks (CNNs) have been developed for use in perception[6], including segmentation of point cloud obstacles and road detection. In regards to planning and decision making, Schwarting et al.[8] noted several end-to-end planning methods, where images are input into a deep neural network (NN) and output paths are directly generated.

Bojarski et al.[1] first pioneered the approach of using end-to-end learning from raw image inputs directly to steering angle commands. Farag[2] applied this learning method successfully to the Udacity Self Driving Car Simulator[10], which we replicate in our baseline system.

We experiment on improving the end to end learning methods used in the baseline by using online learning methods inspired by DAgger[7]. DAgger is a form of online learning from expert data, where new data is collected from the trained policy and then retrained using expert labeling. Since the Udacity simulator does not support expert labeling during autonomous driving, after we train an agent, we only start collecting data when the agent is about to drive off the road. We then retrain the agent using the new data.

Additionally, we also experiment on improving the baseline by using data augmentation methods inspired by DART[3]. DART involves injecting random noise into the expert data policy, which is more helpful for the agent to learn if it encounters a state that it does not see in the training data. In our case, we inject small uniform random noise of ± 0.01 into the steering angle for 60% of images to simulate slightly different actions.

III. Data

We collect data by manually driving around the Udacity Self Driving Car Simulator, which contains a "lake track" and a "forest track". Each track contains different features to denote lane lines and different variations in terrain to serve as 'edge cases' for the agent.

We are able to manually drive around the "lake track" (Figure 1) using a mouse and keyboard. A built-in record

Fig. 1: User view of Lake Track

(a) Left Camera Image

(b) Center Camera Image

(c) Right Camera Image

Fig. 2: Examples of Left, Center, and Right Camera Images from Dataset in Same Timestamp

function captures images at around 20 FPS with three front-facing cameras mounted on the vehicle. After driving around for three laps, we collected 4812 RGB 320x160 pixel images for each of the left, center, and right cameras of the vehicle, for a total of 14,436 images, which served as the training and validation set for the baseline model. Each image (Figure 2) was labeled with the image name, its corresponding steering angle (in radians), throttle command, and speed in a CSV file. As there is only one steering angle for three images, an offset of +/-.2 radians is added to the annotated steering angle for the right and left camera images, respectively. We then process all the images and labels and split it up into a training set and testing set. For each sample in the training set, we choose one of the left, center, or right image uniformly randomly, with the corresponding steering angle. We then perform data augmentation on our dataset by randomly flipping, translating, adding shadows, and adding brightness to the camera images 60% of the time per sample, and keep the original labeling the other 40% of the time. For the experimental DART method, we add random noise from -0.01 to 0.01 to the steering angle. We also resize and crop the top half of our images to 66x200 for faster training. We then train the data on our model, which outputs predicted steering angles.

IV. Methods and Evaluation

A. System architecture

We use the model architecture from Bojarski et al.[1], where we input the RGB images and output a predicted steering angle. We use an implementation of this model for Udacity's simulator by github user nakishibuya[5], trained on our own dataset, as a baseline. We made modifications to this code for our own experimentation with different models and data preprocessing. The baseline model consists of 5 Convolutional layers and 4 fully connected layers with dropout. All layers used ELU activation functions. The model architecture is described in detail in Figures 3 and 4. The model takes in a single RGB image, which in training may be any of the three camera images. During inference, however, only the center camera image is passed to the network. The baseline system uses a static control algorithm to compute the throttle for the car; this is not an output of the model as it only provides predicted steering angle. The control for throttle is given below:

$$throttle = 1.0 - \hat{\theta}^2 - (\frac{v_{cur}}{v_{lim}})^2$$

This reduces the throttle from the maximum of 1.0 based on the magnitude of the steering angle and how close the current speed is to a software-set speed limit.

B. Training

We built the model in Keras and trained for 8 epochs, each consisting of 20000 samples of batch size 40 for a total of 6.4M images consumed. The training of 8 epochs took approximately 10 hours on a MacBook Pro 2018. Additional hyperparameters are described in Table 1.

We also trained the baseline model with the same hyperparameters on a Microsoft Surface Pro 4 for 35 epochs, 10000 samples, 40 batch size (or 14M images consumed), taking 20 hours. The model trained on Windows was used for the experiments involving DAgger and the model trained on Mac is used for the DART experiment. The reason for this is due to inconsistent behavior of the simulator across different hardware, which we discuss in further detail in the Limitations section. From this point on, we will refer to one epoch as 10000 samples with batch size 40 for consistency.

```
model = Sequential()
model.add(Lambda(lambda x: x/127.5-1.0, input_shape=INPUT_SHAPE))
model.add(Conv2D(24, 5, 5, activation='elu', subsample=(2, 2)))
model.add(Conv2D(36, 5, 5, activation='elu', subsample=(2, 2)))
model.add(Conv2D(48, 5, 5, activation='elu', subsample=(2, 2)))
model.add(Conv2D(64, 3, 3, activation='elu'))
model.add(Conv2D(64, 3, 3, activation='elu'))
model.add(Dropout(args.keep_prob))
model.add(Flatten())
model.add(Dense(100, activation='elu'))
model.add(Dense(50, activation='elu'))
model.add(Dense(10, activation='elu'))
model.add(Dense(1))
```

Fig. 3: Baseline Model Architecture

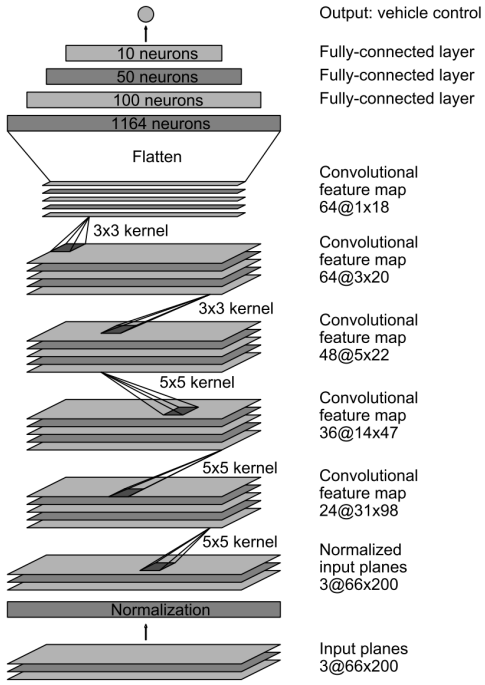

Fig. 4: Baseline Model Architecture Visualization

C. Experiment Setup

For the DAgger algorithm, expert annotated actions over agent-controlled state exploration needed to be collected. We model the expert policy with a human driver using manual control. The Udacity simulator does not support recording data in autonomous mode, so we used the Unity game engine to modify the simulator to support recording data in autonomous mode. Additionally, we modified the simulator to increase the car's speed limit. In order to have expert annotations over states explored purely by the agent, the expert would have to estimate which steering angle to take without the feedback of controlling the car. This proved intractable, but it is not desirable to have a fully agent-controlled state exploration anyway (the car may go off-road or crash, situations which are unrecoverable and thus must be avoided). Therefore, we allow the agent to drive in autonomous mode and take over manual control of the car when the agent is about to drive off the road. We record data during the segment where the expert driver intervenes for the agent, and then retrain the agent using the new data. For each round of data collection, the agent drives three laps and any interventions needed are recorded. The resulting dataset additions throughout our experiments ranged from 1575 to 2406 images.

Table 1: Baseline Hyperparameters

Hyperparameter	Baseline
Input Size	66x200
Dropout Probability	0.5
Activation Function	ELU
Loss Function	MSE
Number of Epochs	8
Samples Per Epoch	20000
Learning rate	.0001
Optimizer	Adam
Batch Size	40

For the DART method, we first collected a smaller dataset after driving only one lap of 1275 images each for left, center, and right cameras, for a total of 3825 images. We then augment the data as usual, and this time adding uniform random noise from -0.01 to 0.01 to the steering angle label. We then train the model the same way as the baseline with the same parameters.

D. Evaluation

We evaluate the model's performance based on the average lap time over a three-lap time trial. In testing we raise the set speed limit for the agent and rerun until it can no longer successfully complete three laps, and we record the best time. In addition, we record the amount of data added and the number of epochs trained to compare how quickly the experimental training methods can train the agent to drive successfully.

V. Experiments

We run three experiments to compare against the baseline: DAgger, DAgger-Lite, and DART. Each of these experiments start with a pre-trained model that is an early checkpoint of the trained baseline model, one which has yet to learn to fully drive around the track. In the case of the Windows baseline, we chose the model trained for 10 epochs, and in the case of the Mac baseline, we chose the model trained for 6 epochs. The experiments are described below:

1) DAgger: 3 rounds of data collection; dataset for each cycle is the union of the previous cycle's dataset and that cycle's expert-annotated data. The initial dataset is the training dataset.
2) DAgger-Lite: 3 rounds of data collection, however the dataset for each cycle is fixed at N=16384 images sampling the expert-annotated data at probability $p = .5$ and the previous cycle's dataset at probability $1-p = .5$. In other words, the dataset weights more heavily the most recent experience given by the expert.
3) DART: A full lap of driving is recorded with noise injected into the steering angle. This dataset (N=3825)

replaces the training dataset for the remainder of the model's training.

The DAgger models were trained for 1 epoch, checkpointed at 5 equally spaced intervals, for each round of data collection. DART was trained for 8 epochs. The performance was evaluated at the first checkpoint where the model could drive around the track at all, and, in the case of the DAgger experiments, at the end of each round of data collection and training.

VI. Results

The Mac baseline model trained for 16 epochs received a final validation loss of 0.0268 and a final training loss of 0.0330. The earliest checkpoint able to drive around the track more than three times was at 8 epochs, with an average lap time of 1:54. The DART Model equalled the Mac baseline performance with an additional 8 epochs (14 total).

The Windows baseline model trained for 40 epochs finished with a training loss of .0234 and a validation loss of .0230. The earliest checkpoint able to drive around the track was at 35 epochs; this model managed an average lap time of 1:34. The DAgger model was able to drive after .4 epochs' worth of additional training in the first data collection cycle, achieving a best lap time of 85 seconds, and this did not improve by the end of the first cycle. It improved its laptime to 63 seconds after the second round of data collection and to 56 seconds after the third round of data collection.

The DAgger-Lite model performed worse than the DAgger model, requiring the full epoch of training in the first round of data collection before being able to complete 3 laps, doing so with a lap time of 116 seconds. It improved to 71 seconds after the second round of data collection, and to 68 seconds after the third round of data collection. The full results and relevant parameters are tabulated in Figure 5.

Table 2: Results

Model	Epochs	Loss (Train)	Loss (Val)	Ave Lap Time (s)	Total Data Collected
Baseline(Win)	35	.0242	.0224	94.3	14436
Baseline(Mac)	8	.0410	.0482	113	14436
Baseline(Mac)	14	.0151	.0164	115	18261
D1-002 (Win)	10.4	.0415	.0102	85.3	16011
D1-005 (Win)	11	.0408	.0311	85.6	16011
D-Lite1-005 (Win)	11	.0522	.0459	116	16011
D2-005 (Win)	12	.0374	.0925	63.3	17634
D-Lite2-005 (Win)	12	.0478	.0119	71.3	18417
D3-005 (Win)	13	.0333	.0784	56.7	19257
D-Lite3-005 (Win)	13	.0304	.0094	68.0	20691

Fig. 5: Results for experiments at points of interest. Nomenclature: x-00y, where x is the data collection cycle and y is the checkpoint, where 005 is 1 epoch trained. Epochs normalized to 10000 samples, batch size 40.

VII. Discussion

One interesting result of our experiments is that we found no strong correlation between the performance of the agent and the validation MSE between the predicted and actual steering angle, although some reduction of training error was needed before the agent was able to drive successfully. The results for each model over the course of training are plotted in Figure 6 and 7. This indicates that similarity to the expert is not crucial for driving proficiency, at least for a problem as unconstrained as ours. For one, there are many states the agent may encounter that it has never seen in the data; in such cases similarity to the expert is not guaranteed to produce a correct action. In addition, there are many ways to take a turn in a track without obstacles in the lanes; since there are many appropriate ways to take a turn, dissimilarity to the expert doesn't imply poor driving.

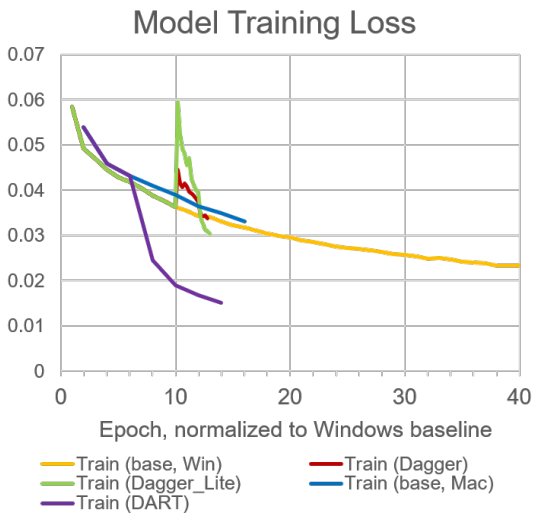

Fig. 6: Model Training Loss over time. Although training loss goes down over time, we found that there is little correlation between training loss and how well the model drives autonomously.

Both DAgger and DAgger-Lite had better lap times compared to our baseline, although DAgger performed better than DAgger-Lite. They were also able to reach a proficient level of driving in considerably less training time than the baseline. In testing some of the earlier baseline model checkpoints, we found that the agent could mostly drive around the track, except for a couple edge cases where it would get confused. By adding data specifically showing how the expert recovers from these edge cases not found in the training data, the DAgger models are quickly able to learn how to overcome these edge cases. We did not test whether a randomly initialized model trained on the aggregated datasets would reach an equivalent level of performance in less total epochs due to lack of time, but this would not be a good comparison to DAgger either. The baseline model was needed to collect the data in the first place - we could not have known which data to collect without first knowing which parts of the track are the edge cases.

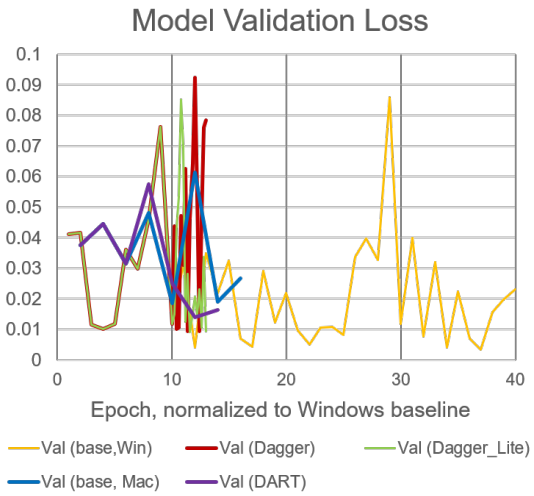

Fig. 7: Model Validation Loss over time. We found that validation loss was not a good predictor of how well the model drives autonomously.

The difference in the training time for the DAgger compared to the baseline model echoes the intuition that most driving situations are very quickly learnable, but the difficult parts involve unforeseen states or scenarios. A driving agent then might be trained most efficiently by starting with simpler situations and progressively adding more complicated ones to teach more complex behaviors, rather than attempting to learn several patterns in the data at once. The question of where to cut off training and add more data is an open-ended one, which we leave to future research.

Between DAgger and DAgger-Lite, we found that DAgger-Lite performed worse at each cycle than unmodified DAgger. This was true even while the data was the same (for the first data collection cycle, the data collected is the same because the models are identical). This suggests that the model may be focusing too much on edge cases with $p = .5$ and consequently forgetting other patterns in the original training dataset. We thought that this might still result in DAgger-Lite learning the edge cases faster in the first cycle due to the annotated edge case data being weighted fivefold in comparison to DAgger, but this was not the case, suggesting that there may have been useful patterns in the training data yet to be learned that DAgger-Lite missed out on due to statistically prioritizing that data less.

A. Limitations

There were several challenges that we dealt with when conducting our experiments. The first (referred to earlier in the Methods section), is that the simulator behaves differently on different hardware, which resulted in models trained on one of our systems (Mac) not being transferable to the other (Windows). This is documented in the Udacity project repository README[9]. Due to the long training time, we were forced to run separate experiments with separate baseline models. The models' performances were not easily comparable across the DAgger and DART experiments because we could not exchange models and compare performance, and the baseline did not perform the same for the same amount of training. In addition, because data had to be collected separately, driving style between different people and even with the same person across different runs may have affected the distribution of data between experiments, so the number of images collected may be a poor point of comparison. Occasional frame rate drops on Windows also affected model behavior because only one action can be made per frame. We tried our best to maintain consistent testing conditions, but a proper test would involve much better hardware and a simulator with a capped frame rate. Lastly, it was observed that the simulator executable published by Udacity (used for the Project Milestone results) behaved differently than the one we built from Unity (with both modified and unmodified source with respect to the Udacity repository) possibly due to differences in Unity versions; as a result we had to discard old results and start over with the modified simulator used for our experiments.

B. Future Work

We were surprised that our DAgger-Lite performed better than DAgger, so we would like to explore why. This could be because DAgger-Lite oversamples edge cases compared to normal driving cases, which hurts overall performance during normal driving conditions.

Additionally, we also would like to explore how well DAgger-Lite generalizes for additional edge cases and complex environments, such as when there are random obstacles on the road or different tracks and environments. This may require using a different simulation environments or heavily modifying our existing simulator.

Lastly, we would like to explore the output control of the throttle as well as the steering angle in an end-to-end learning network. Our current model fixes throttle as a function of the output steering angle, but perhaps that is not the optimal throttle. This means that we would have two independent outputs for the neural network - both throttle and steering angle.

VIII. Conclusion

We replicated the NVIDIA model with a dataset generated from our own manual driving (augmented according to the published randomized techniques described above). We conducted experiments using DAgger and DART and found that incorporating expert annotations over partially-trained agents' failing edge cases results in faster learning towards driving proficiency. DAgger performed the best in terms of lap time, followed by DAgger-Lite, and lastly the baseline model. Focusing too much on edge cases in DAgger-Lite, however, while it allows the dataset size to be limited, may not be as effective as simply taking all the data available. We also found that train loss and validation loss are not incredibly descriptive methods for how well the car drives (other than indicating overfit when training loss is much lower than validation loss); the dataset's coverage of the state space seems to be a more

critical contributor to the quality of agent that can result from training.

Supplementary Material

The source code and driving videos may be found in the following Google Drive link: https://drive.google.com/drive/folders/1ZJG7T89krM1gHkKcp93FsKbx-LwBysqW?usp=sharing.

Acknowledgments

We are grateful to CS 336 teaching staff for their help and support.

References

[1] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016. URL http://arxiv.org/abs/1604.07316.

[2] W. Farag and Z. Saleh. Behavior cloning for autonomous driving using convolutional neural networks. In *2018 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, pages 1–7, Nov 2018. doi: 10.1109/3ICT.2018.8855753.

[3] Michael Laskey, Jonathan Lee, Roy Fox, Anca Dragan, and Ken Goldberg. Dart: Noise injection for robust imitation learning. *arXiv preprint arXiv:1703.09327*, 2017.

[4] Michael Montemerlo, Jan Becker, Suhrid Bhat, Hendrik Dahlkamp, Dmitri Dolgov, Scott Ettinger, Dirk Haehnel, Tim Hilden, Gabe Hoffmann, Burkhard Huhnke, et al. Junior: The stanford entry in the urban challenge. *Journal of field Robotics*, 25(9):569–597, 2008.

[5] Naokishibuya. naokishibuya/car-behavioral-cloning, Apr 2019. URL https://github.com/naokishibuya/car-behavioral-cloning.

[6] Scott Pendleton, Hans Andersen, Xinxin Du, Xiaotong Shen, Malika Meghjani, You Eng, Daniela Rus, and Marcelo Ang. Perception, planning, control, and coordination for autonomous vehicles. *Machines*, 5(1):6, 2017.

[7] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

[8] Wilko Schwarting, Javier Alonso-Mora, and Daniela Rus. Planning and decision-making for autonomous vehicles. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1):187–210, 2018. doi: 10.1146/annurev-control-060117-105157. URL https://doi.org/10.1146/annurev-control-060117-105157.

[9] Udacity. udacity/carnd-behavioral-cloning-p3, Jun 2018. URL https://github.com/udacity/CarND-Behavioral-Cloning-P3.

[10] Udacity. udacity/self-driving-car-sim, Jan 2019. URL https://github.com/udacity/self-driving-car-sim.

[11] Chris Urmson, J Andrew Bagnell, Christopher Baker, Martial Hebert, Alonzo Kelly, Raj Rajkumar, Paul E Rybski, Sebastian Scherer, Reid Simmons, Sanjiv Singh, et al. Tartan racing: A multi-modal approach to the darpa urban challenge. 2007.