Autonomous R/C Car Behavioral Cloning Optimization

Joon Jung
joonjung@stanford.edu

1 Introduction

Behavioral cloning [2] is relatively simple to implement but yields optimal result efficiently. In this project I have used behavioral cloning to train a CNN supervised classifier autopilot, based on an open source platform know as Donkey Car [4]. The goal of the project is to model and optimize the autopilot in a real world setting, other than a simulated one, trying to gain valuable insights to launch a real world machine learning agent. In order to improve the autopilot's performance, I have employed Data Aggregation [3] to augment the training process.

The model car is equipped with a mono frontal wide angle camera, capturing 120x160 RGB images, which are used for the training and testing inputs for the autopilot. In addition, the model car's steering angle and motor throttling values are used for the classification labels, so the autopilot can estimate and output the best steering angle and throttling output given an input image in the testing phase.

2 Related Works

Bojarski et all. [1] have shown that it is possible to use a CNN based supervised model to drive a car. The work has used three frontal cameras, using the middle camera as the main source for the agent's inputs and using the side cameras to compensate the car's shift and rotation movements. It basically has relied only on the frontal captured images to classify the right steering angle to keep the car on the track. This modeling is quite simple to come up with a decent performance, if it is trained with sufficient amount of data. However, the biggest problem is that once it encounters a state which it has not seen before, it is very easy for the agent to drift away significantly (Fig. 1).

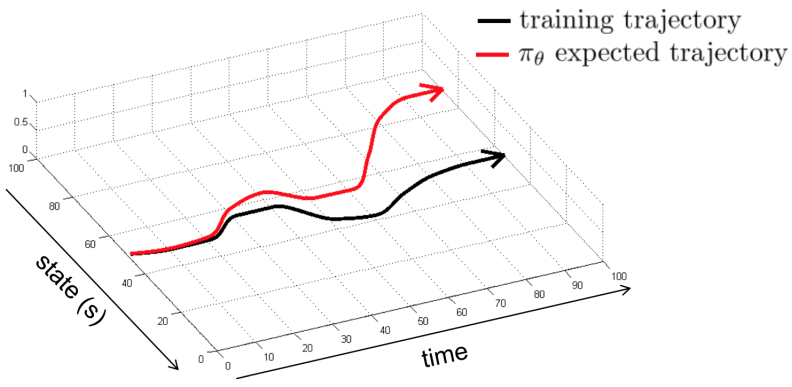


Figure 1: Behavioral Cloning trajectory drifting, Image from [6]

As shown by Ross et all. [3], using a supervised learning to mimic the expert optimal policy π^* based on its state distribution d_{π^*}, therefore ignoring its own state distribution, can only guarantee to

achieve a bound of $O(T^2)$ for its cost function $J(\pi)$ compared to the optimal policy cost function $J(\pi^*)$, where T designates its time varying trajectory. That is, letting $E_{s\sim d_{\pi^*}}[(l(s,\pi)] = \epsilon$, then

$$J(\pi) \leq J(\pi^*) + T^2\epsilon.$$

where $l(s,\pi)$ is the 0-1 loss of π with respect to π^* in state s.

As to optimize the bound of this approach, the same work has suggested Data Aggregation which can achieve the cost $J(\pi)$ to be bounded linear to the trajectory T, as shown from the following theorem from the same work.

Letting N to designates the number of iteration to perform the data aggregation (Section 5.3), if N is $\widetilde{O}(uT)$, then there exists a policy $\hat{\pi} \in \hat{\pi}_{1:N}$ such that

$$J(\hat{\pi}) \leq J(\pi^*) + uT\epsilon N + O(1).$$

3 Donkey Car

Our model car has a mono wide angle camera, a servo controlled steering and a thrust motor (Fig. 2). The model car's brain is a raspberry pi portable computer, capable of running Keras models with Tensorflow backend. The car is trained on an indoor track, constructed simply with white tapes.

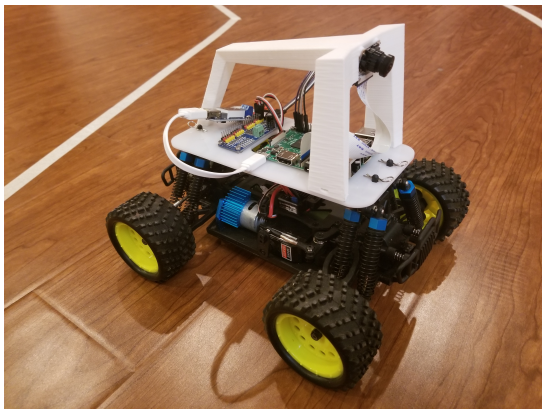


Figure 2: Donkey car

4 Dataset and Features

The primary inputs for the training and testing are the 120x160 RGB images captured from the frontal camera, classifying the best matching steering angles and the throttling values. The agent has performed about total 200 wraps of running on the indoor track in multiple sessions, each wrap equaling to capturing about 520 images, their corresponding steering angles and throttling values. The training and validation is performed with split ratio of 0.8:0.1:0.1 between the training, developing and validation sets.

5 Method

5.1 CNN Autopilot

The Figure 3 shows the architecture of the autopilot. It is consisted of 5 convolution layers, followed by a flattening layer, and 3 fully connected layers. Also after each fully connected layers, there is a dropout layer for regularization which is not shown in the figure. Also not shown in the figure, there is another fully connected layer at the end connected in parallel to the second dropout. These two end full layers classify the given input image(120 x 160 RGB) with the best matching steering angle and throttling values in each.

The image on the right in the same figure shows an input image, super imposed with a masking to show the image segments activating the CNN most [5].

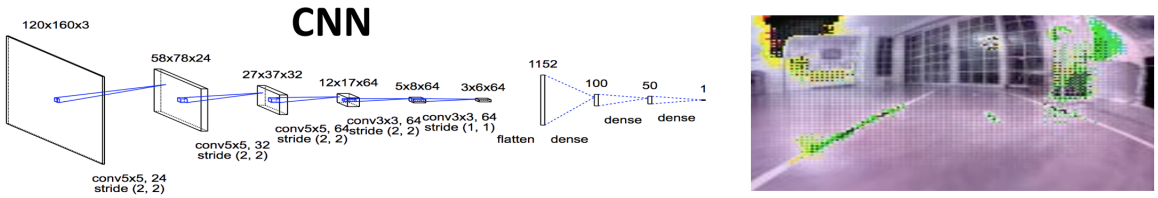


Figure 3: CNN autopilot

5.2 Behavioral Cloning

As disscussed by Yue et all. [2], we can model the supervised classification CNN autopilot using Makov Decision Process. The frontal captured camera image inputs, the steering angles and the motor throttling values together form the states of the agent. The sequence of the states is designated as **s**. The steering and throttling also form each action the agent can exert in the states. The sequence of actions is designated as **a**. The collected timed sequence of the tuple of **s** and **a** is $D = \{\tau := (\mathbf{s}, \mathbf{a})\}$.

Then by collecting the τ^* from the expert policy π^*, we can set the learning objective function as

$$argmin_\theta E_{(s,a^*)\sim P^*} L(a^*, \hat{\pi}_\theta(s))$$

where $P^* = P(s|\pi^*)$ designating the distribution of states visited by the expert.

5.3 Data Aggregation

In order to improve the baseline autopilot policy π, I have employed Data Aggregation [3]. Given the expert policy estimated distribution π^*, we train the first policy π_1 from the training examples of D^*. Then the policy π_1 gets deployed to collect another example set D_1. Next the collected D_1 gets modified by the expert to be $D_1 = \{(\mathbf{s}, \pi^*(\mathbf{s})\}$. Here the state sequence **s** designates the state sequence collected in D_1. Then the process iterates, merging each D_i to form the super set D.

The following lists Data Aggregation algorithm.

D = $\{\}$
Train first policy π_1 based on D^*
for i = 1 to N **do**
  Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\pi^*$.
  Sample T step trajectories using π_i.
  Get dataset $D_i = \{(\mathbf{s}, \pi^*(\mathbf{s})\}$ of visited states by π_i and actions given by expert.
  Aggregate $D = D \cup D_i$.
  Train π_{i+1} on D.
end for
Return best policy π_i on validation.

For this project, β_1 is set to 1 and β_i is set to 0 for $i > 1$.

Also specific to this project, $\pi^*(\mathbf{s})$ in the iteration i is achieved by the expert manually modifying the actions by its best estimation. For a specific example, Figure 5 shows one instance section of the actual D_i with π_i being modified, with which originally the agent failed to stay on the track.

6 Experiment Results

The table in Figure 5 summarizes the experiment results. As the metric, it has used the average number of times the agent going out of the track in percentage.

The first row of the table shows the failure rates of each individual policy π_i without merging all the datasets from each iterations, that is not performing $D = D \cup D_i$ for the next training dataset. The first column shows the failure rate of the human driver. π_1 is the first policy trained with the training

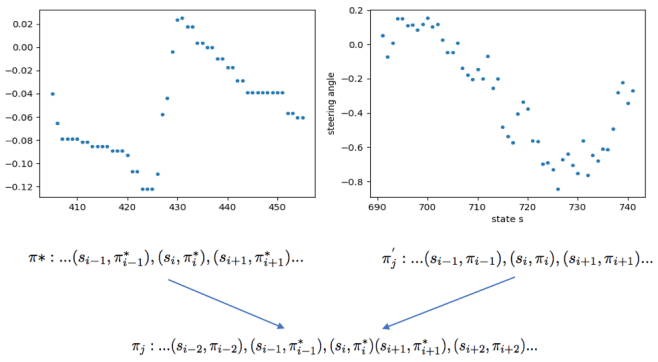


Figure 4: Manual action correction by expert

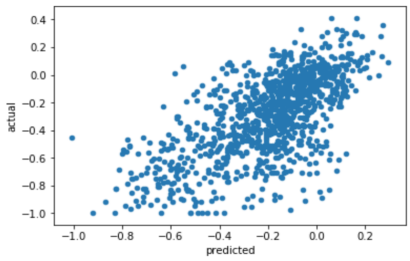
Datasets Aggregated	Human	π_1	π_2	π_3
π_i isolated	10%	26.6%	20%	100%
Aggregating each π_i	n/a	100%	n/a	n/a

Figure 5: Experiment results

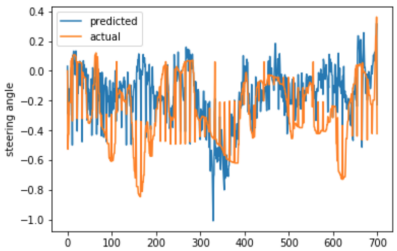
data collected from the human driver. π_2 is the first policy which is generated by manually modifying the misbehavior using π_1. π_3 is the next iteration policy generated from π_2.

The second row shows the case of the dataset being merged, as $D = D \cup D_i$. Only the first iteration is conducted stopping early as the agent failed to track from the start with the given policy.

The Figures 6 thorough 9 show the scatter plots and the trending graphs between the actual and predicted steering values for each policies.



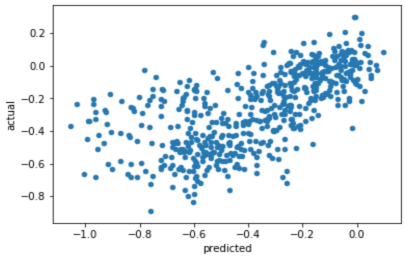
(a) predicted vs. actual scatter plot



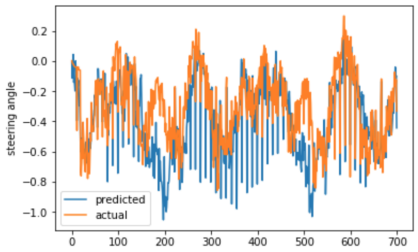
(b) predicted and actual trending plot

Figure 6: π_1

The isolated π_2 has achieved the best performance while the isolated π_3 has failed to detect the track at all, even though the scatter plot and trending graph seem to show better correlations. This might be indicating that the failure reason for π_3 is not actually its trending capability with the actual values, but something else. On the other hand, the case for π_1 merged shows a complete failure of correlating with the actual values.

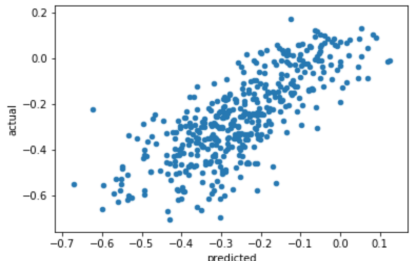


(a) predicted vs. actual scatter plot

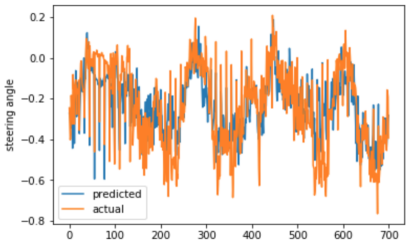


(b) predicted and actual trending plot

Figure 7: π_2

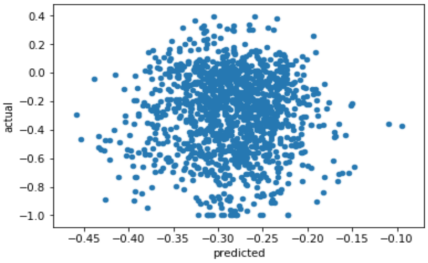


(a) predicted vs. actual scatter plot

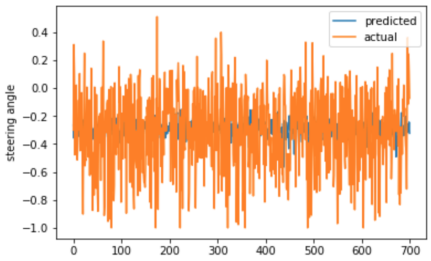


(b) predicted and actual trending plot

Figure 8: π_3



(a) predicted vs. actual scatter plot



(b) predicted and actual trending plot

Figure 9: π_1 merged

7 Conclusion/Future Work

Just by referring the trending graphs, it might be the case that π_3 could have performed better than all of its predecessors. On the other hand, for the merged policy case, I am not certain what destroyed its modeling completely. These will have to be for the future works.

Also If I had more time, I would have tried modeling the policy through a reinforcement learning model other than an imitation learning tried here. Even though I don't have any quantitative data to support, after training the autopilot many times, it seems like confirming the fact that the dependency of current state to its past states plays very important role for the agent's robustness (Section 2). Launching a real world agent using a reinforcement learning, taking it out of the simulated environment will pose very interesting challenging problems.

8 References

[1] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., ... Zhang, X. (2016). End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316.

[2] Yisong Yue, Hoang M. Le: ICML2018: Imitation Learning.

[3] Stéphane Ross, Geoffrey J. Gordon, J. Andrew Bagnell: No-Regret Reductions for Imitation Learning and Structured Prediction. CoRR abs/1011.0686 (2010).

[4] Donkey Car.

[5] Keras Salient Object Visualization.

[6] Sergey Levine: Supervised Learning of Behaviors.

[7] Project Source Codes