Sample Efficient Interactive End-to-End Deep Learning for Self-Driving Cars with Selective Multi-Class Safe Dataset Aggregation

Yunus Bicer\textsuperscript{1}, Ali Alizadeh\textsuperscript{2}, Nazim Kemal Ure\textsuperscript{3}, Ahmetcan Erdogan\textsuperscript{4} and Orkun Kizilirmak\textsuperscript{4}

Abstract—The objective of this paper is to develop a sample efficient end-to-end deep learning method for self-driving cars, where we attempt to increase the value of the information extracted from samples, through careful analysis obtained from each call to expert driver’s policy. End-to-end imitation learning is a popular method for computing self-driving car policies. The standard approach relies on collecting pairs of inputs (camera images) and outputs (steering angle, etc.) from an expert policy and fitting a deep neural network to this data to learn the driving policy. Although this approach had some successful demonstrations in the past, learning a good policy might require a lot of samples from the expert driver, which might be resource-consuming. In this work, we develop a novel framework based on the Safe Dataset Aggregation (Safe DAgger) approach, where the current learned policy is automatically segmented into different trajectory classes, and the algorithm identifies trajectory segments/classes with the weak performance at each step. Once the trajectory segments with weak performance identified, the sampling algorithm focuses on calling the expert policy only on these segments, which improves the convergence rate. The presented simulation results show that the proposed approach can yield significantly better performance compared to the standard Safe DAgger algorithm while using the same amount of samples from the expert.

I. INTRODUCTION

Recent years saw significant advances in self-driving car technologies, mainly due to several breakthroughs in the area of deep learning. In particular, the use of vision-based methods to generate driving policies has been of interest to a vast body of researchers, resulting in a variety of different learning and control architectures, that can be roughly classified into classical and end-to-end methods. Conventional methods approach the problem of autonomous driving in three stages: perception, path planning, and control \cite{1}. In the perception stage, feature extraction and image processing techniques such as color enhancement, edge detection, etc. are applied to image data to detect lane markings. In path planning, reference, and the current path of the car is determined based on the identified features in perception. In the control part, control actions for the vehicle such as steering, speed, etc. are calculated from reference and the current path with an appropriate control algorithm. The performance of the classical methods heavily depends on the performance of the perception stage, and this performance can be sub-optimal because of the manually defined features and rules in this stage \cite{2}. Sequential structure of the classical methods might also lead to the non-robustness against errors, as an error in feature extraction can result in an inaccurate final decision.

On the other hand, end-to-end learning methods learn a function from the samples obtained from an expert driving policy. The learned function can generate the control inputs directly from the vision data, combining the three layers of the classical control sequence into a single step. By far, the most popular approach for representing the mapping from images to controls in end-to-end driving is using neural networks (NN). ALVINN by Pomerleau \cite{3} is one of the initial works in this area, which uses a feedforward neural network that maps frames of the front-facing camera to steering input. Researchers from Nvidia utilized convolutional neural networks (CNN) \cite{4} to automatize the feature extraction process and predict steering input. An FCN-LSTM architecture \cite{5} is proposed to increase learning performance with scene segmentation. In \cite{6}, a visual attention model used to highlight some essential regions of frames for better prediction. Although the steering input prediction in an end-to-end manner is a well-studied problem in the literature, the steering input alone is not sufficient for fully autonomous driving. In \cite{7}, a CNN-LSTM network is proposed to predict the speed and steering inputs synchronously.

Pure end-to-end learning policies are limited to the demonstrated performance, and although the training and validation loss on the data collected from the expert might be low, errors accumulated from the execution of the learned driving policy might lead to poor performance in the long run. This performance loss is partly because the learned driving policy is likely to observe states that do not belong to the distribution of the original expert demonstration data. DAgger \cite{8} algorithm addresses this issue by iteratively collecting training data from both expert and trained policies. The main idea behind DAgger is to actively obtain more samples from the expert to improve the learned policy. Even though DAgger achieves better driving performance, it might end up obtaining a lot of samples from the expert, which can be time and resource-consuming in many real-life scenarios. SafeDAgger \cite{9} algorithm, an extension of DAgger, attempts to minimize the number of calls to the expert by predicting...
the unsafe trajectories of the learned driving policy and only calls the expert on such cases. Another extension of DAgger, EnsembleDAgger [10], predicts the variance of the decisions by using multiple models and takes it as additional safety criteria like SafeDAgger.

In this paper, we propose a novel framework which is sample-efficient compared to the SafeDAgger algorithm (state-of-the-art data aggregation method), named Selective SafeDAgger. The proposed algorithm classifies the trajectories executed by the learned policy to safe and multiple classes of unsafe segments. After the prediction, the model focuses on obtaining the expert policy samples primarily from the identified unsafe segment classes. Our main contribution is an imitation learning algorithm that collects the most promising samples from the expert policy, which enables outperforming the SafeDAgger method while limited to the same number of calls to the expert.

This paper is organized as follows. Section II provides the details of the methodology. The experimental setup is provided in section III, followed by a discussion about results in section IV and conclusions in section V.

II. METHODOLOGY

In this section, driving policies, the architecture of the network, and devised algorithm are explained in detail.

A. Driving Policies

We begin with giving definitions of the used terms to explain driving policies in detail.

A set of states $S$ for the car in this paper is an environment model, and $s \in S$ is one of the states for the car in that environment. Observation of the state $s$ is defined as $\phi(s) \in \Phi(S)$ where $\Phi(S)$ is the observation set for all states. $a(s) \in A(S)$ will be driving action at observation $\phi(s)$ where $A(S)$ is the set of all possible actions.

A set of driving policies $\Pi$ is defined as in Eq. (1).

$$\Pi : \Phi(S) \rightarrow A(S)$$

where $\Pi$ is a mapping from state observations $\phi(s)$ to driving actions $a(s)$ such as steering, throttle, brake, etc.

Two distinct driving policies are defined throughout the paper. The first one is an expert policy $\pi^* \in \Pi$ that drives the car with a reasonable performance that we want to imitate. An expert policy in an autonomous driving scenario is usually chosen as actions of a human driver. Variants of DAgger algorithms, however, have mislabeling problem in case of the human driver, since drivers do not have feedback feelings from their actions and they can give incorrect reactions to the given states. To overcome the mislabeling problem, we have used a rule-based controller which contains speed and steering controllers, as an expert policy in this paper. The second one is a primary policy $\pi_0 \in \Pi$ that is trained to drive a car. This policy is a sub-optimal policy according to the expert policy since it is trained on a subset of observation set $\Phi(S)$.

Training a primary policy to mimic an expert policy is called imitation learning or learning by demonstration. One of the most common methods for imitation learning is based on supervised learning techniques. The loss function for the supervised learning is defined as in Eq. (2) [9].

$$l_{\text{supervised}}(\pi, \pi^*, D_0) = \frac{1}{N} \sum_{i=1}^{N} ||\pi(\phi(s_i)) - \pi^*(\phi(s_i))||^2$$

where $l_{\text{supervised}}$ refers to $l^2$-Norm between trained and expert policy actions.

A primary policy as in Eq. (3) is defined as a policy that minimizes the loss function as follows.

$$\pi_0 = \arg \min_{\pi} l_{\text{supervised}}(\pi, \pi^*, D_0)$$

Minimization of the loss function can be challenging since it is known that the relation between image frames and driving actions is highly nonlinear. So, we have used a deep neural network architecture to find an optimal solution for the primary policy.

B. Network Architecture

The earlier works in end-to-end learning for self-driving cars focus on computing only the steering angle from a single image or a sequence of images. The longitudinal control component is required to reach a higher level of autonomy in the end-to-end framework. In this work, we utilize the multi-task model proposed in [7] as our baseline, which is capable of generating both longitudinal and lateral control inputs for the car. Besides, we utilize a speed controller rather than the classical throttle/brake commands for the longitudinal control. The steering action is predicted from the raw image inputs taken from the cameras located in front of the vehicle through convolution layers, and the speed is predicted from a sequence of speed profiles through a Long-Short Term Memory (LSTM) layer. There exists a single-direction coupling between the longitudinal controller (speed controller) and the lateral steering actions. In particular, the speed of the vehicle has a significant impact on the prediction model, since entering a turn with low speed represents different dynamics for the lateral controller when compared to a high-speed maneuver. Moreover, the straight trajectory dominates the whole other trajectory types (e.g., turn left, turn right); therefore, the trained network will be biased toward the straight path. To recover from this issue, we decided to define various trajectory types including all major maneuvers such as straight, turn left, turn right and low and high-speed scenarios, by which the devised model will learn the other less-occurring maneuvers.

The model architecture is shown in Fig. 1. It takes the current observation and the past speed profile and returns steering action, speed action, and the class of the trajectory segment. The multi-task network predicts the steering angle through a visual encoder using a stack of convolution and fully-connected layers. In the first two convolution layers (Conv1 and Conv2), large kernel size is adopted to better capture the environment features, which is suitable for the front-view camera. Inputs and kernels of the each convolution layer is denoted by "#channels@input height × input width"
define only one class for the safe trajectories.

The multi-class classifier takes the partial observation of the state $\phi(s)$ which contains the visual perception and the past speed profile and returns a label indicating in which part of the trajectory the policy will likely to deviate from the expert policy $\pi^*$. The labels for training the model was generated through one-hot-encoding method, defined by sequential decisions; first, it was decided whether the policy is safe by measuring its distance from the expert policy through $l^2$-Norm metric using Eq. (5).

$$c_s(\pi, \phi(s)) = \begin{cases} 0, & \|\pi(\phi(s)) - \pi^*(\phi(s))\| > \tau_{safe} \\ 1, & \text{otherwise} \end{cases}$$

where $\tau_{safe}$ is a predefined threshold and can be chosen arbitrarily. Furthermore, to distinguish between low-speed and high-speed turn trajectories, steering threshold $\tau_{turn}$, speed thresholds for turn maneuver $\tau_{speed,turn}$ and straight trajectory $\tau_{speed,straight}$ are defined heuristically based on the response of the car dynamics in these trajectories. The threshold values for this work is depicted in Table I.

| Parameter               | Threshold value |
|-------------------------|-----------------|
| $\tau_{safe}$           | 0.5             |
| $\tau_{turn}$           | $0.25^\circ$    |
| $\tau_{speed,turn}$     | 10 m/s          |
| $\tau_{speed,straight}$ | 13.75 m/s       |

TABLE I: Threshold Values in Labeling Process

where $\tau_{safe}$ as 0.5 yields $0.25^\circ$ for the steering angle and 1 m/s for the speed difference between the network prediction and expert policy output.

C. Selective SafeDAgger Algorithm

Algorithm 1 describes the proposed method in detail, which takes the expert policy $\pi^*$ as an input and gives $\pi_i$ as an output. The primary dataset $D_0$ is collected by using $\pi^*$, which is then utilized in training a primary policy $\pi_0$ by a supervised learning method. Having the $\pi_0$ at hand, $c_i$, the unsafe classes of $D_0$ for the trained policy $\pi_i$ are determined. An observation $\phi_k$ taken from environment $\phi(s)$ is evaluated by $\pi_i$ to find its class $c_{\phi_k}$. If $c_{\phi_k}$ is an element of $c_i$, $\pi^*$ takes over the control of the car and $\phi_k$ is appended to $D'$. Otherwise, $\pi_i$ continues to command the car until it encounters an unsafe class. As depicted in lines 9-10, the algorithm continues to append data to $D'$ for $T$ number of iterations. The appended dataset $D'$ is aggregated into $D_{i-1}$ to create $D_i$ and $\pi_{i+1}$ is trained on $D_i$. This loop is repeated for $N$ times, as shown in lines 10-14. In the end, the algorithm returns the best $\pi_i$ over the validation set.

III. EXPERIMENTS

A. System Setup

1) Simulator: AirSim used in this work is an Unreal Engine Plugin based simulator for drones and cars established by Microsoft to create a platform for AI studies to develop and test new deep learning, computer vision and reinforcement learning algorithms for autonomous vehicles.
Algorithm 1: Selective SafeDAgger: Blue fonts

distinguishes the difference between Selective SafeDAgger and SafeDAgger

```
1 Collect \( D_0 \) using \( \pi^* \nabla 
2 \pi_0 = \arg\min_{\pi} l_{\text{supervised}}(\pi, \pi^*, D_0) 
3 \text{for } i = 1:N \text{ do} 
4 \quad c_i \leftarrow \text{Define unsafe classes over } D_0 
5 \quad D' \leftarrow \lfloor \rfloor 
6 \quad \text{while } k \leq T \text{ do} 
7 \quad \quad \phi_k \leftarrow \phi(s) 
8 \quad \quad c_{\phi_k} \leftarrow \text{classifier output of } \pi_i(\phi_k) 
9 \quad \quad \text{if } c_{\phi_k} \in c_i \text{ then} 
10 \quad \quad \quad \text{use } \pi^*(\phi_k) 
11 \quad \quad \quad D' \leftarrow [\phi_k] 
12 \quad \quad \quad k = k + 1 
13 \quad \quad \text{else} 
14 \quad \quad \quad \text{use } \pi_i(\phi_k) 
15 \quad \end{while} 
16 \quad D_i = D_{i-1} \cup D' 
17 \pi_{i+1} = \arg\min_{\pi} l_{\text{supervised}}(\pi, \pi^*, D_i) 
19 \text{return best } \pi_i \text{ over validation set} 
```

with photo-realistic graphs in simulations [11]. It has built-in APIs for interfacing with Python coding language. Furthermore, the engine editor creates custom environments or scenarios.

The road track for the training process of the algorithm is devised in a way to capture all defined scenarios in this work. The geometry of the custom created training track is shown in Fig. 2, in which all the trajectory classes are illustrated.

Representative power of the training set can be increased by collecting data from unseen observations. With that reason, two additional cameras were added to the front-facing camera with an angle of \( \gamma \) to imitate turning-like maneuvers [4]. AirSim APIs provide ground truth labels for the front-facing camera frames, but ground truth labels for the left and right cameras should be adjusted with a threshold as in Eq. (6).

\[
\begin{bmatrix}
L_l \\
L_r
\end{bmatrix} = 
\begin{bmatrix}
L_{c_{\text{steering}}} + \gamma & L_{c_{\text{speed}}} - p_{\text{speed}} \\
L_{c_{\text{steering}}} - \gamma & L_{c_{\text{speed}}} - p_{\text{speed}}
\end{bmatrix}
\] (6)

where \( L_l, L_r, L_{c_{\text{steering}}} \) and \( L_{c_{\text{speed}}} \) refer to the ground truth for the left and right cameras, center camera steering and speed actions respectively. In the turning case, the ground truth speed of the vehicle is adjusted by a parameter \( p_{\text{speed}} \) which is chosen as 4 m/s heuristically.

2) Data Preprocessing: A couple of techniques were utilized in the preprocessing phase. The input raw image was down-sampled to the size of \( 144 \times 256 \times 3 \) (RGB) and a Region of Interest (ROI) defined with the size of \( 59 \times 255 \) to cover almost the entire road and ignore the features above the horizon, which reduces the computational cost. Moreover, to improve the convergence rate and robustness of the neural network model, the processed image was normalized to the range [0,1] and augmented by randomly changing the brightness of the image with a scale of 0.4. The normalization was done by dividing all image pixels by 255.

3) Expert Policy: To automatize the data collection part of the algorithm, a rule-based expert policy is defined as shown in Fig. [3]

![Fig. 2: Train set track](image)

![Fig. 3: Expert policy](image)

For the steering action, \( T_1 \) is a tangent line to the road spline at the position of the car and \( P_1 \) is a point on road spline with \( l_{\text{ref}} \) distance along spline from that positions. Tangent line at \( P_1 \) according to road spline is \( T_2 \). The angle between \( T_1 \) and \( T_2 \) which is \( \alpha \) will be expert steering action as depicted in Eq. (7).

\[ a_{\text{steering}} = \alpha = \arccos \left( \frac{T_1 \cdot T_2}{||T_1|| \cdot ||T_2||} \right) \] (7)

For the speed action, \( P_2 \) is a point on the road spline with a distance \( l_{P_2} \) from the position of the car along the road spline as depicted in Eq. (8).

\[ l_{P_2} = l_{\text{ref}} V_{\text{current}} k_{\text{steering}} \] (8)

where \( V_{\text{current}} \) is current speed and \( k_{\text{steering}} \) is a fine tuned constant. Tangent line at \( P_2 \) according to the road spline is \( T_3 \).

Expert speed action is defined by Eq. (9).

\[ a_{\text{speed}} = V_{\text{cruise}} - \beta k_{\text{speed}} \] (9)

where \( V_{\text{cruise}} \) is a pre-defined cruise speed, \( k_{\text{speed}} \) is a fine tuned gain and \( \beta \) is an angle between \( T_1 \) and \( T_3 \).
For our implementation, the parameters are chosen as $l_{ref} = 1$ m, $k_{steering} = 5$, $V_{cruise} = 13.8$ m/s and $k_{speed} = 10$.

B. Training

For the training of the primary policy $\pi_0$, dataset $D_0$, which contains 2800 image data were collected by using expert policy $\pi^\ast$. Nesterov Adam Optimizer (Nadam) was used as an optimizer for the training of the network with the initial learning rate of $10^{-5}$ and moment of 0.99. The Training continued for ten epochs with the batch size of 32.

Trained primary policy $\pi_0$ is tested on the pre-collected dataset to classify trajectories and calculate the $l^2$-Norm of each sample in the dataset. The weakness of the network over trajectory segments is determined by a coefficient of weakness, which is defined as in Eq. (10).

$$c_i = \frac{N_{L2,i}}{N_i} \times \mu_{L2,i}$$

where $\mu$, $\sigma$ are mean and standard deviations for the $l^2$-Norm of class $i$. $N_{L2,i}$ is the total number of samples in class $i$ that $l^2$-Norm of samples fall in the region of one $\sigma$ away from the mean $\mu$. $N_i$ is the total number of samples in class $i$.

Once the weakness coefficients are calculated, trajectory classes are sorted according to their weakness coefficients, and the two of the most dominant unsafe classes will be chosen for data aggregation as shown in Table I. Additionally, the classes with the mean $l^2$-Norm lower than 1, will be selected as allowable classes.

As depicted in Table I, the weakness coefficients for the class LS and HS are quite low and never chosen as weak classes. The initial dataset for the training is biased toward LS, and HS classes and $l^2$-Norms in those classes are low, which lead to low weakness coefficients. Moreover, training track does not have many samples from class LR so that weakness coefficients for the class LR is also low.

After determination of the weak and allowable classes, the data aggregation phase begins. In this phase, policy $\pi_i$ drives the car to collect 10 batches of data in dominant classes. If policy faces with dominant classes, the expert policy takes control of the vehicle and samples are taken in that time labeled and reserved for aggregation. If policy $\pi_i$ faces with allowable classes which are unsafe, it continues to drive the car. For all the other unsafe classes, the expert policy takes control of the vehicle with the query limit of 10 batch-size. When the number of query reaches the limit, data aggregation freezes and training starts with the new aggregated dataset $D_i$.

After the training, $\pi_i$ becomes $\pi_i+1$, and determination of dominant weak classes on the pre-collected data is repeated for collecting relevant data. This process will be repeated for 10 iterations. As shown in Fig. 4, in the dataset aggregation iteration number 1, a significant fraction of dataset is unsafe, and as it proceeds to recover from the most problematic cases, the model error converges. The progress of this process can be seen from iteration number 1 to 10.

IV. RESULTS

In Fig. 5a we present the performance of the Selective SafeDAgger with using metric of $l^2$-Norm in each class during the training process. For the first iteration, $HR$ and $HL$ are chosen as weak classes and data for new dataset comes from those classes by querying expert policy. It is seen that in the second iteration, $l^2$-Norms drops for all classes by using aggregated dataset. Notice that the performance of the policy for the other classes is also increased without querying expert policy for those classes which are not the case for the SafeDAgger. Sequential decision making is the main idea behind this behavior. In SafeDAgger, when policy shifts from nominal conditions, the expert policy is called, and the new dataset is collected until the safety criterion is met, which leads to an unnecessary query of the expert policy. On the other hand, Selective SafeDAgger tries to solve the problem.

![Fig. 4: Convergence rate of the proposed model; It shows the improvement of the model as the number of dataset aggregation iterations increases.](image-url)

![Fig. 5a](image-url)

| # Iter. | LL  | HL  | LR  | HR  | LS  | HS  |
|--------|-----|-----|-----|-----|-----|-----|
| 1      | 0.004 | 0.321 | 0.019 | 0.694 | 0.002 | 0.010 |
| 2      | 0.505 | 0.122 | 0.037 | 0.278 | 0.001 | 0.023 |
| 3      | 0.635 | 0.264 | 0.028 | 0.607 | 0.001 | 0.062 |
| 4      | 0.751 | 0.515 | 0.046 | 0.646 | 0.001 | 0.010 |
| 5      | 0.018 | 0.678 | 0.034 | 0.755 | 0.001 | 0.010 |
| 6      | 0.009 | 0.752 | 0.039 | 0.849 | 0.000 | 0.006 |
| 7      | 0.717 | 0.790 | 0.038 | 0.780 | 0.001 | 0.004 |
| 8      | 0.028 | 0.787 | 0.017 | 0.794 | 0.001 | 0.006 |
| 9      | 0.670 | 0.634 | 0.011 | 0.713 | 0.001 | 0.005 |
| 10     | 0.012 | 0.768 | 0.020 | 0.809 | 0.001 | 0.003 |

TABLE II: Coefficient of weakness for each class
The selectivity of the proposed algorithm will define the unsafe cases that dominate all other classes, which results in faster convergence of the model error compared to different dataset aggregation methods.

TABLE IV: Mean $l^2$-Norm on Unseen Test Track

|                  | Selective SafeDAgger | SafeDAgger |
|------------------|----------------------|------------|
| 1. Test Track    | 0.4794               | 0.5518     |
| 2. Test Track    | 0.3295               | 0.4986     |
| 3. Test Track    | 0.3254               | 0.3632     |

V. CONCLUSIONS

In this work, we implemented a Selective SafeDAgger algorithm which is sample-efficient in the selection of dataset aggregation. The proposed algorithm evaluates the performance of the trained policy and determines the weakness of the policy over different trajectory classes and recovers the policy from those specific trajectory classes. Our method outperforms the SafeDAgger algorithms in term of sample-efficiency and convergence rate. Next, we aim to cluster the trajectories with unsupervised neural network techniques to have a better realization of the road trajectories.

ACKNOWLEDGMENT

This work is supported by Scientific and Technological Research Council of Turkey (Turkish: TÜBİTAK) under the grant agreement TEYDEB 1515 / 5169901.

REFERENCES

[1] J. Leonard, et al., “A Perception-Driven Autonomous Urban Vehicle.” Journal of Field Robotics. (10):727-748.
[2] Z. Chen and X. Huang, “End-to-end learning for lane keeping of self-driving cars,” 2017 IEEE Intelligent Vehicles Symposium (IV), Los Angeles, CA, 2017, pp. 1856-1860.
[3] D. Pomerleau, “ALVINN: An Autonomous Land Vehicle in a Neural Network.” NIPS (1988).
[4] M. Bojarski, et al., “End to end learning for self-driving cars”, CoRR, vol. abs/1604.07316, 2016.
[5] H. Xu, Y. Gao, F. Yu, and T. Darrell, “End-to-End Learning of Driving Models from Large-Scale Video Datasets,” 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, 2017, pp. 3530-3538.
[6] J. Kim and J. Canny, “Interpretable Learning for Self-Driving Cars by Visualizing Causal Attention,” 2017 IEEE International Conference on Computer Vision (ICCV), Venice, 2017, pp. 2961-2969.
[7] Z. Yang, Y. Zhang, J. Yu, Junjie Cai, and Jiebo Luo. End-to-end multi-modal multi-task vehicle control for self-driving cars with visual perception. arXiv preprint arXiv:1801.06734, 2018.
[8] S. Ross, G. Gordon, and A. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. Journal of Machine Learning Research, 15:627–635, 2011.
[9] J. Zhang and K. Cho. Query-efficient imitation learning for end-to-end simulated driving. In AAAI, 2017.
[10] K. Menda, et al., “EnsembleDAgger: A Bayesian Approach to Safe Imitation Learning.” CoRR abs/1807.08364 (2018): n. pag.
[11] S. Shah, D. Dey, C. Lovett, A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles“, Field and Service Robotics, 2017.