DEGREE PROJECT IN THE FIELD OF TECHNOLOGY
INFORMATION AND COMMUNICATION TECHNOLOGY
AND THE MAIN FIELD OF STUDY
COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2020

Imitation Learning using Reward-Guided DAgger

NORA AL-NAAMI

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Imitation Learning using Reward-Guided DAgger

NORA AL-NAAMI

Master in Machine Learning
Date: July 21, 2020
Supervisor: Pawel Herman, Mika Cohen, Farzad Kamrani
Examiner: Mårten Björkman
School of Electrical Engineering and Computer Science
Host company: Swedish Defense Research Agency

Abstract

End-to-end autonomous driving can be approached by finding a policy function that maps observation (e.g. driving view of the road) to driving action. This is done by imitating an expert driver. This approach can be conducted by supervised learning, where the policy function is tuned to minimize the difference between the ground truth and predicted driver actions. However, using this method leads to poor performance since the policy function is trained only on the states reached by the expert. An algorithm in imitation learning that addresses this problem is Dataset Aggregation (DAgger). The main idea of DAgger is to train a policy iteratively with data collected from expert and the policy function itself. This requires identifying a rule for the interaction of the expert and policy function. The current DAgger variants require querying the expert for a long time and do not explore the state space with both safety and efficiency. In this thesis, we present an extension to DAgger, which attempts to present a decision rule with the safety in state space as a probability measure in order to minimize expert queries and guide the exploration in training. We evaluate the proposed algorithm called Reward-Guided DAgger (RG-DAgger) with other known algorithms Behavior Cloning, Vanilla DAgger and Ensemble DAgger. The different algorithms are evaluated in the context of self-driving cars on a track with twenty minutes of driving in a Virtual Battle Space Simulator 3 (VBS3). The training of the algorithms is carried out on ten randomly generated tracks using a human as an expert. The result shows trends of the expert time during training and number of falls during tests for one trial. The trends seen in the performance of expert time show Behavior Cloning performed worst with the highest expert time while RG-DAgger had the lowest total expert time overall DAgger iterations. The performance of number of falls shows the highest average number of falls in Ensemble DAgger. In conclusion Ensemble DAgger and Vanilla DAgger are more robust algorithms to learning compared to RG-DAgger, while RG-DAgger samples labels from expert only when expert controls the car making it more friendly for a human expert.

Sammanfattning

Autonom körning från end-to-end kan uppnås genom att hitta en policy funktion som kartlägger observation, exempel på en sådan kartläggning är en kör vy till en körnings rörelse. Detta görs genom att imitera en expertförare. Strategin kan göras genom övervakat lärande där policyfunktionen är avstämd för att minimera skillnaden mellan truth labels och förutsagda förarens åtgärder, att använda denna metod leder emellertid till att modellen avviker från expertens beteendespår. Detta beror på att policyfunktionen endast tränas på de tillstånd som experten når. En algoritm i imiteringsinlärning som hanterar detta problem är Dataset Aggregation (DAgger), huvudtanken med DAgger är att utbilda en policy tränad iterativt med data som samlas in från en expertförare och själva policyfunktionen. Detta kräver att man identifierar regel för expertförarens interaktion och policyfunktion. Nuvarande DAgger algoritmer kräver förfrågningar från expertföraren vilket leder till mycket experttid och på grund av algoritmens begränsade utforskningar så finns det risk att algoritmen är varken säker eller effektiv. I denna avhandling presenterar vi en förlängning till DAgger, som försöker presentera en beslutsregel med säkerheten i tillståndets rymd som en sannolikhetsåtgärd för att minimera expertfrågor och vägleda utforskningen i träning. Vi utvärderar den föreslagna algoritmen som kallas för Reward-Guided DAgger (RG-DAgger) med andra kända algoritmer Behavior Cloning, Vanilla DAgger och Ensemble DAgger. De olika algoritmerna utvärderas i samband med självkörande bilar på en bana med tjugo minuters körning i en Virtual Battle Space Simulator 3 (VBS3). Träningen av algoritmerna utförs på tio slumpmässigt genererade spår med en människa som expert. Resultatet visar trender för experttid under träning och antal fall under tester för ett försök. Trenderna i prestandan för experttid visar Beteende kloning fungerade sämst med den högsta experttid medan RG-DAgger hade den lägsta totala experttid för alla DAgger iterationer. Prestandan för antal fall visar det högsta genomsnittliga antalet fall i Ensemble DAgger. Sammanfattningsvis är Ensemble DAgger och Vanilla DAgger mer robusta algoritmer till lärande jämfört med RG-DAgger, medan RG-DAgger samplar etiketter från expert endast när expert kontrollerar bilen vilket gör RG-DAgger mer vänligt för en mänsklig expert.

Contents

Chapter 1

Introduction

One of the ten major causes of death in the world is transportation accidents[1]. The traffic accidents can be decreased by using autonomous cars which have high reliability and faster reaction time compared to humans. There are more benefits to autonomous cars such as reduced traffic congestion, better organized parking lots and less traffic police [1].

This makes automated driving a very popular research area, where many algorithms and methods have been developed to improve it. Many automobile companies, such as Tesla, Volvo [2], as well as Deep-learning companies, NVIDIA [3] are investing considerable amount of resources in this research field. Most approaches to automated driving relies on deep learning algorithms to teach a network how to drive in different environments.

The research of autonomous driving started in 1926 where a radio controlled car 'Linriccan Wonder' was developed [4]. In 1980 Mercedes-Benz robotic Van [5] was developed using vision based system using LIDAR, radar and GPS. This system built the foundation of the current technologies found in modern cars such as lane parking, steer assist etc. Another approach to autonomous driving is the network named ALVINN by Pomerleau [6]. The network ALVINN was built with a single hidden layer which trains on a front-facing camera image, sensor map and steering angle. Recent research of autonomous driving use similar approaches but with deeper networks like convolutional neural networks [3, 7].

Previous attempts of implementing autonomous driving have relied on supervised learning. In those attempts an expert drives collecting training data of image-action pairs. These data is used to train a neural network with a su-

[1] Automated vehicles are more than self-driving cars, Who.int, https://www.who.int/news-room/fact-sheets/detail/the-top-10-causes-of-death, 2020-06-14

pervised learning paradigm. This approach of training a model to imitate an expert is called *Behavior Cloning* [8, 9]. A problem of using Behavior Cloning for autonomous driving is that it violates the assumption of independent and identically distributed (i.i.d.) data made by a purely supervised learning approach [10]. This is problematic since autonomous driving is a sequence prediction problem where a system over time needs to predict a sequence of actions given a sequence of observations [8]. Therefore, using Behavior Cloning is sub-optimal and might lead to a problem called *compounding error* where a mistake made by the system can accumulate due to unseen or different observations than the ones learned during training [11].

Imitation Learning *Imitation learning* [10, 12] is a machine learning method that tries to address the problems observed in Behavior Cloning by training a network on expert demonstrations and solving the sequential decision problem. It has been used in many robot control applications where the sequential decision problem occurs [13–15]. The first attempt to solve Imitation Learning problem was using Behavior Cloning, which suffered from the issue of compounding error. A recent approach proposed to solve this is an iterative algorithm called *DAgger* [10]. Initially, in the DAgger algorithm a data set is collected entirely by the expert and a policy is trained on the collected expert-trajectory data set. In each DAgger iteration a new data set is collected under the current policy with the interaction of an expert and a new policy is trained on the aggregated data sets [10]. Further explanation of the DAgger algorithm can be found in section 2.5.

1.1 Problem statement

The performance of the DAgger algorithm depends heavily on quality of the data collected in each DAgger iteration to train a policy. This quality, in turn, is based on the interaction of the trained policy and the expert [8]. There are several DAgger variants that focus on different performance measures [16–19]. Those DAgger variants differ mostly on their interaction between the expert and the trained policy, which is called a decision rule. Existing DAgger variants require a large amount of expert queries or are not sufficiently good in finding the states where the policy and expert need to switch control. Those flaws in the existing DAgger variants lead to a collection of low quality data. Therefore, a DAgger variant with minimal expert queries and good exploration of the state space with a balanced interaction between the expert and a policy in each DAgger iteration is necessary for good performance. This problem makes up the following research question: Can a model-based DAgger variant

which uses signals of unsafe areas provided by a simulator outperform the existing DAgger methods.

1.2 Scope and objectives

This thesis focuses on testing different DAgger variants: the first DAgger variant Vanilla, the latest DAgger variant Ensemble and the proposed DAgger named as Reward-Guided DAgger. The study of other DAgger variants such as Safe DAgger [17], Human-gated DAgger [19] and Dropout DAgger [16] are outside of the scope. Additionally, optimization of hyperparameters of each DAgger variant is outside of the scope. Another limitation in the study is that the data presented in this work is for a single trial in the experiment. The focus of the thesis is on comparing the performance of the different DAgger variants on the application of autonomous driving. The architecture of the network used is not the focus of the thesis. The weather and illumination of the roads where the experiment is carried out is outside of the scope. The objectives of this thesis is to find a DAgger algorithm for imitation learning that is optimal in performance and expert time during training compared to the state-of-the-art of DAgger variants. Another objective is to design the algorithm to be both suitable for both AI expert and a human expert during the data collection while training. The last objective is to collect quality data during training.

1.3 Thesis outline

The report is orgainzed as follows:

Chapter 2 provides the background and the underlying principles of the work.

Chapter 3 presents the methodology of this work.

Chapter 4 presents the results of the algorithms applied in this work.

Chapter 5 analyzes and discusses the findings.

Chapter 6 concludes the work and outlines future work.

Chapter 2

Background

2.1 Machine Learning

There are three major machine learning methods: supervised learning, unsupervised learning and reinforcement learning.

2.1.1 Supervised Learning

Supervised learning is one of the learning methods used in machine learning. Training with supervised learning means that the input x is known and the labels for the input are also known. Supervised learning is used for classifying or estimating data by building a classifier or estimator. In supervised learning the algorithms will learn by example where the data set collected has true labeled values for each data point. The idea of supervised learning is to learn how to generalize [20]. In order to generalize, there need to be patterns in the data. An example of generalization is classification which is done by separating an input of n-data points into m classes by finding the pattern in the data to predict new examples correctly using a discriminant function [21]. Another example of generalization is regression which is a function approximation or interpolation, it works by predicting the next value in a sequence by fitting a mathematical function to the data points given as the input.

2.2 Artificial Neural Networks

2.2.1 Artificial neural network

There are different ways of solving machine learning problems, one of them is by using aritificial neural networks. The architecture of a deep neural network consists of three main parts: input layer, hidden layers and an output layer [22]. Each layer might contain one or several neurons. The main property of the network is for each neuron to receive input from the previous layer and sends its output to the next neighbouring layer. The input layer is where the data set is fed in the network. The hidden layer consists of a chosen amount of neurons that are independent of each other and are fully connected to the previous layer. A network with two hidden layers and four neurons in each hidden layer can be seen in Fig. 2.1. The signals from a layer are forwarded to the neurons of the next layer via connected weights w [23]. Each neuron in every layer calculates the weighted sum of the incoming input by multiplying the weight with the input and adding bias as in equation 2.1. The weighted sum z is applied to an activation function $f(z)$ which is the output y of the neuron. The non-linearity of the network is determined by the activation function f.

$$\hat{y} = f(w^T x + b) \tag{2.1}$$

Accepting an input x and producing an output $\hat{y}$, information will be propagated forward and this is called forward propagation. A cost function $J(\theta)$ is produced during forward propagation [24] where θ are the networks' parameters. The network learns by minimizing the cost function $J(\theta)$ which is a measure of the errors produced by the network. In order to reduce the errors, gradients of the networks parameters are computed [21]. The networks parameters are then updated based on a gradient optimization method until a minimum in the cost function $J(\theta)$ is found [24]. This is called back-propagation.

2.3 Convolutional Neural Network

Convolutional Neural networks are a neural network used for processing data which has a know grid-like topology [24]. There are three different layers used to build a convolutional neural network: convolutional layers, pooling layers, and fully-connected layers. The convolutional layer is the central part of a convolutional neural network since most of the heavy computation is carried out there. In the convolusional layer a kernel K is multiplied element-wise with

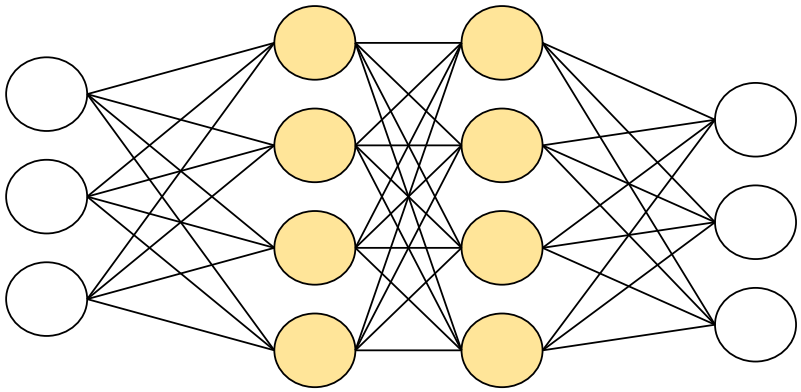

Figure 2.1: A deep neural network with input layer, two hidden layers and output layer.

an input feature map I as shown in Fig. 2.2. The result of this operation is an output which is referred to as a pre-activation map S as seen in equation 2.2.

$$S = K * I \tag{2.2}$$

There are three different parameters that determine the size of the activation map: the depth, stride, and zero-padding. The depth is the number of kernels K used to convolve the input feature map I and can be chosen depending on what needs to be learnt about the input feature map [25].

Stride determines the steps taken when a kernel K is convolved across the input feature map I. If the stride is 1 and the input feature map is an image then the filter should move one pixel at a time as shown in Fig. 2.2. However, when the stride is two then the kernel slides two pixels across the input volume I. When the stride is bigger than one it results in a smaller activation map.

The zero-padding hyperparameter controls the size of the activation map by padding zeros at the borders of the input feature map as shown in Fig. 2.3. It is used to fit the input feature map when the kernel convolves across it.

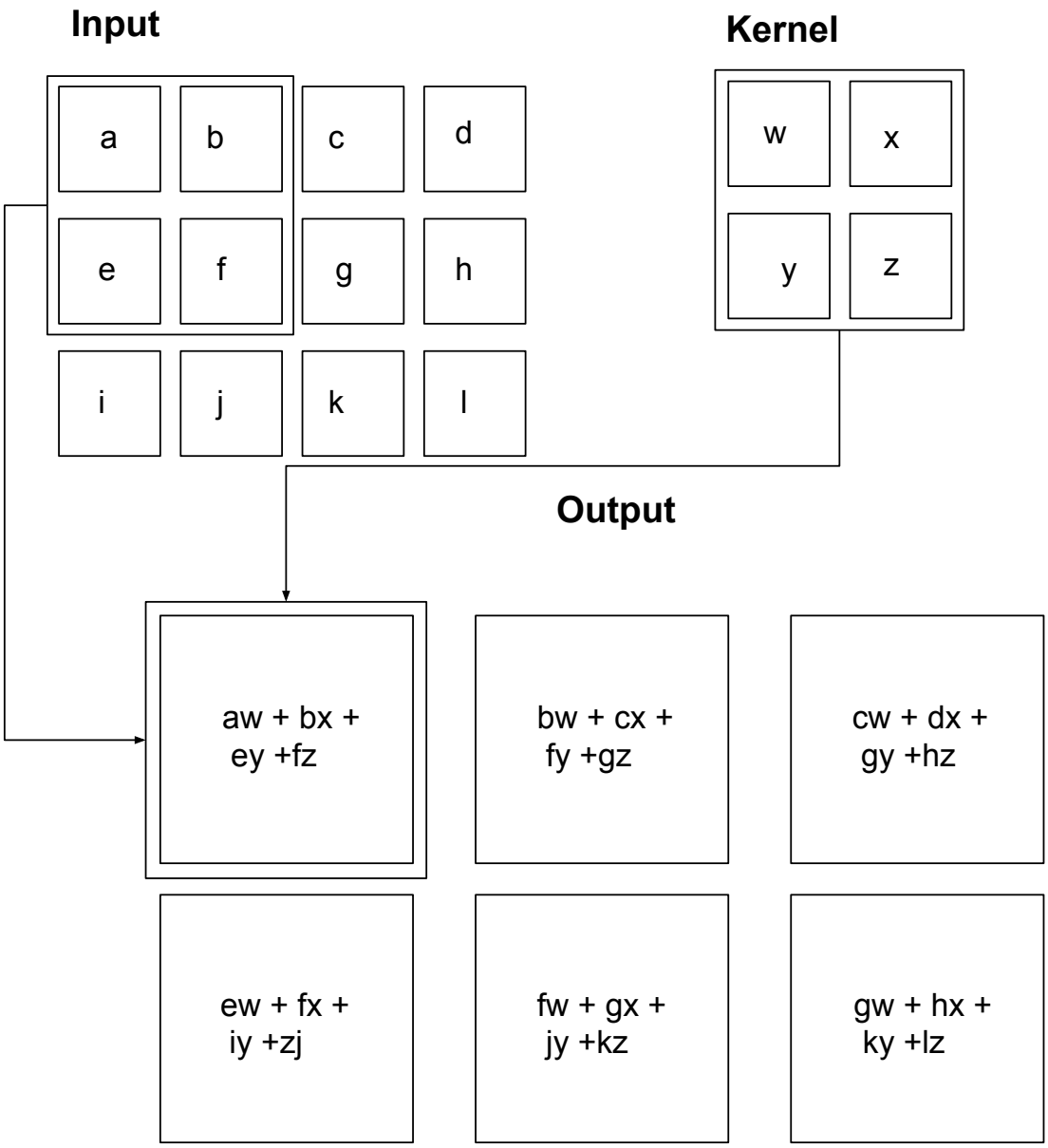

Figure 2.2: A 2-D convolution example with stride 1 where a kernel convolve across the input feature map.

2.3.1 Pooling layer

In a convolutional neural network architecture a pooling layer is usually positioned between convolutional layers. It is used to decrease the size of the

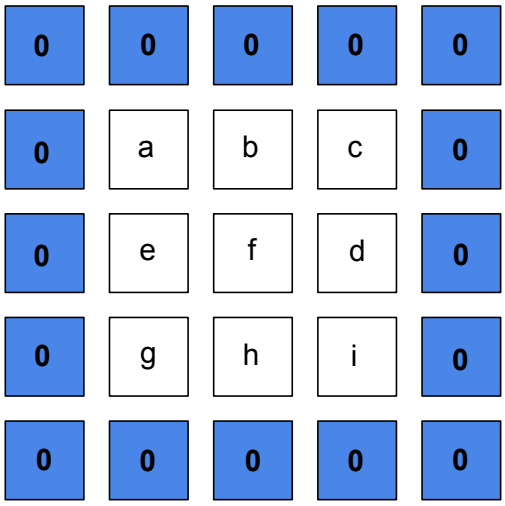

Figure 2.3: An input feature map with zero-padding of size one in the borders marked in blue color.

output volume and control overfitting by reducing the number of parameters [25] which result in less computation time. There are different types of pooling layers such as MAX pooling and average pooling. A MAX pooling layers slide through the input feature map and choose the max value of the input feature map window it covers. The average pooling takes the average of the input feature map volume window covered by the average pooling layer. This decreases the width and height of the output volume but does not affect the depth.

2.3.2 Fully-connected layer

The fully-connected layer used in Convolutional neural network has the same architecture as in regular neural networks. In a fully-connected layer every neuron of an input is connected to every neuron in the next layer as shown in Fig. 2.4.

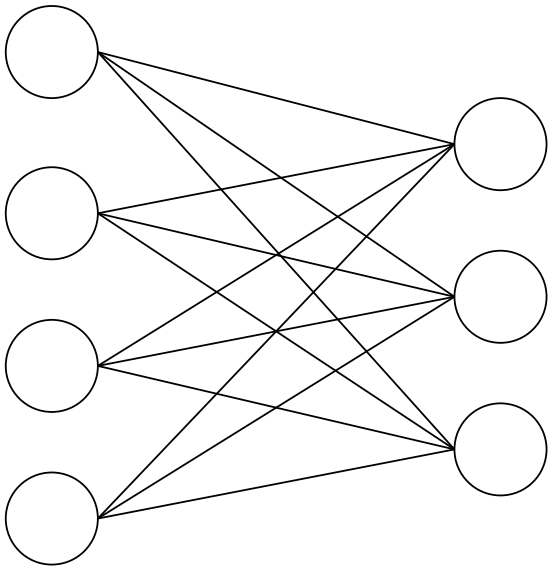

Figure 2.4: A fully-connected layer with four input nodes and three output nodes.

2.4 Imitation Learning

Many real-life applications require making a sequence of decisions while the environment is changing rather than a single prediction. These kind of problems are referred to as sequential decision-making problems. An example of such a problem is automated driving were driving a car requires a sequence of decisions with the environment changing partly based on the actions taken by the car. Automated driving can be modeled as a system that at time step t gets an input o_t as an image of the driving field and generates an action a_t. A loss l_t is defined as a metric for assessing how poorly the system performs for each time step t. This process repeats for each time step and the goal is to find a mapping between the images of the driving field and actions taken by the car. This mapping is called a policy: a good policy $\pi(o_t)$ seek to minimize the total loss [26].

In imitation learning, it is assumed that an expert exists that knows how to drive very well. The idea of imitation learning is to let a novice learn how to

imitate the expert from observations at a specific time [26]. Imitiation learning is related to supervised learning as we learn a policy π from a data set D, particularly to structured prediction [10, 15, 27], where the task is to learn a mapping from input x to a complex output y. The connection of supervised learning and imitation learning is the same to the reduction of structured prediction to sequential decision [10] and reduction of imitation learning to structured prediction [15].

Behavior Cloning is a Supervised imitation learning algorithm that suffers from compounding error. This means that when a novice trained by Behavior Cloning algorithm veers from an expert trajectory, it would be difficult for it to recover from a failure. This is an issue since when a Behavior Cloning trained policy predicts incorrectly at time step t, which will cause it to make mistakes continuously due to unseen states during training [26].

Another idea to solve this problem is to make π learn from its own mistakes meaning training π on the mistakes it encounters. This is done by testing a policy π_i and documenting the states it encounters, then train a new policy π_{i+1} on the documented states the old π_i encountered and repeat. This is how the so called Dataset Aggregation (DAgger) algorithm works [10]. The interaction between a novice policy and an expert policy makes DAgger better than Behavior Cloning in terms of compounding error since a trained DAgger policy is further trained while correcting its mistakes.

2.5 DAgger

DAgger is an algorithm proposed as a solution for the compounding error problem found in Behavior Cloning. Collecting demonstrations for all possible states is infeasible and so DAgger focus on relevant scenarios by requesting additional demonstrations and updating the policy trained [8]. It reduces imitation learning to supervised learning with interaction between the novice and expert [10]. DAgger is an iterative algorithm which is defined as Dataset Aggregation. Another definition of DAgger "DAgger is a meta algorithm which attempts to collect expert demonstrations under the state distribution induced by the learned policy" [8]. In consideration to automated driving, a naive DAgger Algorithm 1 requires an expert to drive a vehicle using Behavior Cloning to generate the data set D. The data set generated D is used to train an initial novice policy $\pi_{nov,i}$. Next, for each DAgger iteration i, a novice policy $\pi_{nov,i}$ drives and the expert policy π_{exp} corrects the novice policy $\pi_{nov,i}$ mistakes. The degree of interaction between the expert policy π_{exp} and the novice policy $\pi_{nov,i}$ is dependent on the choice of decision rule DR $(\cdot)$, which is different

for various DAgger variants. The decision rule determines at every time step t whether the novice policy $\pi_{nov,i}$ or expert policy π_{exp} action is used to interact with the environment. Then, using the decision rule a new training data set D_i that contains information about how to recover from mistakes encountered by the novice policy $\pi_{nov,i}$ is generated. Finally, a novice policy $\pi_{nov,i+1}$ is trained by aggregating the training data set $D \leftarrow D \cup D_i$, where D_i is generated by $\pi_{nov,i}$ and D is the previously accumulated data set. This should be repeated K times and finally the best policy can be chosen given a specified criteria to test the policies [26].

Algorithm 1 DAgger Algorithm

```
procedure DAGGER(DR(·))
    initialize D ← ∅
    initialize π_{nov,i}
    for iteration i = 1 : K do
        sample T-step trajectories using DR
        Get D_i = {o_t, π_exp(o_t) | t = 1 : T}
        Aggregate datasets: D ← D ∪ D_i
        Train π_{nov,i+1} on D
```

Naive DAgger requires many queries from the expert, which could be very expensive depending on the expert being used [17]. Thus, many different variants of DAgger were proposed that are query-effecient, some of them are Safe DAgger and HG-DAgger. In the next sections DAgger variants used in the experiments are presented.

2.5.1 Vanilla DAgger

In the Vanilla DAgger algorithm a dataset D is initially sampled using the expert policy π^*. Then, a novice policy π_1 is trained on the dataset sampled by the expert policy at the beginning. For each iteration, a new dataset D_i is sampled using the novice policy while switching to expert policy depending on the decision rule. The dataset D_i is then aggregated with the previously sampled D and a new novice policy π_{i+1} is trained on the aggregated dataset. Lastly, the best novice policy is chosen given a certain criteria[10].

The decision rule for Vanilla DAgger is shown in Algorithm 2 where novice switch to the expert, sampling the experts' actions in the dataset with probability $\beta_i \in [0, 1]$ where i is a DAgger iteration. if $\beta_i = \lambda\beta_{i-1}$ for some $\lambda \in (0, 1)$ then the novice takes increasingly more actions each iteration.

Algorithm 2 Vanilla DAgger Decision Rule

1: **procedure** DR(o_t, i, β_0, λ)
2: $a_{nov,t} \leftarrow \pi_{nov}(o_t)$
3: $a_{exp,t} \leftarrow \pi_{exp}(o_t)$
4: $\beta_i \leftarrow \lambda^i \beta_0$
5: $z \sim Uniform(0, 1)$
6: **if** $z \leq \beta_i$ **then**
7: **return** $a_{exp,t}$
8: **else**
9: **return** $a_{nov,t}$

2.5.2 Ensemble DAgger

Ensemble DAgger is a variant of DAgger that differs from Vanilla DAgger in the interaction between expert and the novice. The expert π_{exp} in Ensemble DAgger samples data D_i only when the decision rule shown in Algorithm 3 is satisfied. The decision rule of Ensemble DAgger is based on computing two measures: discrepancy τ and doubt χ. The first measure is the Safe DAgger decision rule [17] and is referred to as discrepancy τ in Ensemble DAgger. Discrepancy is a measure of the deviation of the novice from expert as seen in Algorithm 3, line 4. The second measure doubt χ is a variance measure, which represents the confidence of the novice or the familiarity of the current state to the training data [18]. Two thresholds for discrepancy τ and doubt χ are set when using Ensemble DAgger.

Algorithm 3 Ensemble DAgger Decision Rule

1: **procedure** DR(o_t, τ, χ)
2: $\bar{a}_{nov,t}, \sigma^2_{a_{nov,t}} \leftarrow \pi_{nov}(o_t)$
3: $a_{exp,t} \leftarrow \pi_{exp}(o_t)$
4: $\hat{\tau} \leftarrow \|\bar{a}_{nov,t} - a_{exp,t}\|^2$
5: $\hat{\chi} \leftarrow \sigma^2_{a_{nov,t}}$
6: **if** $\hat{\tau} \leq \tau$ and $\hat{\chi} \leq \chi$ **then**
7: **return** $\bar{a}_{nov,t}$
8: **else**
9: **return** $a_{exp,t}$

Ensemble DAgger uses ensemble method, which is a technique to train a collection of neural networks to execute the same task and then combine the output to a single prediction [28]. In Ensemble DAgger doubt is measured

by computing the variance σ^2 of the predictions by the ensemble instances. The action of the novice is measured as the average predictions of ensemble instances. Ensemble DAgger aims to allow the novice to act only when it is close to the expert, which is measured by discrepancy $\hat{\tau}$ and confident in its action, which is measured by the doubt $\hat{\chi}$. Ensemble DAgger is different than Vanilla DAgger in that it only samples training data when the decision rule is satisfied and the decision rule is based on different measures than in Vanilla DAgger.

2.6 Previous work

Some related work to the project include the Super Tux Kart and Super Mario Bros with experiments carried out using Vanilla DAgger [10], SMILe [29] and SEARN algorithms [12]. For Super Tux Kart the different algorithms mentioned above are trained using kart view images of the track as input to steer the kart with fixed speed. The performance of using DAgger, SMILe and Behaviorial Cloning [8] is compared for driving the kart using average number of falls per lap. For Behaviorial Cloning the results of the experiment show that no improvement is made as more data is collected entirely by expert. SMILe performs better than Behaviorial Cloning but it still falls off the track after 20 iterations. However, for DAgger the kart stops falling off the track after 10 iterations which shows that the most optimal algorithm is DAgger.

The other experiment Super Mario Bros. is trained to play the game with the following movements (left, right, jump, speed) with the game images as input. The performance is measured as the average distance travelled by Mario. The algorithms DAgger, SMILe and SEARN are compared and DAgger outperforms SMILe as in the previous experiment Super tux Kart and it also outperforms SEARN.

Another related work that cover the use of DAgger with a safe decision rule is Safe DAgger [17]. A deep convolutional network is used as a novice to predict the angle of the steering wheel and the brake with front-facing camera images as input. The algorithms tested in the experiment are Behavior Cloning, Naive DAgger and Safe DAgger with the following evaluations metrics: average number of laps, damage per lap and mean squared error (MSE) of steering angle. The results show that with Safe DAgger the car stops going out of the lane after three iterations. It can also be seen from the results that with each iteration the querying to the expert policy decreases and this could be due to the novice learning the cases which are seen as difficult by the safety policy.

Other works are the experiments carried out using Ensemble DAgger [18]: the inverted pendulum domain and MuJoCo HalfCheetah. The goal for the Inverted Pendulum domain is to stabilize an inverted pendulum. The DAgger variants compared in those experiments are Safe DAgger where and Ensemble DAgger. The novice policy in the experiment is an ensemble of multi-layer perceptrons with the input as a two-dimensional state space $[\theta, \dot{\theta}]$ and labels as a one-dimensional action space of u. Using a fixed set of hyperparameters for each decision rule, the decision rules are then compared to each other in this experiment. The results show that using the decision rule in Ensemble DAgger helps the novice to navigate the system in safe places while switching to expert during unfamiliar places, which leads to avoiding dangerous places. However, when using the Safe DAgger decision rule the novice policy is not given as much freedom to visit states different than the data set collected by the expert trajectory resulting in the data collected being no different than Behavior Cloning. For the MuJoCo HAlfCheetah domain the goal is to run in a stable gait with eighteen-dimensional ($\mathbb{R}^{18}$) observations and sixth-dimensional ($\mathbb{R}^{6}$) actions. The DAgger variants used are the same as the previous experiment. The decision rules are also compared in this experiment by the performance of the novice and the performance of the expert. The results show that the novice performance is better than the combined system's performance for the same parameters set.

Chapter 3

Methods

This chapter starts with introducing the proposed algorithm Reward-Guided DAgger. It is one of the algorithms used in the experiments including Behavior Cloning, Vanilla DAgger and Ensemble DAgger. The experiments are carried out using the Virtual Battle Space Simulator and the algorithms are trained using a human expert and a NVIDIA network as novice.

3.1 Proposed: Reward-Guided DAgger

The idea of Reward-Guided DAgger (RG-DAgger) is to let the expert take control mostly when the novice is likely to enter an unsafe area. In the driving experiment there are two cases where the vehicle is most likely to enter an unsafe area: 1) when the vehicle is close to the road boundaries, 2) when the vehicle is not oriented towards the continuation of the road. These conditions in the driving experiment are used to distinguish between the permitted states for the novice to drive safely and the unsafe areas where the expert should take control. As seen in algorithm 4 (RG-DAgger), two signals are sent from the simulator to DAgger algorithm: signalLane and signalOrient.

Those signals are in the form of probabilities. The first Signal is used to show how close the vehicle is to the road boundaries. The closer the vehicle is to the road boundaries, the higher the probability that the expert should take control. The second signal shows the difference between the orientation of the vehicle relative to the road continuation. The bigger the difference is, the higher the probability is for the expert to takeover. Both of the signals are significant in finding unsafe states. Therefore, the logical operator OR is used to combine them.

This results to handing full control to the expert if a random number, z,

generated from a uniform distribution between zero and one is less than any of the signalLane or signalOrient probabilities. Full control in this case means the labels are sampled from the expert and the vehicle is controlled by the expert. Otherwise, the novice has control and labels are collected from the states visited by the novice. The expert has control every time the signals' probabilities are larger than a random number z. Thus, the expert is not given full control in long time periods or in fixed time periods. Compared with using a fixed time period of handing control to the expert this is more robust to different unsafe states, minimizing expert time and giving more control to the novice to explore new states.

The network is implemented in python and the environment for the experiment is implemented in the military training simulator Virtual Battle Space simulator (VBS3). The signals were implemented in the simulator and sent to the python program by using a socket with Transmission Control Protocol (TCP). RG-DAgger algorithm in python is always listening to the socket to read new signals continuously. The simulator is continuously sending signals when the simulation session is started.

Algorithm 4 RG-Guided DAgger Decision Rule

procedure DR(o_t, signalLane, signalOrient)
    $a_{nov,t} \leftarrow \pi_{nov}(o_t)$
    $a_{exp,t} \leftarrow \pi_{exp}(o_t)$
    $z \sim Uniform(0, 1)$
    if $z \leq$ signalLane $||z \leq$signalOrient **then** ▷ unsafe area
        return $a_{exp,t}$
    else ▷ safe area
        return $a_{nov,t}$

3.2 Initial setup

In this experiment the hardware used are joystick, computer, screen and mouse. A car was used in the simulator for the experiments. The car speed is set to a constant value of 20 km/h and it is initially oriented facing the road in each track. Additionally, all of the simulation keys are turned off except for the steering angle. The total number of tracks used in this experiment are twelve tracks. Ten of those tracks are used for training the DAgger iteration and Behavior Cloning. Another track is for training initial novice policy. The the last one is a test track with 900 second of driving time. All of the experiments are conducted in a simulation environment using the military simulator VBS3.

The simulator can be used for multi-playing, virtual training environment and as a development platform for modeling and simulation [1]. In the simulator the settings are adjusted for efficiency such as the car drives constantly with 20 km/h and it has infinite amount of fuel. The car and the player are non-damageable in order to avoid interrupting the experiments. The weather in the simulator is set to sunny. The surface of the training and test tracks are paved with road markings.

3.3 Simulation Environment

The data collected consists of observations of the driver view, as seen in Fig. 3.1, which were recorded using the system camera with 5Hz frequency. The images have 480x640 resolution and are paired with the corresponding steering angle command.

Figure 3.1: An example image of the driving view of the car.

The initial data set D_0 is collected using a long track with a driving time of 1200 second. This track is randomly generated in the simulator and it covers

[1] Bohemia Interactive Simulations, https://bisimulations.com/products/virtual-battlespace, 2019-05-20

many environments such as cities and green fields. In the case of DAgger iterations, a different set of tracks are used. The number of tracks used for training each DAgger iteration are ten. Those tracks are randomly generated in order to minimize human bias on the data collection. Furthermore, they cover various areas in the simulator map as can be seen in Fig. 3.2 and do not intersect with each other. All the DAgger iteration tracks are 30 second drive time. The reason for the short amount of driving time chosen is in order to ensure a variety of environments are visited. Thus, in each DAgger iteration a model π_i is trained on all the ten tracks for a total driving time of 300 second.

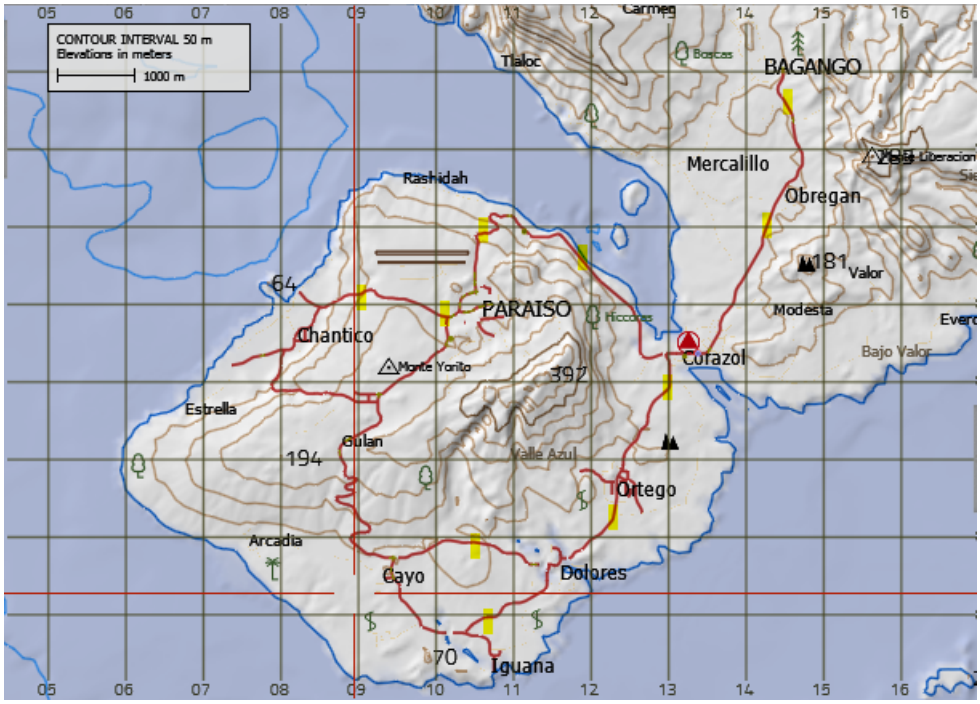

Figure 3.2: The starting position of the ten training tracks used in the DAgger iterations marked as yellow points in the map.

3.4 Data collection

A mission is designated in the simulator VBS3 fulfilling the requirements set for the experiment. For each algorithm the initial data set D_0 is collected using the long track with 1200 second driving time. This data set D_0 is collected to train an initial novice policy π_0 with the supervised learning approach. Then, all DAgger variants are run iteratively with the following steps. At each iter-

ation i, training examples are collected by a mixture of the previous novice model π_{i-1} and the expert π^* driving for 300 second on all ten tracks. At each point in time, the decision rule depending on the type of the DAgger variant, determines whether it is safe to let the novice policy drive. After completing the collection of training examples, they are combined with all the previous data collected so far D_{i-1}. This combination produces the data set D_i for the current iteration. Those steps are repeated for ten iterations. The same number of iterations are used for all algorithms to be fair. In the case of Behavior Cloning, data collection is carried out entirely by the expert. However, it is also for ten iterations and for the same set of tracks as the DAgger variants.

3.5 Policy

There are two main policies used for all algorithms, expert and novice policy. The expert policy in the experiments conducted in this thesis is a human with the assumption that he/she is an expert in driving a car in a simulator. A novice policy is the model trained to imitate the expert.

The model trained and used as a novice is based on the NVIDIA network [3], which has good performance in end-to-end self-driving cars. Small changes to the network were done in order to fit the experiments in this paper. The final network architecture is found in Fig. 3.3. Images of the driving view are fed into the NVIDIA network, which predicts a new steering value for every time step. After each DAgger iteration a new model $\pi_{nov,i+1}$ is trained on the newly collected data set aggregated with the data collected with previous iterations $D \leftarrow D \cup D_i$, as seen in Algorithm 1 line 7. This is done by using backpropagation that adjusts the weights of the network to bring the network closer to the desired steering values as seen in Fig. 3.4 where the green arrow indicates the desired steering angle and red arrow is the predicted steering angle.

In the experiments for Vanilla DAgger and RG-DAgger the NVIDIA architecture is used as the novice. In Ensemble DAgger the novice is an ensemble network where each ensemble instance is made of the same architecture (Fig. 3.3). The number of ensembles are ten and the method used for the ensembles is varying the training data, based on the interpretation of the method conducted in Ensemble DAgger paper [18]. The form of Ensemble network chosen in the experiment is called bootstrap aggregation where a subset of the training data is chosen to train each ensemble instance and returned to the sample, thus the method is sampling with replacement [30]. This might result in each observation being included more than once in the sample drawn

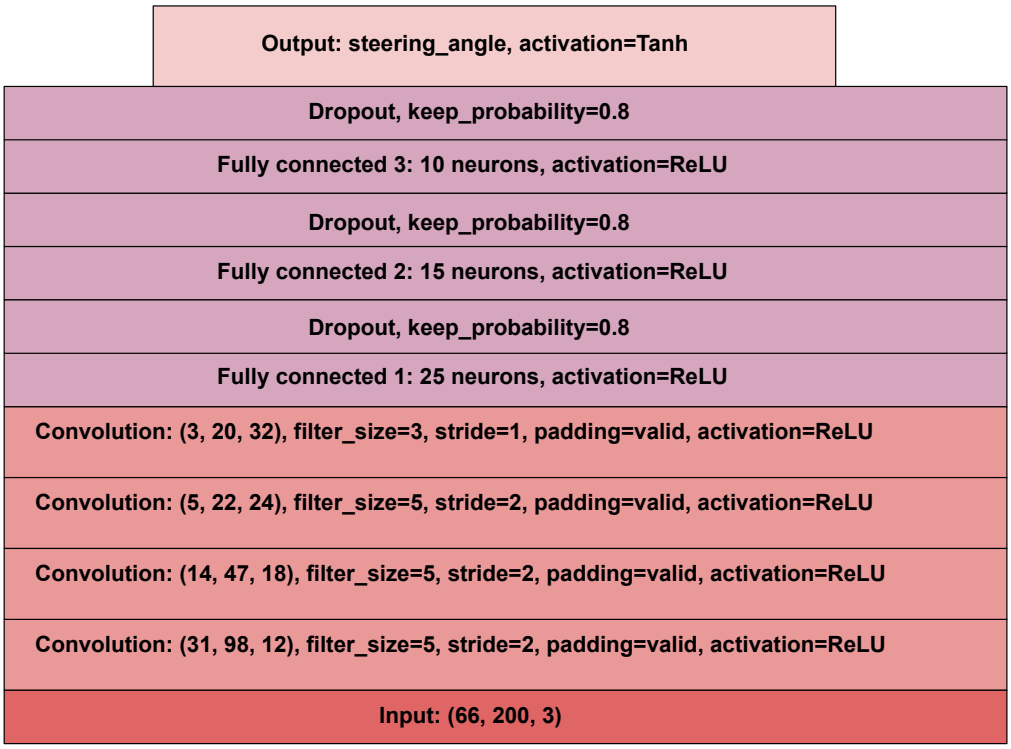

Figure 3.3: The network architecture of the novice policy: NVIDIA architecture.

for training the ensembles. This data sampling method is implemented in the experiment by dividing the training data collected so far into the number of ensembles which result in different data parts. There are two parameters that are set for configuring the Ensemble in this experiment: a percentage of the amount of training data ensemble instance train of the data collected so far and probability of an instance choosing to train on a certain data part. In this experiment all ensembles are trained with the same amount of data in order to preserve fairness among the ensembles. Therefore, each ensemble instance is trained on 80% of the data set collected so far. Additionally, each data part is chosen with 50% chance for each ensemble instance. Those parameters were chosen after some manual tuning. The classifiers are the ten ensemble instances that has a NVIDIA architecture. Those Ensemble instances are trained on 80% of the training sample. Finally, the Ensemble is tested by feeding the Ensemble instances an observation of the driving field at time t and the average of the Ensemble instances predictions is the steering angle predicted.

Figure 3.4: The desired steering angle shown as a green arrow and predicted steering angle as a red arrow.

For all the DAgger variants the human expert has to drive constantly with or without control of the car. This is due to the following reasons. For the Vanilla DAgger expert labels have to be sampled constantly in order to train the network with correct labels in the upcoming iterations. In Ensemble DAgger, the expert labels are necessary for the decision rule of the Ensemble DAgger. Since the decision rule requires a measurement of the discrepancy which is the loss $\tau \leftarrow \|\bar{a}_{nov,t} - a_{exp,t}\|^2$ between novice action predictions $a_{nov,t}$ and expert actions $a_{exp,t}$. Lastly, expert needs to drive consistently for RG-DAgger in order to minimize error during the time for switching to expert when it is given control by the decision rule. It is error prone for the human expert to take over immediately when the switching occurs. Additionally a switching signal needs to be given to the human expert when the control is transferred to the human expert. There is a risk that the reaction time of the human expert to the signal is slow and wrong labels will be sampled at the beginning when the human expert takes control since the joystick is grabbed at that moment.

In training of the DAgger variants a switch from novice to the expert is dependent on a decision rule which differs for different DAgger variants. The decision rule is tested continuously during training. In this experiment it is implemented at the frequency of 5 Hz where the decision rule is tested and

a decision is made if an expert or novice should have control over the car, resulting in the expert having control in different time spans throughout the experiment. In other words, an expert can have control over the car in time step t and control is given back during time step $t + 1$.

3.6 Choice of parameters

All of the parameters chosen for the DAgger variants are set based on manual tuning. This is done since an algorithm of hyperparameter tuning for the DAgger variants is not proposed and all of the DAgger variants except Vanilla DAgger are model based. Another reason is the use of an expert which limits using an approach for hyperparameter optimization. Therefore, manual tuning of the parameters were applied for all DAgger variants. All of the hyperparameters are presented in Table 3.1.

For Vanilla DAgger initial beta β_0 is set to $\beta_0 = 0.75$ since a higher beta value results in slower learning convergence, more iterations are required. Low beta gives little to no control to the expert, which is problematic since it results in unsafe driving. It would take longer to finish the training track with smaller β_0 since the vehicle might get out of the lane several times which results in the car getting stuck in the same place for a long time.

In Ensemble DAgger, doubt χ and discrepancy τ thresholds were chosen based on a trade-off between flexibility and strictness. Using too small discrepancy and doubt thresholds results in being too strict when the novice explores new states. When using a large discrepancy and doubt thresholds, it gives too much flexibility to the novice even in unsafe states where the expert should take control. An example of such an unsafe state is when a vehicle is driving close to the lane which is unsafe since there is a high risk that the vehicle drives outside of the lane. Therefore, discrepancy is set to $\tau = 0.8$ and doubt is $\chi = 0.02$.

In RG-DAgger ten discretization layers are chosen to portray the distance between the vehicle and the road boundaries. The layers are placed with a distance of $roadWidth/10$ where $roadWidth = 3.1$ m. Therefore, a layer is placed for every 0.31 m. Each of the layers give a different probability of when the expert should take over. The chosen probabilities are the following $prob = [1.0, 1.0, 1.0, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1]$. The first three probabilities of 1.0 represent the closest layers to the vehicle with the smallest radius from the center of the vehicle. The last probability in the array of probabilities represents the furthest layer to the vehicle. The orientation of the vehicle to the road is also measured in RG-DAgger by using the simulator. The difference

of the vehicle's orientation relative to the road is represented with probabilities. The bigger the difference, the higher the probability for the expert to take control. A difference in orientation between 25° to 45° has probability 0.4 to switch to the expert. A difference of 45° to 55° has a probability 0.5. A difference in orientation larger than 55° gives control to the expert with probability 1.0 since the car is oriented in a very unsafe way.

Hyperparameters	Values
β	0.75
τ	0.8
χ	0.02
$probLane$	[1.0, 1.0, 1.0, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1]
$probOrient$	[1.0, 0.5, 0.4]

Table 3.1: Hyperparameters for all DAgger variants

3.6.1 Implementation of RG-DAgger in VBS3

Layers of circles surrounding the vehicle with radius starting from the vehicle's center point is implemented in order to define a safe driving area of a vehicle. The circles are defined by lines starting from the center point of the vehicle which are positioned to define distance between car and unsafe area as seen in Fig. 3.6. The vehicle is in a very unsafe state if the most inner circle closest to the car is over or outside of the road boundaries. The most inner layer to the vehicle is the threshold for how close the vehicle is to the road boundaries. Each layer around the car defines different probabilities that indicate when the expert can take over. The closer the car is to the road boundaries, the higher the probability is set to switch to the expert.

The orientation signal is implemented by measuring the direction of the vehicle relative to the road in the simulator. Direction to the road is calculated by finding the next road object segment in the VBS3 simulator and extracting the direction of the object relative to the current road object. The difference in orientation of the vehicle relative to the road results in different probabilities of the expert taking control. An example is shown in Fig. 3.5 where θ is the difference between vehicle and road orientation. The larger θ is, the higher the probability that an expert should take over.

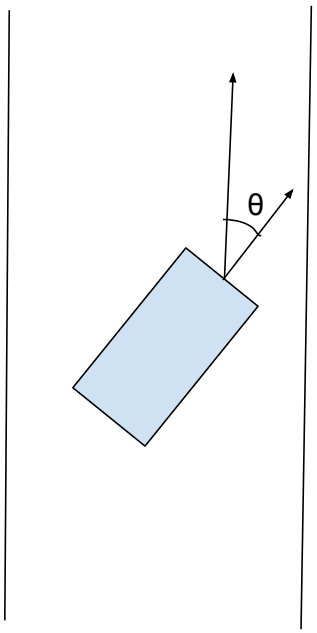

Figure 3.5: The orientation signal shown as θ in the figure, is one of the RG-DAgger algorithms' signals that measures the orientation of a vehicle to the road.

3.6.2 Key contributions of RG-DAgger

RG-DAgger minimizes expert intervention compared to other DAgger variants because it only reacts when there is a need, meaning only if the novice is positioned in unsafe state (in other words, close to the roadside or oriented differently relative to the road). Unlike other DAgger variants where the expert takes over control despite the novice being in a safe state, RG-DAgger focuses on handing over control to the expert during dangerous situations.

As mentioned, RG-DAgger gives control to the expert only when the novice is positioned in an unsafe state. This results in RG-DAgger exploring the state space more efficiently and collecting data that is not similar to the demonstration data. Other DAgger variants might be too restricted in following the trajectory of the demonstrated data during training, which might lead to the consequences of not learning how to recover from mistakes.

In the training phase RG-DAgger is safer since it gives full control to the expert when the vehicle is in an unsafe state. Safety is more important in training when using a real scenario and not only the simulator to train the models.

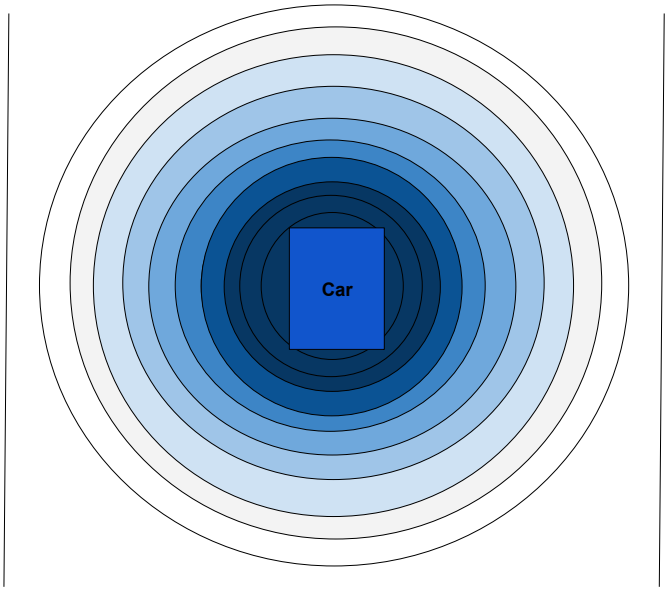

Figure 3.6: The probability distribution of the safety area of the road to the road boundaries as circles with color tones that indicate the different probabilities of safety.

In Vanilla DAgger safety is not considered and in Ensemble DAgger safety is considered at the cost of the flexibility- strictness trade-off. The vehicle will not explore much of the state space if the doubt χ and discrepancy τ thresholds are small, but it will be safe during training.

3.7 Evaluation metrics

For evaluation, we compare the three different DAgger variants 1) Vanilla DAgger, 2) Ensemble DAgger, and 3) RG-DAgger. The models for all iterations in each DAgger variant are evaluated on the test track while recording the evaluation metrics.

The extracted metric from the simulator used for performance evaluation is the number of times the vehicle drives off-road in the test track. This metric is called "number of falls" throughout the thesis. It tests the performance of Behavior Cloning and different DAgger variants' models driving on unseen track. It is a good measure for the experiments, since all models get the chance

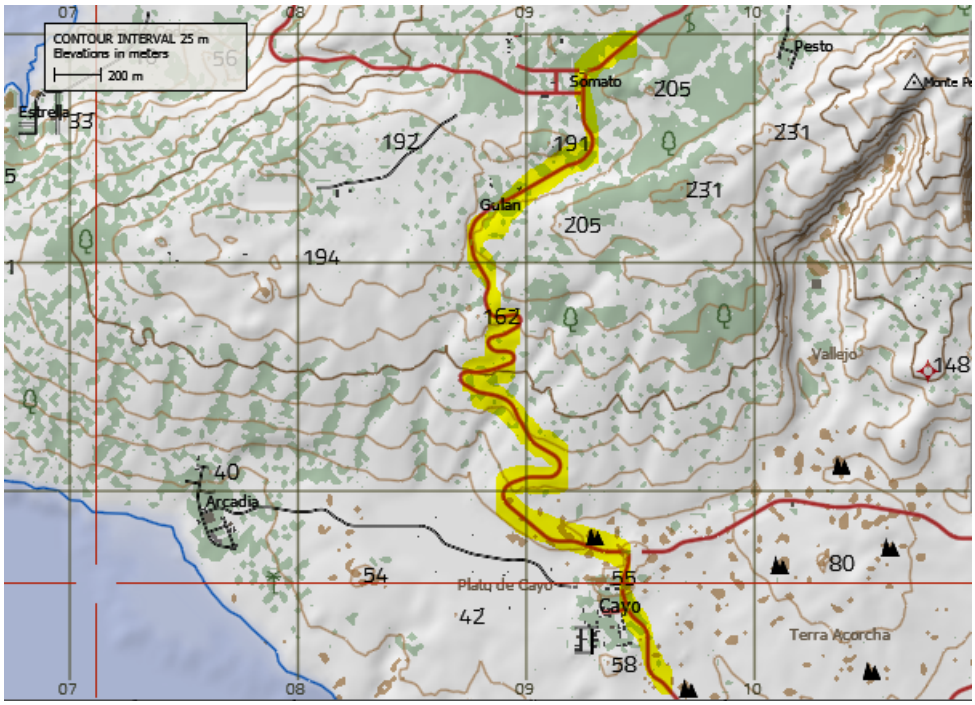

Figure 3.7: The path of the test track highlighted.

to complete driving the test track shown in Fig. 3.7. The vehicle is placed at the center of the road, parallel to the position it drove off-road. This is implemented with the help of the tools in the simulator to return it to a safe position in the test track, in order to fairly evaluate the different DAgger variants and Behavior Cloning. In similar experiments [31] it was shown that a DAgger variant model might have good performance overall but might drive off-road at the beginning. Other models could have a really bad performance but learn how to drive at the beginning of the track. This leads to drawing wrong conclusions if a model is not given the chance to complete a testing track.

Another metric is the expert time, which can be defined in two ways. First, the time $t_{fullControl}$ an expert takes full control of the vehicle by providing labels and exploring the states visited by steering the vehicle. Second, the time $t_{limitedControl}$ the algorithm queries the expert with and without handing control to the expert. For example by expert labelling the data set collected by the novice. For training, the vehicle needs to finish driving the whole track, regardless if it drives off-road. With the help of the simulator tools, the vehicle is returned at the center of the road, parallel to the position where it drove off-road. This is done in order to make sure that all DAgger variants have the same data distribution by collecting approximately the same amount of data.

Chapter 4

Results

To address the research question, an experiment was designed for comparing RG-DAgger with Behavior Cloning, Vanilla DAgger and Ensemble DAgger. The main criteria for comparative analysis are expert time and number of falls explained in Section 3.7. Due to limited resources, all results shown are derived based on a single trial only so the findings are merely indicative rather than conclusive.

4.1 Expert time

One of the metrics used for evaluating the performance of Behavior Cloning and the different DAgger variants relies on what we refer to as expert time as mentioned in section 3.7. Expert time can be interpreted in two ways $t_{fullControl}$ and $t_{limitedControl}$. The first one $t_{fullControl}$ is the time an expert takes full control of the vehicle by providing labels and exploring the states visited by steering the vehicle. The second $t_{limitedControl}$ is the time the algorithm queries the expert with and without handing control to the expert. In the case where the algorithm queries the expert without handing control to it, then it is the novice that has control of exploring the state space. The expert time $t_{fullControl}$ and $t_{limitedControl}$ for Behavior Cloning and all DAgger variants are shown in Table 4.1. The value $t_{fullControl}$ will be considered throughout the report because it is relevant for comparing the expert time for the DAgger variants.

Mode	Behavior Cloning	Vanilla DAgger	Ensemble DAgger	RG-DAgger
$t_{limitedControl}$	3000	3000	3000	249
$t_{fullControl}$	3000	1387	383	249

Table 4.1: Total expert time of Behavior Cloning and DAgger variants in experiment vs full control for the expert

4.1.1 Base model

The base model in the zeroth DAgger Iteration is not shown since it has the same expert time of 1200 second for all the algorithms. The base model for all algorithms is trained on the same data D_0 collected entirely by the expert. However, the base model has the same network in all algorithms except Ensemble DAgger where the base model is 10 ensembles trained with different parts of the initial data collection D_0.

4.1.2 Behavior Cloning

In Behavior Cloning the expert is queried the entire time in all iterations with full control given to the expert. Thus, the expert time in each iteration is the time of collecting data in the training tracks shown in Fig. 4.1. The total expert time shown in Fig. 4.2 is the sum of the expert time over each iteration, which is in total 3000 second.

4.1.3 Vanilla DAgger

Vanilla DAgger queries the expert the entire time while novice having control of exploring the state space, as shown in Algorithm 2 in Section 2.5.1. The expert is given full control meaning both exploring the state space and labeling the states when a chosen hyperparameter β is bigger than a random number sampled from a uniform distribution. In regards to $t_{limitedControl}$ Vanilla DAgger queries the expert all the time for every DAgger iteration. Therefore, the expert time $t_{limitedControl}$ for Vanilla DAgger would be the total driving time which is $10 * 10 * 30 = 3000$ second (Table 4.1). This is the same as the expert time for Behavior Cloning, as shown in Table 4.1. The expert time $t_{fullControl}$ for each DAgger iteration in Vanilla DAgger is shown in Table 4.1. The expert time $t_{fullControl}$ decreases in each iteration giving less control to the expert for every iteration. The decease in the expert time depends on the value beta β which shrinks for each iteration.

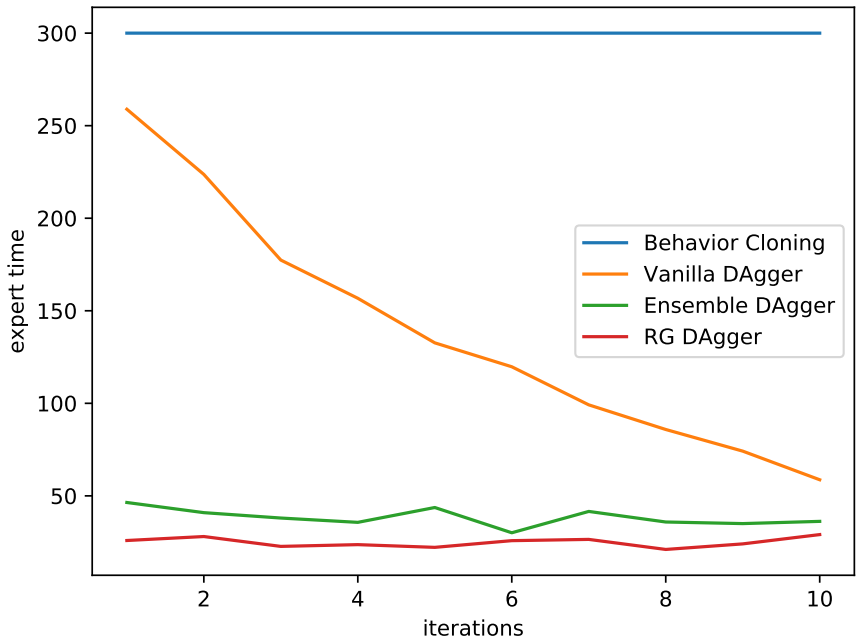

Figure 4.1: The expert time $t_{fullControl}$ for Behavior Cloning and DAgger variants with expert having full control for each DAgger iteration.

For Vanilla DAgger the expert time is large since initially the expert is given control to a large extent. After each iteration less control is given to the expert since it is assumed that the model learns how to imitate the expert for each iteration. The Vanilla DAgger has this setup because of this assumption [10]. Therefore, Vanilla DAgger has the highest total expert time $t_{fullControl}$ shown in Table 4.1 compared to the other DAgger variants but less expert time $t_{fullControl}$ compared to Behavior Cloning as seen in Fig. 4.2.

4.1.4 Ensemble DAgger

As mentioned in Ensemble DAgger the decision rule depends on two measurements: discrepancy rule and doubt rule. The discrepancy rule is the difference between novice predictions and true labels at a given state as shown in Algorithm 3 line 4. True labels in this experiment are labels provided by the human expert for each state o_t at time frame t. This means the algorithm queries the expert the entire time in order to measure discrepancy. However, in the original Safe DAgger algorithm [17], discrepancy is predicted by a trained network

called safety policy given meta data provided by the simulator used. In this thesis a safety policy for measuring discrepancy could not be constructed due to the time limit. Therefore, in Ensemble DAgger a human expert is used for measuring the discrepancy. Ensemble DAgger expert time can also be interpreted in both ways $t_{fullControl}$ and $t_{limitedControl}$. With respect to expert time $t_{limitedControl}$, Ensemble DAgger has an expert time of 3000 second and for $t_{full_control}$ the expert time is 383 second in total (Table 4.1). The expert time $t_{fullControl}$ in Ensemble DAgger is not stable, however a pattern of expert time decrease can be seen at the beginning in Fig. 4.1. In the first four iterations, the expert time decreases. Then it becomes unstable by increasing and decreasing a few iterations later, as seen in Fig. 4.1. This instability in expert

Iterations	Ensemble DAgger 0.5	Ensemble DAgger 0.8
0	1200	1200
1	49.57	46.42
2	41.75	40.93
3	46.82	38.02
4	47.19	35.69
5	41.03	43.71
6	48.74	30.06
7	48.38	41.59
8	46.65	35.88
9	47.4	35.02
10	42.68	36.22

Table 4.2: Expert time $t_{fullControl}$ for Ensemble DAgger with discrepancy 0.5 and with discrepancy 0.8.

time for ensemble is due to the use of a human expert. The algorithm Ensemble DAgger is not an algorithm designed for a human-in-the-loop setting [18]. There are several drawbacks of using a human expert in Ensemble DAgger. The time it takes to train the network in Ensemble DAgger is longer than the other DAgger variants, since there are ten ensembles trained on the data so far collected. Therefore, it takes longer time to train after each iteration. This affects the human expert accuracy, which in turn affects the expert time of Ensemble DAgger. Another factor that affects the expert time of Ensemble DAgger is the threshold chosen for discrepancy and doubt rules. A small value of discrepancy or doubt threshold results in the models following the expert trajectory and small perturbations leads to handing over control to the expert.

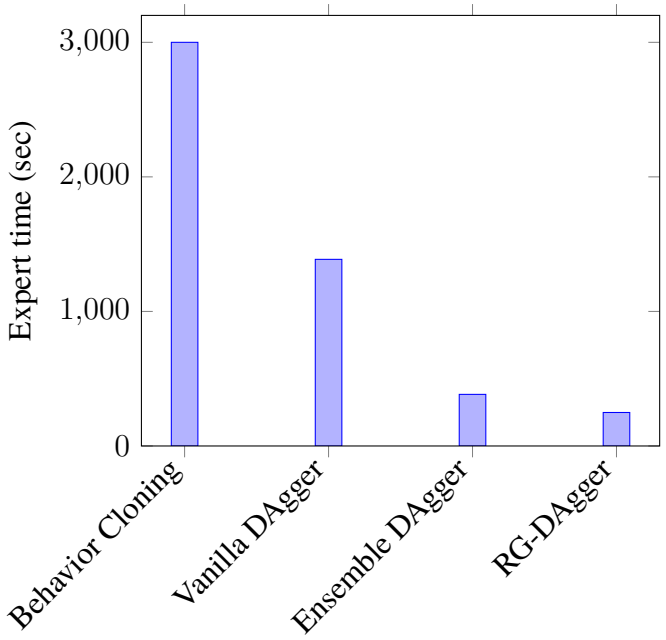

Figure 4.2: The total expert time $t_{fullControl}$ of each DAgger variant excluding the expert time for the base model.

From experiments conducted it was shown that this results to approximately constant expert time in Ensemble DAgger for discrepancy 0.5, as shown in Table 4.2.

4.1.5 RG-DAgger

The RG-DAgger algorithm queries the expert only during the time the decision rule is true and the expert would be given full control. Thus, the expert time $t_{fullControl}$ and $t_{limitedControl}$ are each 249 seconds as shown in Table 4.1. This is due to the nature of the decision rule for RG-DAgger, which only queries the expert when the decision rule is true. In Fig. 4.1 the expert time for RG-DAgger is approximately the same for each iteration around 25 seconds. The expert time does not decrease with more iterations due to the previous model still having a high chance of visiting already learned states.

RG-DAgger has the least total expert time seen in Fig. 4.2 since the expert only takes over when the novice is in an unsafe state. An unsafe state is defined as a state where the vehicle is close to lane boundaries or has an orientation that differs largely from the road continuation. It is more intuitive to define the thresholds in RG-DAgger in order to not make it too strict or too flexible

in comparison to other DAgger variants. In other words, there is better control and definition of when to hand over to the expert. This results in lower expert time in RG-DAgger compared to the other DAgger variants. However, the expert time for each DAgger iteration is approximately around the same value 25 second. The reason for this could be that in RG-DAgger the expert sometimes takes control in states that are already learned by the novice. A solution for this problem is suggested as future work in Chapter 6.

4.1.6 Total expert time

The total expert time $t_{fullControl}$ for each algorithm summed over expert time for each iteration of the algorithm is shown in Fig. 4.2. Both RG-DAgger and Ensemble DAgger have the least expert time in comparison to Behavior Cloning and Vanilla DAgger. RG-DAgger has the least expert time compared to all DAgger variants used in this experiment, due to the focus of its decision rule that intends to sample quality data.

4.2 Performance

This section presents the results of testing Behavior Cloning and each DAgger variant. The performance of each DAgger variant and Behavior Cloning is measured by the number of falls during testing and is presented in Fig. 4.3. All DAgger variants have bad performance with high number of falls in the base model and the first two DAgger iterations. This is due to the small number of states the combined system (expert and novice) has explored in the first iterations causing compounding error during testing. From the third iteration and further there is an improvement in all DAgger variants as seen in Fig. 4.3. This improvement differ depending on the DAgger variant.

Both Vanilla DAgger and RG-DAgger show better results in later iterations staring from the third iteration compared to Behavior Cloning and Ensemble DAgger. RG-DAgger converges faster towards zero number of falls and Vanilla DAgger starts to converge in the sixth iteration.

4.2.1 Average falls

The best performing DAgger variant with the least average falls and lowest standard deviation is RG-DAgger. It has an average performance of 3.80 falls in the test track and a standard deviation of 3.77 as shown in Table 4.3. The next best DAgger variant in terms of average number of falls is Vanilla DAgger

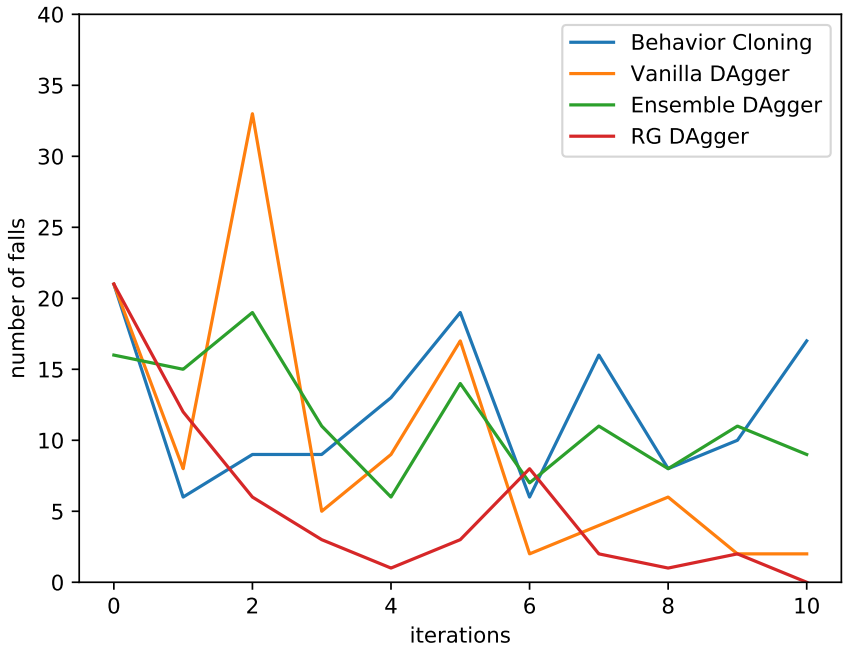

Figure 4.3: The number of falls in the test track for each trained model 0-10 for Behavior Cloning and each DAgger variant

since it has an average of 8.80 falls. However, it has a high standard deviation of 9.65, which shows that the models generated in each DAgger iteration vary significantly on performance. Ensemble DAgger has an average of 11.10 falls, which is close to Behavior Cloning average number of falls 11.30. The standard deviation of Ensemble DAgger is 3.98, which indicates poor performance in Ensemble DAgger and that more iterations would be required to lower the average number of falls.

DAgger variant	Mean (Average)	Standard deviation
Behavior Cloning	11.30	4.67
Vanilla DAgger	8.80	9.65
Ensemble DAgger	11.10	3.98
RG-DAgger	3.80	3.77

Table 4.3: Mean and standard deviation of number of falls for Behavior Cloning and DAgger variants for ten iterations excluding base model

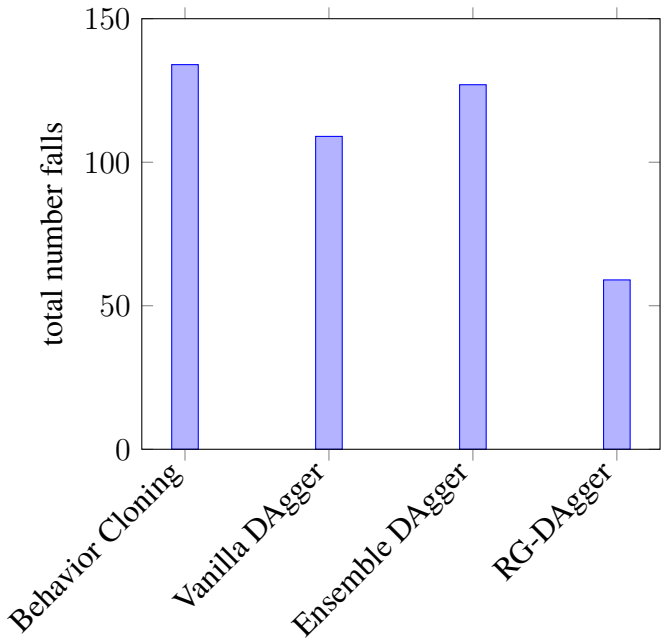

Figure 4.4: The total number of falls during test for Behavior Cloning and all models in each DAgger variant.

4.2.2 Total number of falls

Each DAgger variant has different overall performance of its models for all DAgger iterations. The total number of falls summed over the DAgger iterations indicate the overall performance of each DAgger variant. This is shown in Fig. 4.4 including the total number of falls for Behavior Cloning.

The total number of falls of Ensemble DAgger is higher than other DAgger variants shown in the Fig. 4.4. This results in the overall performance of Ensemble DAgger being worse than other DAgger variants, but it is slightly better than Behavior Cloning since the total number of falls is less than Behavior Cloning. The total number of falls, as shown in Fig. 4.4 for Behavior Cloning is higher compared to all the DAgger variants. This is expected because of compounding error [10] and it validates that a problem exists when using Behavior Cloning.

RG-DAgger has the least total number of falls. During testing it could be noticed that RG-DAgger is better in correcting both a small deviation of the expert trajectory in the test track and large perturbations compared to other DAgger variants.

Chapter 5

Discussion

5.1 Overview

This work presented a new DAgger algorithm RG-DAgger for training a self-driving vehicle. The proposed DAgger algorithm RG-DAgger is compared to a baseline, Behavior Cloning, the original DAgger algorithm, Vanilla DAgger, and the state-of-the-art algorithm, Ensemble DAgger. The idea of the design of RG-DAgger is to focus on improving the number of expert queries and quality of data collected during training. The main objective of different DAgger variants is to achieve training a network, which is able to imitate the behavior of an expert . However, each one focuses on different areas such as increasing safety [16–18], minimizing expert queries [17] or designing a more human expert friendly algorithm [19] while trying to keep the same state-of-the-art performance. The experiments carried out in this work consist of a single trial in both training and testing. The trends of the results collected show that RG-DAgger and Vanilla DAgger differ from Behavior Cloning and Ensemble DAgger in that they are better in training a network that converges faster towards lower number of falls. Moreover, Vanilla DAgger converges in expert time, which indicates learning progress. However, the results show that RG-DAgger is in general less dependent on the expert during training. Compared to other algorithms tested, RG-DAgger is overall better in generating models with good performance and least expert time.

5.2 The effect of RG-DAgger

The proposed DAgger algorithm uses two signals provided by the simulator, in order to focus on the areas that require attention of the expert during training.

This DAgger algorithm contributes to the idea of finding signals that help in defining quality data and provide better guided data collection. The idea of the algorithm can be implemented in other works where signals can be specified depending on identifying situations where quality data can be collected.

RG-DAgger attempts to minimize expert queries by identifying safe and unsafe states. It uses a simple decision boundary between safe and unsafe states, where unsafe states for a driving vehicle is the closer the vehicle drives to the edge lines and the larger the vehicle is oriented to the road continuation. This is a more straightforward concept of identifying unsafe states instead of independently training a third policy such as safety policy in Safe DAgger [17] or measuring continuously discrepancy and doubt as in Ensemble DAgger [18]. Due to how straightforward the way it distinguishes between safe and unsafe state observations, it was assumed to be more reliable in its way of distinguishing the safe and unsafe states compared to the other methods. However, there are some limitations with the algorithm such as when a model in training iterations visits a collected state deemed as unsafe with a specific probability of handing control to the expert. The decision rule hands control to the expert without taking into consideration that the model already knows how to recover from this certain state and thus the expert time will be approximately the same for all iterations. This in turn lowers the reliability of the algorithm in distinguishing between safe and unsafe states, which is a cost of the simplicity the algorithm presents. In this sense, DAgger variants such as Ensemble DAgger are more robust to learning over time the decision boundary by using the concept of doubt. However, the doubt threshold is dependent on the variance of Ensemble instance predictions, which might be in itself not reliable.

There are many factors that deteriorate the expert time for each DAgger variant where the expert time decreases or stays constant. Some of the factors that the expert time is dependent on the rate of learning convergence of the models, a hyperparameter value that tunes the expert time directly such as the hyperparameter in Vanilla DAgger beta β and finally the stability in learning. All of those factors excluding the hyperparameter value make the assumption that the expert the models are imitating is error-free. Those factors are the main factors which affect expert time in training the DAgger models. However, the choice of hyperparameters and the use of a human expert might affect the results of expert time during data collection (training). For example, a human expert's energy decreases over time or taking a break might make the human expert lose touch with how to control the joystick which might decrease the performance in the next iterations. The novice learning performance can

only be as good as the expert-level performance. The human expert introduces many issues due to the expert-novice system as mentioned in HG-DAgger [19]. This limits the safety and learning performance measured in the experiments since the RG-DAgger is dependent on signals which requires a reaction faster than the human experts' reaction. However, it is more friendly towards a human expert than some of the state-of-the-art DAgger variants such as Vanilla DAgger [10]. It provides the human expert with control authority (or full control) compared to Vanilla DAgger which provides a human expert with limited control authority that degrade the quality of action labels [19].

According to [18] there is a relation between safety and performance of a novice [18]. The safer the choices of discrepancy and doubt are, the lower the performance becomes. In the experiments reported in Chapter 4 hyperparameters were chosen manually for Ensemble DAgger and safety was taken into consideration. Therefore, the results of Ensemble DAgger when using a human expert reflect rather poor performance and instability, which is an indicator of variance described at the end of Section 4.2. Ensemble DAgger is not a human-in-the-loop algorithm where the human-in-the-loop algorithm takes into consideration human reaction time when acting as an expert [19]. In turn, this results in Ensemble Dagger suffering from actuator lag [19]. As mentioned, it was found later that a safety policy [17] can be used for measuring discrepancy but the simulator used for the experiments does not provide meta data for training a safety policy.

In the case of self-driving cars, RG-DAgger can be implemented in the scenario where a road is specified for the collection of training data set. The orientation of the car relative to the road can be measured after finding the orientation of the road and using a gyroscope to measure the orientation of the car to the road. This requires a designated training area for the training data collection. Since RG-DAgger is a very safe algorithm, the distance between the car and road boundaries can be measured using Ladar sensing according to [32]. This designated scenario is necessary for using RG-DAgger.

5.3 Behavioral cloning and DAgger data set

An underlying relation between the quality and quantity of the training data set collected exists in both Behavior Cloning and the DAgger variants. In Behavior Cloning the data collected is large and the quality is bad since the data contains no states that correct a network when it deviates from the expert trajectory [26]. The data set also contains many repetitions of the expert trajectory already learned by the network. A solution introduced for partially

improving the quality of the data is Vanilla DAgger, because it combines both network and expert when collecting training data [10]. This combination in Vanilla DAgger leads to exploring more states in the state space, which helps in returning the network to the expert trajectory when deviating from it. However, the repeated data issue still exists because during training in Vanilla DAgger the data is always collected whether the network is in control or the expert is in control. This leads to Vanilla DAgger collecting the same amount of training data as Behavior Cloning but with improved quality in the data set. Afterwards, Safe DAgger and Ensemble DAgger were introduced where their algorithms help in decreasing the issue of repeated states in the data set. In Safe DAgger, Ensemble DAgger and the newly proposed RG-DAgger the base data set is collected entirely from the expert trajectory where the network gets no control. For those algorithms the DAgger iterations consist of a combination of both expert and network where only states visited by the expert are collected. Those DAgger variants attempt to teach the network to return to the expert trajectory by handing control to the expert when the network deviates from the expert trajectory during training. They collect a higher quality and smaller data set than Vanilla DAgger and Behavior Cloning. Those algorithms are worth experimenting more on, but the trade-off in collecting a good data set using those algorithms is the complexity that they bring. RG-DAgger attempts to collect better data quality than Safe DAgger and Ensemble DAgger, which it might accomplish but it does not decrease the complexity introduced in the Safe DAgger and Ensemble DAgger. It might even add more complexity due to the increased number of hyperparameters to adjust and the relation between those hyperparameters, which is a relation that is found in Ensemble DAgger's hyperparameters: doubt and discrapancy [18].

5.4 Limitations

One significant limitation of this work is limited trials conducted due to lack of time. Each trial took extensive amount of time, especially since it is dependent on a human expert. Therefore, one trial was carried out and the results are not statistically validated. The roads chosen for training and testing purposes in the experiments are all paved roads and contain lane markings. The expert is limited to a human expert since the built-in AI in the VBS3 could not be used limited access to necessary variables. In different scenarios an AI could exist but there is still a need to develop an algorithm that is less complex and less expensive than the built-in AI. Other limitations include the functions used in VBS3 to distinguish roads from other surfaces such as the built-in function

isRoad that is not optimal since it cannot detect the sidewalks of a bridge roadway. Therefore, the one-hour-long track chosen did not contain any bridges. Otherwise, the function isOnRoad is good at detecting surfaces that are not roads.

Discrepancy is measured by using a human expert in order to find the true labels due to limitations in the data provided by the simulator. In the simulator no meta data can be read in order to train a safety policy that predicts the discrepancy, as done in [17].

The road architecture of the simulator is not optimal for automating the testing in order to carry out many tests.

5.5 Ethics and Sustainability

Imitation learning improves automation of several tasks, which in turn affects all three components of sustainability: social, environmental and economic. From the economic perspective, more companies strive to automate minor work in order to cut costs. This increases revenue for the companies and enhances product quality, but from a social pessimistic perspective it increases technological unemployment [33] if no counter measures are taken. An optimistic view is that more qualified jobs will be generated by the new technology. From the environmental sustainbility perspective, automation can save energy when using a scheduling system of the automation. An example of automation that raises up ethical questions is self-driving cars, which reduces traffic accidents but they raise ethical questions in the case where either people inside the car or outside the car will die. This is an ethical challenge for the developers and users of automated systems of how to implement decision-making of such systems.

Chapter 6

Conclusion

In conclusion, one can see that the decision rule of RG-DAgger constructed with simplicity of the concept in mind is more complex than what one has expected. Since it introduces more hyperparameters than the other DAgger variants and like in other works we do not present rules for how to set the hyperparameters in this work. Therefore, future work would include more investigations designing heuristics or strategies for finding hyperparameters for the DAgger variants. RG-DAgger is also not robust to learning over time as the other DAgger variants. Future work would be combining RG-DAgger and the idea of a doubt metric to measure the familiarity of states. As mentioned in the discussion, RG-DAgger gives more freedom for the novice to explore the state space. However, the algorithm RG-DAgger has a flaw where the signals give the same value for already learned states by the novice as mentioned in section 5.2. Adding a doubt metric will help in avoiding similar situations since the doubt metric can measure the familiarity of the current state to already trained data. The point is to minimize the probability of handing control to the expert after the expert has learned how to recover from mistakes, which is assumed to be after few iterations.

The results in this experiment are not statistically proven since they are based on a single trial. More tests needs to be taken to provide a better indication of the statistical proof of the results and perform proper comparative analyses with statistical testing. Each trial takes a lot of time to conduct and thus more time is required for ensuring the validity of the results.

Another factor that affects the reliability of the results is the use of a human expert. Further investigations are required using an AI as the expert in order to ensure accuracy of expert labels and consistency of labeling the visited states. Additionally, the use of AI helps in avoiding actuator lag, which Ensemble

DAgger suffers from in the experiments carried out. Especially, in Vanilla DAgger it is important to use an AI in order to ensure accuracy. This is because a human expert is not as efficient in labeling if not provided feedback of the input a human makes. It is also necessary to use an AI for Ensemble DAgger in order to ensure validity of results. The Ensemble DAgger algorithm is not a human-in-the-loop algorithm, which result in actuator lag that is the same problems as in Vanilla DAgger.

Lastly, even though RG-DAgger introduces more hyperparameters which adds to the complexity of the algorithm, the relation of the hyperparameters is much more straightforward than in Ensemble DAgger. Also, from the trends seen in results RG-DAgger should be investigated further before determining its value.

Bibliography

[1] G. Dodig-Crnkovic T. Holstein and P. Pelliccione. “Ethical and Social Aspects of Self-Driving Cars”. In: *arXiv preprint arXiv 1802.04103* (2018).

[2] H. Lind E. Coelingh L. Jakobsson and M. Lindman. “Collision warning with auto brake: a real-life safety perspective”. In: *Innovations for Safety: Opportunities and Challenges* (2007).

[3] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L.D. Jackel, M. Monfort, U. Muller, U. Zhang, and X. Zhang. “End to End Learning for Self-Driving Cars”. In: *arXiv preprint arXiv 1604.07316* (2016).

[4] Keshav Bimbraw. “Autonomous Cars: Past, Present and Future”. In: *Proceedings of the 12th International Conference on Informatics in Control, Automation and Robotics - Volume 1*. ICINCO 2015. Colmar, Alsace, France: Scitepress - Science and Technology Publications, Lda, 2015, pp. 191–198.

[5] K. Bimbraw. “Autonomous Cars: Past, Present and Future - A Review of the Developments in the Last Century, the Present Scenario and the Expected Future of Autonomous Vehicle Technology”. In: *ICINCO 2015 - 12th International Conference on Informatics in Control, Automation and Robotics, Proceedings* 1 (Jan. 2015), pp. 191–198.

[6] D. A. Pomerleau. “ALVINN: An Autonomous Land Vehicle in a Neural Network”. In: *in NIPS*. 1989, pp. 305–313.

[7] Y. LeCun, U. Muller, J. Ben, E. Cosatto, and B. Flepp. “Off-Road Obstacle Avoidance through End-to-End Learning”. In: *in NIPS*. 2005, pp. 739–746.

[8] T. Osa, J. Pajarinen, G. Neumann, J. Bagnell, P. Abbeel, and J. Peters. "An Algorithmic Perspective on Imitation Learning". In: *Foundations and Trends in Robotics* 7.1–2 (2018), pp. 1–179. DOI: 10.1561/2300000053.

[9] F. Codevilla, E. Santana, A. M. López, and A. Gaidon. "Exploring the Limitations of Behavior Cloning for Autonomous Driving". In: *arXiv preprint arXiv 1904.08980* (2019).

[10] S. Ross, G. J. Gordon, and D. Bagnell. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: *AISTATS*. Vol. 15. 2011, pp. 627–635.

[11] D. Misra K. Asadi and M. L. Littman. "Lipschitz Continuity in Model-based Reinforcement Learning". In: *arXiv preprint arXiv 1804.07193* (2018).

[12] J. Langford H. Daumé and D. Marcu. "Search-based structured prediction". In: *Machine Learning* 75.3 (2009), pp. 297–325. DOI: 10.1007/s10994-009-5106-x.

[13] S. Schaal. "Is imitation learning the route to humanoid robots?" In: *Trends in Cognitive Sciences* 3.6 (1999), pp. 233–242. DOI: 10.1016/S1364-6613(99)01327-3.

[14] P. Abbeel and A. Y. Ng. "Apprenticeship Learning via Inverse Reinforcement Learning". In: *Proceedings of the Twenty-First International Conference on Machine Learning*. Banff, Alberta, Canada: Association for Computing Machinery New York, NY, USA, 2004, pp. 1–8. DOI: 10.1145/1015330.1015430.

[15] *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*. MIT Press, Sept. 2007, p. 1690.

[16] K. Driggs-Campbell K. Menda and M. J. Kochenderfer. "DropoutDAgger: A Bayesian Approach to Safe Imitation Learning". In: *arXiv preprint arXiv 1709.06166* (2017).

[17] J. Zhang and K. Cho. "Query-Efficient Imitation Learning for End-to-End Autonomous Driving". In: *arXiv preprint arXiv 1605.06450* (2016).

[18] K. Driggs-Campbell K. Menda and M. J. Kochenderfer. "EnsembleDAgger: A Bayesian Approach to Safe Imitation Learning". In: *arXiv preprint arXiv 1807.08364* (2018).

[19] K. Driggs-Campbell M. Kelly C. Sidrane and M. J. Kochenderfer. "HG-DAgger: Interactive Imitation Learning with Human Experts". In: *arXiv preprint arXiv 1810.02890* (2018).

[20] D. L. Hudson and M. E. Cohen. *Neural Networks and Artificial Intelligence for Biomedical Engineering*. Wiley-IEEE Press, 2000, pp. 59–77.

[21] S. Marsland. *Machine Learning: An Algorithmic Perspective*. Second Edition. Chapman and Hall/CRC, 2014, pp. 1–452.

[22] U. Michelucci. *Applied Deep Learning A Case-Based Approach to Understanding Deep Neural Networks*. First Edition. Berkeley, CA, USA, Apress, 2018, pp. 1–401.

[23] W. Di. *Deep learning essentials your hands-on guide to the fundamentals of deep learning and neural network modeling*. Birmingham, 2018, pp. 60–115.

[24] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016, pp. 164–223.

[25] V. Dumoulin and F. Visin. "A guide to convolution arithmetic for deep learning". In: *arXiv preprint arXiv 1603.07285* (2016).

[26] H. He, H. D. III, and J. Eisner. "Imitation Learning by Coaching". In: *in NIPS*. 2012, pp. 3158–3166.

[27] T. Benjamin, C. Vassil, K. Daphne, and G. Carlos. "Learning Structured Prediction Models: A Large Margin Approach". In: Jan. 2005, pp. 896–903. DOI: 10.1145/1102351.1102464.

[28] Z. Zhi-Hua, W. Jianxin, and T. Wei. "Ensembling Neural Networks: Many Could Be Better Than All". In: *Artificial Intelligence* 137 (2002), pp. 239–263. DOI: 10.1016/S0004-3702(02)00190-X.

[29] R. Stephane and J. A. Bagnell. "Efficient Reductions for Imitation Learning". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. Sardinia, Italy, 2010, pp. 661–668.

[30] A. Buja and W. Stuetzle. "Observations on bagging". In: *Statistica Sinica* 16.2 (2006), pp. 323–351.

[31] A. Elers. "Continual imitation learning : Enhancing safe data set aggregation with elastic weight consolidation". M.S. thesis. Dept. Elect. Eng. and Comp. Sci.: Royal Institute Of Technology KTH, Stockholm, Sweden, 2019.

[32] W.S. Wijesoma, S. Kodagoda, and A. Balasuriya. “Road-Boundary Detection and Tracking Using Ladar Sensing”. In: *Robotics and Automation, IEEE Transactions on* 20 (2004), pp. 456–464. DOI: 10.1109/TRA.2004.825269.

[33] K. yvind. “Automation and Ethics”. In: *Moral Reasoning at Work: Rethinking Ethics in Organizations*. Springer International Publishing, 2019, pp. 69–77. DOI: 10.1007/978-3-030-15191-1_8.

TRITA -EECS-EX-2020:796