DEGREE PROJECT IN INFORMATION AND COMMUNICATION TECHNOLOGY,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, *SWEDEN 2019*

Continual imitation learning

Enhancing safe data set aggregation with elastic weight consolidation

ANDREAS ELERS

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

Continual imitation learning: Enhancing safe data set aggregation with elastic weight consolidation

ANDREAS ELERS

Master of Science in Information and Communication Technology
Date: June 7, 2019
Supervisor: Farzad Kamrani, Amir Payberah
Examiner: Henrik Boström
School of Electrical Engineering and Computer Science
Host company: Swedish Defence Research Agency (FOI)
Swedish title: Stegvis imitationsinlärning: Förbättring av säker datasetsaggregering via elastisk viktkonsolidering

Abstract

The field of machine learning currently draws massive attention due to advancements and successful applications announced in the last few years. One of these applications is self-driving vehicles. A machine learning model can learn to drive through behavior cloning. Behavior cloning uses an expert's behavioral traces as training data. However, the model's steering predictions influence the succeeding input to the model and thus the model's input data will vary depending on earlier predictions. Eventually the vehicle may deviate from the expert's behavioral traces and fail due to encountering data it has not been trained on. This is the problem of sequential predictions. DAGGER and its improvement SafeDAGGER are algorithms that enable training models in the sequential prediction domain. Both algorithms iteratively collect new data, aggregate new and old data and retrain models on all data to avoid catastrophically forgetting previous knowledge. The aggregation of data leads to problems with increasing model training times, memory requirements and requires that previous data is maintained forever. This thesis's purpose is investigate whether or not SafeDAGGER can be improved with continual learning to create a more scalable and flexible algorithm. This thesis presents an improved algorithm called EWC-SD that uses the continual learning algorithm EWC to protect a model's previous knowledge and thereby only train on new data. Training only on new data allows EWC-SD to have lower training times, memory requirements and avoid storing old data forever compared to the original SafeDAGGER. The different algorithms are evaluated in the context of self-driving vehicles on three tracks in the VBS3 simulator. The results show EWC-SD when trained on new data only does not reach the performance of SafeDAGGER. Adding a rehearsal buffer containing only 23 training examples to EWC-SD allows it to outperform SafeDAGGER by reaching the same performance in half as many iterations. The conclusion is that EWC-SD with rehearsal solves the problems of increasing model training times, memory requirements and requiring access to all previous data imposed by data aggregation.

Keywords: Elastic weight consolidation, SafeDAGGER, DAGGER, Rehearsal buffer, Self-driving vehicle, Continual learning

Sammanfattning

Fältet för maskininlärning drar för närvarande massiv uppmärksamhet på grund av framsteg och framgångsrika applikationer som meddelats under de senaste åren. En av dessa applikationer är självkörande fordon. En maskininlärningsmodell kan lära sig att köra ett fordon genom beteendekloning. Beteendekloning använder en experts beteendespår som träningsdata. En modells styrförutsägelser påverkar emellertid efterföljande indata till modellen och således varierar modellens indata utifrån tidigare förutsägelser. Så småningom kan fordonet avvika från expertens beteendespår och misslyckas på grund av att modellen stöter på indata som den inte har tränats på. Det här är problemet med sekventiella förutsägelser. DAGGER och dess förbättring SafeDAGGER är algoritmer som möjliggör att träna modeller i domänen sekventiella förutsägelser. Båda algoritmerna samlar iterativt nya data, aggregerar nya och gamla data och tränar om modeller på alla data för att undvika att katastrofalt glömma tidigare kunskaper. Aggregeringen av data leder till problem med ökande träningstider, ökande minneskrav och kräver att man behåller åtkomst till all tidigare data för alltid. Avhandlingens syfte är att undersöka om SafeDAGGER kan förbättras med stegvis inlärning för att skapa en mer skalbar och flexibel algoritm. Avhandlingen presenterar en förbättrad algoritm som heter EWC-SD, som använder stegvis inlärningsalgoritmen EWC för att skydda en modells tidigare kunskaper och därigenom enbart träna på nya data. Att endast träna på nya data gör det möjligt för EWC-SD att ha lägre träningstider, ökande minneskrav och undvika att lagra gamla data för evigt jämfört med den ursprungliga SafeDAGGER. De olika algoritmerna utvärderas i kontexten självkörande fordon på tre banor i VBS3-simulatorn. Resultaten visar att EWC-SD tränad enbart på nya data inte uppnår prestanda likvärdig SafeDAGGER. Ifall en lägger till en repeteringsbuffert som innehåller enbart 23 träningsexemplar till EWC-SD kan den överträffa SafeDAGGER genom att uppnå likvärdig prestanda i hälften så många iterationer. Slutsatsen är att EWC-SD med repeteringsbuffert löser problemen med ökande träningstider, ökande minneskrav samt kravet att alla tidigare data ständigt är tillgängliga som påtvingas av dataaggregering.

Nyckelord: Elastisk viktkonsolidering, SafeDAGGER, DAGGER, Repeteringsbuffert, Självkörande fordon, Stegvis inlärning

Acknowledgments

I would like to thank my academic supervisor Amir Payberah for giving me feedback and support throughout the thesis. My examinator Henrik Boström also deserves my gratitude for providing me with a structured thesis process and feedback.

During my work I have had many giving discussions with Farzad Kamrani and Mika Cohen at FOI. Your help and input have been of great value, thank you.

Contents

List of Figures

List of Tables

Abbreviations

ANN artificial neural network

CNN convolutional neural network

DAGGER data set aggregation

DNN deep neural network

EWC Elastic weight consolidation

FOI Swedish Defence Research Agency

iCaRL incremental classifier and representation learning

i.i.d. independent and identically distributed

LML lifelong machine learning

MLP multi-level perceptron

MSE mean squared error

VBS3 Virtual Battlespace 3

Chapter 1

Introduction

The interest for machine learning, especially deep learning, has skyrocketed during the last few years [1]. This is due to the last decade of increase in compute power, increase in data generation rates [2, 3] as well as algorithmic and tooling improvements. Another reason the interest for deep learning has increased is that it simply works really well for many domains. Google Translate's performance increased drastically when deep learning was applied [4, 5]. In 2015, a deep learning model from Microsoft surpassed human performance in object recognition [6] in the ImageNet competition and researchers could identify cancer metastases with an accuracy rivaling a trained pathologist [7]. Recording an expert's actions in certain states and using the data to train models is called *behavior cloning* [8]. Lane keeping autonomous vehicles can be achieved with behavior cloning by recording a human driver's actions, i.e., the steering wheel's angle, together with corresponding pictures of the road, and use the data to train a machine learning model [9].

This work is commissioned by the Swedish Defence Research Agency (FOI) [10] as an endeavor to build and spread knowledge within the organization. FOI is a leading research institute in defence and security. Their main activities include research, development of methods and technologies, analyses and studies.

This chapter's remaining parts are outlined as follows: Section 1.1 gives a background, section 1.2 defines the problem, section 1.3 defines purpose and research question. Section 1.4 defines the goal, section 1.5 discuss ethics and sustainability, section 1.6 describes the research methodology. Section 1.7 presents delimitations and section 1.8 outlines the rest of this report.

1.1 Background

Autonomous vehicles is a sequential prediction problem [11] where each steering prediction will affect the next prediction since the previous prediction influences the input distribution, i.e., the view of the road in front of the car. Mispredictions causes a model to deviate from the expert's behavior it is trained on and by deviating the model will eventually encounter input data it is not trained on. This is the problem of *compounding errors*. Behavior cloning suffers from compounding errors and data set aggregation (DAGGER) [12] is an algorithm that can reduce these errors. DAGGER reduces the problem of compounding errors by iteratively collecting additional data, appending it to the previous data and then retraining the model from scratch on all data to avoid forgetting previously learned knowledge. An enhanced version of DAGGER is SafeDAGGER [13] that is more data efficient than DAGGER since it only retrains models with data that is deemed difficult. SafeDAGGER is further described in section 2.3.

Trained models lack flexibility, if one wants to extend a model to handle an additional task or a skewed input distribution then one needs to retrain the model on both old and new data. Otherwise the model will learn to model the new data but forget how to model the old data. This is the reason DAGGER and SafeDAGGER aggregate new and old data. The issue of new knowledge overwriting older knowledge is called *catastrophic forgetting* [14] and is indicated by a model performing well on the new data but its performance on the old data has severely degraded. Retraining models when one has collected a sufficient amount of new data can be prohibitively expensive in the long run and may not even be feasible at all since one may not have access to the old data anymore. Given a streaming context where a model is trained in an online fashion it may even be impossible to save all data due to the data generation rates. *Continual learning* [15] is an area of research that focuses on enabling models to handle shifts in input distribution and to learn new tasks incrementally, without forgetting earlier tasks.

Elastic weight consolidation (EWC) [16] is an continual learning algorithm that enables models to learn tasks incrementally without catastrophic forgetting. EWC assumes there exists an explicit loss function in order to be applicable. EWC protects parameters that are important for a task by adding a quadratic term to the loss function. Thus parameters important for a task are not frozen when learning new tasks but changing them incur a high cost. The

notion of adding tasks in continual learning papers is twofold and can to refer to (1) changes in the input while the target domain remains unchanged, and (2) expand the target domain by adding an actual task, which may or may not be similar to previous tasks. An example of adding a similar task can be to add the ability to also recognize an additional car to a model that recognize cars.

In this thesis the notion of knowledge refers to a model's capabilities after being trained on data for some task. Thus the phrase *forgetting previous knowledge* means that a model's performance for a previous task has degraded. The notion of task in this thesis only refers to the first definition used in continual learning papers described earlier, i.e., a new task equals a change in the input distribution.

1.2 Problem

In order to avoid forgetting earlier learned knowledge while learning new knowledge, DAGGER and SafeDAGGER need to retrain models on the aggregated data set containing both new and old data. Training only on new data cause catastrophic forgetting [14] of previous knowledge. In each iteration of DAGGER and SafeDAGGER, new data is collected by deploying models and recording input while using a human expert to provide correcting actions that are used as labels. This data aggregation process is iterative, thereby leading to larger and larger memory requirements and also longer training periods. In the real world it might even be unfeasible to store all previous data, access to the data might become restricted or the data may simply become lost due. Retraining a model on huge data sets may also be unfeasible as it can require a lot of time. Thus, it is of great value if models can be trained solely on newly collected data in each iteration, without aggregating new and old data and without forgetting previously learned knowledge. EWC's evaluation [16] shows that it can protect against catastrophic forgetting in the permuted MNIST test, described in section 2.5, which simulates shifting input distributions. However, as the permuted MNIST test has been criticized of giving unrealistically good results [17, 18] it is unknown whether or not EWC can protect against catastrophic forgetting in a more realistic context such as training solely on new data in each iteration of DAGGER and SafeDAGGER.

1.3 Purpose

This purpose of this work is to investigate whether the SafeDAGGER algorithm can be made more scalable in terms of memory and training time by utilizing EWC to protect knowledge learned from previous data. This approach would enable training only on new data instead of all aggregated data. This leads to the following research question: *can the SafeDAGGER-algorithm be enhanced with the continual learning technique EWC to avoid aggregating new and old data in each iteration and instead allow training models only on new data, yet maintaining the same performance as the ordinary SafeDAGGER?*

1.4 Goal

The goal of combining EWC and SafeDAGGER to creating a more scalable version of SafeDAGGER is to enable others to use it where it was previously unfeasible to use.

This thesis's contribution is EWC-SD. EWC-SD is a scalable and flexible version of SafeDAGGER that lacks the need of saving earlier data by using EWC. The result is a viable algorithm suitable for usage when it is unfeasible to keep aggregating data.

1.5 Ethics and Sustainability

This section presents the ethical concerns related to continual learning as well as the possible implications on social, environmental and economical sustainability.

Few ethical concerns are related to continual learning as it is a technique to extend machine learning models. Though concerns may arise from how continual learning is applied, e.g., continual learning may enable previously unfeasible systems that are ethically questionable. Continual learning is still believed to be appropriate to study since the work aligns with the fifth point in IEEE's Code of Ethics [19], one should make explicit the implications of emerging technologies.

More and more jobs are being automated, especially low-skill jobs [20]. Achieving continual learning would bring humanity one step closer to general artificial intelligence, which would further increase the share of jobs being automated. Such a situation would bear resemblance to the industrial revolution, which was characterized by social upheaval [21], but led to more prosperity in the long run. At the same time, automation enables humans to focus on less menial tasks and possibly reduce the need for work altogether.

The ability to continuously learn new tasks without catastrophic forgetting would provide environmental benefits as the need for retraining models decreases. The benefit lies in a reduced power usage as entire models do not have to be retrained when additional tasks appear. Even though the power savings from a single model may be insignificant, the sum of all power savings due to continual learning can be significant and thus a step toward combating climate change, which is the 13th United Nations sustainable development goal [22].

From the point of view of economical sustainability, continual learning can increase companies' profits as more jobs can be automated. While automating jobs increase monetary profits, it can also harm social sustainability if societies around the world do not change together with technological advancements.

1.6 Research Methodology

The research approach is deductive since the work originates in the theory [23] of continual learning and imitation learning. A measurable hypothesis is formulated and evaluated through quantitative experiments on an autonomous vehicle's driving performance. The research methods are experimental as different hyperparameters are experimented with and applied as practical problem is solved by combining EWC and SafeDAGGER. The research strategy is experimental as externally affecting factors are minimized, a hypothesis is tested and a lot of data is used. Data consisting of road images labeled with steering angles is collected through experiments. The data is analyzed with statistics. Chapter 3 gives a more in-depth explanation of the research methodology.

A literature study is performed to research previous approaches to continual learning, imitation learning and the current state-of-the-art. Adjacent research

areas are also investigated in order to construct a solid starting point for the work. The literature study's results are presented in Chapter 2.

1.7 Delimitations

Delimitations are made to limit the scope of the thesis. This thesis's delimitation regard the choice of continual learning algorithm. EWC is the only used continual learning algorithm. It is plausible that other continual learning algorithms may perform even better than EWC, however other algorithms are not considered in order to make the work's scope manageable.

1.8 Outline

The rest of this report is outlined as follows:

Chapter 2 - Background: This chapter gives the knowledge needed to understand this work and also presents related work.

Chapter 3 - Methodology: This chapter presents the chosen research methods, discusses validity, reliability and reproducibility and describes how the research methods were applied.

Chapter 4 - Results: This chapter describes the EWC-SD algorithm and the experiments' empirical results.

Chapter 5 - Discussion: This chapter analyzes and discusses the results.

Chapter 6 - Conclusions and Future Work: This chapter provides conclusions and future work.

Chapter 2

Background

This chapter provides the reader with the necessary information needed to understand the contents of this thesis and also the related work. Section 2.1 presents supervised learning, section 2.2 explains deep learning, sections 2.3 describes the area of behavior cloning and section 2.4 gives an in depth description of continual learning and catastrophic forgetting. Related work is presented at the end of the chapter in section 2.5.

2.1 Supervised Learning

Machine learning can be decomposed into the three branches supervised, unsupervised and reinforcement learning [24]. Supervised learning utilizes pairs of labeled data, (X, Y), called *training examples*. X is a data point and Y is a label providing some ground truth. The goal of supervised learning is to learn a function that maps X to Y, i.e, $f : X \rightarrow Y$. The idea is that the learned function should be able to predict the information previously given by the label when new unlabeled data is fed into the function [25]. An example is to feed a house's size to the function that then predicts the value of a house after being fed many training examples of houses' sizes, X, and labels, Y, indicating corresponding house valuations.

Learning a mapping from input, X, to output, Y, can be done in several ways, but the most prominent approach is to minimize a *loss function* [25]. A loss function signifies how badly the function's predictions are through a scalar value. A low loss indicates a good mapping and a high loss indicates a bad

mapping. An example of a loss function is *mean squared error (MSE)*, see Equation 2.1. MSE is a commonly used loss function when the output is a continuous value. MSE calculates the mean of the squared errors, i.e., difference between predictions, Y_i, and ground truth, $\hat{Y}_i$.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{2.1}$$

Gradient descent is an optimization algorithm used to find a function's minimum value and can thus be used to minimizes loss functions. It works through repeatedly tweaking parameters by taking small steps in the opposite direction of the gradient, thus slowly descending the loss function's slope. A gradient is a vector containing the partial derivative of each of the function's parameters. The derivative gives a function's rate of change, it is positive when the function increases, negative when the function decreases and zero when the function is not changing. The intuition of taking a step of opposite direction to the gradient is to go down the function's slope toward its minimum, see Figure 2.1.

Deciding what data that is important for some task can be difficult. The input, X, must be relevant to the output, Y, otherwise the learned function's predictions will be awful. Using something unrelated as input, e.g., ice-cream sales, instead of house sizes in the earlier example would not work well at all. The process of identifying, combining and creating appropriate data to learn from

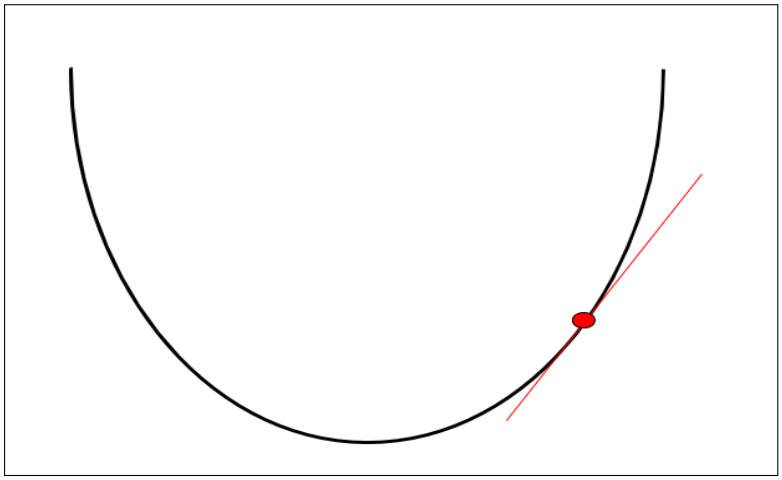


Figure 2.1: Convex function. Red dot shows a point with a positive derivative. The function's minimum is as the bottom of the bowl. [Created by author]

is called *feature engineering* [25]. Feature engineering is both difficult and expensive as it requires experts with domain knowledge to determine what is important for the task at hand. Other methods such as deep learning can reduce the need of feature engineering.

2.2 Deep Learning

The first artificial neural network (ANN) [26] was inspired by the structure of the brain, i.e., a network of interconnected neurons. The simplest ANN is the perceptron [27], which consists of an input layer and a single layer of neurons. An ANN is a deep neural network (DNN), hence the name deep learning, when there exists at least one layer of neurons between the input and output layer. These layers are called *hidden layers* and Figure 2.2 shows a *multi-level perceptron (MLP)* with one hidden layer. A MLP approximates a function which maps input to output [24] through feedforward computations, i.e., each layer's neurons results are fed to the succeeding layer. However when talking about DNNs it implies networks with many hidden layers, e.g., ResNet-152 that has 152 layers [28].

An ANN consists of an architecture, the network's topology, and a set of tuneable parameters that consist of weights and biases. Tweaking these parameters through the *backpropagation* [29] training process allows the network to learn mappings from input to output by creating an internal representation [29]. There is less need for feature engineering since an internal representation is learned but it comes at a cost since deep models are more computationally expensive. Some type of data, e.g., a picture's pixels, can be input to ANNs without any processing and still give good results but one can reduce the training time and and improve results by pre-processing the data. An example is face recognition where a cropped and scaled picture is given as input to an ANN instead of a person together with its surrounding environment. In the latter case the ANN will have to learn to differentiate the face from the surroundings and also deal with faces of different sizes.

The process of an ANN making a prediction is as follows. Each neuron in an ANN calculates a weighted sum of its inputs, adds a bias and applies an activation function to the result. The output of the activation function is the neuron's output, which is fed to the neurons in the succeeding layer. Input neurons and biases do not perform any calculations, input neurons are simply the input data that is forwarded to the neurons in the next layer and biases are

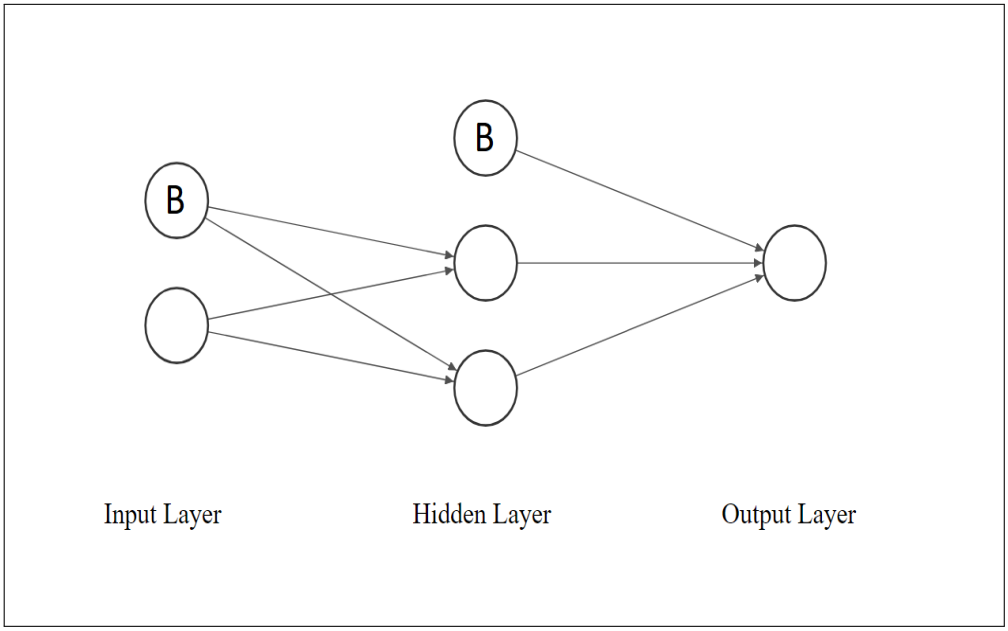


Figure 2.2: Multi-level perceptron with one hidden layer. Input layer consists of one neuron, hidden layer has two neurons and the output layer has one neuron. All layers, except the output layer, have one bias neuron that is denoted with a B in the figure. [Created by author]

extra parameters giving an additional degree of freedom. The output of an ANN's final layer is the prediction. An example is an ANN trained to say yes or no whether there is a dog present in an image, i.e., binary classification. Such an ANN would have a single output neuron in its final layer that receives its input from the previous layer, using the sigmoid activation function that outputs values between zero and one and if the output would be higher than some threshold it would be interpreted as a yes prediction.

2.3 Behavior Cloning

Imitation learning is the process of learning behaviors from demonstrations. A simple form of imitation learning is behavior cloning, which is succinctly and precisely described as "The process of reconstructing a skill from an operator's behavioural traces by means of Machine Learning techniques" [8]. An example of behavior cloning is to learn a model that clones a human driver's behavior. A human drives a car meanwhile a camera records pictures of the

road as well as the steering wheel's angle that is used to label the camera's images. These angle labeled road images can then be applied in a normal supervised learning fashion to train a model that maps camera inputs to appropriate steering wheel angles.

Behavioral cloning has been successful in many areas. ALVINN [30] uses a three-layered neural network to follow a road using road images and a laser range finder as input. The neural network's prediction represents the most appropriate steering direction in order to stay on the road. Another, more recent work, uses a convolutional neural network (CNN) to learn a self-driving car to stay within a lane based on steering wheel angle-labeled images alone [9]. Other successful applications of behavior cloning consist of a quadrocopter being able to follow a forest trail [31] and piloting an aircraft [32].

The difference between supervised learning and imitation learning is that the former assumes training and test data to be independent and identically distributed (i.i.d.) which is not true in the latter. The learned model in imitation learning affects the distribution of succeeding input data, i.e., predictions are independent of each other in supervised learning meanwhile predictions in imitation learning are sequential and have an effect on succeeding predictions [11]. As an example, if a model predicts steering left in a right turn then the policy's next action needs to be a sharp right to prevent the vehicle from going off-road. However such recovery examples may be rare or non-existing in the training data as the training data is assumed to originate from a human expert. This can lead to compounding errors where deviations from the human expert's behavior trace will cascade and lead to further errors [33]. Collecting data that contains recovery examples mitigates these errors, e.g., by using DAGGER [12], see algorithm 1.

DAGGER is a algorithm that is used to iteratively collect additional data, merge new and old data and then retrain models on all data to create improved models. Initially, one gathers data and trains a basic model $\hat{\pi}_1$. When collecting additional data, the model controls the entity, e.g., a self-driving car, but the human expert's actions are recorded and used as labels. Based on a parameter β, see Algorithm 1 line 4, the human expert is sometimes allowed to control the vehicle and correct its trajectory. If β is one, the human expert is in control all the time. $\hat{\pi}_i$ is the model, π^* is the human expert and π_i is the combination of both based on β. By recording the expert's actions when the model itself is controlling the vehicle, one collects data indicating what the model should actually have done in each state. These collected data representing correcting actions can then be used to retrain the model together with all

Algorithm 1 DAGGER

1: Initialize $\mathcal{D} \leftarrow \emptyset$
2: Initialize $\hat{\pi}_1$ to any policy in $\prod$
3: **for** i=1...N **do**
4: Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$
5: Sample T-step trajectories using π_i
6: Get dataset $\mathcal{D}_i = (s, \pi^*(s))$ of visited states by π_i and actions given by expert
7: Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$
8: Train classifier $\hat{\pi}_{i+1}$ on $\mathcal{D}$
9: **return** best $\hat{\pi}_i$ on validation

earlier data in order to learn the model to handle states that it previously could not handle. Retraining on both new and old data is done to avoid forgetting previously learned knowledge. DAGGER returns all trained models as it is not guarantee that each model improves monotonically, thus one has to check which of the models has the best performance. The pseudocode for DAGGER can be seen in Algorithm 1.

SafeDAgger [13] is a variant of DAGGER that uses a control model that drives the vehicle as in DAGGER but adds a safety model. The safety model determines if the control model's prediction are good enough. If the predictions are good enough the control model can control the vehicle, otherwise control is given to the human expert. Thus SafeDAGGER only collects additional data from "difficult" states that causes the control model to fail at its task, e.g., a sharp turn causing a trained model to drive off the road. The purpose of only collecting data from difficult states is to reduce the involvement of a human expert. The pseudocode for SafeDAGGER can be seen in Algorithm 2. Lines 1-4 use a *reference policy* π^*, i.e., a human expert, to collect initial data $\mathcal{D}$ and train an initial control model π_0 and an initial safety model $\pi_{safe,0}$. The safety strategy at line 6 is the process of giving control to the human expert π^* when the control model π_i cannot drive safely. Line 7 formalizes the safety strategy. The safety strategy's purpose is to allow more data collection without crashing. Lines 8-10 aggregate data and update both models by retraining them on the aggregated data.

Algorithm 2 SafeDAGGER

Collect $\mathcal{D}_0$ using a reference policy π^*
Collect $\mathcal{D}_{safe}$ using a reference policy π^*
$\pi_0 = argmin_\pi loss_{supervised}(\pi, \pi^*, D_0)$
$\pi_{safe,0} = argmin_{\pi_{safe}} loss_{safe}(\pi_{safe}, \pi_0, \pi^*, D_{safe} \cup D_0)$
for i=1...M **do**
    Collect D' using safety strategy using π_{i-1} and $\pi_{safe,i-1}$
    Subset Selection: $D' \leftarrow \phi(s) \in D' | \pi_{safe,i-1}(\pi_{i-1}, \phi(s)) = 0$
    $\mathcal{D}_i = \mathcal{D}_{i-1} \cup \mathcal{D}'$
    $\pi_i = argmin_\pi loss_{supervised}(\pi, \pi^*, D_i)$
    $\pi_{safe,i} = argmin_{\pi_{safe}} loss_{safe}(\pi_{safe}, \pi_i, \pi^*, D_{safe} \cup D_i)$
return π_M and $\pi_{safe,M}$

2.4 Continual learning and catastrophic forgetting

Catastrophic forgetting, also known as catastrophic interference, is a phenomenon witnessed in neural networks when incrementally learning multiple tasks. It is signified by a drastic decrease in the network's performance on previously learned tasks after learning additional tasks [14]. The performance loss occurs since a neural network uses a single set of tunable parameters to learn mappings from input to output. A preceding task's mapping will suffer as the parameters are tuned for a succeeding task's mapping. This issue is generalized by the stability-plasticity dilemma [34]. When learning new tasks, parameters should to be stable enough to retain previous knowledge but also plastic enough to learn new knowledge. Continual learning is about allowing a model to learn new tasks in an incremental fashion without forgetting previously learned tasks. The research area is not standardized and goes by several name besides continual learning, e.g., sequential learning, incremental learning, lifelong learning and continuous learning. Achieving continual learning is key in order to enable more versatile models and lifelong machine learning (LML), which brings mankind one step closer to artificial general intelligence. LML has a system perspective on incremental learning [35] meanwhile continual learning has a more narrow algorithmic perspective. Artificial general intelligence is a machine with the same or higher intelligence and problem solving capabilities as a human.

Subsection 2.4.1 describes areas related to continual learning and subsection 2.4.2 presents a number of desiderata for continual learning.

2.4.1 Areas related to continual learning

Continual learning is related to several other machine learning areas [36]: transfer learning, multi-task learning and online learning. Below, each related area is given a short description and how it relates to continual learning.

Transfer learning is a technique that allows one to create a machine learning model for some task that lacks sufficient labeled data by training the model on a related task that has plenty of labeled data. The idea is that the model will learn a good representation by training on the related task and then the model is fine-tuned to the target, thus knowledge learned from the related task is transferred the target task. In continual learning, there exists multiple tasks and knowledge should be transferred forward and backward between tasks [36]. Forward transfer implies performance on future tasks is improved as the knowledge learned from earlier tasks is beneficial for future tasks. Backward transfer is the opposite, learning an additional task improves the performance on earlier tasks [37].

Multi-task learning interleaves data from all tasks and optimizes the network's parameters for all tasks during training. This technique assumes that all tasks are known and that all data is available. Continual learning learns from each example in an online fashion meanwhile multi-task learning learns in batches [36]. Another difference is that in continual learning, one cannot assume to have access to all previous data.

Online learning is when training data appear one example at a time instead of appearing in batches. The model is trained and updated for each incoming example. This allows the model to adapt to changes in the data distribution but if the distribution changes too much the model will fail on the original data distribution. Continual learning needs to be able to learn from each example, adapt to changes in data distributions yet also maintain performance as the distributions change, over all tasks [36].

2.4.2 Continual learning desiderata

There exists several ways to mitigate the problems of catastrophic forgetting, e.g., regularization, ensembles, retraining and rehearsing, but each option has drawbacks. Regularization approaches can put too much constraints on the optimization, leading to sets of parameters having low or no performance at all on new tasks. Creating an ensemble with an additional model for each task is not scalable as the memory usage will scale with the number of tasks, each model will also have to learn its own representation that can be common among tasks. Retraining a model with the entire old data set interleaved with the new task's data can be very inefficient and expensive due to long training times. As reference, a ResNet-50 model training 90 epochs on ImageNet-1k with a single NVIDIA M40 GPU takes 14 days [38]. Rehearsing on data of already learned tasks is a way to maintain good performance while learning new tasks, the issue is that it is costly since past data proportional to the number of tasks needs to be saved. Retraining and to a lesser extent rehearsing relies on the assumption that past data remains available.

The definition of continual learning is not entirely agreed upon and is currently defined by a non-finished set of desiderata [15]. These desiderata consist of online learning, presence of transfer, resistance to catastrophic forgetting, bounded system size and no direct access to previous experience, see Figure 2.3. The desiderata describe ideal properties of a continual learning algorithm, however relaxations are needed as it may not be possible to achieve all desiderata together. As of this being written, there exists no continual learning algorithm satisfying all of these desiderata. A de facto way of evaluating continual learning is also absent. Below, each desiderata is presented.

Online learning. Learn from every data point in an online fashion. Data sets and tasks are not fixed and tasks lack boundaries.

Presence of transfer. Bidirectional knowledge transfer, i.e., previously learned tasks should improve performance on new tasks and learning new tasks should improve performance on previously learned tasks.

Resistance to catastrophic forgetting. Performance on previously learned tasks should not decrease greatly as new tasks are learned.

Bounded system size. Model capacity should be fixed, i.e., the model cannot expand in order to learn new tasks. This constrains the model to use its capacity well and also means that the model has to forget older tasks gracefully as its capacity is exceeded.

No direct access to previous experience. Access to previous data or rewinding environments is not allowed for continual learning algorithms.

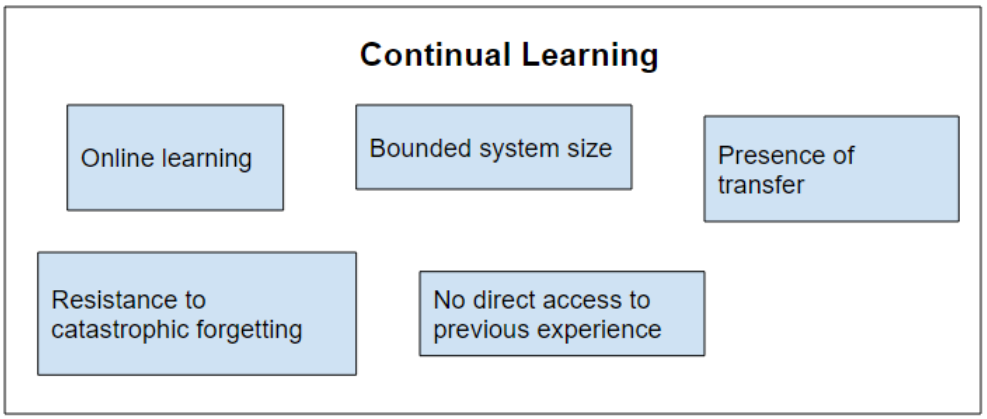


Figure 2.3: Set of continual learning desiderata as defined by the NIPS 2016 workshop. [Created by author]

2.5 Related Work

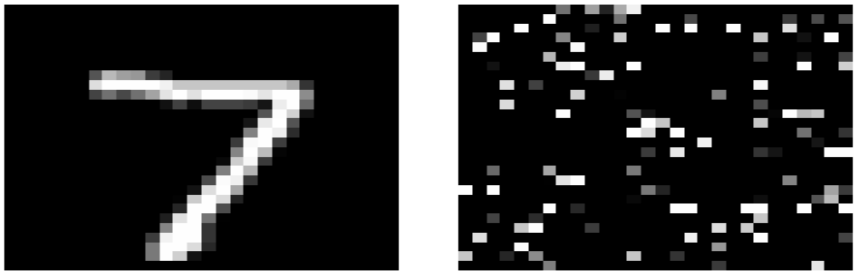
The DAGGER algorithm [12] deals with the issue of compounding errors in sequential predictions by iteratively querying a human expert for more data and retraining the model on past and new data combined. Note that a human expert is required at all times while collecting data, which is expensive. Sequential prediction means that a model's future input depends on its earlier predictions, thus sequential prediction input data does not fulfill the usual i.i.d. assumption. As the input data is not i.i.d. each prediction that is not perfect causes the model to depart slightly from the states visited by the expert and eventually the model may encounter states significantly different from those it was trained on, leading to undefined performance. DAGGER solves this issue by deploying the model, allowing the model to encounter new states based on its predictions while the human expert is simultaneously providing correcting actions, i.e., what the model should do when it encounters each state. The collected states together with the corresponding correcting actions are used as additional training data that is appended to the previous data and the model is retrained. This process repeats until the model performs well enough. The results shows DAGGER outperforming the compared techniques SMILe [33] and SEARN [39] in two imitation learning tasks, Super Tux Cart and Super Mario Bros, and handwriting recognition. Tests show that DAGGER is the

only technique capable of creating a model that never falls drives off the road in Super Tux Cart.
SafeDAGGER [13] is an improvement upon DAGGER that aims to reduce the amount of correcting actions needed from the human expert, thus making the algorithm cheaper as human experts are costly. This reduction is achieved by introducing another predictor, i.e., another machine learning model. It is a safety model that learns to predict whether or not the main model can perform its task well enough, i.e., the main model's prediction is close enough to the ground truth. What is deemed well enough is decided by some threshold chosen by the user. Thus SafeDAGGER can select a subset of all collected data that represents difficult input. These difficult data are used together with old data to retrain a new model as in the original DAGGER algorithm. SafeDAGGER is evaluated via a autonomous driving scenario in TORCS [40]. The results shows SafeDAGGER is reducing the number of actions needed from the human expert, fewer crashes and less damage per driven lap and also that SafeDAGGER trains a good model faster and with less data compared to DAGGER.

Kirkpatrick et al. presented the EWC algorithm [16] to overcome catastrophic forgetting. EWC is a regularization technique that protects tasks' important parameters by reducing their plasticity, thus mainly non-important parameters are optimized during training. EWC relies on there being multiple parameter configurations for neural networks that give good performance [41, 42] and thus it is possible to find a set of parameters for task B, θ_B, where task A's important parameters are fairly unchanged. EWC measures parameters' importance with Fisher information [43] matrices and adds a quadratic penalty on the important parameters for each new task to the overall loss. Equation 2.2 shows EWC's loss function with two tasks, A and B. $\mathcal{L}_B(\theta)$ is the usual loss for some task B and $\sum_i \frac{\lambda}{2} F_i(\theta_i - \theta^*_{A,i})^2$ is the added regularization term that protects parameters that are important for some task A. The hyperparameter λ signifies how important the old task is compared to the new task.

$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \sum_i \frac{\lambda}{2} F_i(\theta_i - \theta^*_{A,i})^2 \tag{2.2}$$

F is the Fisher information matrix, θ_i is network's current parameters and $\theta^*_{A,i}$ the set of good parameters extracted previously by training on task A. The Fisher information matrix is calculated from a set of examples from the previous task. MNIST [44] is a data set of handwritten digits and is a common classification task. An evaluation of shifting input distributions is the permuted MNIST test. Permuted MNIST is the most commonly used scenario



(a) MNIST seven (b) Permuted MNIST seven

Figure 2.4: Visualization of the permuted MNIST test for testing shifts in input distributions. An image of a seven (a) and the same image but the pixels are permuted (b). Both images have the same label. [Created by author]

in continual learning to evaluate shifting input distributions and creates new input distributions by permuting the input image's pixels but keeping the labels unchanged. EWC is evaluated with permuted MNIST, see an example in Figure 2.4, and on multiple Atari games. The results show that a neural network can retain knowledge and perform well on multiple tasks when trained on tasks sequentially. However, the permuted MNIST test is criticized of giving unrealistically good results [17, 18], thus other tests are needed. Another issue is that there is only a finite amount of parameters that can be deemed important the network will eventually fail to learn anything or even start forgetting previous knowledge as the network is saturated and tasks' important parameters will be tuned.

Rebuffi et al. presents incremental classifier and representation learning (iCaRL) [45] for learning tasks incrementally while recording a small set of examples for each class. iCaRL uses these sets to classify new data through nearest-mean-of-exemplars and to reduce catastrophic forgetting through rehearsal. The representation is updated by using a loss function combining classification loss and distillation loss. The results shows that iCaRL performes better than the compared methods and that its accuracy is not biased towards recently learned classes as other methods are [46].

This thesis differs from the related work in the following ways. This thesis differs from DAGGER as it uses EWC to maintain previously learned knowledge while training only on the newly collected data instead of retraining on the union of old and new data. Also, DAGGER is evaluated in Super Tux Kart while this thesis uses the Virtual Battlespace 3 (VBS3) simulator [47]. This

thesis's difference to SafeDAGGER is the same as with DAGGER and also that this thesis uses a human instead of a model to decide when the control model is driving well enough. If the human decided the control model is failing, the human will take over control and data is collected until the human returns the control to the model. This thesis does not alter EWC, but evaluates EWC in the more realistic context of autonomous driving with shifting input distribution instead of the criticized permuted MNIST test. This thesis takes the idea of a rehearsal buffer containing data from earlier tasks to investigate whether it can give a significant performance improvement with EWC.

Chapter 3

Methodology

To achieve valid and reproducible research results it is important to select and plan an appropriate research design. Figure 3.1 summarizes the methods used in this work's research design and described throughout section 3.1.

Section 3.1 presents the chosen research methodology and discusses alternative methods. Section 3.2 describes how the chosen research methodology was applied in practice.

3.1 Choice of Research Method

This section presents the chosen research methods. The research question is restated to help the reader follow the discussion about choosing appropriate research methods. This work's research question is: *can the SafeDAGGER-algorithm be enhanced with the continual learning technique EWC to avoid aggregating new and old data in each iteration and instead allow training models only on new data, yet maintaining the same performance as the ordinary SafeDAGGER?*

Quantitative and qualitative research are the two main types of research. Quantitative research deals with work of numerical character meanwhile qualitative research deals with non-numerical work [23]. This work utilizes the quantitative research method as training and evaluating deep learning models is inherently numerical and provides objective measures whether or not the combination of SafeDAGGER and EWC works. The qualitative approach could be useful for evaluating driving quality of an autonomous vehicle, but that would

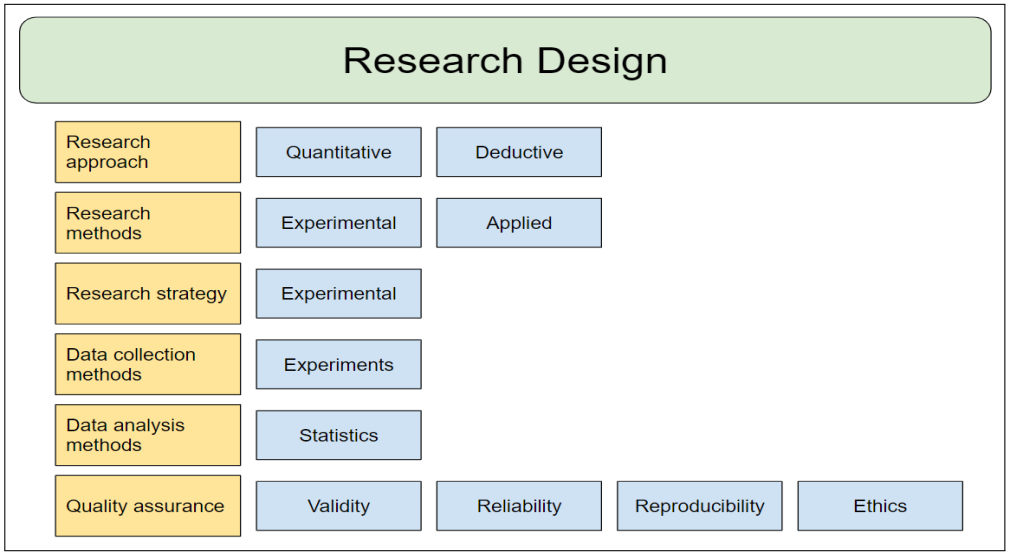


Figure 3.1: Research design used in this work. [Created by author]

be more suitable after establishing that the SafeDAGGER and EWC combination works.

Research methods provides a theoretical framework describing how to conduct research [23]. Possible research methods suitable for this work's research question are experimental, analytical, applied. The experimental research method examines connections between variables by altering one variable while keeping others set to see how the result is affected [23]. The analytical research method tests hypotheses with already existing data and already existing theories [23]. The applied research method is a practical method that solves specific questions or practical problems based on existing theory in order to either solve problems or develop solutions [23]. The experimental research method was chosen as answering the research question required experimentation with hyperparameters and the applied research method was chosen since existing theory, SafeDAGGER and EWC, was used to develop a new algorithm.

Research strategies provides practical guidelines for performing the research [23]. Research strategies suitable for quantitative research are experimental, ex post facto, surveys and case study [23]. The experimental research strategy is about minimizing factors affecting the measurements through well design experiments in order to test hypotheses using huge data sets. The ex post facto research strategy uses already collected data to test hypotheses by searching back in time to find relationships between variables. Surveys are used to find re-

lationships between variables and create information on events that are not directly observed [23]. The case study strategy empirically study real life events by using several sources of evidence. The experimental research strategy was the only strategy that fitted this work as it was a strategy for testing hypotheses with large data sets. The hypothesis tested in this work is whether or not SafeDAGGER's requirement of aggregating data can be removed by using EWC.

Induction and deduction are research approaches that provides structured pathways of how to draw conclusions [23]. The inductive approach proceeds from observations from which patterns are detected and hypotheses formulated and tested, finally resulting in new theory. The deductive approach proceeds from known theory to formulate a hypothesis that is verified or falsified through observations. This thesis applied a deductive approach since the work's research question investigates and tests if the combination of SafeDAGGER and EWC, i.e., known theory, can result in a better algorithm. The inductive approach would be appropriate if the work instead investigated why such an approach would work, using qualitative methods.

Below, methods used for data collection is presented in section 3.1.1, methods for data analysis in section 3.1.2 and quality assurance in section 3.1.3.

3.1.1 Data Collection

Data collection methods suitable for quantitative research are experiments, questionnaires, case study and observations [23]. Experiments gather large data sets. Questionnaires utilizes questions to gather data. Case studies uses few participants but gathers data in-depth. Experiments are used in this work to collect data sets of road images labeled with the corresponding angle of the steering wheel. This data is recorded when a human expert drives a vehicle along roads in the VBS3 simulator.

3.1.2 Data Analysis

The purpose of data analysis methods is to inspect, clean, transform and model data in order to provide a reliable foundation from which conclusions could be drawn and decisions made [23]. Common data analysis methods for quantitative research are statistics and computational mathematics. Statistics calcu-

late results for samples [23] and computational mathematics utilizes numerical methods, modelling and simulations to analyze data [23]. Statistics were chosen as data analysis method since evaluation was based on numerical comparisons and it allowed the performance of the combination of SafeDAGGER and EWC to be compared easily with ordinary SafeDAGGER.

3.1.3 Quality Assurance

Research is about producing new knowledge through the scientific method to ensure the new knowledge is as correct as possible. To ensure the quality of quantitative research with a deductive approach one should discuss validity, reliability, replicability and ethics [23]. Ethics have already been discussed in section 1.5, the other three are discussed below.

Validity refers to measuring what is actually supposed to be measured [23]. Validity was achieved by thoroughly assessing whether the evaluation criteria could properly measure changes in performance when EWC was combined with SafeDAGGER.

Reliability deals with the stability of the tests, i.e., ensuring the test results are consistent and not dependent random factors [23]. Reliablity was ensured by removing or minimizing any variance between tests. Concretely, it was done by opting for determinism wherever possible by assigning seeds to random number generators and by using saved scenarios in the VBS3 simulator, which allowed all tests to have the same initial conditions. The choice of using a human instead of a safety model in SafeDAGGER introduced some subjectivity as to deciding when the human should have taken control of the vehicle. However, as it was the same human doing all experiments, all decisions were kept as similar as possible thus minimizing some uncertainty.

Replicability means that someone else can repeat the research based on the information contained in this work and attain the same results [23]. This was achieved in several steps. Descriptions were detailed and explicit to ensure its possible to reproduce this work without needing to make any guesses or logical leaps. Hyperparameters, seeds and model architecture were provided. The used hardware and software were listed to enable others to use the same hardware and software versions.

3.2 Method Application

Providing the reader with the work's practical methodology is important for reproducibility purposes and also for guaranteeing correctness as the work becomes transparent and traceable. The hardware and software used in this work can be viewed in Appendix B.

Section 3.2.1 presents all things related to the data used in this work, section 3.2.2 explains how training was performed and section 3.2.3 describes how the trained models were evaluated.

The application self-driving vehicles was chosen as a basis to apply the selected methods and execute the work. The VBS3 simulator [47] provided an environment for collecting data, extracting metrics and evaluating trained models. The models were evaluated by deploying them in the VBS3 simulator observing how well they did according to the metrics. The extracted metrics were (1) driven distance until the vehicle left its lane and (2) if the model managed to finish a track or not, see the example in Table 3.1. The first metric showed how far the vehicle could travel, which was easy to compare, and the second metric made it easy to summarize a model's performance. If a model failed to finish the tracks, it was deployed again to collect additional training data. Each time a model failed, a human expert took control of the vehicle and corrected the vehicles trajectory before returning control to the model again. Data was recorded while the vehicle was under human control, thus giving additional data representing difficult input needed for further SafeDAGGER *iterations*. An iteration in SafeDAGGER is the process of collecting additional data, aggregating data and retraining a model. The purpose of each iteration is to produce an improved model.

An initial data set was collected by manually driving a vehicle around the first training track. The initial data set was used to train a base model that had some driving capabilities but not good enough to finish any track. The base model provided a common foundation for the other models to build upon. Four models were trained from the base model, (1) a naive model training only on new data without EWC, (2) a SafeDAGGER model as a baseline, (3) a SafeDAGGER model using EWC called EWC-SD and (4) an EWC-SD model with a rehearsal buffer. The naive model was used to show there existed an actual problem and it gave a lower bound that the other models could be compared against. The second model provided a baseline that the third and fourth model could be compared with to show if EWC-SD and rehearsal worked well.

All models were deployed independently in VBS3 and iteratively collected additional data and were retrained until the models either finished all tracks or had run for ten iterations. Note that the models collected their own data sets during iterations and that they were not trained on the same data, except for the inital data set. The reason for this was that each model's collected data had to reflect its own weaknesses that varied between the different models.

For each iteration, each model was deployed on both training tracks and also the test track in order to evaluate its performance. Additional data was collected from situations where the models failed during deployment. Situations similar to difficult parts of the test track were recorded on the training tracks when the models could finish the training tracks but not the test track. This was done as the models were not allowed to train on the test track.

Table 3.1: Example table of evaluation containing the two metrics, an iteration and three runs. Yes and no labels are color coded to facilitate interpretation

	Training track 1		Training track 2		Test track	
	Distance driven (m)	Training track 1 finished	Distance driven (m)	Training track 2 finished	Distance driven (m)	Test track finished
Iteration 1						
Run 1	1925	yes	687	no	1939	yes
Run 2	1925	yes	531	no	1939	yes
Run 3	1925	yes	1448	yes	795	no

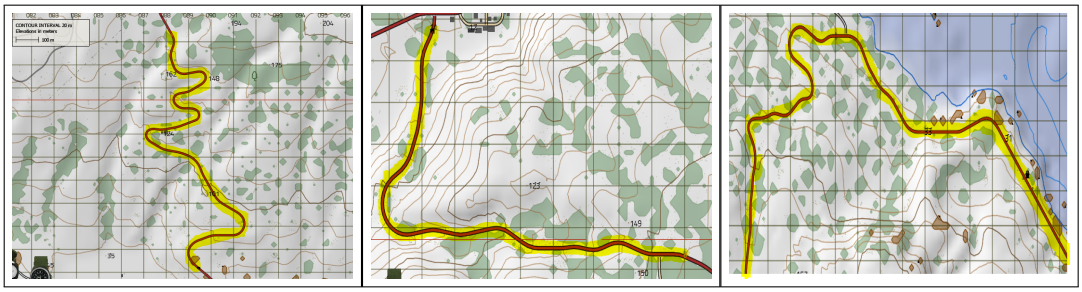
3.2.1 Data

The data was collected by driving a vehicle on roads in the VBS3 simulator with a fixed velocity of 20km/h. The speed limit was imposed in order to enable the human expert to provide good labels. The weather was sunny and the view was unobstructed. All images depicted the same type of roads, paved and marked with white line markings and lighting conditions were similar throughout. Pictures were recorded at a rate of 5Hz with 600x800 resolution and saved with the corresponding angle of the steering wheel. Two training tracks were used and one test track, see Figure 3.2. Training track one was 1925 meters, training track two was 1448 meters and the test track was 1939 meters. Data was only collected from the training tracks as the test track was used solely for evaluation.

The initial data set used to train the base model consisted of 901 labeled training examples collected from the first training track. All other data was collected from either one of the training tracks. The test track was only used for evaluation and data was never collected from it. The number of data collected for each iteration varied between 418 to 491 training examples.

The data collection had two phases in which the data collection method differed, (1) a model failed on one or both training tracks and (2) a model finished both training tracks but not the test track. In the first phase, a failing model was deployed on the track it failed and a human expert gave correcting actions, which were recorded, as the model performed badly. The second phase needed an alternate way of collecting data as the models in this phase could complete the training tracks, thus never needing correcting actions. Thus, the second phase recorded data from environments in the training tracks that were very similar to failing environments in the test track. The recorded data in both phases represented difficult input as the models failed in those situations. Figure 3.3 shows the difference between a normal state and a difficult state. Note that the difficult state was difficult because the model encountered a state that it had barely or not at all been trained on since it was a position where the vehicle was drifting into the other lane, something the human expert would never do.

Collected data was pre-processed in two steps in order to remove redundant information and reduce training time. The collected images were cropped to remove unnecessary sections and then downsampled to a lower resolution. As the road was the only important section in each image, cropping away every-



(a) Training track 1 (b) Training track 2 (c) Test track

Figure 3.2: Bird's eye view of the training set tracks (a) & (b) and the test set track (c). Roads are the red lines and used roads are highlighted in yellow. [Created by author]

(a) Normal image (b) Hard image

Figure 3.3: Image from initial training set (a) and image captured through SafeDAGGER denoting a difficult state for the model (b). [Created by author]

thing slightly above the horizon created images without redundant information. Downsampling the pictures to a lower resolution of 66x200 pixels further decreased the computational cost of training on the data set. Figure 3.4 shows each step of the data pre-processing.

(a) Before cropping, 600x800 pixels (b) After cropping, 300x800 pixels

(c) After downsampling, 66x200 pixels

Figure 3.4: Before (a) and after (b) cropping is applied to a picture and then (c) the cropped picture is downsampled to a lower resolution. [Created by author]

3.2.2 Training

MSE was used as loss function since it is a commonly used loss function for regression problems. MSE computed the squared difference between predicted steering wheel angle and the recorded ground truth.

EWC uses task boundaries to do two things, (1) compute a Fisher information matrix for the previous task and (2) copying the network's parameters for the previous task. The Fisher information matrix captured parameters' importance for a specific task and by saving the network's parameters after training it was possible to see how much the model's parameters deviated from the saved ones after training on a new task. EWC uses the deviation from a task's saved parameters and their importance to constrain the learning of new tasks to find a set of parameters similar to previous tasks' sets of saved parameters. This work utilized each iteration of SafeDAGGER as a task boundary. In order to calculate the Fisher information matrix it was necessary to keep a set of training examples from the previous task. This work kept a set of 80 training examples for each iteration in order to compute the Fisher information matrix, these examples were however not used in training. Using 80 examples were chosen by experimenting with different amounts. In this work, a new task meant a shift in the input distribution. Thus each additional Fisher matrix and set of saved parameters further constrained the training procedure to find a set of parameters suitable for all input distributions. The intuition of EWC's effect was similar to a Venn diagram where each Fisher matrix and set of saved parameters created a circle and the set of parameters that minimized the loss function was the intersection, see Figure 3.5.

Each model used the base model as a foundation and then they were trained in different ways. The naive model was trained only on new data but did not use EWC to protect against catastrophic forgetting. The SafeDAGGER model was trained on all data, both new and old. The EWC-SD model was only trained on the newly collected data in each iteration and used EWC to protect against catastrophic forgetting. The EWC-SD model used a rehearsal buffer and was trained on the newly collected data and the contents of the rehearsal buffer. The rehearsal buffer was updated between each iteration and implemented as a reservoir sample [48] of size 23. The buffer's size was chosen by experimenting with different sizes. A reservoir sample of size s simply stored the first s examples and then when the n*:th* example arrived it was stored in the sample with probability s/n, thus each encountered example had the probability s/n of being in the sample. When the sample was full and a new example was added

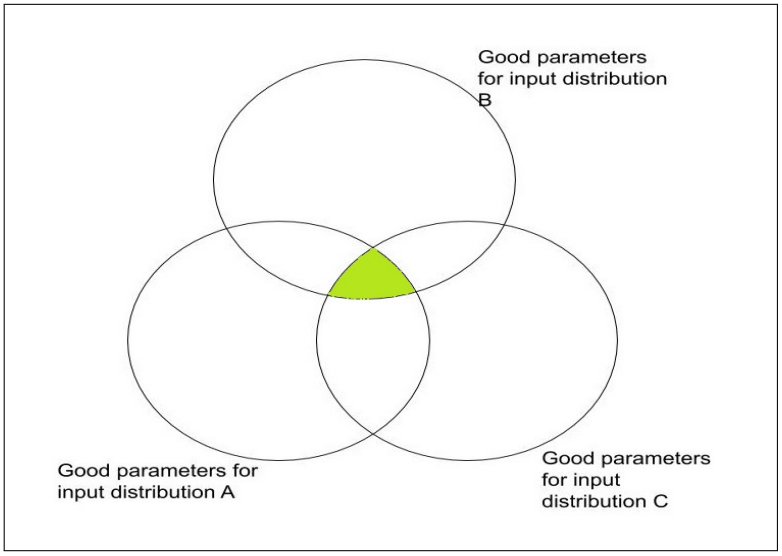


Figure 3.5: Intuition behind EWC's imposed constraints. Each additional task in EWC adds a constraint to the loss function. A constraint is signified by a circle in the picture. A set of parameters providing a low loss and good performance for all constraints is found at the intersection of all circles, marked in green. [Created by author]

to it, an already saved example was evicted chosen uniformly at random.

Automatic hyperparameter search was shown to be impossible since a model's test error did not reveal if it performed better or worse to another model with a higher test error. Each model had to be deployed in the simulator where its driving performance could be visually inspected and also distance driven measured. This also meant that early stopping [24] could not be used to prevent overfitting. Chosen hyperparameter values, see Appendix A, were selected after manually searching among different values and finding values giving decent performance.

Adding dropout layers [24] to the model reduced the risk of overfitting. This risk was further reduced by also using L2-regularization [24]. The models were deemed to not have overfitted the training data if they could perform on the test track and if the training MSE and validation MSE were fairly similar, a significantly lower training MSE compared to the validation MSE would indicate overfitting.

3.2.3 Evaluation

Each model was deployed and run thrice per track in VBS3 to evaluate its performance. Multiple runs were necessary as the results varied slightly between runs. Three runs per track were chosen as a trade off between statistical validity and time cost. The models were, for each iteration, evaluated on driven distance measured in meters and whether they finished the training tracks and test track. A model was allowed to drive until it either reached the target, went off road or veered into the opposite lane. The first two conditions were checked via the VBS3 simulator's API and a human checked the third condition as the API lacked support for it.

Driven distance was chosen because it is numerical and straight forward to compare between iterations. Distance also measures how well the vehicle drives as the speed is constant, a vehicle moving in a zig-zag pattern will have to travel further than a model that drives straight. By checking whether or not the models finished the training tracks it was possible to see if they monotonically improved or if they regressed in performance, e.g., a model manages to finish the first training track during some iteration but fails on the same track during the next iteration. Measuring MSE allowed comparisons to see if MSE was representative of a models performance or not in this context. Presenting the amount of data collected for each model for each iteration made it possible to see that there were no major differences in the number of collected training examples leading to unfair advantages.

Chapter 4

Results

This chapter presents the altered and improved version of the SafeDAGGER algorithm, called EWC-SD, in section 4.1. EWC-SD is an acronym for elastic weight consolidation safe dataset aggregation. Section 4.2 presents the empirical data from experiments conducted with variants of SafeDAGGER.

4.1 EWC-SD

EWC-SD is similar to SafeDAGGER and uses the idea of a safety model from SafeDAGGER, but in the form of a human. An initial data set is collected to train an initial model, see line 1 & 2 in Algorithm 3. The for-loop in the algorithm, at line 5, is the iterative phase of deploying the model to collect additional data where it fails, line 6, and then retraining the model to make it succeed in states where it previously failed, line 7. A Fisher information matrix that shows parameter importance for a task is calculated from a set of 80 training examples saved from the previous iteration, see line 8. A model's parameters are saved after training and used in the EWC constraint during the suceeding training iterations. Both Fisher information matrices and saved parameters are stored in lists and the results on lines 8-9 are appended to the corresponding lists. EWC is used when training and only data D_i from the current iteration is used for training instead of all data, see line 7. This is a significant difference from both DAGGER and SafeDAGGER.

Algorithm 3 EWC-SD

Collect $\mathcal{D}_0$ using a human expert
$\pi_0 = argmin_\pi loss(D_0)$
$F =$ Calculate Fisher information matrix from $\mathcal{D}_0$
$\theta^* =$ parameters of π_0
for i=1...M **do**
    $D_i =$ Collect difficult data using using π_{i-1} and human intervention
    $\pi_i = argmin_\pi EWC\ loss(\pi_{i-1}, D_i, F, \theta^*)$
    $F =$ Calculate Fisher information matrix with data saved from D_{i-1}. Append result to list of Fisher matrices
    $\theta^* =$ Append parameters of π_i to list of parameters
return π_M

These changes are motivated by the fact that EWC can allow a model to learn shifts in input distributions yet remain performant on previous input distributions. Imperfect predictions causes a model to drift from the human expert's trajectory, resulting in input that the models have not been trained on. This new input may consists of images where the vehicle is drifting out of the lane, e.g., being very close to the road's center line or being very close to going off road. Such input represents a shift from the human expert's trajectory that runs along the center of the lane. Collecting data that is not similar to the human expert's usual trajectory represents an altered input distribution. Thus, applying EWC to SafeDAGGER is suitable to remove the need of saving previous data, lowering training times and memory requirements.

4.2 Empirical Results

All models originate from the same model, called the base model. The base model is trained on some initial data that allows it to drive for 200 to 800 meters depending on the track it is driving on.

The naive approach that trains only on new data without using EWC cannot retain its performance, see Table 4.1. The naive approach does improve from the base model in the first iteration, however it catastrophically forgets earlier knowledge in the following iterations. An example of this is that the model is trained on left turns as it is a weakness during iteration two but it causes the model to mostly steer left in the following iteration. Thus the model fails to stay on the road and quickly ends the test. This can be seen in iteration two, three

Table 4.1: Empirical results of naive model

	Training track 1		Training track 2		Test track	
	Distance driven (m)	Training track 1 finished	Distance driven (m)	Training track 2 finished	Distance driven (m)	Test track finished
Iteration 1						
Run 1	1927	yes	849.8	no	1462.4	no
Run 2	1926	yes	384.3	no	1458.7	no
Run 3	1929.1	yes	834	no	1455.8	no
Iteration 2						
Run 1	8.8	no	11	no	12	no
Run 2	8.7	no	11	no	11.9	no
Run 3	8.8	no	11	no	12	no
Iteration 3						
Run 1	16.8	no	308.2	no	181.3	no
Run 2	14.5	no	832.7	no	181.7	no
Run 3	15.6	no	823.8	no	181.6	no
Iteration 4						
Run 1	9.3	no	12.1	no	13.4	no
Run 2	9.3	no	12.1	no	13.4	no
Run 3	9.3	no	12.1	no	13.2	no

and four as the model, almost instantly, drives off the road. The naive model is only tested for four iterations as it is clear that it does not improve.

SafeDAGGER, which is trained on all data, improves its performance in each iteration except for the third iteration, see Table 4.2. In the third iteration, the model fails after 204 meters on the test track which corresponds to a sharp right turn. In the next iteration the model becomes perfect and manages to finish all tracks. These results highlight that models trained with SafeDAGGER do not improve monotonically as iteration two performs better than iteration three.

EWC-SD, which is trained only on new data with EWC, shows that it can resist catastrophic forgetting and improve its performance as the number of iterations increase, see Table 4.3. However, EWC-SD requires ten iterations before it can complete the test track and even then it only manages to complete the test track in two out of three runs. As comparison, SafeDAGGER only requires four iterations before it completes all the test track in all runs and SafeDAGGER even

manages to complete the test track in one run during iteration two. One thing to note is that EWC-SD seems to catastrophically forget between iterations one and two as its performance severely degrades, however the model regains its performance in the following iterations.

EWC-SD with rehearsal, which injects 23 earlier examples into the training data, has the best performance, see Table 4.4. It manages to finish the test track in all three runs in iteration two. That is two iterations less than the baseline SafeDAGGER and eight iterations less than EWC-SD. This shows is a significant improvement over SafeDAGGER and also that rehearsal is an extremely useful technique with low cost as 23 examples corresponds to approximately 5% of the training data used per iteration.

Table 4.2: Empirical results of SafeDAGGER

	Training track 1		Training track 2		Test track	
	Distance driven (m)	Training track 1 finished	Distance driven (m)	Training track 2 finished	Distance driven (m)	Test track finished
Iteration 1						
Run 1	1929.4	yes	1448.4	yes	502.7	no
Run 2	1929	yes	1448.1	yes	1888.1	no
Run 3	1930.2	yes	1448.1	yes	1472.9	no
Iteration 2						
Run 1	1927.1	yes	1449.6	yes	1552.6	no
Run 2	1926.6	yes	1447.1	yes	1941.9	yes
Run 3	1927.8	yes	1449.6	yes	621.3	no
Iteration 3						
Run 1	1927.3	yes	1448.1	yes	204.4	no
Run 2	1929.8	yes	1449.6	yes	204.5	no
Run 3	1928.4	yes	1449.2	yes	204.4	no
Iteration 4						
Run 1	1928.4	yes	1446.5	yes	1941.6	yes
Run 2	1927.1	yes	1447.5	yes	1940.9	yes
Run 3	1926.8	yes	1448	yes	1939.9	yes

By cross referencing iterations two and four in SafeDAGGER's MSE Table 4.5 and SafeDAGGER's performance in Table 4.2 it is possible to see that a lower MSE does not always correspond to better performance. SafeDAGGER is better in iteration four than iteration two but it's MSE in iteration four is 0.00296 which is higher than the MSE of iteration two which is 0.00237. Thus a higher MSE does not always indicate better performance in this application.

As each algorithm needs to be trained on data relevant to their individual weaknesses, their training data is not identical. The amount of training examples used in each iteration is between 418 to 491. The mean number of examples used to train each algorithm is fairly similar, the largest difference is between EWC-SD and the rehearsal approach with a mean difference of 13.6 training examples per iteration. Table 4.6 shows the number of training examples used for each algorithm and iteration as well as mean number of training examples per algorithm.

To summarize the performance of each algorithm, the test track completion per iteration is calculated as the three run mean for each algorithm and presented in Figure 4.1. Each algorithm is included in the Figure until the algorithm manages to finish the test track during all three runs. SafeDAGGER reaches 100% after four iterations, the rehearsal approach reaches 100% after two iterations and EWC-SD only reaches 87% test track completion as it failed one of its runs during the tenth iteration, which is the last iteration. Figure 4.1 also reveals that EWC-SD's performance seems to oscillate up and down between iterations four to nine.

Table 4.3: Empirical results of EWC-SD

	Training track 1		Training track 2		Test track	
	Distance driven (m)	Training track 1 finished	Distance driven (m)	Training track 2 finished	Distance driven (m)	Test track finished
Iteration 1						
Run 1	500.6	no	1448.3	yes	619.8	no
Run 2	501	no	1446.1	yes	620.3	no
Run 3	501.7	no	1446.9	yes	1259.6	no
Iteration 2						
Run 1	38	no	49.7	no	192.4	no
Run 2	37.9	no	49.5	no	106.8	no
Run 3	37.7	no	51.5	no	183.7	no
Iteration 3						
Run 1	312.4	no	1449	yes	186.1	no
Run 2	310.6	no	1448.3	yes	185.8	no
Run 3	308.4	no	1448.9	yes	177	no
Iteration 4						
Run 1	501.3	no	1448.8	yes	195.3	no
Run 2	501.6	no	1447.4	yes	187.8	no
Run 3	326	no	1447.9	yes	195.2	no
Iteration 5						
Run 1	490.2	no	1447.3	yes	487.6	no
Run 2	487.2	no	1448.4	yes	494.4	no
Run 3	486.6	no	1446.5	yes	493.9	no
Iteration 6						
Run 1	329.5	no	866.5	no	220.6	no
Run 2	328.8	no	833.4	no	219.6	no
Run 3	329.2	no	834.7	no	214.1	no
Iteration 7						
Run 1	1923.8	yes	1447	yes	602.8	no
Run 2	1925	yes	1445.6	yes	602.3	no
Run 3	1924.2	yes	1445.6	yes	602.8	no
Iteration 8						
Run 1	675.7	no	875.3	no	204.9	no
Run 2	322.2	no	844.4	no	196.8	no
Run 3	322.9	no	845.3	no	203.4	no
Iteration 9						
Run 1	1924.4	yes	1444.8	yes	617.3	no
Run 2	1924	yes	1447.2	yes	607	no
Run 3	1925.7	yes	1445.1	yes	1372.9	no
Iteration 10						
Run 1	1923.9	yes	1446.8	yes	1938	yes
Run 2	1925.9	yes	1446.7	yes	1174.2	no
Run 3	1923.6	yes	1447.9	yes	1939	yes

Table 4.4: Empirical results of EWC-SD with a rehearsal buffer

	Training track 1		Training track 2		Test track	
	Distance driven (m)	Training track 1 finished	Distance driven (m)	Training track 2 finished	Distance driven (m)	Test track finished
Iteration 1						
Run 1	1929.2	yes	834.4	no	780.9	no
Run 2	1929	yes	834.2	no	1175.2	no
Run 3	1928.9	yes	832.9	no	1176.9	no
Iteration 2						
Run 1	1928.7	yes	1445.6	yes	1939.9	yes
Run 2	1926.3	yes	1445.5	yes	1939.1	yes
Run 3	1927.2	yes	1445.5	yes	1938.1	yes

Table 4.5: Mean squared error of the different algorithms during each iteration. Three of the algorithms have two MSE values that are written as validation error / previous task's data validation error

Mean squared error per iteration				
	SafeDagger	EWC-SD	EWC-SD with rehearsal	Naive
Iteration 1	0.00469	0.00698 / 0.00376	0.00567 / 0.0034	0.00496 / 0.00436
Iteration 2	0.00237	0.00158 / 0.02136	0.01042 / 0.00444	0.01523 / 0.04225
Iteration 3	0.00247	0.00464 / 0.00175	N/A	0.00654 / 0.0761
Iteration 4	0.00296	0.00534 / 0.0046	N/A	0.00684 / 0.11328
Iteration 5	N/A	0.0116 / 0.00596	N/A	N/A
Iteration 6	N/A	0.01735 / 0.01632	N/A	N/A
Iteration 7	N/A	0.00655 / 0.02994	N/A	N/A
Iteration 8	N/A	0.00855 / 0.02227	N/A	N/A
Iteration 9	N/A	0.00447 / 0.01569	N/A	N/A
Iteration 10	N/A	0.00935 / 0.01101	N/A	N/A

Table 4.6: Number of samples collected in each iteration for the the tested algorithms

Number of samples collected				
	SafeDagger	EWC-SD	EWC-SD with rehearsal	Naive
Iteration 1	458	458	458	458
Iteration 2	455	422	451	455
Iteration 3	430	491	N/A	423
Iteration 4	N/A	460	N/A	457
Iteration 5	N/A	427	N/A	N/A
Iteration 6	N/A	435	N/A	N/A
Iteration 7	N/A	453	N/A	N/A
Iteration 8	N/A	419	N/A	N/A
Iteration 9	N/A	426	N/A	N/A
Iteration 10	N/A	418	N/A	N/A
Mean	447,6	440,9	454,5	448,3

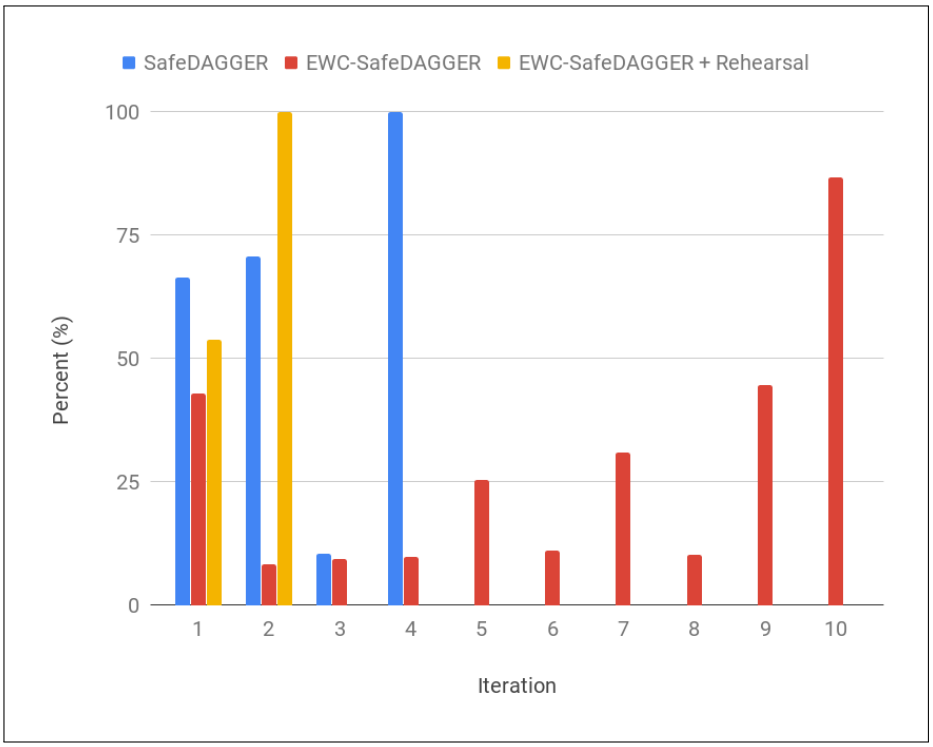


Figure 4.1: Performance summary of the different approaches. Showing mean test track completion per iteration. [Created by author]

Chapter 5

Discussion

The empirical results shows that the naive approach is not viable and that SafeDAGGER or EWC-SD is needed to not forget previously learned knowledge. EWC-SD does not achieve the same performance as SafeDAGGER, but it does improve over time and given more iterations it might reach performance equal to SafeDAGGER. Adding rehearsal to EWC-SD significantly improves its performance and it even produces a very good model in fewer iterations than the ordinary SafeDAGGER algorithm. This result is very interesting as it might be a definitive improvement that can replace SafeDAGGER, but additional research is needed to verify such a claim as there are multiple sources of uncertainty that may have affected the outcome. The sources of uncertainty include that the models are trained on different data, slightly different data amounts and human error during data collection.

Adding a rehearsal buffer to EWC-SD using reservoir sampling provides a significant improvement at a very low cost. The buffer size is roughly five percent of the number of training examples used for each iteration, i.e., 23 training examples. This simple addition allowed EWC-SD to achieve excellent performance already after two iterations, which is half of what SafeDAGGER requires. Thus, rehearsal buffers are an simple and cheap way of greatly increasing performance. Interestingly, another study at FOI [49] also noted that rehearsing on 5% of old data together with EWC gave an significant increase in performance. Questions regarding optimal buffer eviction policies, e.g., first in first out or random, and buffer sizes are not answered in this work and are suitable for examination in future work.

As seen in Table 4.5, a lower MSE does not always indicate a better performing model. This causes a range of issues such as not being able to use early stopping to prevent overfitting and not being able to perform automatic hyperparameter search. A theory is that the discrepancy is caused due to data not being i.i.d. which affects the input distribution when a model is deployed in the VBS3 simulator, but not when calculating the validation MSE during a model's training. Slight mispredictions of the steering angle compound while driving in VBS3 which cause the models to slightly deviate from the human expert's trajectory and eventually encounter input that they have not been trained on. An example is a model trained on straight roads that is driving a vehicle along a straight road and compounding errors causes it to drift to the left. Eventually the model will drive along the left edge of the road, which it has not been trained on, and thus its actions are unpredictable. If a model with a low MSE makes a slight misprediction at the wrong time, e.g., in a sharp curve, it may be possible that it veers off the human expert's trajectory and fails meanwhile another model with a higher MSE might not make the same misprediction in the sharp curve and thus manages to drive further. In short, the MSE inconcistency might be due to non i.i.d. data and inconvenient mispredictions at critical moments that cause models to encounter input that they have not been trained on.

As each algorithm requires data tailored to its specific weaknesses the different algorithms are trained on different data sets. This is an issue as comparing algorithms trained on different data is not truly fair, however it is needed in this case. To mitigate this issue data was collected in the same way for each algorithm to ensure that no algorithm received dissimilar data. The amount of data used is low and should be larger. Mean number of training examples, see Figure 4.6, are close to each other and is deemed to not have a significant effect on the results but the number of training examples should have been truncated to make all data sets identical in size.

The evaluation metrics are decent, but they could be improved. The improvement lies in that the metrics do not reveal a model's performance in all cases. The metrics only reveal a model's performance up until the model's first failure. As such, imagine a scenario where a model is able to complete the first 99% of a track but goes off track when it encounters that last 1%. The model would be deemed to perform well. However if the scenario is reversed and the model fails right at the start it will be deemed worthless while it can actually complete the rest of the track by itself. This is the problem of only evaluating up until the first failure, instead some other metric should be used. A propo-

sition is to use the number of human interventions to keep the vehicle in its lane instead of measuring whether or not the vehicle finished the track. The number of interventions gives the same information as measuring whether or not the vehicle completes a track and it also allows one to evaluate the vehicles performance on the whole track. Counting human interventions should have been used in this work as several of the models failed early but could drive quite far on their own beyond the initial failing point, thus making the models seem worse than they really were. However, this metric was not thought of until after the evaluation was performed.

The chosen research methods have worked very well for this work and it is hard to see benefits of using other research methods. However with the applied methodology, the number of training examples should have been the same for all models over all iterations. Even though the difference in number of training examples between the algorithms is small it is still a source of unevenness and should be eliminated. Moreover, additional evaluation runs should have been used to provide results with higher statistical guarantees. As the results could vary between runs it is not guaranteed that the models that managed to finish the tracks in all three runs would finish the tracks if ten runs were evaluated. Additional runs would identify such situations and produce more reliable results. However, three runs were deemed to be good enough and a decent trade off given that the increased time cost of using more runs.

As SafeDAGGER's safety model was replaced with a human to determine when a model required correcting actions, there is concerns for potential bias and validity issues. The human giving correcting actions knew which model was driving and could thus provide better correcting actions to the some model, e.g., EWC-SD, in order to improve its results. However, such bias did not occur and the human supplied correcting actions were as consistent as possible over all models in order to provide valid and authentic results. Another issue is whether or not it is appropriate to replace SafeDAGGER's safety model with a human. It was deemed okay to use a human as safety model in this work as it was easy to see when a model deviated from the desired driving pattern in VBS3. However, this work should be reproduced with a proper safety model to ensure its validity.

During the work many problems were encountered and worked around. Aside from practical issues with hardware and drivers, a major issue was the problem of a lower MSE not guaranteeing a model to perform better. This constrained the work to use manual hyperparameter search that took a lot of time since each model had to be evaluated by being deployed in the VBS3 simulator and

waiting for it to drive in real-time to monitor how far the model was able to drive. Even though this is a problem it also means that there exists potential for improvement as it is highly unlikely that the best hyperparameters were found.

The delimitations are appropriate. This work is a proof of concept that indicates that it is possible to train models only on the most recent data when using EWC together with SafeDAGGER. Delimiting this work to only use EWC is appropriate since including additional continual learning algorithms quickly makes the work significantly larger. By using only EWC, this work shows that it is possible to improve SafeDAGGER through continual learning and shows that it may be worth to continue this work with improved versions of EWC such as online EWC [50] or EWC++ [51].

Chapter 6

Conclusions and Future Work

This thesis presents an improvement upon the SafeDAGGER algorithm that is used to mitigate compounding errors in imitation learning. The improvement is called EWC-SD and is a flexible alternative that uses EWC to train only on new data instead of retraining on both new and old data combined. This allows EWC-SD to scale with larger amount of data as previous data does not need to be stored, lowers training time and allows extending trained model's when old data is not available any more. Another contribution of this work is showing that EWC can handle shifting input distributions in the context of autonomous vehicles, a more real task than the commonly used and criticized permuted MNIST test.

The results show that EWC-SD works but it takes longer than SafeDAGGER to reach good performance. However, by adding a small rehearsal buffer containing only 23 training examples allows EWC-SD to reach excellent performance in half as many iterations as SafeDAGGER. The conclusion is that EWC-SD does not achieve the same performance as SafeDAGGER but EWC-SD with rehearsal does. EWC-SD with rehearsal solves the problems associated with aggregating data in SafeDAGGER, i.e., increasing training times, memory requirements and maintaining access to all previous data.

The purpose of this thesis is to investigate whether SafeDAGGER can be made more scalable in terms of memory and training time by utilizing EWC to train only on new data instead of all aggregated data. The purpose is fulfilled as the results show EWC-SD being able to learn and improve over time when only training on new data. Memory efficiency is achieved since old data can be discarded, thus allowing low memory systems to train models on data sets

that are otherwise too large. When using EWC-SD a Fisher information matrix must be calculated, which takes some time. Thus the aggregated data used in SafeDAGGER must reach a certain size before EWC-SD reduces the training time.

The goal of creating a more scalable version of SafeDAGGER that allows previously unfeasible applications has been fully fulfilled. EWC-SD provides a more scalable version of SafeDAGGER, however it requires more training before reaching the same performance as SafeDAGGER. Adding rehearsal to EWC-SD greatly reduces the amount of training needed and makes EWC-SD outperform SafeDAGGER. Thus EWC-SD can be used in applications where it is unfeasible to store and aggregate data as EWC-SD lacks the need to store all data.

The research question asks whether or not SafeDAGGER can be enhanced with EWC to train only on new data yet maintain the performance given by training on all data. EWC-SD does not maintain the same performance as SafeDAGGER, however by adding rehearsal EWC-SD performs even better than SafeDAGGER.

Aggregating data iteratively is a problem as it increases memory requirements and model training time. Applying EWC to SafeDAGGER shows that it is possible to train only on new data and that aggregating data may not be neccessary, thus relieving the problem. However, this approach needs further research to validate these results.

Areas of future work include solving the MSE discrepancy problem, evaluating additional continual learning techniques, finding the optimal rehearsal buffer type and size and reproducing of this work using hyperparameter search and larger data sets.

Solving the MSE discrepancy problem will enable early stopping and automatic hyperparameter search. Both techniques are key to achieving better results.

Trying improved versions of EWC such as EWC++ [51] or online EWC [50] is interesting since they both perform better than EWC and does not require storing multiple Fisher matrices and variables, thus further lowering the memory requirements.

This work uses a rehearsal buffer implemented as a reservoir sample. This may not be the optimal type of buffer and other variants should be researched. Buffer sizes are also an area where further research is needed to find the opti-

mal trade off between increasing performance and additional memory cost of keeping previous data.
Reproducing this work with larger data sets and with hyperparameter search will validate the idea of applying EWC to SafeDAGGER and might also provide better results due to better hyperparameters.

Bibliography

[1] *Google Trends*. Accessed: 2019-01-22. Google. URL: https://trends.google.com/trends/explore?date=2013-01-01%202019-01-01&q=Deep%20learning,%2Fm%2F0h1fn8h.

[2] Åse Dragland. *Big Data – for better or worse*. Accessed: 2019-01-21. SINTEF. URL: https://www.sintef.no/en/latest-news/big-data-for-better-or-worse/.

[3] Bernard Marr. *How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read*. Accessed: 2019-01-21. Forbes. URL: https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/#6b9a2e9260ba.

[4] Yonghui Wu et al. "Google's neural machine translation system: Bridging the gap between human and machine translation". In: *arXiv preprint arXiv:1609.08144* (2016).

[5] Gideon Lewis-Kraus. *The Great A.I. Awakening*. Accessed: 2019-01-22. The New York Times Magazine. URL: https://www.nytimes.com/2016/12/14/magazine/the-great-ai-awakening.html.

[6] Kaiming He et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[7] Yun Liu et al. "Detecting cancer metastases on gigapixel pathology images". In: *arXiv preprint arXiv:1703.02442* (2017).

[8] Ivan Bratko, Tanja Urbančič, and Claude Sammut. "Behavioural cloning: phenomena, results and problems". In: *IFAC Proceedings Volumes* 28.21 (1995), pp. 143–149.

[9] Zhilu Chen and Xinming Huang. "End-to-end learning for lane keeping of self-driving cars". In: *Intelligent Vehicles Symposium (IV), 2017 IEEE*. IEEE. 2017, pp. 1856–1860.

[10] *FOI | About FOI*. Accessed: 2019-02-11. FOI. URL: https://www.foi.se/en/foi/about-foi.html.

[11] Hal Daumé III. *A Course In Machine Learning*. Second. Page 212. Self-Published, Jan. 2017. URL: http://ciml.info/dl/v0_99/ciml-v0_99-all.pdf.

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

[13] Jiakai Zhang and Kyunghyun Cho. "Query-efficient imitation learning for end-to-end autonomous driving". In: *arXiv preprint arXiv:1605.06450* (2016).

[14] Michael McCloskey and Neal J Cohen. "Catastrophic interference in connectionist networks: The sequential learning problem". In: *Psychology of learning and motivation*. Vol. 24. Elsevier, 1989, pp. 109–165.

[15] *NeurIPS | 2018. Continual Learning Workshop*. Accessed: 2019-01-30. NIPS. URL: https://nips.cc/Conferences/2018/Schedule?showEvent=10910.

[16] James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the National Academy of Sciences* 114.13 (2017), pp. 3521–3526.

[17] Ronald Kemker et al. "Measuring catastrophic forgetting in neural networks". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[18] Sebastian Farquhar and Yarin Gal. "Towards robust evaluations of continual learning". In: *arXiv preprint arXiv:1805.09733* (2018).

[19] *IEEE Code of Ethics*. Accessed: 2019-01-21. IEEE. URL: https://www.ieee.org/about/corporate/governance/p7-8.html.

[20] Ljubica Nedelkoska and Glenda Quintini. "Automation, skills use and training". In: 202 (2018). DOI: https://doi.org/https://doi.org/10.1787/2e2f4eea-en. URL: https://www.oecd-ilibrary.org/content/paper/2e2f4eea-en.

[21] Judith Eleanor Herrin, Michael Frassetto. "History of Europe". In: *Encyclopaedia Britannica* (Dec. 2018). Accessed: 2019-01-23. URL: https:

//www.britannica.com/topic/history-of-Europe/Social-upheaval.

[22] *Goal 13: Take urgent action to combat climate change and its impacts*. Accessed: 2019-01-23. UN. URL: https://www.un.org/sustainabledevelopment/climate-change-2/.

[23] Anne Håkansson. "Portal of research methods and methodologies for research projects and degree projects". In: *The 2013 World Congress in Computer Science, Computer Engineering, and Applied Computing WORLDCOMP 2013; Las Vegas, Nevada, USA, 22-25 July*. CSREA Press USA. 2013, pp. 67–73.

[24] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[25] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. " O'Reilly Media, Inc.", 2017.

[26] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[27] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[28] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[29] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[30] Dean A Pomerleau. "Alvinn: An autonomous land vehicle in a neural network". In: *Advances in neural information processing systems*. 1989, pp. 305–313.

[31] Alessandro Giusti et al. "A machine learning approach to visual perception of forest trails for mobile robots". In: *IEEE Robotics and Automation Letters* 1.2 (2016), pp. 661–667.

[32] Andrew Isaac and Claude Sammut. "Goal-directed learning to fly". In: *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. 2003, pp. 258–265.

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

[34] Martial Mermillod, Aurélia Bugaiska, and Patrick Bonin. "The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects". In: *Frontiers in psychology* 4 (2013), p. 504.

[35] Daniel L Silver, Qiang Yang, and Lianghao Li. "Lifelong machine learning systems: Beyond learning algorithms". In: *2013 AAAI spring symposium series*. 2013.

[36] Zhiyuan Chen and Bing Liu. "Lifelong machine learning". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 10.3 (2016).

[37] David Lopez-Paz et al. "Gradient episodic memory for continual learning". In: *Advances in Neural Information Processing Systems*. 2017, pp. 6467–6476.

[38] Yang You et al. "ImageNet Training in Minutes". In: *Proceedings of the 47th International Conference on Parallel Processing*. ICPP 2018. Eugene, OR, USA: ACM, 2018, 1:1–1:10. ISBN: 978-1-4503-6510-9. DOI: 10.1145/3225058.3225069. URL: http://doi.acm.org/10.1145/3225058.3225069.

[39] Hal Daumé, John Langford, and Daniel Marcu. "Search-based structured prediction". In: *Machine learning* 75.3 (2009), pp. 297–325.

[40] *The Open Racing Car Simulator*. Accessed: 2019-04-05. TORCS. URL: http://torcs.sourceforge.net/.

[41] Robert H. Nielsen. "Theory of the backpropagation neural network". In: *Proceedings of the International Joint Conference on Neural Networks* volume I (1989), pages 593–605.

[42] Héctor J Sussmann. "Uniqueness of the weights for minimal feedforward nets with a given input-output map". In: *Neural networks* 5.4 (1992), pp. 589–593.

[43] Alexander Ly et al. "A tutorial on Fisher information". In: *Journal of Mathematical Psychology* 80 (2017), pp. 40–55.

[44] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[45] Sylvestre-Alvise Rebuffi et al. "icarl: Incremental classifier and representation learning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 2001–2010.

[46] Zhizhong Li and Derek Hoiem. "Learning without forgetting". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.12 (2018), pp. 2935–2947.

[47] *VBS3*. Accessed: 2019-01-22. Bohemia Interactive Simulations. URL: https://bisimulations.com/products/virtual-battlespace.

[48] Jeffrey S Vitter. “Random sampling with a reservoir”. In: *ACM Transactions on Mathematical Software (TOMS)* 11.1 (1985), pp. 37–57.

[49] Farzad Kamrani, Mika Cohen, Emil Larsson. *Lagom intelligenta datorgenererade styrkor*. FOI Memo 6587. FOI, Dec. 2018.

[50] Jonathan Schwarz et al. “Progress Compress: A scalable framework for continual learning”. In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, Oct. 2018, pp. 4528–4537. URL: http://proceedings.mlr.press/v80/schwarz18a.html.

[51] Arslan Chaudhry et al. “Riemannian walk for incremental learning: Understanding forgetting and intransigence”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 532–547.

[52] *End-to-End Deep Learning for Self-Driving Cars*. Accessed: 2019-03-21. Nvidia. URL: https://devblogs.nvidia.com/deep-learning-self-driving-cars/.

[53] *Python3.6.6*. Accessed: 2019-02-27. Python Software Foundation, Feb. 27, 2019. URL: https://www.python.org/downloads/release/python-366/.

[54] *Install TensorFlow*. Accessed: 2019-02-27. TensorFlow, Feb. 27, 2019. URL: https://www.tensorflow.org/install.

[55] *NumPy*. Accessed: 2019-02-27. NumPy Developers. URL: http://www.numpy.org/index.html.

[56] *Matplotlib*. Accessed: 2019-02-27. Matplotlib Development Team. URL: https://www.matplotlib.org/.

[57] *Virtual Joystick*. Accessed: 2019-05-03. VJoy. URL: http://vjoystick.sourceforge.net/site/index.php/81-news/135-vjoy-2-1-8-rel1.

[58] *Pyvjoy*. Accessed: 2019-05-03. Tidzo. URL: https://github.com/tidzo/pyvjoy.

[59] *AutoHotkey*. Accessed: 2019-05-03. AutoHotkey Foundation LLC. URL: https://www.autohotkey.com/.

[60] *UJR*. Accessed: 2019-05-03. evilC. URL: https://github.com/evilC/AHK-Universal-Joystick-Remapper.

Appendix A

Model and Hyperparameter Information

The model's architecture is an altered version of Nvidia's architecture for self driving vehicles [52].The model consists of four convolutional layers, three fully connected layers and a single output giving a steering angle, see Figure A.1. The model takes RGB images of 66x200 pixels as input. Dropout is used between the fully connected layers to prevent overfitting. The used hyperparameters can be seen in Table A.1. SafeDAGGER used 75 epochs as its training had more data and 28 epochs were used for the other approaches.

Table A.1: Hyperparameters

Optimizer = Adam
Learning rate = 0.0001
Dropout keep probability = 0.8
Epochs = 75 or 28
Batch size = 50
L2-regularization = 0.0001
EWC-lambda = 5000
Number of Fisher samples = 80

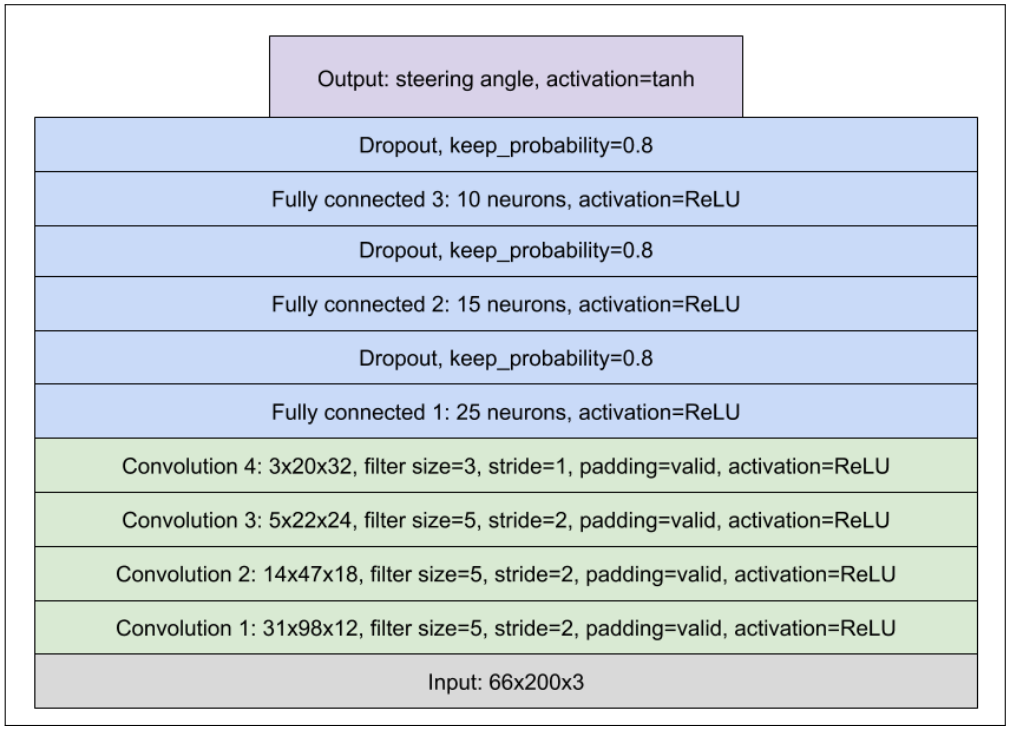


Figure A.1: Deep learning model architecture consisting of four convolutional layers, three fully connected layers and one output. [Created by author]

Appendix B

Hardware and Software Information

This appendix presents all software used, including their versions, and the hardware used during model training and testing. It is done to facilitate reproduction of this work. Table B.1 lists the used software and their versions. The used seed was 20180423, the code snippet below shows the seeds that were set.

```
import tensorflow as tf
import numpy as np
import os

os.environ['PYTHONHASHSEED'] = '20180423'
np.random.seed(20180423)
tf.set_random_seed(20180423)
```

The implementation was written in the programming language Python [53]. Tensorflow [54] was used for building and training the models. NumPy [55] was used for representing data and manipulating it. Matplotlib [56] was used to visualize data. Windows 10 was used as development platform and for deploying the models in VBS3. The machine used for training used Ubuntu 18.04. The VBS3 [47] simulator was used to collect data and test the model. vJoy [57] was used to create a virtual joystick that a model could use to control the vehicle inside VBS3 and pyvjoy [58] is a set of Python bindings that allows one to manipulate vJoy devices. AutoHotkey [59] is a scripting language

Table B.1: Exhaustive information about the used software

Python 3.6.7
Tensorflow 1.12.0
NumPy 1.15.4
Matplotlib 3.0.2
Virtual Battlespace 3 17.4
Ubuntu 18.04.1 64-bit
Windows 10 Version 10.0.14393 Build 14393
CUDA 10.1
Nvidia driver 418.39
vJoy 2.1.8
pyvjoy
UJR 6.10
AutoHotkey

and URJ [60] is an AutohotKey script that was used used to map the physical joystick to the virtual joystick.

The Windows 10 machine was equipped with a Intel Core i7-4800MQ CPU and 32GiB of RAM. The Ubuntu 18.04 machine was equipped with a GTX 1080TI GPU, a AMD Ryzen Threadripper 1950X 16-core CPU and 64GiB RAM. A Logitech Extreme 3D Pro joystick was used to drive a vehicle in the VBS3 simulator to collect data.

TRITA-EECS-EX-2019:526