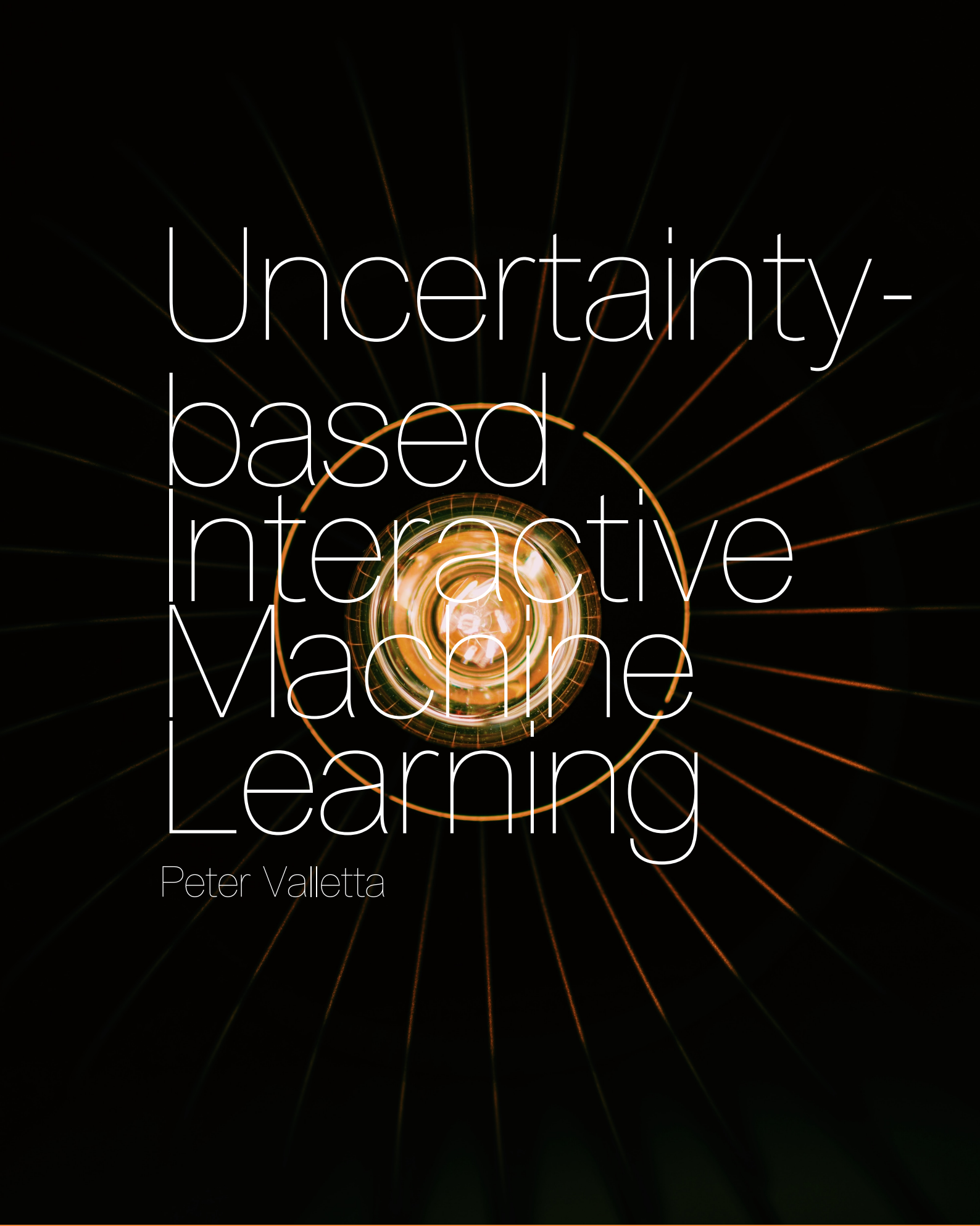
Uncertainty-based Interactive Machine Learning



Peter Valletta

TU Delft

Uncertainty-based Interactive Machine Learning

by

Peter Valletta

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday August 24, 2020 at 02:00 PM.

Student number:	4788168	
Project duration:	September 3, 2019 – August 24, 2020	
Thesis committee:	Dr. ing. J. Kober,	TU Delft, supervisor
	ir. R. Pérez-Dattari,	TU Delft, supervisor
	Dr. P. Mohajerin Esfahani,	TU Delft, reader
	Dr. W. Pan,	TU Delft, reader

TU Delft

The research work disclosed in this publication is partially funded by the
Endeavour Scholarship Scheme (Malta). Scholarships are part-financed
by the European Union - European Social Fund (ESF) -
Operational Programme II – Cohesion Policy 2014-2020
"*Investing in human capital to create more opportunities and promote the well-being of society*".

European Union – European Structural and Investment Funds
Operational Programme II – Cohesion Policy 2014-2020
"Investing in human capital to create more opportunities and promote the well-being of society"
Scholarships are part financed by the European Union - European Social Funds (ESF)
Co-financing rate: 80% EU Funds;20% National Funds

Abstract

Interactive Machine Learning (IML) encompasses numerous Machine Learning (ML) methods in which a human user participates in a novice agent's learning process, through providing either corrective or evaluative feedback or demonstrative states and actions. A primary assumption of IML methodologies is that user input is optimal, or near-optimal. Realistically, a set of demonstrations provided to a *Learning from Demonstration (LfD)* algorithm will contain conflicting or poor examples, which often degrade the quality of the learnt policies. While this issue is sporadically discussed in literature, the formulation of a comprehensive solution remains an open problem.

Accordingly, the objective of this project is to develop and evaluate an algorithm with the capability to successfully train policies in spite of suboptimal training data, while solving sequential decision-making problems in robotics. Direct conflict analysis and Uncertainty Estimation (UE) are studied to enable the detection of unwanted elements in IML training data, while an extension of the Option Class (OC) framework (Chernova and Veloso, 2008) is proposed for the interactive training of Artificial Neural Network (ANN) policies in scenarios with imperfect training data.

The conflict analysis algorithm is designed to quantify conflict values between data points in a dataset intended for supervised ANN training. A learning conflict is encountered when two similar input samples have different target values, while for two such input samples with comparable target values, there is no conflict. The implementation of this conflict analysis algorithm largely followed the process described in the proposing publication (Ledesma et al., 2018) and was intended as a metric for assessing the ensuing UE methodology. This method was validated for the detection of learning conflicts in demonstration data and successfully applied within the OC algorithm for the same purpose.

UE presents a structured approach for the quantification of a network's confidence in the accuracy of its output, based on the observed training data. Intuitively, an ANN's uncertainty on a specific input sample should be low when the corresponding training labels are consistent, and high when said training labels are conflicting. A split-network architecture that performs function approximation alongside UE was initially devised for settings with one-dimensional action spaces and subsequently extended to environments which consider multiple, independent actions.

The scope of the OC algorithm is to resolve ANN training problems arising from inconsistent demonstrations in environments with equivalent action choices, such as obstacle avoidance scenarios. This project leverages the generalisation and scalability qualities of ANNs and extends the applicability of this approach to a significantly more diverse set of environments. A multiple-stage process was undertaken to implement this approach on regression-based networks, as opposed to the original classifier-based methodology which employed Gaussian Mixture Models. The initial stage involved the use of an ANN classifier, which was rapidly superseded by a regression-based network incorporating UE for the identification of conflicting data in continuous, multi-dimensional state and action spaces.

Broadening the relevance of this algorithm to continuous action spaces, which may take an infinite set of values, implies that it is likewise able to process discrete spaces, which are intrinsically finite and in this case, easier to analyse. Combinations of continuous and discrete spaces encompass all conceivable environments, hence the OC algorithm may be used in virtually any scenario, irrespective of the dimensionality and nature of the spaces considered. Through testing in uni- and multi-dimensional settings, this methodology was verified to successfully train a novice agent to achieve some desired objective, irrespective of any conflicts in demonstration data. Nevertheless, various features of this approach may be improved through additional development, in order to make this method unerring and increasingly efficient.

The research work disclosed in this publication is partially funded by the Endeavour Scholarship Scheme (Malta). Scholarships are part-financed by the European Union - European Social Fund (ESF) - Operational Programme II – Cohesion Policy 2014-2020 "Investing in human capital to create more opportunities and promote the well-being of society".

Our deepest fear is not that we are inadequate.
Our deepest fear is that we are powerful beyond measure.
It is our light, not our darkness
That most frightens us.

We ask ourselves
Who am I to be brilliant, gorgeous, talented, fabulous?
Actually, who are you *not* to be?

— *Marianne Williamson*

Preface & Acknowledgements

Peter Valletta
Delft, August 2020

All the work described within this thesis represents my efforts and progress towards fulfilling the requirements of the Master of Science in Systems and Control program at the Delft University of Technology. These two years I have spent learning and researching at this university have been arduous but enjoyable, and have provided invaluable experiences and learning opportunities.

Furthermore, I would like to express my appreciation of a number of persons that have greatly helped me, in one way or another, to complete this thesis and along with it, my degree. First of all, I would like to thank Dr. ing. Jens Kober and Mr. Rodrigo Pérez Dattari for their invaluable insight, guidance and feedback throughout this project.

Immense gratitude goes towards my parents, who have incessantly supported my choices and have been essential for the continuation of my studies. I am also extremely grateful for my uncle, Kenneth, without whom I would not have started my degree and who has continually offered guidance on technical matters.

Finally, my heartfelt thanks go to my other half, Althea, who has supported me day in, day out for over six years, irrespective of the physical distance between us.

Glossary

List of Acronyms

i.e.	id est (‘it is’)
i.i.d.	independent and identically distributed
ANN	Artificial Neural Network
BC	Behavioural Cloning
CNN	Convolutional Neural Network
CVA	Conflict Value Algorithm
DBSCAN	Density-based Spatial Clustering of Applications with Noise
EAC	Equivalent Action Choices
FNN	Feedforward Neural Network
GMM	Gaussian Mixture Model
LfD	Learning from Demonstration
LfF	Learning from Feedback
IML	Interactive Machine Learning
KDE	Kernel Density Estimation
KL	Kullback-Liebler
MAP	Maximum A Posteriori
MC	Monte Carlo
ML	Machine Learning
MSE	mean squared error
NLL	negative log-likelihood
NLP	Natural Language Processing
NN	Nearest Neighbour
OC	Option Class
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
SGD	Stochastic Gradient Descent
SL	Supervised Learning
UE	Uncertainty Estimation
UL	Unsupervised Learning
VI	Variational Inference

List of Symbols

Greek Symbols

$\boldsymbol{\Gamma}$	Regularisation matrix
$\boldsymbol{\theta}$	Parameter vector in policy representation
λ	Regularisation parameter
ϕ	Non-linear activation function
π	Exact policy representation
π_θ	Approximate policy representation
Ψ	General function approximator
σ	Variance of a set of data elements
τ	Model precision
υ	Uncertainty estimate

Latin Symbols

$\bar{x}$	Normalised data point
$\boldsymbol{x}$	Input data point for a model
$\mathcal{A}$	Action Space
$\mathcal{D}$	Dataset of demonstrated state-action pairs
$\mathcal{L}$	Loss function
$\mathcal{S}$	State Space
$y(\boldsymbol{x})$	Output data point for a model

Other Symbols

$\boldsymbol{A}$	Matrix
$\boldsymbol{a}$	Vector
$\mathbb{R}$	The set of real numbers
$\mathbb{Z}$	The set of integer numbers
a	Scalar variable

Contents

1

Introduction

A vast amount of development and innovation have propelled Machine Learning (ML) from a fledgeling concept to a mainstream area of research and an integral component of daily life, in a few decades. This affluence of content has enabled the ML field to progress from tackling intellectually difficult but structured, mathematically straightforward problems, such as chess-playing (Campbell et al., 2002), to more abstract and practical applications, including the automation of labour, Natural Language Processing (NLP) and medical diagnoses (Goodfellow et al., 2016a; Mayo et al., 2018).

An intrinsic quality of ML systems is the capability of generating knowledge through the inference of patterns and relationships from data. Individual pieces of information, termed *features*, are correlated to various possible outcomes relevant to a specific task. Feature extraction is essential for intelligent learning, which implies that a system should be able to *generalise*, i.e., extend its applicability to data instances outside of what it has already experienced.

The Artificial Neural Network (ANN) is a widely popular paradigm, inspired by and roughly modelled on the highly interconnected, parallel structure of the biological brain and its learning model. The motivation for applying this paradigm within the scope of ML rests on the idea that the brain provides a proof-by-example of intelligent behaviour and duplicating its functionality is an effective way of enabling this process (Goodfellow et al., 2016a; Nielsen, 2015a). As a result of their composition, ANNs are uniquely suitable for certain applications, such as classification, dimensionality reduction and sequence or pattern generation. Furthermore, ANNs are applicable in control systems, owing to their capability for non-linear modelling and data processing and shorter development time, when compared to traditional methods. These models are also able to generalise and are easily scalable to different scenarios (Eberhart and Shi, 2007; Joshi, 2017; Mahanta, 2017). In light of their presently extensive use and various qualities, ANNs are exclusively used as the ML framework within this project.

Thus far, the scope of this report encompasses most scenarios where ML may be applied, however, this project primarily considers ML within the rapidly expanding field of robotics. As this field permeates into various domains, the necessity for accurate and robust control algorithms increases correspondingly, particularly in the case of autonomous robots. Such systems are commonly required to perform complex motions in dynamic environments, often in the presence of humans, hence safe operation is imperative. As the task and environment complexity increase, manually prescribing this behaviour into deterministic algorithms quickly becomes a daunting task. Additionally, in specific scenarios, domain experts may not possess the skills or robotics knowledge to create such algorithms, however their expertise is necessary to successfully devising a control algorithm, or policy, for robotic agents. A *policy* defines a learning agent's behaviour, through mapping sensed environment states to the set of actions that the robot is able to perform (Argall, 2009; Sutton and Barto, 2018).

Interactive Machine Learning (IML) offers an intuitive methodology to teaching robots a set of desired motions, without obliging domain experts to also be skilled in robotics. In this work, IML is designated as a general term encompassing the ML approaches in which a human user participates in the agent's

learning process. These approaches are further classified depending on the user's interaction, which may take the form of demonstrations, corrective feedback or evaluative feedback. Consequently, the IML field is divided into two categories - Learning from Demonstration (LfD) and Learning from Feedback (LfF), respectively. The latter category is not commonly found in literature, however in keeping with the approach in Argall (2009), Celemin (2018) and Pérez-Dattari (2019), it is applied in this report to denote IML methods which shape policies through feedback.

In general, IML builds on the premise that it is frequently easier for a human teacher to demonstrate the sought behaviour instead of explicitly engineering it. This methodology facilitates the application of robotics in new domains since it allows users without a background in robotics to effectively teach the learning agent. An associated benefit of IML is that it incorporates expert and prior knowledge into the learning process and therefore allows for a significant decrease in sample complexity, in comparison to methods that attempt to learn the same task, without this prior information (Osa et al., 2018).

Regardless of their advantageous qualities, IML approaches often require a comprehensive approach in order to answer four main queries: when, what, who and when to imitate a teacher's input. These considerations are thoroughly examined in related literature (Billard et al., 2016; Billing and Hellström, 2010) and are reviewed in more depth in Chapter 2. This project is concerned with alternative issues within this field that have not been addressed with the same vigour, and persist as open problems.

A primary assumption of IML methodologies is that user input is optimal, or at best sub-optimal. Realistically, a set of demonstrations provided to an LfD algorithm will contain conflicting or poor examples which often degrade the quality of the learnt policies, which in turn, may not meet the required goals. Furthermore, human feedback which is processed in real time by an LfF algorithm may be erroneous or otherwise non-ideal, leading a learnt policy to fail to achieve the desired performance in multiple tasks, notwithstanding a significant amount of feedback. An additional pitfall of IML methods lies in their dependence on the human teacher to provide training data, frequently without an indication of progress in learning a specific task. This approach allows the teacher to independently determine when to cease to provide inputs to the algorithm and is susceptible to early stopping. In such a scenario, the novice agent is not sufficiently trained on a task and does not achieve the desired performance (Argall et al., 2009; Billing and Hellström, 2010; Schaal, 1997; Scholten, 2019).

While these issues are sporadically discussed in literature, a comprehensive solution to either case has not yet been proposed. Particularly, Osa et al. (2018) state that the detection of undesirable demonstration data within a set of such inputs and explicitly removing it remains an open problem.

Accordingly, this project studies the use of direct conflict analysis (Ledesma et al., 2018) and uncertainty estimation to detect such unwanted elements in IML training data and proposes alterations to the Option Class (OC) framework (Chernova and Veloso, 2008), which is used for the interactive training of ANNs in scenarios with imperfect training data.

In this context, this thesis and consequently, this report, were divided into two distinct, yet related segments. In the former, two separate methods are proposed for the assessment of demonstration data in LfD scenarios, with focus on the effectiveness of uncertainty estimation towards achieving this goal. Directly obtaining a measure of the significance of conflicts within a dataset presents an effective response to the considered task, however, it disregards the most important facet of training a robotic system, i.e., the performance of the trained policy, which is inexorably linked to this data. Conversely, uncertainty estimation links policy performance to dataset quality and provides a more informative metric to assess dataset issues, at the expense of the requirement to train an ANN.

The latter part of this project studies the existing OC framework as an approach to successfully training novice agents in the presence of conflicting demonstrations arising from equivalent action choices (EAC). This methodology, which was previously limited to Gaussian Mixture Model (GMM) based classifier networks and thus, discrete action spaces, was extended to an ANN-based implementation that is scalable to low and high dimensional state spaces and generalised to continuous action spaces, which are the prevailing case in robotics.

1.1 Aim & Objectives

The aim of this thesis is to develop and evaluate a methodology with the capability to successfully train policies or improve agent's performance in spite of suboptimal training data, when solving sequential decision-making problems in robotics. This goal was subdivided into a series of progressive objectives, which are listed hereunder.

- Define and implement an algorithm that, through a trained ANN, effectively identifies regions of high uncertainty and conflicting data within a set of demonstrations obtained while teaching robots to perform specific tasks with IML.
- Validate the proposed algorithm using simulated platforms.
- Define and implement an algorithm that, by employing the inference provided from the prior method, allows robots to successfully learn to solve tasks through LfD, despite conflicting and sub-optimal demonstrations. In scenarios where poor demonstrations induce inefficient performance rather than absolute failure, the algorithm is desired to improve said performance.
- Validate the proposed algorithm using simulated platforms.

1.2 Outline

This thesis report comprehensively presents the work undertaken in this project with the aim of developing solutions for the outstanding problems defined previously. Specifically, the content within this report is laid out in a progressive manner, analogous to the series of project objectives detailed above.

Chapter 1 has briefly reviewed the background to this project, focused this research onto two areas within the IML field and introduced the content of this report. Chapter 2 reviews the fundamental theoretical concepts and describes a number of existing publications relevant to the area of research introduced above. Two approaches for the identification of sub-optimal and conflicting entries in a set of demonstrations are proposed in Chapter 3.

In Chapter 4, an algorithm exploiting the inference provided by these methods is proposed to enable successful policy learning with IML in settings with sub-optimal demonstrations. Furthermore, this chapter contains a comprehensive analysis of the proposed algorithm, which is applied on realistic simulated scenarios including a simulated robotic platform. A review of the contributions of this project and recommendations for the future development of these algorithms are presented in Chapter 5.

2

Theoretical Framework

This chapter provides a concise review of the core theoretical underpinnings of the algorithms and methods applied in this project and a brief description of the publications which are relevant to its scope. Section 2.1 explains the Interactive Machine Learning (IML) field in more depth. Section 2.2 presents a number of function approximation methods that were applied or studied within this project and the Supervised Learning (SL) methodology for policy training, which is central to Learning from Demonstration (LfD) algorithms. Finally, an assessment of state-of-the-art methods within the Uncertainty Estimation (UE) field is included in Section 2.3, in anticipation of the application of these methods to dataset analysis.

2.1 Interactive Machine Learning

As previously described in Chapter 1, IML encompasses a set of Machine Learning (ML) methodologies in which a human participates in the learning process by supervising the agent's performance and providing feedback to improve the learnt policy. These approaches are distinguished based on the type of feedback received from the human user and are classified accordingly as LfD or Learning from Feedback (LfF) methods. This project's objective is concerned with the analysis of demonstration data, thus henceforth, this report is predominantly concerned with the former class of IML methods, since LfF approaches shape policies without explicitly providing demonstration data.

LfD is a well-established technique by which novice agents learn a policy, which shall be denoted π, through the observation of demonstrations provided by an expert teacher. In this sense, a novice agent signifies an action-taking player in an environment that does not possess any prior knowledge of the task it must accomplish and must be taught to traverse this environment through demonstrations. Incoming demonstrations are perceived as a sequence of state-action pairs (i.e., labelled data), that instruct the learning to take specific actions at a given state (Argall et al., 2009). The learning agent's task is to reproduce this behaviour as accurately as possible, which it seeks to do by shaping its policy, π_θ, in a supervised manner (Bagnell, 2015; Kober and Peters, 2010). A database of demonstrated behaviour collected prior to and during the algorithm's learning phase is used to adapt this policy.

Several variations of this approach have been proposed under different terminologies, such as *Imitation Learning*, *Learning from Experience* and others (Billing and Hellström, 2010). A primitive LfD approach, known as Behavioural Cloning (BC), directly and naively learns the demonstrated state-action mapping, as illustrated in Figure 2.1.

Billard et al. (2016) highlight the implicit scope within the LfD field of developing algorithms that are generic in their generation and representation of the skills to be taught to a novice agent. A set of key problems - *what, when, who* and *how* to imitate - exist within the LfD framework and must be addressed in order to realise this generality (Argall et al., 2009; Billard et al., 2016; Billing and Hellström, 2010).

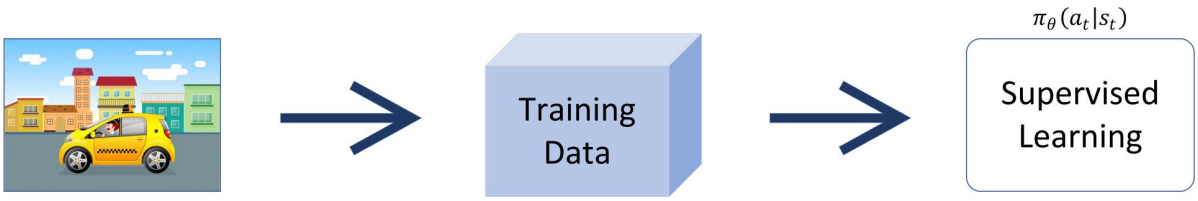


Figure 2.1: A simplified depiction of the Behavioural Cloning approach (adapted from Pérez-Dattari (2019)), where demonstrations are acquired from a domain expert as a dataset of labelled state-action pairs and a novice agent is trained in a supervised fashion to imitate the performance of the expert.

What to imitate refers to the problem of devising an appropriate metric to evaluate the action reproduction of the learning agent. This task involves the determination of which behaviour should be copied and which actions are irrelevant or may be safely ignored. In a continuous control scenario, this problem relates to the definition of the feature space, constraints and cost function used for learning.

When to imitate concerns problems where the teacher does not provide an explicit set of demonstrations, thus the learning agent must identify when relevant actions are being performed. *Who* to imitate is an issue in settings where multiple motions are present in a robot's observation space and it is required to determine which actions were performed by the demonstrator. *How* to imitate involves the determination of the movements a robot must perform in order to replicate specific tasks, for example, moving a block from one position to another. This issue is concerned with the correspondence between the demonstrator and the learning agent and is presented in detail in Section 2.1.1.

This set of fundamental issues underlies all LfD approaches and must be appropriately treated in order for any methodology to successfully achieve its goal. In addition to these concerns, a learnt policy must be able to recover from inevitable mistakes in execution. Such behaviour is not considered in the BC approach, hence policies defined with such a method often fail in practice. This scenario is thoroughly explained in Section 2.1.2 and is accompanied by the description of the DAgger (Ross et al., 2010) algorithm, which effectively addresses this problem.

2.1.1 The Correspondence Problem

Any form of imitation or social learning, where the learning agent acquires knowledge through observation of a more skilled agent, or demonstrator, must involve some level of *correspondence* between these two agents. In the case where both agents are humans of a similar gender, age and size, their correspondence becomes trivial, as similar body parts are mapped together - left arm to left arm, right leg to right leg, and so on.

A significant difficulty arises in IML when there are notable dissimilarities between the capabilities of the demonstrator and those of the agent. For example, when the goal of the learning task is to direct a robotic arm to perform some action and the demonstrations are provided by a human expert, the differences in actuation, reach and degrees-of-freedom may be severe. A visual comparison between a human arm and a robotic arm is provided in Figure 2.2. This issue is termed the Correspondence Problem (Nehaniv and Dautenhahn, 2001) and may lead to failure unless the LfD procedure is adapted to account for the learner's constraints.

This issue expands the LfD objective from directly reproducing a teacher's actions to the added consideration of identifying the elements within a demonstration which should be imitated, such that the actions of the learning agent are as close as possible to those of the demonstrator. A plausible solution in certain problems is to learn the forward dynamics model of a system and apply this model to relate the demonstrator's input to the agent's structure (Meriçli and Veloso, 2011; Osa et al., 2018).

A direct teacher-learner mapping may not exist in multiple scenarios due to discrepancies in structure, mechanics or sensing ability. For example, a robot learner obtaining visual input through a camera will not observe states and state changes in the same manner as a human demonstrator. In such cases,

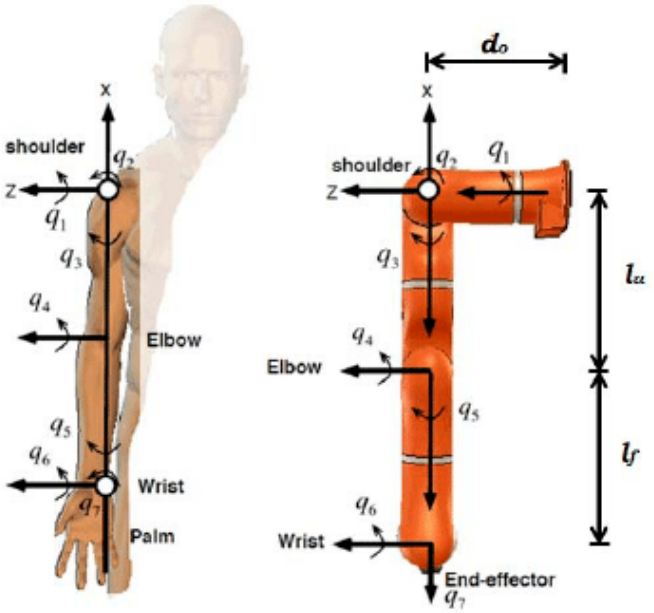


Figure 2.2: A comparison of the kinematic arrangements of a human arm and a robotic arm (Ficuciello et al., 2014). In this setting, the systems are particularly similar and thus, correspondence at a mechanical level is not a significant issue. Nevertheless, the same problem may require considerable effort elsewhere, e.g., in processing observations obtained through a camera, which will inevitably differ from observations acquired by a human.

the main concern becomes how to map the observed demonstrations to the agent's domain in order to transfer information between the two (Argall, 2009; Argall et al., 2009).

2.1.2 State Distribution Mismatch

In addition to the complexities introduced due to correspondence issues, LfD approaches are greatly susceptible to state distribution mismatch, which inevitably leads to catastrophic learning failure unless this problem is taken into account *a priori*. Bagnell (2015) describes the scenario where a car in a video game is taught to drive by directly mapping images captured from a real car to steering angles in the game using a classical SL approach. In this instance, the approach fails severely and the virtual vehicle drives off the road.

In this scenario, learning errors *cascade*, as opposed to the independent errors in SL. Inexorably, the learning agent will err at some point and steer differently to the human driver, displacing the vehicle from driving down the centre of the road. The training examples encountered by the learning agent may not account for this scenario, thus the agent has a higher probability of performing another error, which would lead to a more significant state deviation, further increasing the probability of error, etcetera. An illustration of this scenario is provided in Figure 2.3.

A naive solution to this problem would involve gathering demonstrations for all the scenarios that the learner might encounter during execution. Clearly, such an approach would not be viable on modest-scale problems, let alone the large-scale situations being considered presently, and thus focus should be directed to correcting the most relevant scenarios (Bagnell, 2015; Osa et al., 2018).

The DAgger (Dataset Aggregation) algorithm (Ross et al., 2010) is a deft approach to addressing the cascading error problem that iteratively trains a deterministic policy which, for the distribution of states it traverses during training, is shown to guarantee good performance. In its most basic form, DAgger trains a new policy, π_θ, for each of its T iterations. Algorithm initialisation requires the demonstration dataset to be set to an empty set, $\emptyset$, while the new policy is set to any policy in Π, a class of policies considered by the learner.

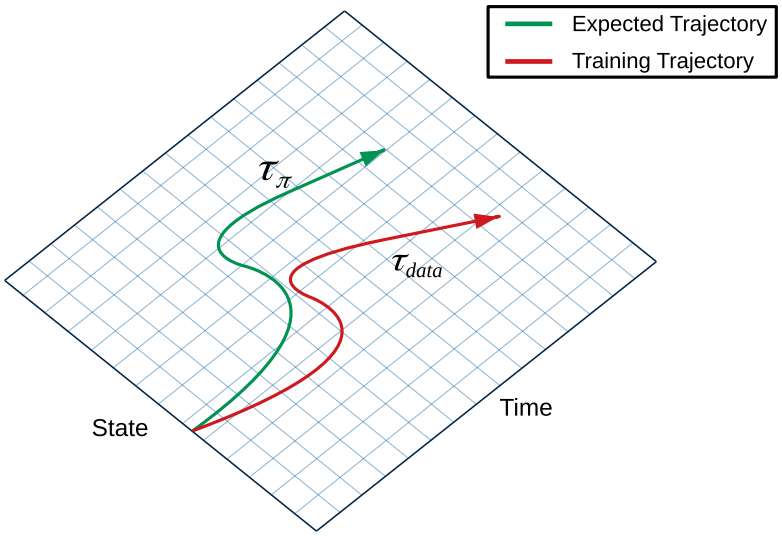


Figure 2.3: A visual representation of the state-distribution mismatch problem (adapted from Pérez-Dattari (2019)), where a small initial deviation from the ideal trajectory (τ_π), initiates a cascade of errors causing the resultant policy to significantly differ from its desired behaviour.

The initial algorithm iteration trains the novice policy in typical SL fashion, using the expert policy to gather demonstrations. A second policy is trained by executing the novice policy for one time step, followed by releasing control to the expert and gathering the state-action pairs traversed under the expert policy as demonstrations. The new demonstration set is added to a global dataset, $\mathcal{D}$, which is used to define a new novice policy. This process is repeated to train the i^{th} step of the novice by observing the expert's decisions after running '$i-1$' steps of the learnt policy.

In addition to this training procedure, the action policy is modified to partly query the expert policy π_θ^* with probability β when choosing controls, such that the number of mistakes is minimised. With this approach, every policy encounters states derived from the same distribution, although not necessarily identical to its training states. Errors of early policies are re-visited by successive ones, which learn to recover by imitating the expert's corrections. A pseudo-code implementation is given in Algorithm 1.

Algorithm 1: DAgger (Ross et al., 2010)

procedure DAgger(Π)
  Initialise $\mathcal{D} \leftarrow \emptyset$
  Initialise $\tilde{\pi}_{\theta_1}$ to any policy in Π
  for $i \leftarrow 1, N$ **do**
    Let $\pi_{\theta_i} = \beta_i \pi_\theta^* + (1-\beta_i)\tilde{\pi}_{\theta_i}$
    Sample T-step trajectories using $\tilde{\pi}_{\theta_i}$
    Get dataset $\mathcal{D}_i = \{(s, \pi^*(s))\}$ of visited states and actions
    Aggregate datasets: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$
    Train classifier $\tilde{\pi}_{\theta_{i+1}}$ on $\mathcal{D}$
  end for
  return π_θ as the best $\tilde{\pi}_{\theta_i}$ on validation
end procedure

EnsembleDAgger (Menda et al., 2018) is an extension of the DAgger approach that highlights the lack of safety considerations in the original algorithm while training the novice policy. The underlying reasoning behind this approach postulates that when the novice acts, the system will likely deviate from the expert trajectory and sample new states, which may be unsafe or lead to failure. Conversely, it assumes that the expert exclusively traverses safe trajectories, therefore, the goal of EnsembleDAgger is to allow the novice to act as much as possible, in order to experience diverse states and gather relevant demonstrations, while limiting the probability of failure. With this algorithm, the novice is

allowed to select actions when it is sufficiently similar to the expert policy, as proposed in SafeDAgger (Zhang and Cho, 2016), and when the novice's doubt is below a certain threshold. Novice doubt (a.k.a. uncertainty) is estimated through the variance computed with an ensemble of ANNs, following the approach outlined in Section 2.3.2.

Human Gated (HG) DAgger (Kelly et al., 2018) is a variation of EnsembleDAgger that moves away from the Robot-Centric (RC) sampling applied in all prior approaches. This is motivated by the assumption that when the human expert is allowed extended periods of control authority, the observed action labels will be of higher quality. Unlike DAgger, which allows the novice policy to act in a pre-determined sequence and unlike EnsembleDAgger, which passes control between the novice and expert policies through an automatic gating function, HG-DAgger permits the human to take control at any point, when they detect that the system is traversing unsafe states. Expert demonstrations are gathered exclusively during these periods, otherwise the novice policy is allowed to act.

In this project, HG-DAgger is applied to train novice agents when an evaluation of dataset quality, in terms of an agent's performance, or an assessment of the developed algorithm are required. The manual gating approach allows for the reuse of pre-collected datasets in certain test cases, while retaining the resilience to the state distribution mismatch problem afforded by the DAgger method. The reuse of certain datasets enables a direct comparison of trained agents and hence, a comparative evaluation of the effects of the developed algorithms.

2.2 Function Approximation

In the context of sequential decision making, i.e., training a digital system (physical robot or simulated agent), the *function* is understood to represent a mapping between the learning agent's observations and its actions. This function is generally termed the *policy*, π, as defined in Section 2.1. While in restricted scenarios, π may be characterised by a look-up table or any other exact representation, this swiftly becomes intractable as the state and action spaces increase in dimension.

Function approximation, which denotes a family of mathematical and statistical methods, is the standard approach to representing policies in large (typically continuous) state-action spaces (Kober et al., 2013). While a large number of function approximators exists, this section briefly introduces the Gaussian Mixture Model (GMM), in light of its relevance to the Option Class (OC) framework (Chernova and Veloso, 2008) which is discussed in Chapter 4 and Feedforward Neural Networks, which are predominantly applied in this project.

Henceforth, the parametrically-approximated policy shall be denoted π_θ, where $\boldsymbol{\theta}$ represents a vector of model parameters. The function approximator Ψ shall represent the policy as $\pi_\theta = \Psi(s; \boldsymbol{\theta})$ such that $\Psi : \mathcal{S} \rightarrow \mathcal{A}$ defines the mapping from the state space to the action space. The reference to states and actions is made from the viewpoint of the robotics field, where observed states are supplied as inputs to Ψ, which should predict (thus, output) a corresponding action.

2.2.1 Gaussian Mixture Models

A general mixture model is a composition of like functions that are individually limited in their representation power, however, may be superimposed to allow for an improved characterisation of the data being considered. Individual components have simple parametric forms, however the aggregation of such components enables the representation of complex functions (Bishop, 2006c; Grosse and Srivastava, 2015).

A GMM involves a combination of multiple Gaussian distributions, with each component having a unique mean μ_i and covariance $\boldsymbol{\Sigma}_i$, such that

$$\pi(\boldsymbol{x}) = p(\boldsymbol{x}) = \sum_{i=1}^{N} a_i \, \mathcal{N}(\boldsymbol{x} \,|\, \mu_i, \boldsymbol{\Sigma}_i) \tag{2.1}$$

where a_i is a mixing coefficient. Bishop (2006c) defines that with an adequate amount of Gaussian distributions and appropriate tuning, a GMM is able to approximate most continuous functions to some arbitrary accuracy.

The set of mixing coefficients $\boldsymbol{a}$ must respect specific constraints in order to be probabilities, such that the GMM output is a probability distribution. First, assuming that $p(\boldsymbol{x})$ and all individual distributions are normalised, the integration of Equation 2.1 defines:

$$\sum_{i=1}^{N} a_i = 1 \tag{2.2}$$

Additionally, consider the conditions $p(\boldsymbol{x}) \geq 0$ and $\mathcal{N}(\boldsymbol{x} \,|\, \mu_i, \boldsymbol{\Sigma}_i) \geq 0$, where the latter implies that $a_i \geq 0 \; \forall \; i$). These two constraints, coupled with Equation 2.2, define boundaries on the mixing coefficients:

$$0 \leq a_i \leq 1 \tag{2.3}$$

A representation of function approximation with a GMM is included in Figure 2.4, where the plots are normalised for the individual Gaussian distributions.

2.2.2 Feedforward Neural Networks

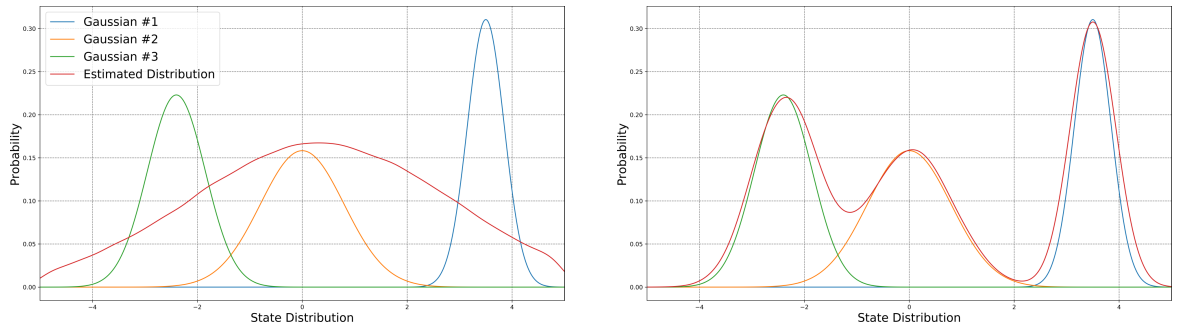
The Feedforward Neural Network (FNN) is the classic Artificial Neural Network (ANN) architecture, aptly named because information exclusively traverses the network in one direction, from the input to the output, without any loops (Nielsen, 2015a). With sufficient complexity (i.e., number of units with appropriate structure), FNNs can approximate any continuous function (Engelbrecht, 2007). In a mathematical sense, these networks aim to approximate some function $f(\cdot)$ of their input, which may be composed of multiple mappings. For example, three functions $f^{(1)}$, $f^{(2)}$ and $f^{(3)}$ may be chained such that:

$$\Psi(\boldsymbol{x}) = f(\boldsymbol{x}) = f^{(3)}\left(f^{(2)}\left(f^{(1)}\left(\boldsymbol{x}\right)\right)\right) \tag{2.4}$$

These structures are the most common in FNNs and denote a network's layers. In this case, $f^{(1)}$ is the first layer, and so forth. The overall number of layers within a network is called its *depth*, while its *width* is defined by the number of neurons (units) in a layer. A *neuron* is the most basic decision-making structure within a network and is defined by a function h composed of weights $\boldsymbol{w}$, a bias b and a non-linear activation function ϕ:

$$h(\boldsymbol{x}) = \phi\left(\boldsymbol{w} \cdot \boldsymbol{x} + b\right) \tag{2.5}$$

FNN layers are classified into three types - input, hidden and output. The input layer is unique within a network and is arguably not a layer, since it does not perform any decision-making function. The output layer is the final set of neurons of a network and expresses the outermost function of the concatenation $f(\cdot)$, which may result in a singular or n-dimensional signal. All the layers in between the input and



(a) Gaussian distribution estimate **(b)** GMM estimate

Figure 2.4: A 1-dimensional comparison of the estimation accuracy of a GMM and a singular Gaussian distribution for a combination of three distinct distributions. Naturally, a singular Gaussian is insufficient in this scenario, whereas the GMM effectively approximates the desired data distribution.

output layers are the FNN's hidden layers, which may be of arbitrary dimension. An FNN with one hidden layer is shown in Figure 2.5.

In order for FNNs to be considered *universal approximators* (Hornik, 1991), they must include some non-linearity within their structure, otherwise they would be limited to representing linear functions. Activation functions characterise this non-linearity in such networks. As previously mentioned, ANNs hold the ability to accurately approximate any function, however this depends on the choice of activation functions, network depth and individual layer width, which are outstanding design parameters. Multiple types of activation functions exist, with distinct learning properties. A classical activation function is the sigmoid, or logistic function

$$\phi(x) = \frac{1}{1 + e^{-x}} \tag{2.6}$$

which is bound such that $\phi(x) \in (0, 1)$. The bounds on the value of this function do not allow for a decrease in network weights, thus a more permissive function, such as the hyperbolic tangent may be used. A similar, heuristic argument may be used to justify the application of other activation functions, given that rigorous proofs defining the benefits of one function over another do not exist (Nielsen, 2015b). A set of common activation functions is displayed in Figure 2.6.

Lastly, returning to the concept of policy parametrisation, in this scenario the weight vector $\boldsymbol{\theta}$ defining the approximate policy π_θ corresponds to a flattened concatenation of all network weights and biases.

2.2.3 The Basics of Machine Learning

A prime target of ML is learning from existing data and discovering hidden patterns, which can be applied to the evaluation of new objects in the future (Yang, 2018). ML algorithms may be divided into three categories, i.e., SL, Unsupervised Learning (UL) and Reinforcement Learning (RL) methods (Goodfellow et al., 2016b; MacKay, 2003; Sutton and Barto, 2018).

In the context of ML, both SL and UL may be viewed as algorithms that are able to learn from data. An algorithm may be said to *learn* if its performance at tasks *T*, as quantified by some measure *P*, improves with experience *E* (Goodfellow et al., 2016b). This definition posits that these three elements can be used to construct ML algorithms.

Reinforcement Learning differs from both these methods in that it collects data through repeated rollouts while concurrently learning from rewards, rather than relying on available (labelled) data (Sutton and Barto, 2018). Although RL has been shown to be a powerful tool for solving sequential decision making problems, this project is exclusively concerned with IML strategies that rely on SL and utilises some UL algorithms, thus RL shall not be discussed further.

The Experience, *E*

In a broad sense, *E* defines the interaction that ML algorithms have with the dataset during their learning phase and explains the distinction between SL and UL:

- **Supervised Learning**: The idea behind SL is that an algorithm learns from a training set containing features associated with a label or target, provided by an expert, external supervisor (Goodfellow et al., 2016b; MacKay, 2003; Sutton and Barto, 2018). The provided label indicates the correct action the learning agent should take, which differs depending on the task, *T*. In a regression setting, a label will commonly denote the correct output of a function or a direction in which an agent should move, whereas for classification, the label represents the true class of the feature being assessed.
- **Unsupervised Learning**: The UL approach is concerned with finding structure embedded within sets of unlabelled data. In this scenario, feature labels are not required and the algorithms attempt to 'make sense' of the data by identifying patterns or grouping similar features. Some unsupervised algorithms are used for predictions on unseen data, based on the structure identified in the training set (Goodfellow et al., 2016b; MacKay, 2003; Sutton and Barto, 2018).

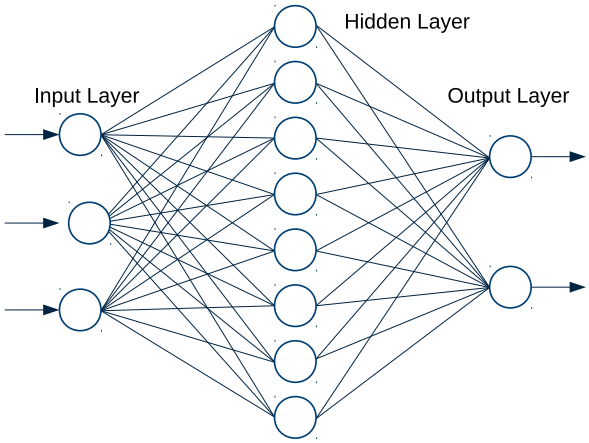
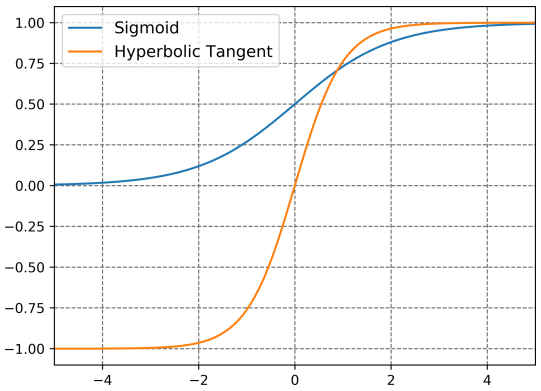


Figure 2.5: A 3-input, 2-output FNN with a single hidden layer.

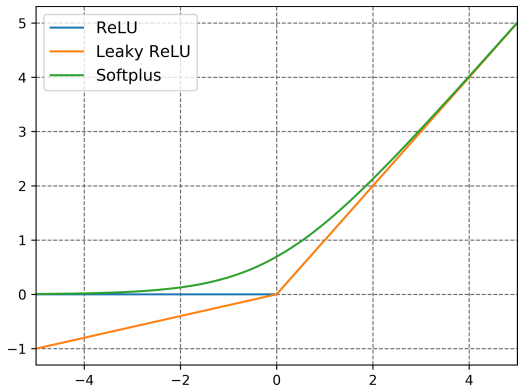
The Task, *T*

Tasks in ML are often problems that are too complex to solve with human-designed and coded programs. In this context, the process of learning is not the task, e.g., if the desired goal is to make a robot walk, the task is walking. The distinction between various problems in ML is often noted in how the algorithm should process an example, which is commonly a collection of features measured from some object or event which the system should process. The two most common ML tasks are classification and regression, which are described as follows:

- **Classification**: In such tasks, the learning system is asked to identify to which one of *k* distinct classes an input belongs. The output of this system is commonly configured in one of two ways, either numeric such that $y : \mathbb{R}^{n\times m} \rightarrow \{1, \ldots, k\}$, where $n, m \in \mathbb{Z}_{>0}$ or a one-hot encoding, where a k-dimensional vector is produced, with each element corresponding to one class. This vector should contain strictly one non-zero element that indicates the class the input belongs to. A variant to this direct approach is the classification task where the system outputs a probability distribution over classes.
- **Regression**: This class of tasks is concerned with the prediction of a numeric value(s) given a set of features - a significant similarity with classification tasks exists, however the output format is different. In this scenario, the ML algorithm learns some underlying function and is expected to provide an output $y : \mathbb{R}^{n\times m} \rightarrow \mathbb{R}$, given the training data.



(a) A comparison of the sigmoid and hyperbolic tangent activation functions.



(b) A comparison of the Rectified Linear Unit (ReLU), leaky ReLU and softplus activation functions.

Figure 2.6: These figures compare common activation functions in ANN training. Figure 2.6b compares the ReLU function to a variant, leaky ReLU, which is not forced to zero for negative inputs and avoids the dying ReLU problem Lu et al. (2019). The softplus function is a smooth alternative to ReLU activation.

The Performance Measure, *P*

As a quantitative measure of any ML algorithm, *P* is necessary to evaluate its ability at given tasks and often must be specifically designed for every *T*. For classification, two common measures are the model's accuracy and error rate, i.e., the proportion of examples for which the model produces a correct or incorrect output, respectively. Alternatively, for regression and density-estimation tasks, a metric evaluating a continuous-valued score is more informative. Naturally, the choice of *P* is not always straightforward due to difficulty in selecting the feature or quantity to measure. A common design choice in a regression setting involves whether to heavily penalise a system for frequent, moderate mistakes or rare, significant mistakes (Goodfellow et al., 2016b).

Typically, the aim of ML is to extend the learnt algorithm to unseen examples with good performance, rather than limiting the scope to training examples, an ability referred to as *generalisation*. In order to evaluate an algorithm's generalisation capability, a test dataset is separated from the training dataset and exclusively used to compute the generalisation error. An underlying query pertaining to this approach questions how the performance on an unseen test set is improved through observations of the training set. The performance on both datasets is related, assuming that their samples are independent and identically distributed (i.i.d.) i.e., independent from each other in a statistical sense and randomly drawn from the same data generating distribution.

This relation promotes the conclusion that the expected performance of a randomly selected model is equal on both datasets. However, the ML process of sampling the training set, adapting model parameters to minimise training error and subsequently evaluating test error, implies that the expected test set error is greater than or at best, equal to the expected error on the training set. This is encapsulated by the conflicting goals of any ML algorithm (Goodfellow et al., 2016b):

- Minimise the training error
- Minimise the generalisation error (gap between training and test errors)

These two factors lead to the two main challenges of learning algorithms, i.e. *under-fitting* and *over-fitting*. Under-fitting refers to a setting where the model is not able to sufficiently minimise the training error, while over-fitting implies that the generalisation error is too large, often due to excessive focus on the training set. Altering a model's capacity, i.e., its ability to fit a variety of functions, is one method to control a model's tendency to under- or over-fit. For ANNs, a basic way of achieving this is to vary a network's depth and width. Modifying the performance of UL algorithms, where error or performance measures are often unavailable, is typically effected through varying an algorithm's hyper-parameters (Aggarwal, 2017; Xu and Tian, 2015).

A more widespread method for the reduction of generalisation error in SL algorithms is regularisation, which encompasses approaches that affect a model's preference for small weights (Nielsen, 2015b). Two specific methods, namely L1 and L2 regularisation, penalise a model's weights proportionately to their magnitude, in linear and quadratic fashion, respectively. These regularisation functions are implemented as modifications to an algorithm's cost function, which is iteratively minimised during training. Equations 2.7 and 2.8 define L1 and L2 regularisation respectively, where J_0 is the unregularised cost function, w represents weights in the trained function, λ is the regularisation parameter and N is the size of the training set.

$$J(w) = J_0 + \frac{\lambda}{2N} \sum_w \big|w\big| \tag{2.7}$$

$$J(w) = J_0 + \frac{\lambda}{2N} \sum_w w^2 \tag{2.8}$$

While this section has discussed the goals of ML methods and ways to reducing the conflict between these goals, specific functions to evaluate the prediction error have not yet been mentioned. A common function to compute the prediction error in regression tasks is the mean squared error (MSE), which is computed as the average of the squared differences between the feature label, y and the network's

prediction, $\tilde{y}$:

$$MSE = \frac{1}{N}\sum_{N} \|\tilde{y}(\boldsymbol{x}) - y(\boldsymbol{x})\|^2 \tag{2.9}$$

For classification tasks, the negative log-likelihood (NLL) cost, also referred to as the cross-entropy cost, is often a better choice for computing the error (Nielsen, 2015b):

$$NLL = -\frac{1}{N}\sum_{N} [y \ln(\tilde{y}) + (1-y)\ln(1-\tilde{y})] \tag{2.10}$$

In UL scenarios, such functions are not available due to the unlabelled nature of the provided data. The evaluation of these methods is carried out using either *internal* or *external* validity measures. Internal validity indicators consider information held within an algorithm, such as the mean radius of clusters, however they are unable to provide a direct comparison between different methods. A severe issue with these indicators is that their score may be manipulated through the selection of an algorithm that matches the measure applied. External measures consider the results of the UL method and quantify its precision and recall (Aggarwal, 2017; Xu and Tian, 2015).

As alluded to previously, ML algorithms include numerous settings that can be used to affect their learning behaviour, which are termed *hyper-parameters*. Commonly, the determination of values for these parameters is heuristic, however an additional, held-out validation dataset may also be used to make these model choices. A validation dataset is usually defined in the same manner as the test set, i.e. samples are taken from the data generating distribution (Goodfellow et al., 2016b).

2.3 Uncertainty Estimation with Neural Networks

A general use of the term *uncertainty* refers to all the situations where there is a lack of knowledge with regards to the outcome of a problem, its structure or the nature of the factors motivating the problem. An elementary example concerns a coin, which may be biased or unbiased. Intuitively, the result of its toss will result in either heads or tails with some probability, however, these probabilities, the real outcome of the toss, and whether the toss in itself is fair, are uncertain quantities prior to this toss.

The definition of the probabilities characterising this toss embodies the *aleatoric* or *data* uncertainty within this problem, which stems from inherent randomness within the environment or the agent. In machine vision applications, sensor or motion noise within observations, which are inherent to these problems, are categorised as aleatoric uncertainty (Clements et al., 2019; Kiureghian and Ditlevsen, 2009).

Two additional terms are introduced, *homoscedasticity* and *heteroscedasticity*. The former refers to constant uncertainties which do not vary with model inputs, such as sensor noise, while the latter encompasses varying, typically input-dependent, sources. A general property of such uncertainties is their recurrence, i.e., aleatoric uncertainty may be characterised, however it cannot be reduced or removed from a system, irrespective of the amount of data considered (Gal, 2016; Kendall and Gal, 2017; Segú et al., 2019).

A second form of uncertainty, present in most problems, is termed *epistemic* uncertainty. With reference to the prior example, consider a biased coin which yields strictly one of either heads or tails for every toss. Prior to the first flip, the uncertainty about its outcome is epistemic, however this is entirely eliminated after the flip is performed (Clements et al., 2019). An alternative term to epistemic uncertainty is *model* uncertainty, derived from the consideration that this accounts for doubt in model parameters, model structure or a general lack of knowledge about the problem at hand. Conversely to aleatoric uncertainty (or *data* uncertainty), given sufficient data, epistemic uncertainty may be nullified (Gal, 2016; Kendall and Gal, 2017; Kiureghian and Ditlevsen, 2009; Segú et al., 2019).

The amalgamation of epistemic and aleatoric uncertainties is defined as *predictive* uncertainty and encapsulates a model's certainty (or lack of) in its output (Gal, 2016). While the previous discussion

connotes a clear separation between uncertainties, practically, the classification of various sources into either the aleatoric or epistemic classes is unclear and reliant on the knowledge of the practitioner as a system expert (Kiureghian and Ditlevsen, 2009). Nevertheless, the distinction between uncertainty types is essential, since they enable specific and distinct functions.

2.3.1 Bayesian Modelling for Uncertainty Estimation

The Bayesian framework is widely regarded as a principled approach to quantifying uncertainty in neural network predictions. In order to determine the relationship between UE and Bayesian modelling, examining the practical significance of different uncertainty types may be beneficial. A model cannot be trained to predict either type of uncertainty, since no apparent labels exist for such a quantity. Any model must be able to infer this doubt through observation of its training data.

Aleatoric uncertainty is a measure of the network's confidence in the accuracy of its output. For example, if for a set of similar input data, the corresponding output labels differ significantly, a network's uncertainty should intuitively be large, since the predicted value will not match the supplied data. Conversely, if the output labels were identical or very similar, the network's predictions should be accurate to this data and thus, the resulting uncertainty would be small.

In terms of the Bayesian framework, aleatoric uncertainty is modelled by placing a distribution over a model's output, with the variance of this distribution quantifying the uncertainty. Conveniently, the estimation of this variance may also be accurately performed while learning a regression task through SL, with the objective of maximising the log likelihood of the target data. A more thorough discussion of this network configuration and the definition of its objective is presented in Section 2.3.3.

Epistemic uncertainty is an evaluation of a network's doubt in its predictions, which does not depend on the nature of its training data. In effect, such a network's uncertainty should be low when it is trained sufficiently on input samples similar to those it is being asked to formulate predictions for. Vice-versa, this uncertainty should be high for input samples largely dissimilar to the training dataset. Clearly, this uncertainty depends on an ANN's weights (Kendall and Gal, 2017).

Retaining the same Bayesian framework, epistemic uncertainty is estimated through inference over a model's parameters, given a set of training inputs $\boldsymbol{X}$ and corresponding output labels $\boldsymbol{Y}$. The inference goal is defined as finding the set of parameters $\boldsymbol{\theta}$ that is likely to have generated these outputs, i.e., finding the *posterior distribution* over the parameter space. When the model is used to generate output predictions for new data points, the inferred prediction variance is interpreted as the model's doubt.

Following the Bayesian approach, a distribution $p(\boldsymbol{\theta})$ is placed over the parameters, which embodies any prior knowledge on these values. This distribution is transformed as data is observed, capturing the more or less likely parameters defining these data points. The probabilistic model by which the outputs are generated from input data points, given $\boldsymbol{\theta}$, is termed the likelihood distribution $p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\theta})$ and for regression, it is often assumed to take the form of a Gaussian distribution. The posterior distribution is hence obtained using Bayes' theorem (Bertsekas and Tsitsiklis, 2008; Gal, 2016):

$$p(\boldsymbol{\theta} \mid \boldsymbol{X}, \boldsymbol{Y}) = \frac{p(\boldsymbol{Y} \mid \boldsymbol{X}, \boldsymbol{\theta})\, p(\boldsymbol{\theta})}{\int p(\boldsymbol{Y} \mid \boldsymbol{X}, \boldsymbol{\theta})\, p(\boldsymbol{\theta} \mid \boldsymbol{X}, \boldsymbol{Y})\, d\boldsymbol{\theta}} \tag{2.11}$$

Exact Bayesian inference for ANNs may be performed with Bayesian Neural Networks (MacKay, 1992), which replace individual, deterministic weights with distributions, and hence, output predictive distributions rather than point estimates. While such networks are appealing, the additional cost involved in training networks with distributions make these architectures infeasible over medium and large-scale settings (Blundell et al., 2015; Gal and Ghahramani, 2016; Kendall and Gal, 2017; Pearce et al., 2018a).

The impracticality of exact inference brings forth the necessity for approximate methods that are similar in complexity to *regular* ANNs, however retain a structured approach to estimating uncertainty. The following sections present common approaches for UE with ANNs.

2.3.2 Estimating Epistemic Uncertainty

Disagreement between Ensembles

A simple approach to estimating epistemic uncertainty involves computing the variance between a set of outputs obtained from an ensemble of deterministic models, such as networks of basis functions or ANNs (Clements et al., 2019; Gal, 2016; Menda et al., 2018; Tomczak et al., 2018). This project is concerned with the latter, however any model may be used to construct an ensemble, in the fashion shown in Figure 2.7.

The ensembling method is assumed to fit M models to one dataset, ensuring variability through bagging (training on different subsets of available data) or random initialisation, batch selection and other methods. Individual models produce M predictions $y_i(\boldsymbol{x}),\ i = 1, 2, \dots M$, given an input $\boldsymbol{x}$, which are combined to produce an ensemble mean $\bar{y}(\boldsymbol{x})$ and variance a $\sigma^2(\boldsymbol{x})$:

$$\begin{aligned} \mathbb{E}[y(\boldsymbol{x})] \approx \bar{y}(\boldsymbol{x}) &= \frac{1}{M}\sum_{i=1}^{M} y_i(\boldsymbol{x}) \\ \text{Var}[y(\boldsymbol{x})] = \sigma^2(\boldsymbol{x}) \approx \upsilon(\boldsymbol{x}) &= \frac{1}{M}\sum_{i=1}^{M}\left(y_i(\boldsymbol{x})^T y_i(\boldsymbol{x}) - \bar{y}(\boldsymbol{x})^T \bar{y}(\boldsymbol{x})\right) \end{aligned} \tag{2.12}$$

The ensemble variance is proposed to quantify the uncertainty, however the produced value may not always be accurate. For example, radial basis function networks predict zero for out-of-distribution data points, although the true output may be notably different. In these cases, the ensemble will have very high confidence (low uncertainty) in an incorrect output (Gal, 2016). A disadvantage of ensembling in this form is that the methodology is non-Bayesian, i.e., it does not perform Bayesian inference. Nevertheless, this method remains attractive due to its computational efficiency and scalability, in comparison to Bayesian approaches (Pearce et al., 2018a,b).

Disagreement between Anchored Ensembles

Ensembling ANNs, in the fashion described previously, forms an intuitive way of estimating epistemic uncertainty, however it has not received widespread acceptance. Although *vanilla* ensembling has seen some empirical success in Tibshirani (1996) and Lakshminarayanan et al. (2016), its non-conformance to (approximate) Bayesian inference has generated some criticism. Anchored ensembling (Pearce et al., 2018a,b) modifies this methodology by leveraging randomised Maximum A Posteriori (MAP) sampling to produce the desired Bayesian behaviour.

This approach returns estimates of desired parameters by adding a regularisation term to pre-defined loss functions. The addition of noise into such an augmented loss, followed by repeated sampling, produces a distribution of estimates comparable to a true posterior obtained through Bayesian inference. The most appealing quality of this method is its practicality - it does not induce any changes to normal ANN training, rather, it involves training a small set of networks in the customary fashion.

MAP estimation returns an evaluation of an unknown parameter that is equivalent to the mode of the posterior distribution about this quantity (Horii, 2019; Pearce et al., 2018a). For multivariate normal prior and likelihood distributions, $\mathcal{N}(\boldsymbol{\mu}_{prior}, \boldsymbol{\Sigma}_{prior})$ and $\mathcal{N}(\boldsymbol{\mu}_{like}, \boldsymbol{\Sigma}_{like})$ respectively, the resultant (multivariate normal) posterior distribution defines the MAP solution as (Petersen and Pedersen, 2012a):

$$\boldsymbol{\mu}_{MAP} = \boldsymbol{\mu}_{post} = \left(\boldsymbol{\Sigma}_{like}^{-1} + \boldsymbol{\Sigma}_{prior}^{-1}\right)^{-1}\left(\boldsymbol{\Sigma}_{like}^{-1}\boldsymbol{\mu}_{like} + \boldsymbol{\Sigma}_{prior}^{-1}\boldsymbol{\mu}_{prior}\right) \tag{2.13}$$

Anchored ensembling relies on the selection of a noise source and distribution to inject into this expression, such that the returned distribution approximates the true posterior. An amenable source is the prior mean, $\boldsymbol{\mu}_{prior}$, which is replaced by a noisy variable $\boldsymbol{\theta}_0 \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$. This distribution is defined with $\boldsymbol{\mu}_0 = \boldsymbol{\mu}_{prior}$ and $\boldsymbol{\Sigma}_0 = \boldsymbol{\Sigma}_{prior} + \boldsymbol{\Sigma}_{prior}^2\boldsymbol{\Sigma}_{like}^{-1}$, as derived in Pearce et al. (2018a,b).

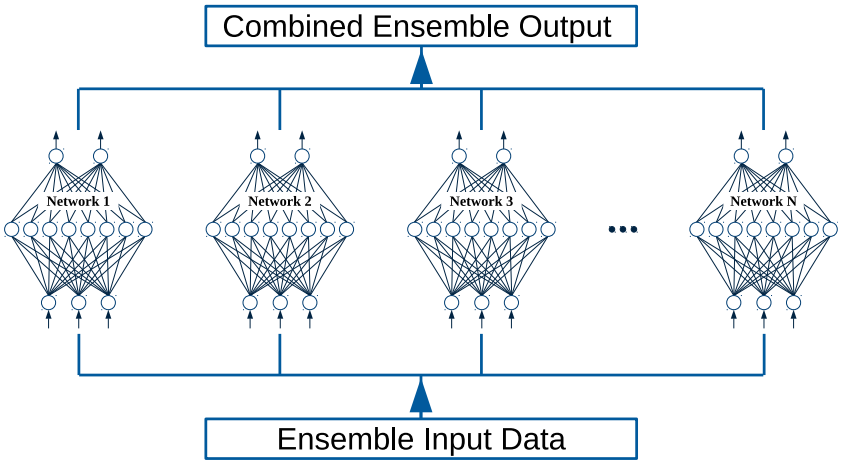


Figure 2.7: An ensemble of ANNs supplied with the same input data, whose output is combined, often through a mean or weighted average computation. Individual networks are often trained on different subsets of a dataset or uniquely diversified through random initialisation, different hyper-parameter selection and other methods.

This MAP inference may be implemented in ANNs performing both regression and classification tasks, with the former being predominantly more relevant to this project. The definition of the loss function for a network within an anchored ensemble departs from the established MSE loss with L2 regularisation:

$$\mathcal{L}_{regularised} = \frac{1}{N}\|y - \tilde{y}\|_2^2 + \frac{1}{N}\left\|\mathbf{\Gamma}^{1/2}\boldsymbol{\theta}\right\|_2^2 \tag{2.14}$$

where $\mathbf{\Gamma}$ is a diagonal, square matrix which allows different regularisation coefficients for every ANN layer (Pearce et al., 2018a). This is less restrictive than the typical formulation with a scalar coefficient, however, less commonly encountered in literature. A Bayesian view of parameters minimising this loss interprets them as MAP estimates with zero-centred, normal priors (MacKay, 1992). The inference scheme proposed above considers prior distributions centred at non-zero values, which are obtained with a small modification to Equation 2.14:

$$\mathcal{L}_{anchor} = \frac{1}{N}\|y - \tilde{y}\|_2^2 + \frac{1}{N}\left\|\mathbf{\Gamma}^{1/2} \cdot (\boldsymbol{\theta} - \boldsymbol{\theta}_0)\right\|_2^2 \tag{2.15}$$

The major concern in this method lies with the evaluation of $\mathbf{\Sigma}_0$ for ANNs, since likelihood variances and correlations in such models vary continuously during training and across parameters. Relying on strong inter-parameter correlations in ANNs, setting $\mathbf{\Sigma}_0 = \mathbf{\Sigma}_{prior}$ is shown to approximate the posterior distribution, with the inference accuracy improving as these correlations increase. Notably, strong parameter correlations are more likely with a greater number of parameters within the network (Pearce et al., 2018a,b).

Dropout as Bayesian Approximation

Bayesian modelling presents a mathematically-sound framework for uncertainty estimation, however, as presented previously, exact inference is not possible for most models other than particularly simple ones. This has led to the search of increasingly efficient models that may be cast as performing approximate Bayesian inference - one especially promising solution considers dropout training applied to deep feedforward, convolutional and recurrent ANNs.

Dropout (Hinton et al., 2012; Srivastava et al., 2014) is one of several regularisation techniques in deep ANNs applied to prevent over-fitting. The term refers to temporarily removing units (hidden and visible neurons) and connections from a network. During training, neurons are dropped randomly, with simple implementations basing this choice on a fixed probability p, either selected heuristically or using a validation data set. Training with dropout may be viewed as sampling a simplified ANN from the larger network at every time step. At test time, an approximate average of the outputs of the 'sampled' networks is obtained by using the full network with downscaled weights.

Monte Carlo (MC) dropout (Gal, 2016; Gal and Ghahramani, 2016) proposes to approximate the posterior over a network's parameters through repeated sampling of output predictions from multiple simplified networks obtained with dropout, given a single input (Segú et al., 2019). The dropout training objective is shown to be identical to that of Variational Inference (VI) (Gal, 2016; Jordan et al., 1999), a standard approximate Bayesian inference method. Both techniques may be viewed as minimising the Kullback-Liebler (KL) divergence (Kullback and Leibler, 1951) between an approximate distribution and the true posterior.

Network predictions for the expected model output (mean) and variance, quantifying the network's uncertainty, are obtained by extending the dropout process to the testing phase. Given a vector of input data, $\boldsymbol{x}$, a set of i.i.d. model outputs $\tilde{y}_i(\boldsymbol{x}), i = 1, 2, \dots M$ are obtained from M simplified ANNs and combined as:

$$\begin{aligned} \mathbb{E}[y(\boldsymbol{x})] \approx \bar{y}(\boldsymbol{x}) &= \frac{1}{M}\sum_{i=1}^{M}\tilde{y}_i(\boldsymbol{x}) \\ \mathsf{Var}[y(\boldsymbol{x})] = \sigma^2(\boldsymbol{x}) \approx \upsilon(\boldsymbol{x}) &= \tau^{-1}\boldsymbol{I} + \frac{1}{M}\sum_{i=1}^{M}\left(\tilde{y}_i(\boldsymbol{x})^T\tilde{y}_i(\boldsymbol{x}) - \bar{y}(\boldsymbol{x})^T\bar{y}(\boldsymbol{x})\right) \end{aligned} \tag{2.16}$$

which is identical to the association of results for a vanilla ensemble, as shown in Equation 2.12.

Intuitively, a notable advantage of this method is its ease of application. Epistemic uncertainty estimates may be obtained from any ANN formulation trained using dropout, without any structural modifications. A limitation of this approach is the unrealistic assumption of constant model noise τ^{-1} (Segú et al., 2019), which makes it insensitive to sources of aleatoric uncertainty. The inclusion of τ^{-1} in the variance computation implies that this method quantifies homoscedastic aleatoric uncertainty. Nevertheless, to truly capture aleatoric uncertainty, the noise parameter should be tuned (Kendall and Gal, 2017), hence MC Dropout is referenced as an epistemic uncertainty estimation method.

As described in Gal (2016), the same derivation employed to show the validity of dropout as an approximate Bayesian inference technique may be applied to other stochastic regularisation techniques, such as multiplicative Gaussian noise and dropConnect.

2.3.3 Estimating Heteroscedastic Aleatoric Uncertainty

Although the estimation of epistemic uncertainty presently remains an active area of research, a methodology for aleatoric uncertainty estimation, specifically for the FNN architecture, was proposed in Nix and Weigend (1994) and is widely applied in literature. Nix and Weigend (1994) maintain that paired input-target data (alternatively, state-action pairs in the robotics field) contains noise, thus for an input $\boldsymbol{x}$, the collected data is defined as:

$$d(\boldsymbol{x}) = f(\boldsymbol{x}) + n(\boldsymbol{x}) \tag{2.17}$$

where $f(\boldsymbol{x})$ is the true function generating said data and $n(\boldsymbol{x})$ is data-dependent, additive noise. Defining observation noise as a function of the data itself is the crucial distinction between typical FNNs, where it is fixed and often, ignored. As this noise is related to data, it may be learnt by the network (Kendall and Gal, 2017).

In a function-approximation task, a network's output $y(\boldsymbol{x})$ may be interpreted as an estimate of the true mean $\mu(\boldsymbol{x})$ of this noisy target distribution, given an appropriate model for $n(\boldsymbol{x})$. An estimate of the degree of this noise, as the variance of the error distribution about $f(\boldsymbol{x})$, is often desirable and represents the network's uncertainty about its prediction. An estimate, $\upsilon(\boldsymbol{x})$, of the variance $\sigma^2(\boldsymbol{x})$, quantifying heteroscedastic uncertainty (Kendall and Gal, 2017), may be obtained alongside the mean prediction from a single network.

Obtaining an additional estimate from an FNN requires modification to its architecture, as shown for the uni-dimensional output case in Figure 2.8. The network's hidden layer is divided to account for

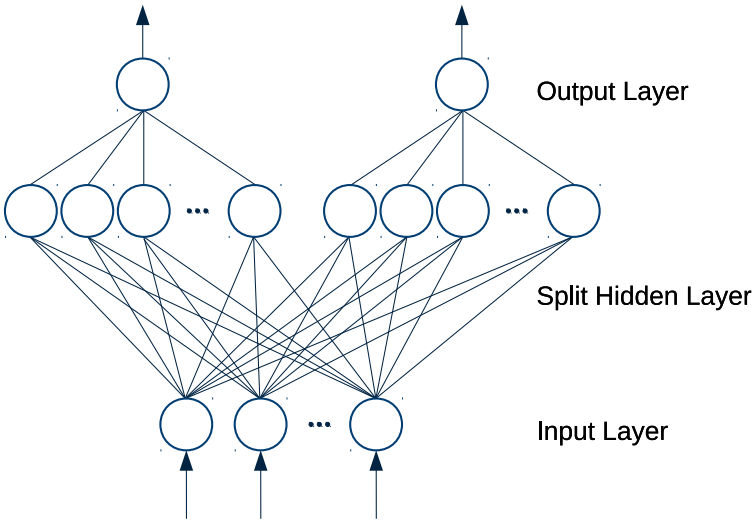


Figure 2.8: A visual representation of the split-network architecture proposed in Nix and Weigend (1994) for the estimation of heteroscedastic uncertainty. This configuration may be extended to include additional hidden layers and a multi-dimensional output layer, which must be accompanied by appropriate modification of the training criterion.

an added output neuron estimating $\sigma^2(\boldsymbol{x})$, however the input layer is unchanged and common to both segments. A similar change may be applied in a multi-dimensional output scenario, while additional hidden layers in either a split or unsplit configuration may be added in either scenario. As the uncertainty must always be a positive value (being a variance estimate), the activation function for the neurons estimating $\upsilon(\boldsymbol{x})$ must enforce this constraint. Nix and Weigend (1994) use the exponential function, however in this work, the softplus function $\ln(1+e^x)$, which is plotted in Figure 2.6b, was preferred. In a multi-dimensional setting, the predicted uncertainty matrix must be positive semi-definite - exploiting the Cholesky decomposition is an effective way of enforcing this property.

A loss function, $\mathcal{L}$, is derived for this FNN based on maximum-likelihood estimation and the assumption of i.i.d. input data, $\boldsymbol{x}$. A form for this loss is obtained by defining the network's goal as the maximisation of the log likelihood of the target values, thus:

$$\mathcal{L} = \sum_i \ln P(d_i \mid \boldsymbol{x}_i, \mathfrak{N}) \tag{2.18}$$

where $\mathfrak{N}$ denotes the split FNN being trained.

Furthermore, by assuming a normal prior distribution for $n(\boldsymbol{x})$ around $f(\boldsymbol{x})$, the target probability distribution may be expressed as:

$$P(d_i|\boldsymbol{x}_i, \mathfrak{N}) = \frac{1}{\sqrt{2\pi\sigma^2(\boldsymbol{x}_i)}} \exp\left(\frac{-\left[d_i - y(\boldsymbol{x}_i)\right]^2}{2\sigma^2(\boldsymbol{x}_i)}\right) \tag{2.19}$$

After including this result in Equation 2.18, taking the natural logarithm on both sides of the resulting expression and negating, again on both sides, the network loss function is defined as:

$$\mathcal{L}_{uni-dim} = \frac{1}{2N}\sum_{i=1}^{N} \frac{\left[d_i - y(\boldsymbol{x}_i)\right]^2}{\sigma^2(\boldsymbol{x}_i)} + \ln[\sigma^2(\boldsymbol{x}_i)] \tag{2.20}$$

Upon training with this criterion, an FNN is able to approximate the underlying function $f(\boldsymbol{x})$ and quantify its uncertainty on this prediction, albeit in the restrictive scenario of a uni-dimensional output. Gal (2016) extends this formulation to the multi-dimensional setting, however his proposal considers model precision τ, i.e., the inverse of the uncertainty term υ. Regardless, a simple inversion restores the predicted quantity to the desired uncertainty value.

To predict heteroscedastic uncertainty for a multi-dimensional output, the network loss function from Equation 2.20 is adapted to:

$$\mathcal{L}_{multi-dim} = \frac{1}{2}\left(\boldsymbol{d}(\boldsymbol{x}) - \boldsymbol{y}(\boldsymbol{x})\right)\boldsymbol{\tau}(\boldsymbol{x})\left(\boldsymbol{d}(\boldsymbol{x}) - \boldsymbol{y}(\boldsymbol{x})\right)^T - \frac{1}{2}\ln\left[\det\left(\boldsymbol{\tau}(\boldsymbol{x})\right)\right] \tag{2.21}$$

$$= \frac{1}{2}\frac{\left(\boldsymbol{d}(\boldsymbol{x}) - \boldsymbol{y}(\boldsymbol{x})\right)\left(\boldsymbol{d}(\boldsymbol{x}) + \boldsymbol{y}(\boldsymbol{x})\right)^T}{\boldsymbol{v}(\boldsymbol{x})} + \frac{1}{2}\ln\left[\det\left(\boldsymbol{v}(\boldsymbol{x})\right)\right] \tag{2.22}$$

3

Conflict Value Analysis and Uncertainty Estimation for Issue Detection in Demonstration Data

In this chapter, two distinct approaches are presented for the identification of conflicting data and sub-optimal demonstrations within demonstration datasets acquired during policy training, in this setting specifically with HG-DAgger. The first methodology proposed in this chapter posits that a measure of conflict between individual datapoints may be computed by exclusively examining the underlying dataset, therefore, without consideration of the learnt policy.

The second approach estimates the learning uncertainty induced by sub-optimal demonstrations during the learning phase through the application of the Uncertainty Estimation (UE) methods presented in Section 2.3. Counter to the previous approach, UE uses the learnt policy to perform this estimation. The characteristics of these methods and consequentially, the differences in their performance, are examined in the latter part of this chapter.

3.1 Overview

Expert demonstrations play a fundamental role in Learning from Demonstration (LfD) methods and are necessary to teach a policy to a novice agent. In general, all LfD approaches including primitive Behavioural Cloning (BC) and iterative, DAgger-like methods, assume that the demonstrator is a domain expert and infallibly provides optimal demonstrations. Nevertheless, it is recognised that a human expert may not fulfil this requirement, i.e., demonstrations from a human user may be sub-optimal and contain undesired trajectories or conflicting data points that hinder effective task completion.

As previously noted in Chapter 1, the detection of issues within a set of demonstration data remains an unexplored area in this field (Osa et al., 2018). Ledesma et al. (2018) propose a deterministic method to compute a measure of the conflict between data points within datasets compiled for Supervised Learning (SL), however this is not linked with a methodology for the removal or correction of data, which is the objective of this work. In the context of LfD and the objectives of this project, it is worthwhile to re-examine this approach. Uncertainty estimation is a structured approach for the identification of imperfections in Artificial Neural Network (ANN) training, thus it is logical to examine the applicability of this approach to LfD.

3.2 Methodology

This section illustrates the development of the algorithms cited previously, which are intended for the analysis of datasets and the identification of learning conflicts within them. As the theoretical underpinnings of most of these methods have already been discussed in Chapter 2, this section explores the practical aspects of these algorithms and their implementation.

3.2.1 Analysis of Datasets with Learning Conflicts

The algorithm proposed by Ledesma et al. (2018) is designed to quantify conflict values between data points in a dataset intended for supervised ANN training, thus every data sample consists of an input-target (or, from a robotics perspective, state-action) pair. A learning conflict is encountered when two similar input samples have different target values. Conversely, for two such input samples with comparable target values, there is no conflict.

In order to generalise the algorithm and ensure its applicability to various datasets, the data in question is normalised such that the range of values is controlled and does not impact performance. A set of N-dimensional input values $\boldsymbol{X} \in \mathbb{R}^{M,N}$ is hence rescaled such that individual elements $x_{i,j} \in [0, 1]$ for $i = 1, 2, \dots M$ and $j = 1, 2, \dots N$, where M denotes the number of data points in a set. A similar computation is performed over the set of target entries, $\boldsymbol{y}$, which are assumed to be one-dimensional.

Subsequently, the algorithm computes a degree of similarity between the input entries in the dataset and a degree of difference between target entries. The similarity between states, δ, is computed using the normalised Euclidean distance:

$$\delta_{ij} = \sqrt{\frac{1}{N} \sum_{k=1}^{N} \left(\bar{x}_{ik} - \bar{x}_{jk}\right)^2} \tag{3.1}$$

while the dissimilarity between target values is defined as the difference between two normalised values, thus $\epsilon_{ij} = |\, \bar{y}_i - \bar{y}_j \,|$. In these expressions, $\bar{x}$ and $\bar{y}$ denote normalised input and target values. The computation of a distance value between every pair of data points leads to two $M \times M$ distance matrices $\boldsymbol{\Delta}$ and $\boldsymbol{E}$, with zero values along the leading diagonal in each case.

The final part of this algorithm involves the combination of these two quantities into a measure that meets the pre-established criteria, i.e., produces a minimal value when ϵ is small and an increasingly larger value as ϵ increases, with a maximum value when $\epsilon = 1$ and $\delta = 0$. An effective way of combining these two values into a single conflict measurement is $c_{ij} = \epsilon_{ij} W(\delta_{ij})$, where $W(\delta_{ij})$ is a weight function such that $0 \leq W(\delta_{ij}) \leq 1$ with a peak for $\delta_{ij} = 0$. As $\delta_{ij} \to 1$, this function should decrease and approach zero.

A candidate for this weighting function is the Gaussian function

$$W(\delta_{ij}) = \alpha \exp\left[-\frac{(\delta_{ij} - \beta)^2}{2\sigma^2}\right] \tag{3.2}$$

for arbitrary real constants α, β and σ. These constants define the shape of the Gaussian function, with σ, which defines its width, retained as a parameter within the algorithm. In this scenario, the desired maximum magnitude of this function is unity, hence $\alpha = 1$, while β, which defines the central point of the Gaussian function, should be zero, leading to the revised weight function:

$$W(\delta_{ij}) = \exp\left[-\frac{\delta_{ij}^2}{2\sigma^2}\right] \tag{3.3}$$

The above expressions may be combined to form a direct expression for the conflict value c_{ij}, which defines an $M \times M$ matrix of values, $\boldsymbol{C}$. Individual elements are computed as:

$$c_{ij} = |\,\bar{y}_i - \bar{y}_j\,| \exp\left[\frac{1}{2N\sigma^2}\sum_{k=1}^{N}(\bar{x}_{ik} - \bar{x}_{jk})^2\right] \tag{3.4}$$

This method, which shall henceforth be referred to as the Conflict Value Algorithm (CVA), is summarised in Algorithm 2.

Algorithm 2: Conflict Value Analysis (Ledesma et al., 2018)

1: **procedure** Conflict Value Algorithm($\mathcal{D}$, σ)
2: Rescale input data such that input entries lie in the range $[0, 1]$
3: Rescale target data such that target entries lie in the range $[0, 1]$
4: Compute a degree of similarity between input data points
5: Compute a degree of difference between target data points
6: Compute the conflict value between data points
7: **return** Matrix of conflict values $\boldsymbol{C}$
8: **end procedure**

At this stage, it must be emphasised that such a matrix $\boldsymbol{C}$ generally provides an excessive $(\frac{M}{2} - 1)$[1] amount of values for every data point, which individually relay very little meaningful information, particularly for similar input and target values. An additional metric must be applied to produce a tractable quantity indicative of this metric - taking the maximum conflict value for each data point is sufficient. Furthermore, the average conflict value over the matrix $\boldsymbol{C}$ may be applied to assess and compare conflicts between two or more datasets.

3.2.2 Uncertainty Estimation

A General Machine Learning (ML) Framework

A primary step towards the implementation of the epistemic and aleatoric UE methods described in Section 2.3 was the definition of a framework for Feedforward Neural Network (FNN) training. This framework was coded in Python and based on the Tensorflow 2 ML platform (Abadi et al., 2015). Incorporating all necessary functions for network training, obtaining predictions, passing and retrieving data to and from this network and other ancillaries, this structure facilitated the effortless implementation of UE on top of this base network. Moreover, this framework established various parameters important in FNN applications, such as the number of hidden layers and neurons per layer, activation functions and the optimisation method applied during training. While these settings were easily varied, two hidden layers with a Rectified Linear Unit (ReLU) or leaky ReLU activation, trained with the Adam optimiser (Kingma and Ba, 2014), were found to work optimally.

Anchored Ensembling

The creation of an anchored ensemble to estimate epistemic uncertainty largely built upon this framework, with the most significant modification being the creation of multiple instances of the base FNN and handling these networks to form an ensemble, as shown in Figure 2.7. It follows that managing such a structure increased the network's complexity and training time.

This UE approach necessitated the introduction of L2 regularisation into the training procedure and subsequently, the incorporation of the *anchoring* term, $\boldsymbol{\theta}_0$, in the regularised loss function, as shown in Equation 2.15. As is outlined in Pearce et al. (2018b), randomly initialising individual network's weights and adopting these values as $\boldsymbol{\theta}_0$ is sufficient for the ensemble's uncertainty predictions to approximate the posterior data distribution, as desired, as the correlations between parameters increase.

[1] Given that $c_{ij} = c_{ji}$ and $c_{ii} = 0\,\forall\, i, j$, for M data points, $\boldsymbol{C}$ provides $\frac{M}{2} - 1$ unique conflict values for every datum.

Epistemic UE through Dropout

The application of dropout for UE was generally straightforward - whilst dropout regularisation was easily implementable, its extension to network predictions was more involved, given that this is not a common use of this methodology. Nevertheless, these were the only modifications required to adapt the basic framework to this application, alongside the uncertainty computation function, which outputs the variance of *T* sampled action predictions from the FNN.

Aleatoric Uncertainty Estimation

Aleatoric UE required rather less discreet modifications to the aforementioned FNN framework, the most fundamental being the addition of hidden layers and an output layer to form the split-network architecture visualised in Figure 2.8. Consequently, the network's loss function was modified to train both outputs, i.e., training both the value and uncertainty predictions simultaneously. An initial implementation of this methodology with a uni-dimensional output space was validated to perform as desired, however, the extension of this approach to a multi-dimensional space required additional modifications to ensure its correct operation.

Naturally, once the data dimensionality is increased beyond unity, the prediction of a covariance matrix is required, which results in n^2 elements for a n-dimensional output space. Furthermore, this matrix must be symmetric and positive semi-definite (Bishop, 2006a; McAllester, 2007), while the precision matrix defined in Gal (2016) must be positive definite. A convenient way to ensure this property is the exploitation of the Cholesky decomposition (Horn and Johnson, 2012; Petersen and Pedersen, 2012b)

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T \tag{3.5}$$

where $\boldsymbol{A}$ is a Hermitian, positive definite matrix and $\boldsymbol{L}$ is a lower triangular matrix with real and positive diagonal entries. While the theorem primarily refers to the decomposition of $\boldsymbol{A}$ into two matrices, the converse also holds trivially. Thus, forming a lower triangular matrix from $\frac{n(n+1)}{2}$ real and positive variables and multiplying by its transpose yields a positive definite matrix, while also reducing the number of values the network must estimate. As for the uni-dimensional setting, the softplus activation function was applied at the network's output to ensure the predictions were indeed positive.

At this point, it is apt to note that multi-dimensional UE with dropout and anchored ensembles did not require the same treatment, since epistemic uncertainty is computed as the variance of multiple samples of a network's output, which may easily be set to any arbitrary dimension. In practical terms, the uncertainty was evaluated using readily available functions.

While validating the implementation of this approach, the network's value (alternatively, action) prediction was observed to incorrectly tend towards a constant value, notwithstanding the uncertainty prediction's correctness. This behaviour was caused due to the concurrent training of both halves of the network, which was unable to meet both prediction goals. Addressing this issue required separating the training procedure into two distinct processes - first exclusively training the value prediction with a mean squared error (MSE) loss function and subsequently fixing the relevant network parameters, followed by training the remaining branch of the network with loss function defined in Equation 2.21.

3.3 Experiments and Results

The CVA and UE algorithms discussed previously were validated using synthetic datasets devised explicitly for such testing and more realistic datasets created through the use of customised environments defined with OpenAI Gym (Brockman et al., 2016). These basic environments were defined to include equivalent action choices in order to create data conflicts in the resulting dataset.

The first of two gym environments consists of a simple, one-dimensional setting in which a black, rectangular agent, shown in Figure 3.1, may move left or right on a horizontal plane and is initialised in a region enclosed by two vertical lines. The objective of this agent is to reach either of two red

crosses, placed at either end of this space, in the fastest manner possible. There is no restriction on the agent's movement and it may freely choose to move in either direction. The second gym environment, visualised in Figure 3.2, represents a classic obstacle avoidance setting. A black, rectangular agent may move freely within the region and is initialised close to its bottom border. The agent's objective is to reach the upper green border without colliding with the gray obstacle or other boundaries.

3.3.1 Uni-dimensional Validation

A preliminary assessment of the performance of the implemented UE methods was performed with a small set of one-dimensional data, created using the non-linear function $x * \sin x$. Around twenty training points, shown in red in the following plots, were used to shape policies for the epistemic and aleatoric uncertainty-predicting networks detailed in Chapter 2. A significantly larger and uniform dataset was used to obtain predictions from each network, which are shown in Figures 3.3, 3.4 and 3.5.

The first figure in this set validates the epistemic UE methods considered in this project, namely *anchored ensembling* and *dropout*. To re-iterate, dropout in this scenario is retained during the network's prediction phase, rather than only applied during training, as is customary. In both Figure 3.3a and

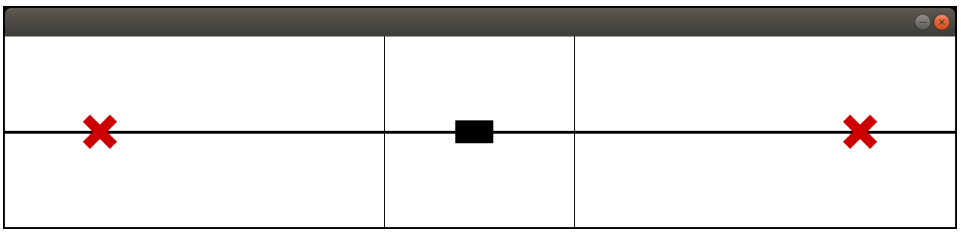


Figure 3.1: One of two validation environments for the Option Class (OC) algorithm. The black, rectangular agent may move left or right on the horizontal line and is initialised in the region enclosed by the two vertical lines. The objective of this agent is to reach either of the two red crosses in the fastest manner possible.

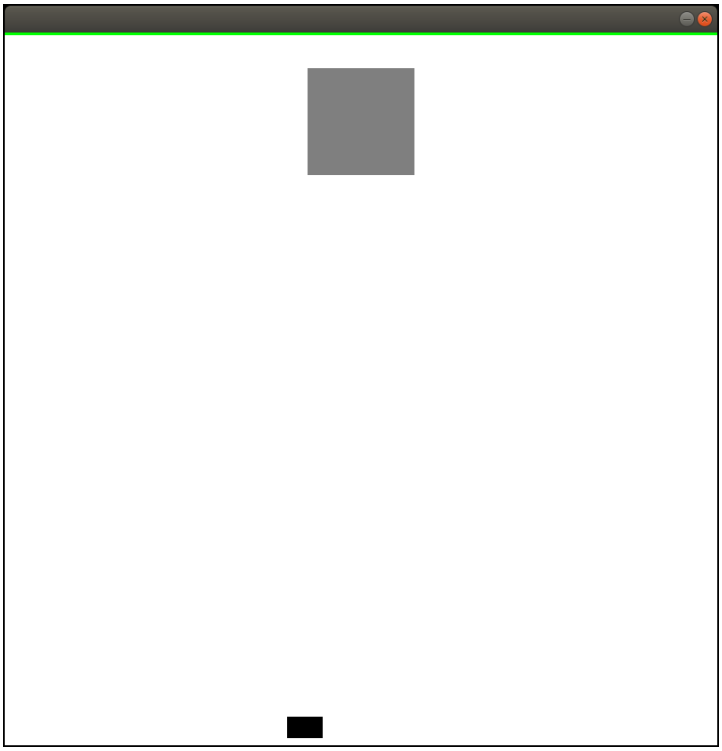
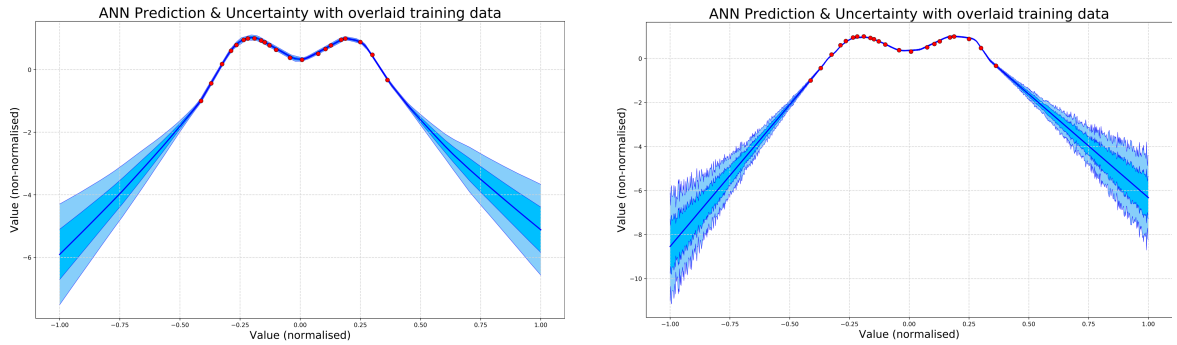
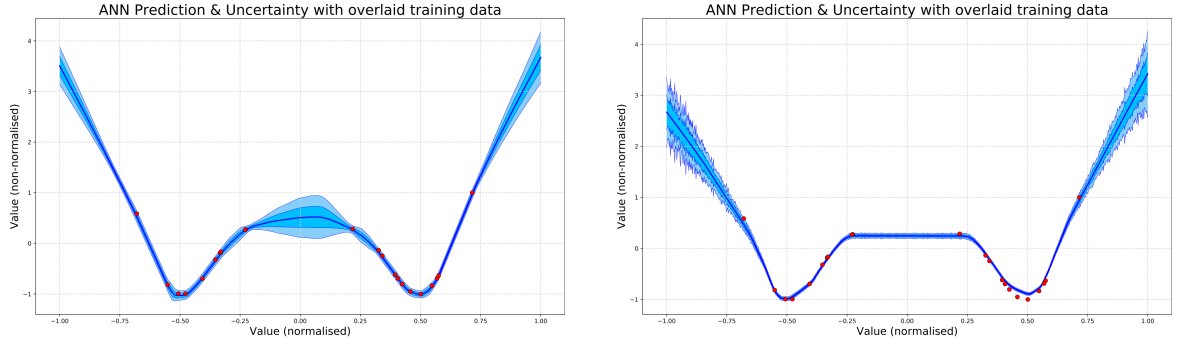


Figure 3.2: One of two validation environments for the OC algorithm. The black, rectangular agent may move freely within the region and is initialised close to the bottom border. The objective of this agent is to reach the upper green border without colliding with the gray obstacle or any other boundary.



(a) Epistemic UE with an anchored ensemble **(b)** Epistemic UE with dropout

Figure 3.3: Estimation of epistemic uncertainty with training data points overlaid (red dots) on the ANN predictions. In both plots, the trained policies accurately predict values close to the training data with low uncertainty and recognise their lack of training on additional points which are predicted with high uncertainty.



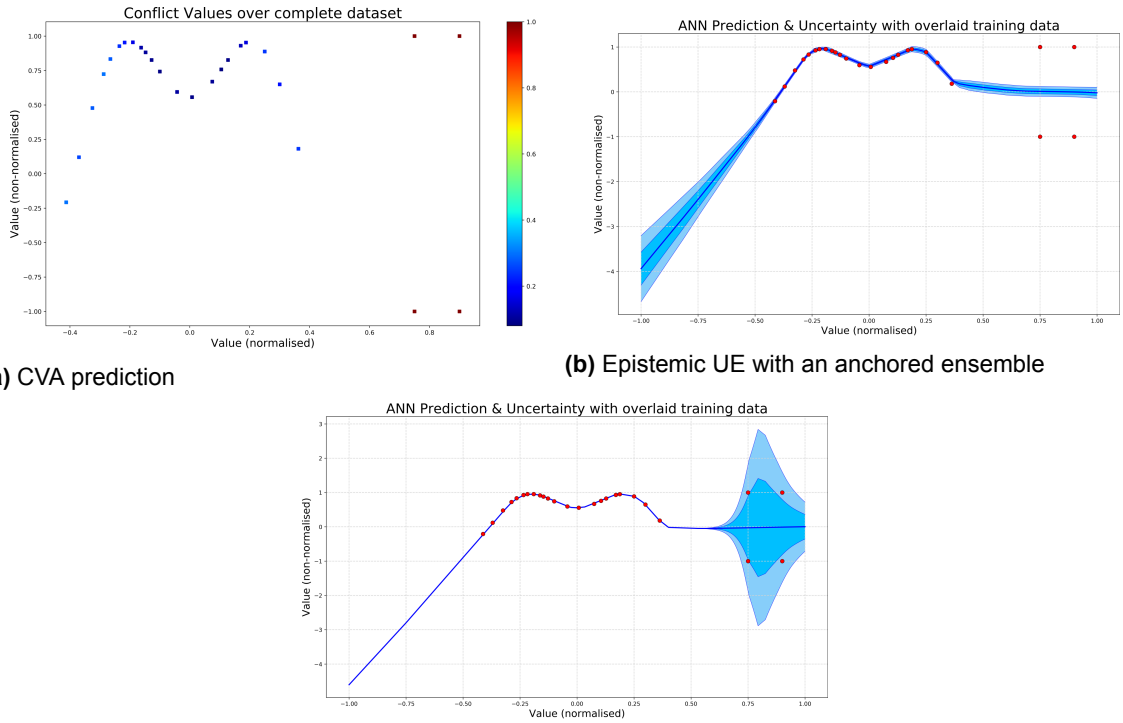
(a) Epistemic UE with an anchored ensemble **(b)** Epistemic UE with dropout

Figure 3.4: Estimation of epistemic uncertainty with training data points overlaid (red dots) on the ANN predictions. In both plots, the trained policies accurately predict values close to the training data with low uncertainty, however in Figure 3.4b fails to recognise its lack of training for points in the range $[-0.25, 0.25]$.

3.3b, the networks' predictions are clearly increasingly uncertain as they diverge from the training data points. This is the intended result with model UE, which is expected to indicate regions where the agent has not been trained to assimilate supplied data. A similar output is noted in Figure 3.4a and 3.4b.

A notable difference between the predictions with anchored ensembling and those with dropout is their smoothness, which arises from each algorithm's underlying architecture. The former trains multiple networks and computes its uncertainty as the standard deviation between their outputs. Given that individual predictions are continuous and smooth over a set of values, the resultant value, based on a small number of similar values, is intuitively expected to be smooth. Conversely, dropout is applied over a single network and predictions are performed after randomly eliminating neurons from the network. While the predictions from the complete network (without dropout) are smooth, the exclusion of different random elements for each data point causes a non-uniform change in the network, in turn causing the observed behaviour.

Another difference between these methods which is more significant in terms of the UE objective can be observed in plots in Figure 3.4. The training data in these plots is split in two distinct regions, with an empty central area. The expected result for this scenario would be an increased uncertainty in this central region, which is observed in the predictions of the anchored ensemble but not in the predictions of the dropout network. Once again, this variation is attributed to varying architectures - where networks within the anchored ensemble intrinsically vary from each other where training data is lacking, differences induced from dropout within this limited central region do not produce an appreciable increase in uncertainty.



(a) CVA prediction

(b) Epistemic UE with an anchored ensemble

(c) Aleatoric UE with split-network architecture

Figure 3.5: A comparison of conflict values, epistemic and aleatoric UE for a dataset with conflicting training data. The anchored ensemble does not associate the conflicting data with high uncertainty, whereas the split-network architecture outputs a significantly large uncertainty in this region. The computed conflict values are comparable to the latter, with the four rightmost points denoted as highly conflicting.

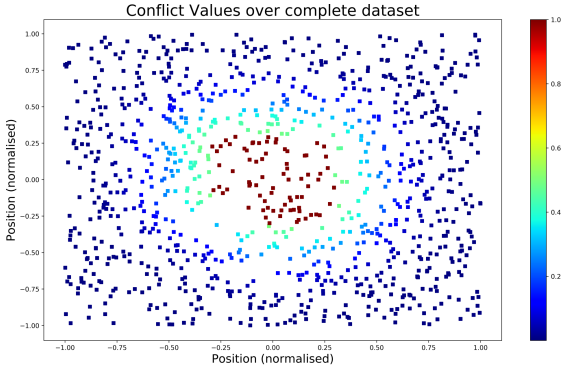
Figure 3.5 compares the CVA, epistemic UE (with anchored ensembling) and aleatoric uncertainty, estimated through the split-network architecture described in Section 2.3.3 for a training dataset with conflicting actions. The CVA prediction is shown in Figure 3.5a, which clearly indicates that the four right-most data points are conflicting (in red, with a large conflict value) while the remaining data points are less so (in blue, with a significantly lower conflict value). Furthermore, as anticipated, anchored ensembling does not predict an increase in uncertainty for conflicting training data, as can be observed in the right-most segment of Figure 3.5b. Conversely, the split-network architecture is highly uncertain in this region and notably certain in its predictions over the remainder of the state-space.

3.3.2 Multi-dimensional Experiments

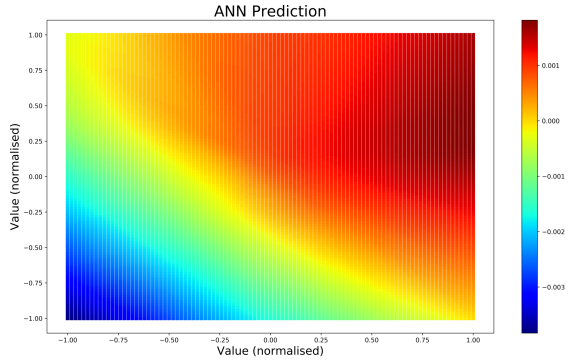
A more in-depth assessment of the algorithms proposed in this chapter was carried out with multi-dimensional validation data and datasets obtained from the aforementioned environments. The following set of plots in Figure 3.6 validate the aleatoric UE algorithm and CVA in a two-dimensional setting with a sparse dataset which includes conflicting data points. As shown in Figure 3.6a, the training data consists of a majority of zero-valued points and a central region of blue points with magnitude '-1'. An alternative set of data with magnitude '+1' is concealed beneath these points, due to visualisation restrictions. Figure 3.6b displays the output from the CVA, which attributes a high conflict value to the data points in the central region and a low conflict for all remaining points in the dataset.

Figures 3.6c and 3.6d show the value and uncertainty predictions, respectively, from the split-network architecture applied to estimate aleatoric uncertainty. The former produces values very close to zero throughout, as expected, with minor variations. The uncertainty prediction is similar to the CVA prediction, with a high uncertainty centrally and low uncertainty elsewhere. Intuitively, a high uncertainty (or conflict value) is expected for input points with both '+1' and '-1' values, which represent an ambiguous situation for the trained algorithm. Additional points are expected to be non-conflicting and thus, the network should be certain in its predictions.

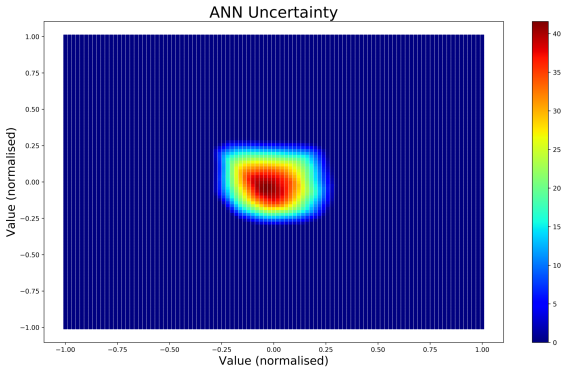
(a) Training dataset



(b) CVA prediction



(c) ANN prediction with split-network architecture

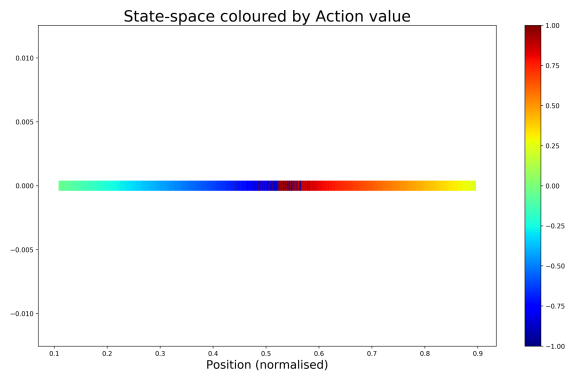


(d) Aleatoric UE with split-network architecture

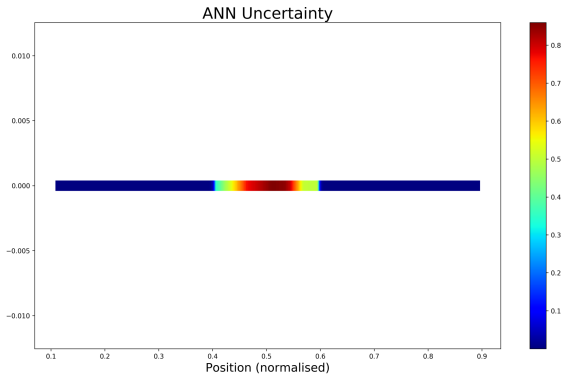
Figure 3.6: A comparison of computed conflict values and aleatoric uncertainty for training data with conflicting target values. Figure 3.6a shows the training data with border points set to zero and central points (in blue) set to -1. Additional data points in this central region set to $+1$ are obscured. The network prediction is approximately zero as anticipated, while high conflict and high uncertainty are observed in the central region of the respective plots.

Lastly, the following plots exhibit generated data and corresponding UEs from the aforementioned OpenAI Gym environments. Figure 3.7 contains the results from the one-dimensional, two-goal environment - the generated data contains numerous conflicts in its central region, corresponding to the agent's range of initialisation positions, shown in Figure 3.1. Figure 3.7b contains the corresponding uncertainty estimates, with a high uncertainty predicted for this central area and low uncertainty in the remainder of the space.

Figure 3.8 similarly presents the generated data and uncertainty predictions for the second environment, visualised in Figure 3.2. The actions displayed in Figure 3.8a correspond to the magnitude of the lateral movement of the agent (i.e., actions defining the agent's movement right or left), while the second action dimension, which controls movement forward and backward is excluded, since it contains minimal conflicts and the corresponding uncertainty predictions are low over the complete space. The network trained on the visualised data predicts a low uncertainty for most data points, while a high uncertainty centrally corresponds to a region with adjacent actions of differing magnitudes, as can be observed in Figure 3.8b.

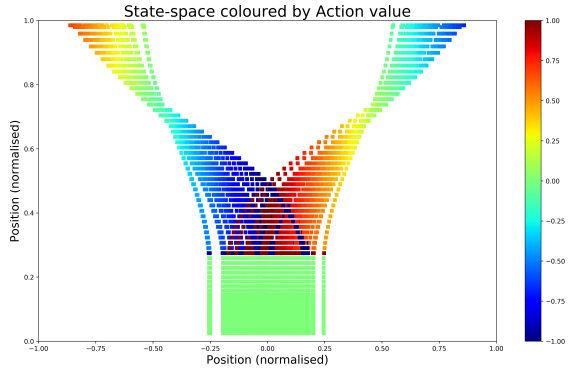


(a) Training dataset

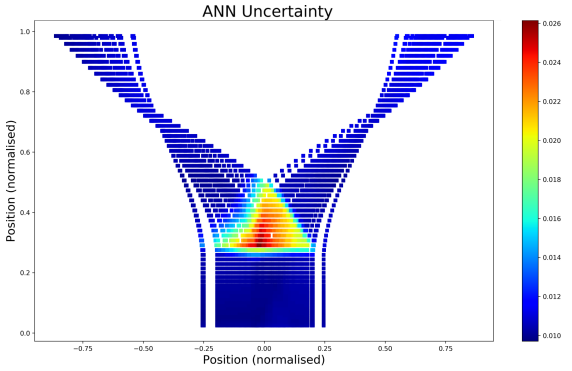


(b) Aleatoric UE with split-network architecture

Figure 3.7: A visualisation of a dataset with learning conflicts and the predicted aleatoric uncertainty ensuing from training an ANN on this data. Figure 3.7a depicts the training data, while Figure 3.7b is the resulting heteroscedastic aleatoric UE, which notably produces large values in the central region with the most conflicting data.



(a) Training dataset



(b) Aleatoric UE with split-network architecture

Figure 3.8: A visualisation of a dataset with learning conflicts and the predicted aleatoric uncertainty ensuing from training an ANN on this data. Figure 3.8a depicts the training data, while Figure 3.8b is the resulting heteroscedastic aleatoric UE, which notably produces large values in the central region with the most conflicting data.

3.4 Discussion

An evaluation of the methodology and results presented previously, combined with additional validation tests, some of which are included in Appendix A, yields a multitude of observations about the approaches explored in this chapter. Firstly, the most apparent conclusion is that both the CVA and heteroscedastic aleatoric UE achieve the desired results and are capable of effectively detecting conflicts in data. As remarked previously, epistemic UE is not aimed at and hence, does not capture these conflicts, thus is not applicable to this objective and will not be pursued further in this project.

The Conflict Value Algorithm is a deterministic method and thus produces a repeatable result, as opposed to UE, whose results depend on the network's weights and a stochastic learning process, namely Stochastic Gradient Descent (SGD). Nevertheless, this repeatability comes at the cost of the capability for generalisation, which is a feature of UE but not of CVA, which implies that this approach can only infer on data within the assessed data set.

In the description of the CVA in Section 3.2, reference was made to an assumption of a uni-dimensional action space, which would be a significant disadvantage for this method, in comparison to the UE approach. Nevertheless, a multi-dimensional action space may be evaluated with this algorithm by simply considering each dimension independently or replacing the action dissimilarity ϵ_{ij} with a multi-dimensional alternative, such as an inverse distance measure. In this project, it was sufficient to use the former approach, given the limited dimensionality considered, however the latter would be useful with larger action spaces due to the computational complexity introduced by this algorithm.

This is a fundamental disadvantage of the CVA, which must compare every point with every other, thus introducing an $\mathcal{O}(N^2)$ complexity. While the UE approach must train an FNN for a number of iterations, the training time of the CVA typically exceeds this and moreover, increases as the dataset grows in size. Generally, training an FNN to predict uncertainties is faster than directly evaluating the conflicts within a dataset.

All in all, both approaches have merit and meet the objective set at the beginning of this project, i.e., identifying regions of high uncertainty and conflicting data within a dataset. These methods have been shown to work on both synthetic data and demonstration datasets acquired from simulation environments, again as set out in Chapter 1. Notwithstanding, the UE approach is preferred due to its capability to predict action values and uncertainties, shorter computation time and versatility to any data-generating environment. The CVA will still be applied for validation and analysis of future results, however will not be retained as the principal methodology for the assessment of demonstration data.

4

Dataset Conflict Resolution with Option Classes

This chapter introduces the Option Class (OC) algorithm proposed by Chernova and Veloso (2008) and presents a number of modifications to this approach, extending its applicability to the Feedforward Neural Network (FNN) and in turn, making this algorithm scalable to a greater number of environments. Additional changes to this algorithm are described in detail in Section 4.2, while the latter parts of this chapter present and discuss the results obtained from experiments validating this methodology.

The data considered henceforth is presumed to take the form of input-target points, as required for Supervised Learning (SL) and specifically Learning from Demonstration (LfD) approaches. This data shall be observed from the frame of reference of the robotics field, thus the terms state and action shall be mentioned frequently and interchanged with the terms input and target, respectively.

4.1 Overview

After the detection of conflicting data points within a demonstration dataset as presented in Chapter 3, a methodology to resolve these conflicts by removing, replacing or altering this data is desired to allow for training with such datasets. In effect, this is a re-statement of the scope of this project and is explored in this chapter.

The algorithm proposed hereunder leverages the qualities of FNNs, namely their capability for generalisation and scalability, and thus extends its applicability to environments with large (multi-dimensional) state and action spaces. This feature is not a property of the original OC algorithm proposed in Chernova and Veloso (2008), which is based on a Gaussian Mixture Model (GMM) classifier and is notably more computationally demanding than an FNN for such scenarios.

A transition from a classifier to a regression-based function approximator was performed to avoid classifiers' inherent limitation of only selecting a single outcome at any time. The selection of multiple actions may be performed by designing certain outputs to imply the performance of multiple actions, nevertheless this would cause the action space to grow significantly, eventually making problems intractable. The use of a regression-based network allows for multiple concurrent and continuous outputs and avoids this growing action space issue, in turn extending the algorithm's applicability to more general scenarios, where simultaneous actions may be useful.

Aleatoric Uncertainty Estimation (UE) is applied within the proposed approach to identify conflicting data points. This subset of data is isolated from the complete dataset and is subsequently exclusively considered by the algorithm. Furthermore, an important property of this algorithm, with particular relevance to LfD, is its applicability to an iteratively increasing number of points within a dataset. The nature of the OC algorithm allows it to be used in scenarios with pre-existing datasets, such as

Behavioural Cloning (BC), or in iterative methods, such as DAgger, where demonstrations are being generated concurrently with the learning process.

4.2 Methodology

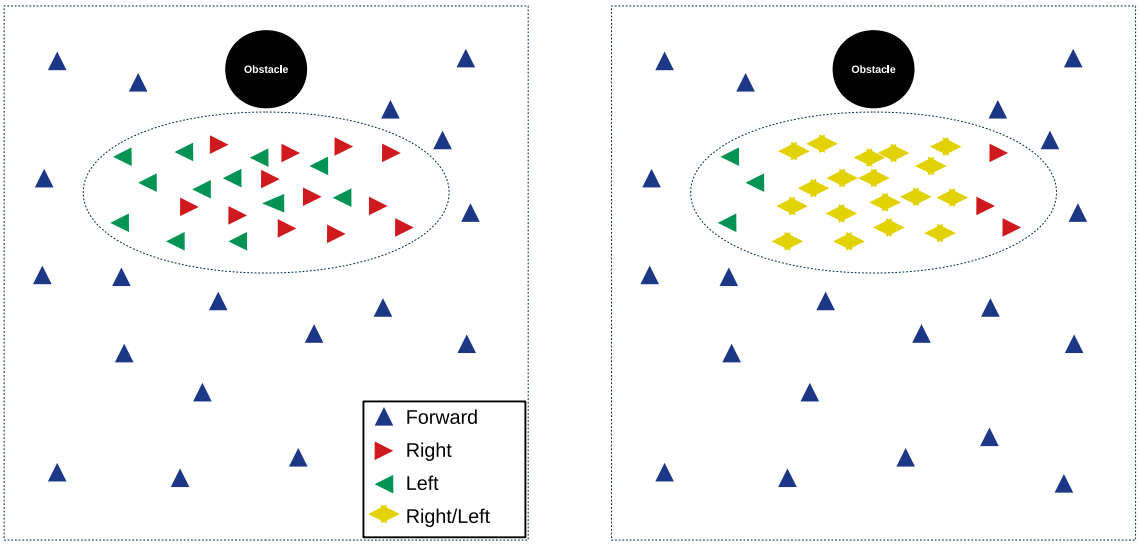
A deep understanding of the existing process is essential prior to implementing any changes, therefore this stage of the project involved a thorough review of the algorithm proposed in Chernova and Veloso (2008). The following is a brief overview of the OC algorithm, after which the variations defined in this project are explained in depth.

4.2.1 The Option Class Algorithm

The scope of the algorithm is to resolve problems of *inconsistent demonstrations*, or conflicting actions, arising from scenarios with multiple equivalent action choices. An adept example is the obstacle avoidance scenario, where a moving agent must avoid an obstacle by moving either left or right, with either action being equally valid. This example is represented in Figure 4.1, alongside the output from the OC method.

The OC algorithm, which is concisely presented in Algorithm 3, may be subdivided into a number of elements. First, the whole dataset $\mathcal{D}$ is examined to identify points with low classification confidence, i.e., points which the classifying function (based on a GMM in Chernova and Veloso (2008)) does not predict with certainty, rather multiple actions may be predicted with a similar probability. In this algorithm's context, low confidence is indicative of conflicting actions. These low confidence points are gathered and identified as part of a set $M \in \mathcal{D}$. Subsequently, a parameter d is defined as the mean Nearest Neighbour (NN) distance between all the points in the dataset. This value is the mean of the Euclidean distance between a point and the point closest to it, for all points in $\mathcal{D}$.

This parameter is applied to identify clusters of adjacent low confidence points, by searching over the restricted set M. A cluster is defined as a subset of points in M forming a connected component with a maximum distance $d * mf$ between a data point and at least one other point in the cluster. The tuning parameter mf is a user-supplied value, aimed at optimally adapting d to the dataset at hand. The aim of defining these clusters of closely-located points is to isolate potentially different types of



(a) State-space with conflicting data **(b)** State-space with option classes

Figure 4.1: A representation of the output from the OC algorithm. The plot on the left demonstrates the unaltered action space from an obstacle avoidance scenario, with evident conflicts in the central region, which is enclosed by an ellipse. After processing with the OC algorithm, the conflicts in this space are replaced with an option class, as shown in yellow, in Figure 4.1b.

conflicts and consider each individually. For example, one set of points may hold the actions *right* and *left*, while a second set of points could hold the actions *forward* and *left*. While both sets hold conflicts, the type of conflict is different and hence, they must be resolved differently and independently.

Every cluster $c \in C$ is first examined by the algorithm, which identifies all the action labels within c. In a consistent (non-conflicting) cluster, only one action label is expected, thus the reception of multiple actions indicates a dataset conflict. Subsequently, for every cluster with at least three data points and more than one action label, the algorithm creates an *Option Class*, i.e., an action label that indicates a choice between two or more actions for a given state. An example of such an OC are the yellow points in Figure 4.1b.

The algorithm defines a new dataset with these OCs, which is used to update the learnt policy. A unique action label is selected for every OC by randomly choosing an action from this class, with selection probabilities proportional to the incidence of each action in the respective cluster, c. Finally, the dataset labels are reset, such that unnecessary corrections from early iterations do not affect ensuing evaluations of this algorithm, which, based on additional data might create different classes or select other action labels.

Algorithm 3: Option Class Definition (Chernova and Veloso, 2008)

```
1:  procedure OptionClass(𝒟)
2:      M ← low confidence points in 𝒟
3:      d ← mean nearest neighbour distance from 𝒟
4:      C ← set of connected components from M, given d
5:      for all c ∈ C do
6:          A ← Action Classes from c
7:          if size(c) ≥ 3 and size(A) > 1 then
8:              Create Option Class from c, given A
9:          end if
10:     end for
11:     Update policy on 𝒟_new
12:     Reset demonstration dataset to 𝒟
13: end procedure
```

A notable feature of the OC algorithm is that it may be evaluated over a whole dataset in one run, in scenarios with pre-existing demonstration data, or it may be applied iteratively, e.g., with LfD methods, after every new demonstration. The latter use is the more relevant for this project, given the focus on Interactive Machine Learning (IML), however the additional flexibility is desirable.

4.2.2 Outline

The adaptation of the OC algorithm in this project was performed in a number of incremental steps, with new elements introduced in each iteration. The methodology followed in this project may be summarised as follows:

- Implement the OC algorithm with an FNN classifier (with a softmax output) as the function approximator, with data confidence evaluated through either the classifier's output probabilities or the Conflict Value Algorithm (CVA).
- Implement the OC algorithm with a regression-based FNN, replacing the confidence metrics with uncertainty estimates obtained with the split-network architecture introduced in Section 2.3.3.
- Consider a generalised discrete action space with fully independent action dimensions, which differs from the previous scenario with co-dependent actions, where only one non-zero action is permitted at any time.
- Consider a continuous action space with fully independent action dimensions.
  - Investigate the CVA as an alternative for UE.

4.2.3 Option Class algorithm with an FNN classifier

As the first implementation within this project, the aim of this iteration of the OC algorithm was to establish a basic framework and validate the fundamental elements of this method which remain unchanged in forthcoming implementations, such as the clustering method and action label selection strategy. Accordingly, this implementation was minimally different from the algorithm proposed in Chernova and Veloso (2008) - the main alteration was the change of approximating function, previously a GMM-based classifer and now an FNN-based classifier.

This change was effected in light of the ultimate goal of implementing this algorithm on an Artificial Neural Network (ANN) architecture, as highlighted in Section 4.1. In practical terms, the circumvention of a GMM-based implementation promoted an efficient implementation process, since a valid FNN architecture was readily available from the work detailed in Chapter 3.

Identification of low confidence data points

One of the most critical components of the OC algorithm is the identification of data points which the policy classifies with low confidence. The classifier's classification probabilities form an obvious metric for this task, however their use required that the FNN's output be normalised to form a probability distribution with the softmax function, rather than transformed to a one-hot encoding. Every element of the network's output was thus translated to the interval $(0, 1)$ and collectively, all terms summed to 1, such that they were interpretable as probabilities. A low classification confidence was determined when the maximum output probability for any input datum was lower than a specified threshold. Data points satisfying this criterion were passed to the subset M of low confidence points.

An alternative criterion for the selection of low confidence data relies on the assumption that low confidence is the result of conflicting action labels in a specific region. Consequently, the CVA, which is presented in Section 3.2.1, was applied to identify conflicts in the dataset, $\mathcal{D}$. Data points with a conflict score exceeding a threshold were gathered in the set M. Crucially, while the former criterion was only suitable for a classifier network, the CVA is independent from the function approximator and applicable to all forms of the OC algorithm.

After the definition of the subset M and the computation of the mean NN distance d, the algorithm proceeds to create a set of connected components, or clusters, of closely-located low confidence points. Chernova and Veloso (2008) discuss the definition of connected components in terms of d, however do not refer to a specific algorithm. A method that performs clustering in the approach described in this paper, and was hence applied in this project, is DBSCAN.

Action Clustering within the OC algorithm

Density-based Spatial Clustering of Applications with Noise (DBSCAN) (Ester et al., 1996) is a methodology proposed to deal with spatial data of variable density and potentially contaminated with noise and outliers. The key benefits of DBSCAN lie in its high efficiency and thus applicability to large datasets, its ability to identify arbitrarily-shaped clusters and a limited dependence of prior knowledge. Specifically, this method requires two parameters that must be pre-specified - the minimum number of data points, mp, within a cluster and the maximum distance between two points, ϵ, below which said samples are defined as neighbours.

These parameters control the definition of density regions, where a cluster is defined when at least mp points lie within a region with radius ϵ from a given point. The algorithm classifies individual data points into three distinct classes, two of which indicate some closeness to a cluster. Core points are those samples that lie within a cluster, while border points are neighbours to such samples, however have less than mp data points in their ϵ-neighbourhood. The remaining points are gathered under the definition of noise (Salton do Prado, 2017; Saptashwa, 2019). A visual representation of DBSCAN's classifier-like operation is provided in Figure 4.2.

Intuitively, the parameter ϵ is equivalent to the computed distance d in the OC algorithm and furthermore, with $mp = 3$ the DBSCAN method is restricted to only form clusters with three or more points.

This is a particularly relevant property, since clusters with at least three points are a requirement for this algorithm and the prevention of small clusters alleviated some of the computational effort.

The assumption that conflicting actions result from areas with equivalent action choices implies that any action within an option class is equally correct. Chernova and Veloso (2008) propose random selection, biased by the distribution of actions within a cluster, such that the most frequently demonstrated action is the most likely to be selected. Within this project, this approach was considered alongside a 'greedy' selection approach, where the most frequently demonstrated action is selected for every cluster.

4.2.4 Option Class algorithm with a FNN

While the implementation described previously was mostly concerned with defining a functioning framework for the OC approach, the upcoming realisation of this algorithm was the first step in extending this method's applicability to virtually any environment. To retain the approach of gradual modifications, the function-approximating FNN was changed to a regression-based network, however the scope of the algorithm was kept as close as possible to the previous scenario. In this sense, a discrete action space was considered, where only one action could be active (non-zero) at one time, thus the network's output could be represented by a one-hot encoding, similarly to a classifier's output.

The consideration of co-dependent actions limited the applicability of this approach in the short-term, however the minimal change in setting allowed for the implementation focus to shift to the data selection strategy. Aleatoric UE, with a split-network architecture as described in Section 2.3.3, was applied for this purpose to replace the confidence metric, given the unsuitability of classification probabilities without a classifier network.

A change in terminology was necessary at this stage. In previous realisations of the OC algorithm, data points were evaluated on a measure of confidence in their action labels, however in this setting, the same evaluation must be performed with a measure of uncertainty. These quantities are opposites - a high confidence in a set of data implies low uncertainty - thus a new objective was defined. The subset M of previously low confidence points became a set containing highly uncertain data points.

The procedure of defining this dataset was largely unchanged from the previous implementation. An uncertainty estimate was obtained for all the data points within the set $\mathcal{D}$ and any points with an uncertainty value exceeding a threshold were passed to the set M. A threshold value within the range $[0, 1]$ was provided to the algorithm, after normalising all uncertainty predictions to the same range. This methodology allowed for greater accuracy and flexibility, however it introduced an additional parameter to the algorithm.

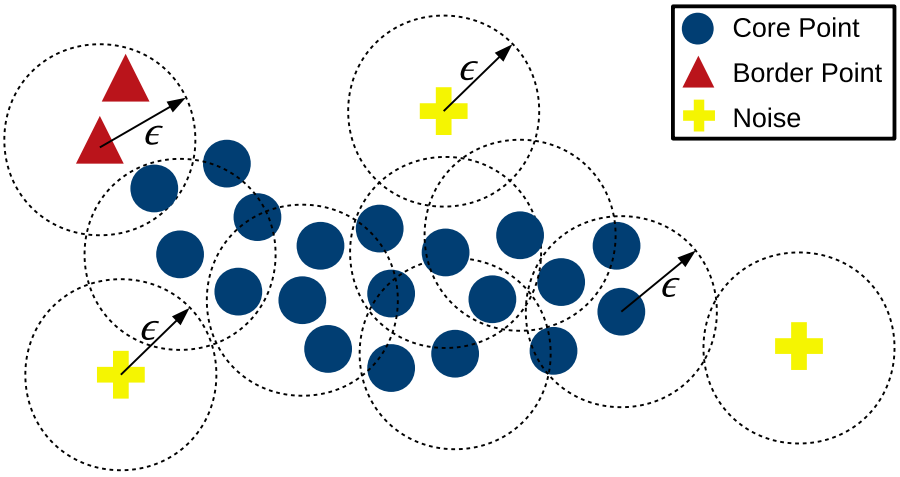


Figure 4.2: DBSCAN with one large cluster and limited outliers (adapted from Salton do Prado (2017))

4.2.5 Option Class algorithm with independent action dimensions

The modifications to the OC algorithm presented so far have retained the co-dependence between actions, producing a classifier-like output and focused on the structural changes necessary to transition to a UE-based algorithm. Conversely, the upcoming changes are predominantly concerned with generalising its applicability, as discussed in Section 4.1, to a wide range of more realistic environments.

Notably, the co-dependence between actions does not fully exploit the benefit of a regression-based FNN, which can produce predictions for multiple, continuous-valued outputs simultaneously without increasing the dimension of the action space, as would happen with a classifier network. The elimination of this co-dependence was hence a natural step, with the consequence that the algorithm now considered environments with independent action dimensions, e.g., a moving vehicle which may simultaneously accelerate (or brake) and steer.

In terms of the OC algorithm, the modifications necessary to assure its suitability to such environments were notably minor and involved the simplification of the action selection process, which occurs after the definition of option classes for every cluster. The selection of a new action label with co-dependent actions necessitated that the algorithm consider all dimensions simultaneously, with the chosen label having the least uncertainty within all said dimensions. Independence between actions allowed for the individual treatment of every dimension in the action selection process. For instance, a conflict between *left* and *right* actions could be resolved without a search for additional conflicts between *forward* and *back* labels for the same state values, since these pairs of actions were now permitted to co-exist.

4.2.6 Option Class algorithm with a continuous action space

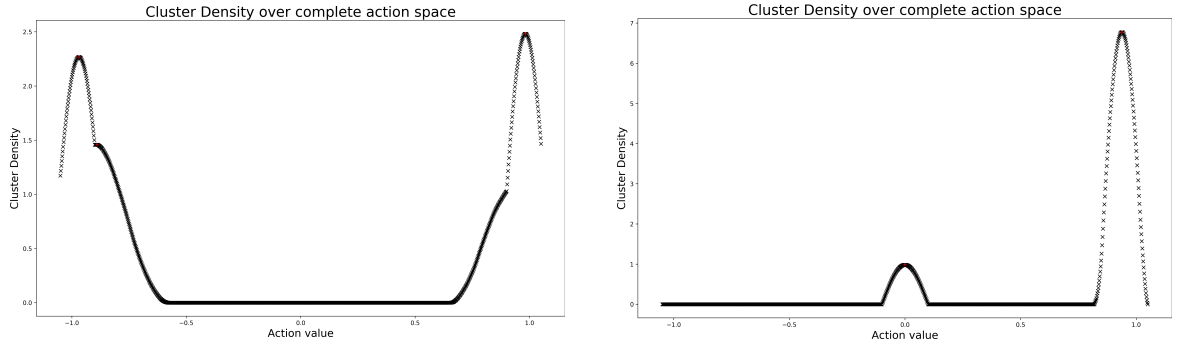
The final significant change to the OC algorithm is similar to the above and entailed a transition from environments with discrete, independent action spaces to environments with continuous, independent actions. This modification truly ensures this method may be used in virtually any scenario, however it is also the farthest from the original case. A discrete action space implies a finite and often quite limited set of possibilities, thus the algorithm is able to compute the number of distinct actions in each cluster and select a new label from within this set. A continuous action space brings forth the possibility of a large number of actions within each cluster with minute differences between each other, making the option class definition and label selection processes overly complex and ultimately infeasible.

Conflict Correction with continuous actions

In order to address data conflicts within identified clusters, the incidence of actions within a cluster must be computed such that the action selection process is based on some metric. An efficient approach to achieving this result is Kernel Density Estimation (KDE) (Bishop, 2006b), a non-parametric approach for probability density modelling. This method approximates a probability density function $p(x)$ from a set of observations which are assumed to be drawn from the same function. If one defines the variable x as a (uni-dimensional) action taken by an agent in a specific environment, then the subset of data contained within a specific cluster in the OC algorithm may be considered as a set of observations of x. This data is hence considered to characterise $p(x)$.

A simplified explanation of KDE specifies that the density model $\hat{p}(x)$ is obtained by placing a given kernel function over every available data point and summing their contributions along the whole dataset. The computed density function should be normalised by dividing throughout by the number of data points available. The kernel is an arbitrarily-chosen, non-negative function that should integrate to one, such that the resulting distribution is similarly non-negative and integrates to one. The selection of this kernel determines the smoothness and accuracy of the estimated density function - a hypercubic kernel may create discontinuities in $\hat{p}(x)$, similar to histogram methods, thus smoother kernels such as the Gaussian function are preferred.

In this project, the KDE function was defined with a cosine kernel, which was validated to closely approximate sampled distributions. After fitting a density model to cluster data, the same model was used to evaluate probability densities along the complete, discretised action space, as shown in Figure 4.3.



(a) Densities for cluster in 1st action dimension

(b) Densities for cluster in 2nd action dimension

Figure 4.3: Evaluations of action densities within specific clusters after KDE training with data points within respective clusters. Evaluating the densities over the full action space and performing peak detection captures the most frequent action values within a cluster, at a resolution determined by the underlying test set, which due to finite computational resources, cannot be continuous. Identified peaks are marked in red.

A finite set of candidates for the new cluster label were defined through peak detection over this space - identified peak values are highlighted in red in the same figure. Finally, in a similar fashion to previous versions of the OC algorithm, the value with the highest density (i.e., most frequently demonstrated action) is selected as the new label for data points in a cluster.

Ultimately, notwithstanding the list of changes detailed in this section, the OC process remains consistent with the procedure in Algorithm 3. Overlooking the change to a regression-based FNN, the only notable variations are the search for low confidence data points, which is replaced with the identification of highly uncertain data and the definition of the set of connected components, which is realised through the DBSCAN methodology. Alongside the dataset $\mathcal{D}$, this method requires two user-defined parameters - a multiplication factor for the distance value d and a threshold for normalised uncertainty predictions.

Option Class algorithm with the CVA

As shown in Chapter 3, the Conflict Value Algorithm presents a viable alternative to UE, thus it was applied within the OC methodology for this purpose. This algorithm removes the need to train the UE network, hence alleviating the computational burden prior to initiating the OC algorithm. Nevertheless, the previous chapter strongly highlighted the high computational cost of the CVA for large datasets, thus this benefit decreases as the dataset in question grows in size. As shall be shown in the following section, this algorithm is a valid replacement for UE within the OC method for limited datasets, however its use with large datasets should be avoided due to lengthy computation times.

4.3 Experiments and Results

A thorough validation of the OC algorithm was performed using synthetic and real datasets generated using the customised OpenAI Gym environments presented previously in Section 3.3, as well as a realistic simulation scenario devised within the environment shown in Figure 4.4. Simulations performed with this environment were designed to emulate a real robotic system and allow for the validation of the algorithm in a realistic setting, which is as close as possible to a real-world scenario.

4.3.1 Single-run Evaluations

Firstly, the OC algorithm's performance was tested in *single-run* format, which evaluates a complete set of demonstration data in one attempt. This configuration was tested with synthetic data from both custom environments and relevant plots are shown in Figures 4.5, 4.6 and 4.7.

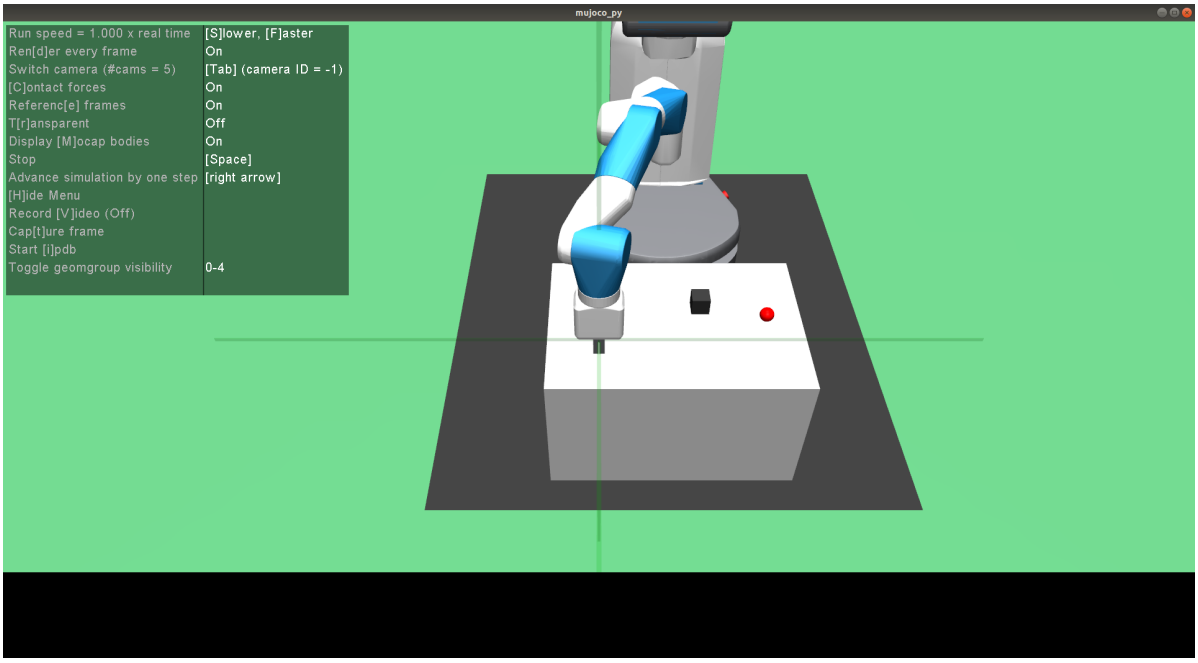


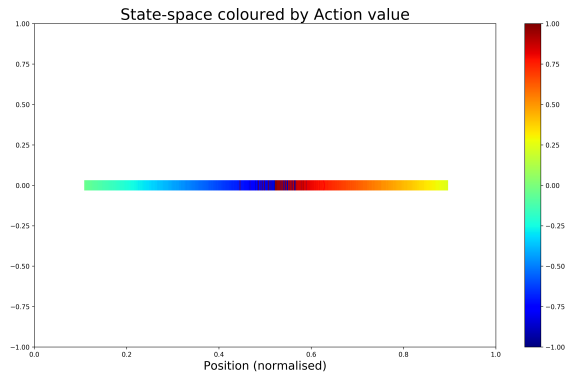
Figure 4.4: A simulated robotics environment with a 6 degree-of-freedom robot arm, a cubic object and a red, spherical goal position to which the object should be moved.

Figure 4.5 demonstrates the operation of the algorithm with the one-dimensional, two-goal environment, which is ideal to understand intermediate results and the algorithm's behaviour. The first two plots show the original, conflicting dataset and FNN predictions after training with this data. When compared to the FNN prediction in Figure 4.5f, it is clear that the dataset conflicts negatively impacted the network's training and lead to a region with low magnitude actions. While the effects of these conflicts are not catastrophic in this basic example, they often lead to failed execution in more complex environments, such as the two-dimensional obstacle avoidance setting.

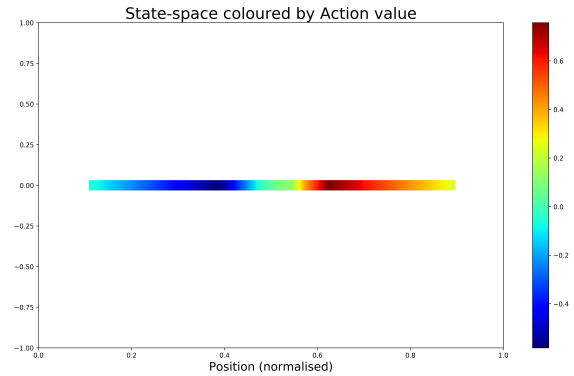
Figures 4.5c and 4.5d show intermediate results leading to a successfully corrected dataset, as displayed in Figure 4.5e. In this and all the following uncertainty plots, the colour of each data point represents its uncertainty, with a dark blue implying the minimum uncertainty in that plot, and dark red being the peak uncertainty in that set. Figure 4.5c shows the uncertainty predictions for the conflicting dataset, where the highly uncertain points clearly correlate to the conflicts in the dataset. The subsequent figure shows the clustering result with DBSCAN, where the selected highly uncertain points are classified into three distinct clusters based on the distance parameter d. A larger multiplier mf would result in one larger cluster, with the corrected dataset remaining void of conflicts.

Comparably to the previous case, the FNN predictions in Figure 4.6, which holds results for the first action dimension from the obstacle avoidance environment, are significantly different. Conflicting demonstrations create an inconsistent prediction space, which can affect the agent's performance in this environment. Akin to the previous set of plots, the peak uncertainty corresponds to the conflicting data points, while the defined clusters effectively address these inconsistencies and create a uniform space, as shown in Figure 4.6e and the corresponding FNN prediction in Figure 4.6f.

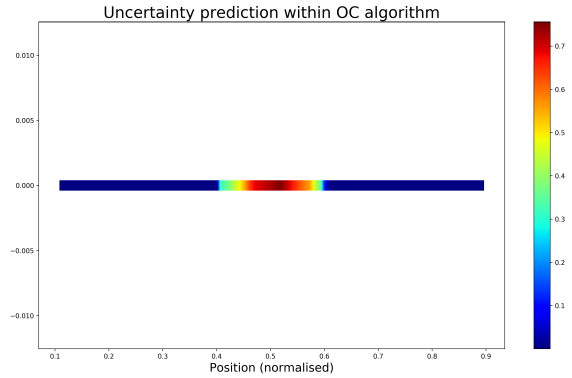
Conversely, the action values in the second dimension, as shown in Figure 4.7 present minimal conflicts, thus the selected uncertain actions form a very small proportion of actions in the set. In such situations, it is expected that the dataset remains unchanged and in fact, this is the observed result.



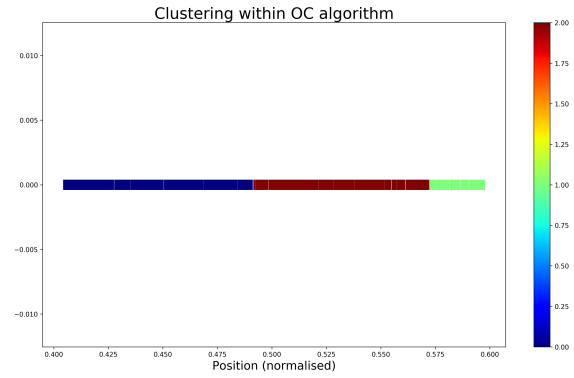
(a) Complete dataset prior to correction



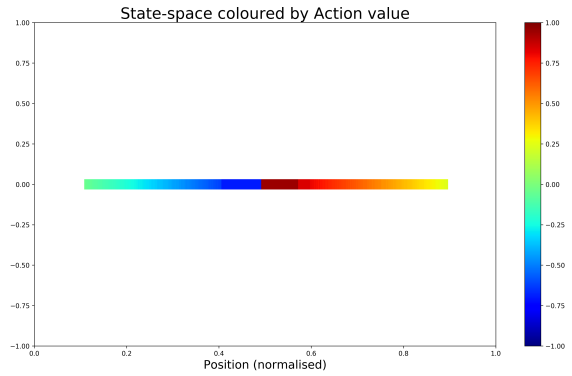
(b) FNN prediction



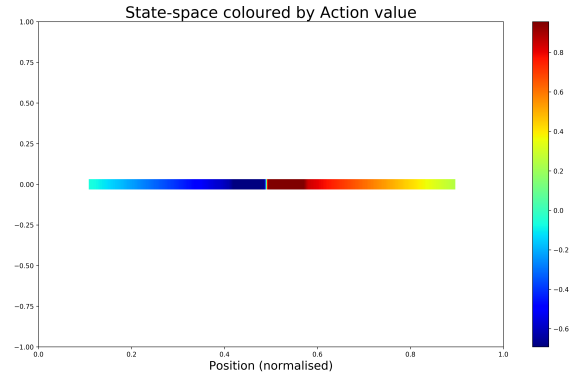
(c) Aleatoric uncertainty estimate



(d) Density-based clustering



(e) Complete dataset after correction



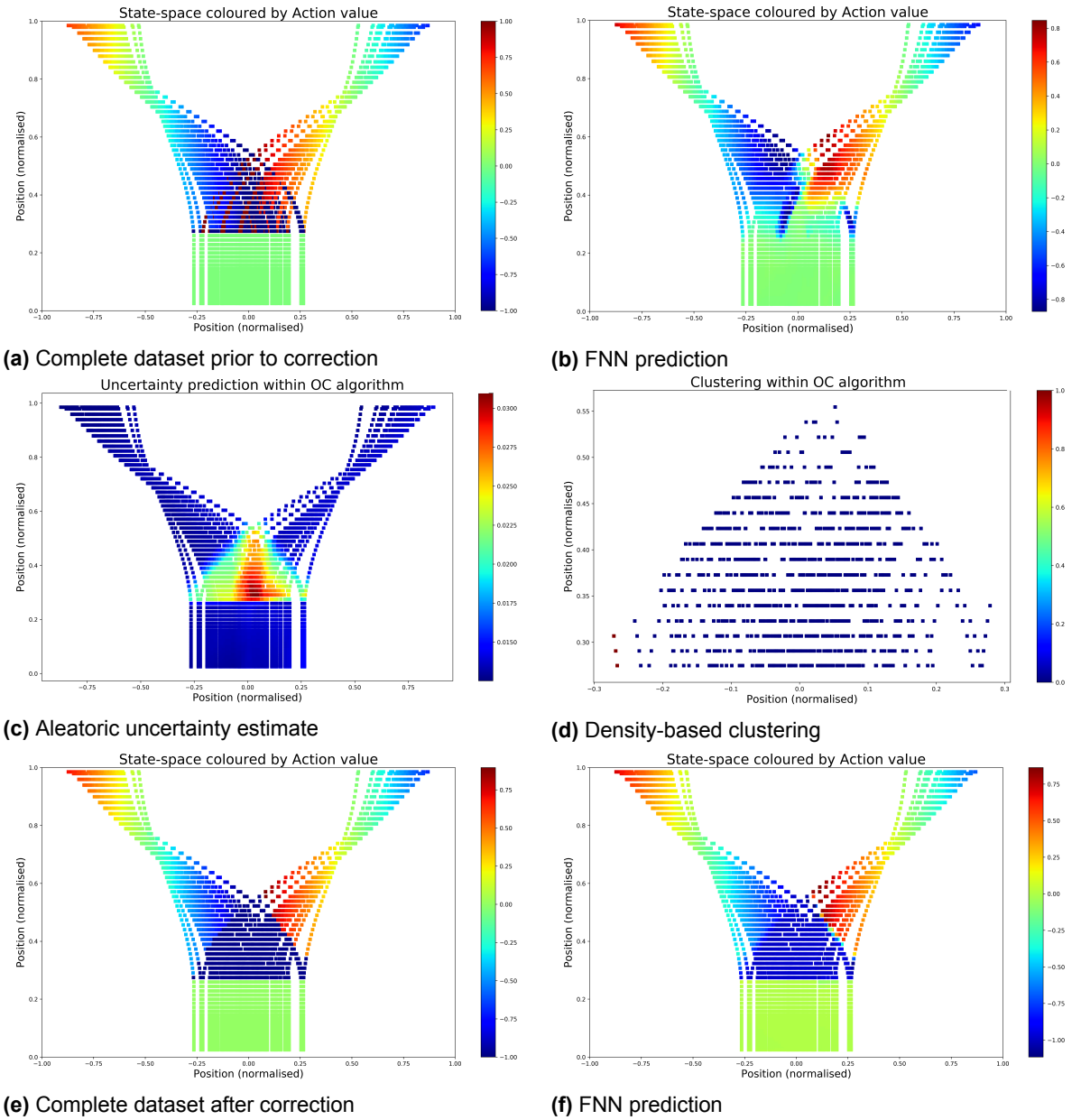
(f) FNN prediction

Figure 4.5: A set of plots exhibiting the operation of the OC algorithm for a 1-dimensional environment, as shown in Figure 3.1, with conflicting data.

4.3.2 Iterative Evaluations

As specified in Section 4.2, the OC algorithm may be applied *iteratively*, hence the upcoming plots demonstrate its output at intermediate stages of such a procedure. Figure 4.8a shows the complete dataset, prior to any corrections. The following figures present the output from the OC algorithm with an increasing number of demonstrations - the algorithm is applied after individual demonstrations are added, however there is an excessive number to show in this report. The final plot in Figure 4.8f shows the agent's output after all demonstrations were processed by the algorithm.

Evidently, the algorithm's output varies significantly between successive runs, depending on the data supplied as well as the number of epochs the UE network is trained for. Due to the iterative nature of this process, this network is trained for a limited number of epochs per iteration, thus early predictions are often overly optimistic and lead to excessive corrections, as can be seen in Figure 4.8b. As the



(a) Complete dataset prior to correction

(b) FNN prediction

(c) Aleatoric uncertainty estimate

(d) Density-based clustering

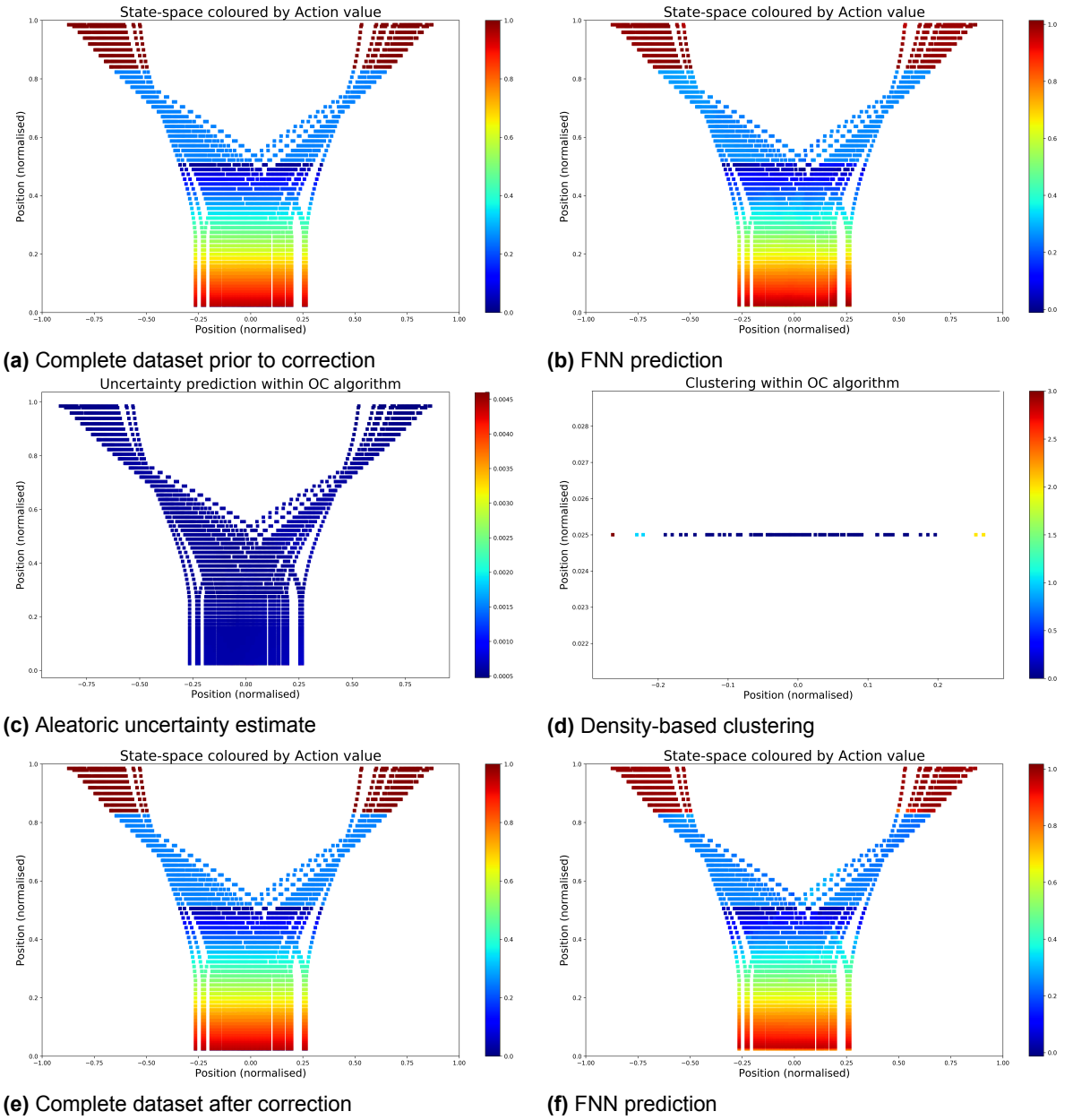
(e) Complete dataset after correction

(f) FNN prediction

Figure 4.6: A set of plots exhibiting the operation of the OC algorithm in the first action dimension of a 2-dimensional environment, as shown in Figure 3.2, with conflicting data.

number of iterations increases the uncertainty predictions become progressively more reliable and consequentially, the OC corrections are observed to be more accurate.

The sequence of plots in Figure 4.8 highlights two important factors deriving from this iterative format. First, the new action labels may take different magnitudes over multiple iterations. While this observation may be trivial given that subsequent evaluations are independent, it must be noted that this feature does not hinder the performance of the algorithm, which results in a consistent FNN prediction, as desired. Secondly, this format allows for occasional imperfect corrections, as evidenced by the dataset in Figure 4.8b. A sub-optimally corrected dataset will only impact the novice agent for a limited number of iterations, with such flaws being addressed by later corrections. Nevertheless, the OC algorithm is intended to work optimally at all times in order to assure good results.



(a) Complete dataset prior to correction

(b) FNN prediction

(c) Aleatoric uncertainty estimate

(d) Density-based clustering

(e) Complete dataset after correction

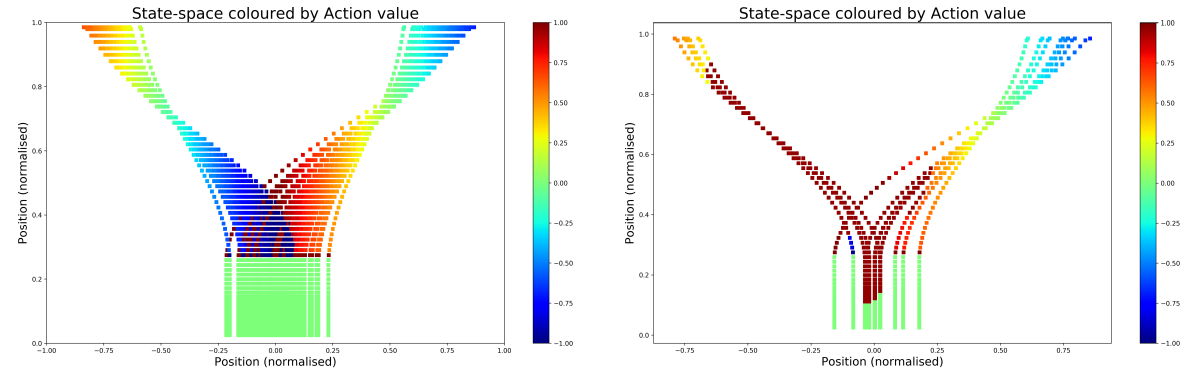
(f) FNN prediction

Figure 4.7: A set of plots exhibiting the operation of the OC algorithm in the second action dimension of a 2-dimensional environment, as shown in Figure 3.2, with conflicting data.

4.3.3 Performance Evaluations

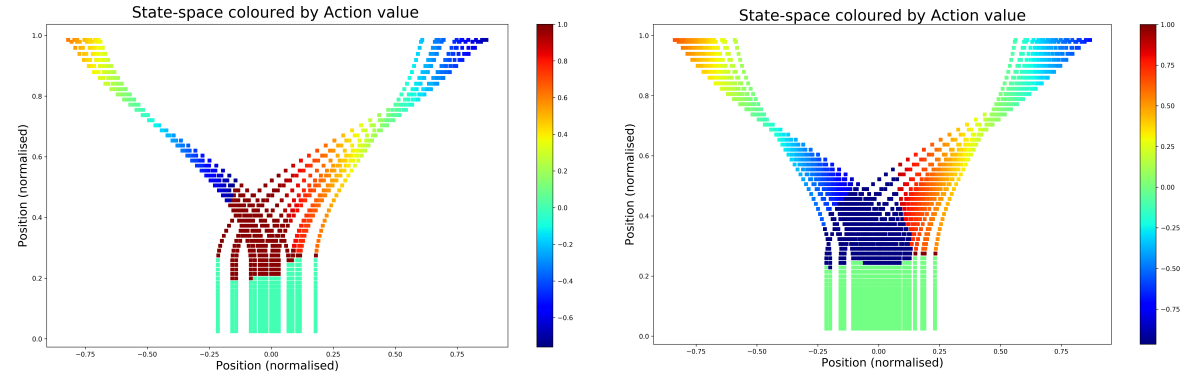
In addition to pictorial results showing the operation of the OC algorithm and resulting data, numerical results are included to provide further insight and demonstrate the benefits of this method. The upcoming results follow from an evaluation of this algorithm over five distinct datasets, generated with the obstacle avoidance environment described previously.

The upcoming tables present the outcome of an assessment of the original and corrected datasets, performed with the Conflict Value Algorithm. This algorithm allowed for a direct and reliable comparison between every dataset pair and permitted inference on the benefits of using the OC algorithm on this data. Tables 4.1 and 4.2 hold results for individual action dimensions, given that the underlying environment defines a two-dimensional action space. The plots observed previously show that the second dimension contains little conflict and thus, the algorithm's corrections are notably minor.



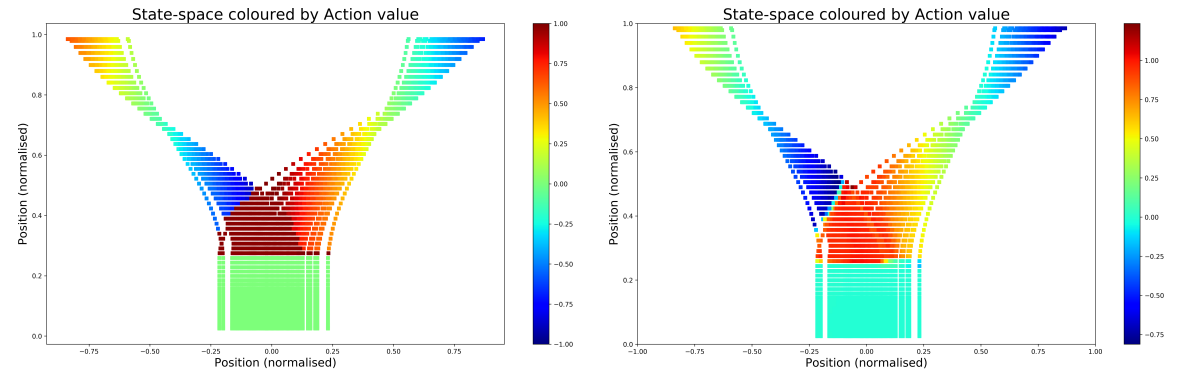
(a) Complete dataset prior to correction

(b) Partial corrected dataset (1)



(c) Partial corrected dataset (2)

(d) Partial corrected dataset (3)



(e) Partial corrected dataset (4)

(f) FNN prediction

Figure 4.8: A set of plots exhibiting the iterative operation of the OC algorithm in the first action dimension of a 2-dimensional environment, as shown in Figure 3.2, with conflicting data. Subsequent plots show the evolution of the algorithm's corrections as new trajectories are added to the dataset.

This is evidenced in the second table, where the differences in the mean conflict value between dirty and clean datasets are negligible. Conversely, the clean datasets in Table 4.1 result in a mean conflict value which is around half of the corresponding quantity computed from their uncorrected versions. The maximum conflict value also decreases after applying the OC algorithm, however it remains notably large due to the ever-present discontinuity between actions of magnitude '+1' and '-1' in all action spaces. The magnitude of the maximum conflict values in Table 4.2 are also due to a similar effect, however in both scenarios, this does not indicate severe issues in these datasets.

Following this data analysis with the CVA, a novice agent was trained through Behavioural Cloning on each of the corrected datasets. Additionally, five FNNs were trained on the original datasets in order to compare the performance differences induced by optimising their training data. Table 4.3

contains results from 25 rollouts for each network, shown as the number of successful rollouts (S.R.) and the average cumulative reward from each run in this environment. A reward value was generated by penalising the agent for every time step spent within the environment, assigning a large reward for reaching the environment objective and significantly deducting from this reward value if the agent hits a boundary wall or the obstacle.

A comparison between the networks trained on the original datasets and corrected datasets show that the rewards for the latter are consistently larger. Furthermore, the networks trained on the clean data always achieve their objective, whereas the other networks have at least one failure, with an observed maximum of 7 failures over 25 attempts. Undoubtedly, the OC algorithm was beneficial and reduced the failure rate of the trained networks, notwithstanding the non-ideal LfD approach used.

4.3.4 The OC algorithm with Conflict Values

The previous section made reference to the application of the CVA within the OC algorithm, replacing UE. It was also established that this algorithm introduces an excessive computational burden when dataset size increases, which negated the benefit of not training a UE network. Nevertheless, Figure 4.9 contains the result of applying the OC algorithm to a conflicting dataset with the CVA.

At first glance, Figure 4.9e demonstrates that the algorithm successfully resolved all the data conflicts, in a similar fashion to the output in Figure 4.6e. A closer analysis indicates that the cluster of points selected by the CVA is larger than selected through UE, leading the correction to extend to points that did not need replacement. This is a direct result of the output of the CVA, in Figure 4.9c, which computes higher values over a larger area due to its point-by-point analysis. Uncertainty prediction peaks are localised to the most conflicting area of a space, with the magnitude slowly decreasing away from this region, which allows a more precise selection of points, which is more desirable. Notably, taking either the maximum or average conflict value for every data point (and adjusting the threshold accordingly) does not significantly affect the prediction.

4.3.5 The OC algorithm in a realistic environment

The final set of results contained within this report is derived from the robotics environment introduced in Figure 4.4. This environment was revised for the purposes of this experiment - an obstacle was added between the robot and the goal position to create an equivalent action choice for the OC algorithm to resolve, while the cubic object was removed to simplify the robot's task. The objective in this scenario required the robot to move towards the goal position, while going around the obstacle. This simulated environment is shown in Figure 4.10.

A novice agent based on an FNN was trained using HG-DAgger, which involved providing an initial (small) set of full demonstrations and subsequently allowing the agent to act and providing corrective

	Set 1		**Set 2**		**Set 3**	
Set type	Mean	Max	Mean	Max	Mean	Max
Dirty	0.00462	1.0	0.0043	1.0	0.00492	1.0
Clean	0.00258	0.97689	0.00279	0.97961	0.00231	0.92981

	Set 4		**Set 5**	
Set type	Mean	Max	Mean	Max
Dirty	0.00487	1.0	0.00443	1.0
Clean	0.00249	0.95037	0.00222	0.93420

Table 4.1: An evaluation of the mean and peak conflict values in original (dirty) and OC-corrected demonstration datasets. The values shown were computed using the CVA for the first action dimension of a set of datasets created using the obstacle avoidance environment shown in Figure 3.2.

	Set 1		Set 2		Set 3	
Set type	Mean	Max	Mean	Max	Mean	Max
Dirty	0.00397	0.71455	0.00397	0.71455	0.00378	0.71455
Clean	0.00382	0.71455	0.00387	0.71455	0.00378	0.71455

	Set 4		Set 5	
Set type	Mean	Max	Mean	Max
Dirty	0.00398	0.71455	0.00376	0.71455
Clean	0.00382	0.71455	0.00376	0.71455

Table 4.2: An evaluation of the mean and peak conflict values in original (dirty) and OC-corrected demonstration datasets. The values shown were computed using the CVA for the second action dimension of a set of datasets created using the obstacle avoidance environment shown in Figure 3.2.

	Set 1		Set 2		Set 3		Set 4		Set 5	
Set type	S.R.	Reward	S.R.	Reward	S.R.	Reward	S.R.	Reward	S.R.	Reward
Dirty	24	85.248	24	84.784	19	45.556	18	37.256	22	69.364
Clean	25	93.300	25	93.280	25	93.308	25	93.404	25	93.288

Table 4.3: An evaluation of five agents' performance in an obstacle avoidance setting, after BC training on synthetic demonstration data. These agents' performance is evaluated on the number of successful runs, with a total of 25 runs for each agent and an averaged cumulative reward from every run. Obtained rewards are akin to those in Reinforcement Learning (RL), with the value of said reward dependent on the performance of the agent.

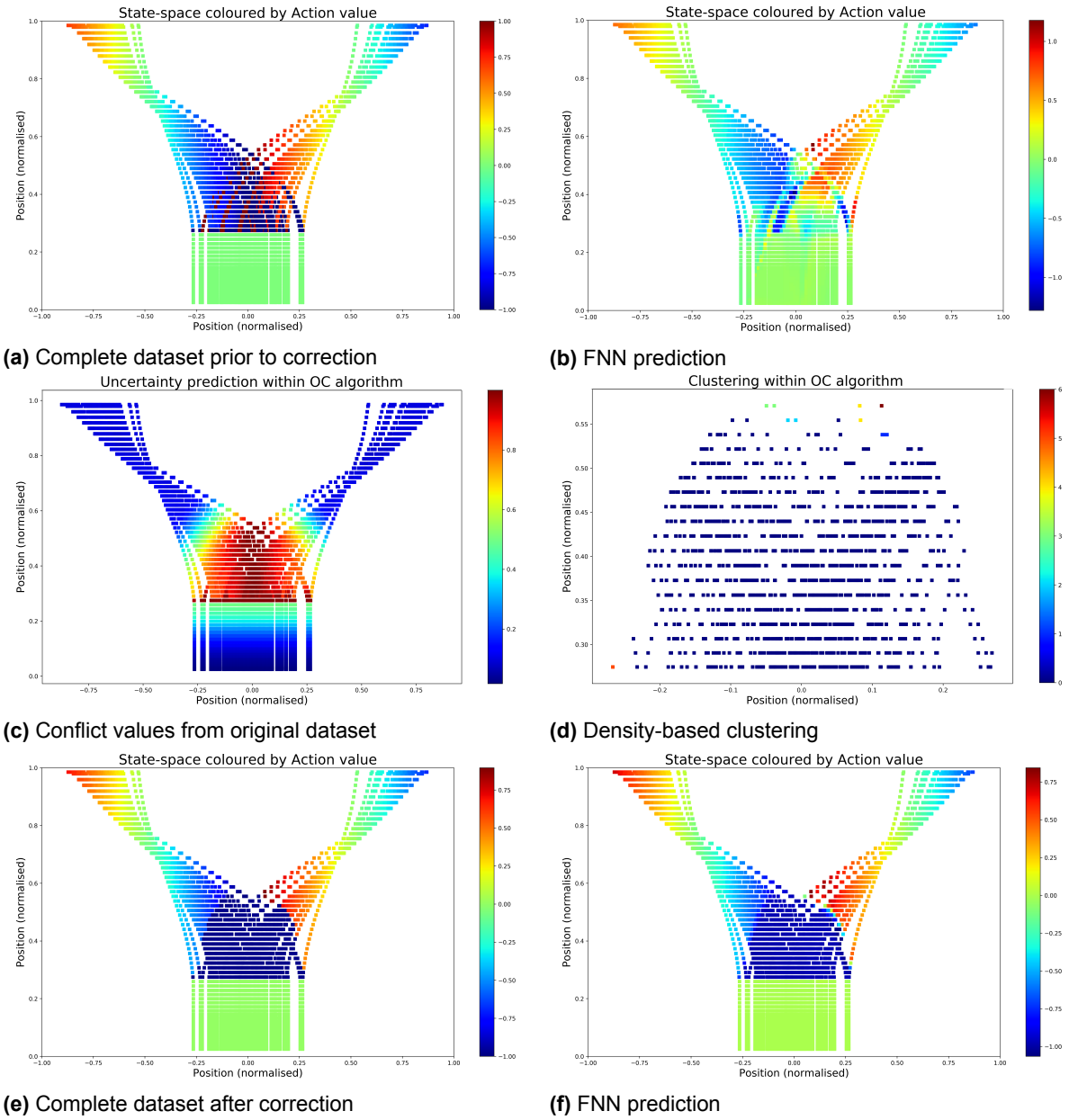
demonstrations when it deviated from the desired behaviour. The OC algorithm was applied following a new demonstration, prior to training the agent.

The original dataset, in two distinct dimensions, is shown in Figures 4.11a and 4.11b. The former plot presents action magnitudes defining lateral movement within this environment, while the latter shows the actions defining movement forward and back. A third dimension representing vertical arm movement up and down is available, however the arm was constrained from moving in this dimension, hence these actions do not hold any information and are excluded from this report.

A comparison of the original datasets and the corrected data in Figures 4.11g and 4.11h brings forth a number of differences introduced by the OC algorithm. The most notable change is observed in the first action dimension, where conflicts between actions of magnitude '+1' and '-1' are replaced with actions of magnitude '-1'. A similar correction is performed in the second action dimension, where a conflict between '+1' and '0' actions is resolved by assigning the latter value to these data points. The corresponding predictions from the trained agent, which are broadly consistent as desired, are presented in Figure 4.11i and Figure 4.11j.

Figure 4.11e and Figure 4.11f contain the uncertainty predictions from the last iteration of this training process. The peak uncertainties within these plots coincide with the most conflicting data points, however do not extend to all the conflicting areas in the dataset - a low uncertainty value is predicted for some areas with visible conflicts.

The main cause of this behaviour is the limited number of training iterations available to the learning network. As this FNN is trained, the nature of the predicted uncertainty evolves from a single, round and rather large region to a more focused area with a smaller proportion of large values and lastly to multiple, smaller regions, if the dataset contains numerous regions of conflicting data. A substantial issue, however, is the notion of diminishing returns, as the network must be trained for an extreme number of iterations to achieve the final result. In an iterative LfD setting, this becomes prohibitive for the user, which must wait for the agent to finish training before providing additional input. As observed from this result, when a balance is sought between training time and the quality of the uncertainty estimates, the result remains sub-optimal.



(a) Complete dataset prior to correction

(b) FNN prediction

(c) Conflict values from original dataset

(d) Density-based clustering

(e) Complete dataset after correction

(f) FNN prediction

Figure 4.9: A set of plots exhibiting the operation of the OC algorithm with the CVA in the first action dimension of a 2-dimensional environment, as shown in Figure 3.2, with conflicting data.

A more subtle concern with this approach to UE is its inclination to expressly consider the most significant conflict within a dataset and ignoring less apparent ones. For example, a region containing actions of magnitude '+1' and '-1' will be determined as highly uncertain, while a separate region with action magnitudes of '+1' and '0' may be ignored. This may be observed in Figures 4.11e and 4.11f and is explored in more detail in Appendix B.

An assessment of the benefit of the OC algorithm in this scenario may be obtained from the comparison of the FNN predictions based on the uncorrected and corrected datasets, respectively. Figure 4.11i contains notably more consistent and well defined regions, which achieve an action magnitude closer to '+1' or '-1', as intended by the demonstrator. Conversely, in Figure 4.11c, individual trajectories may easily be identified and take smaller values due to action conflicts. A similar, but less pronounced difference, may be noted in the comparison between Figures 4.11d and 4.11j. In this sense, the OC algorithm led to an improvement in the agent's policy, however this result remains unoptimised and

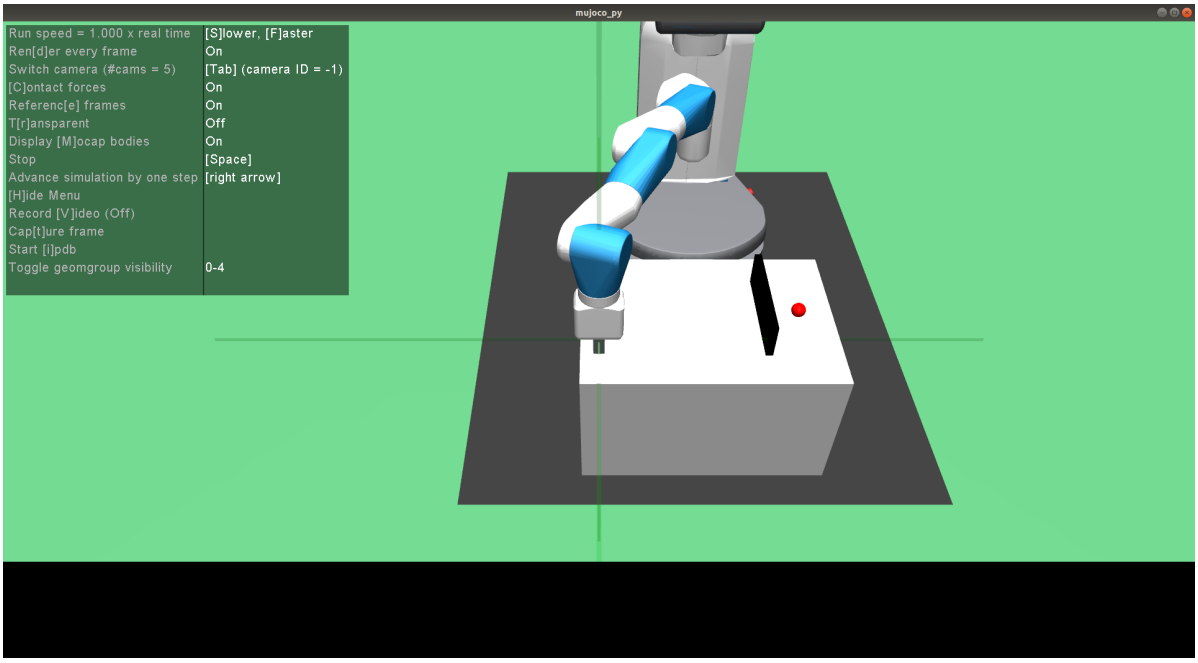


Figure 4.10: A simulated robotics environment with a 6 degree-of-freedom robot arm, an obstacle and a red, spherical goal position to which the arm should be moved.

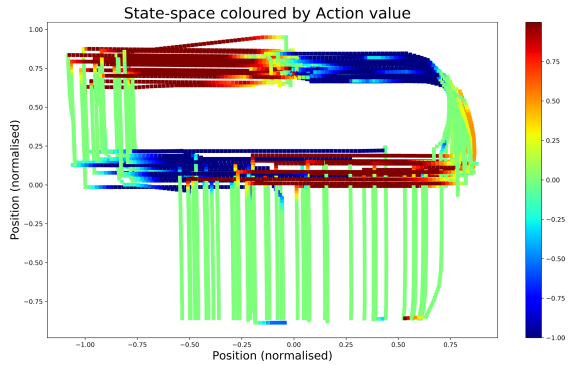
reveals a necessity for future development on this algorithm.

Finally, the OC algorithm's success rate in identifying and completely correcting conflicting data was assessed by evaluating this algorithm five times for five datasets, generated using the obstacle avoidance environment. After every evaluation, the number of successful runs was noted - a *success* implies the resolution of all data conflicts in a dataset. In the results displayed in Table 4.4, no dataset was successfully corrected in all attempts, however it must be noted that a few of these failures were very minor, with only one or two conflicting points in the whole dataset.

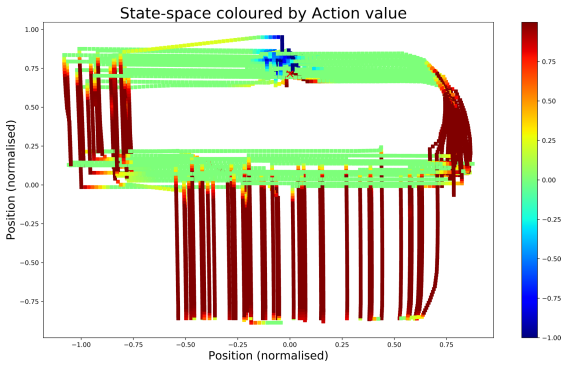
OC type	**Set 1**	**Set 2**	**Set 3**	**Set 4**	**Set 5**	**Total**
Single-Run	4/5	2/5	3/5	1/5	4/5	14/25
Iterative	3/5	3/5	3/5	2/5	4/5	15/25

Table 4.4: An evaluation of the OC algorithm's success rate in addressing conflicting data points over five distinct datasets. The algorithm is applied multiple times over each dataset in both a single-run and iterative fashion, where a successful evaluation implies that all conflicts are eliminated in the resulting dataset.

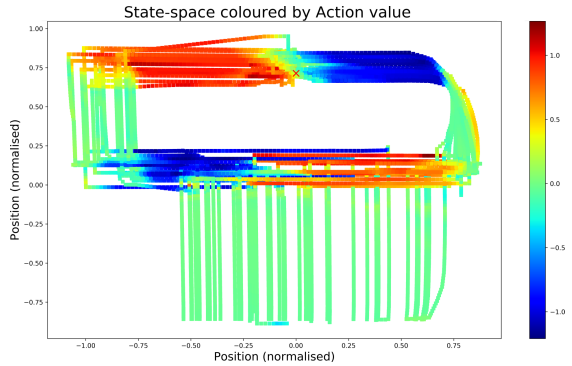
This table also indicates that specific datasets are more likely to be successfully processed than others. For example, Set 4 (Figure 4.12a) contains one conflicting trajectory that is evidently separate from the region containing the majority of the conflicts in this set. Conversely, the data in Set 5 (Figure 4.12b) is arranged such that the conflicting data forms one region. The UE network is likely to attribute a high uncertainty to the central regions and often assigns a lower uncertainty to the lone trajectory, leading to these conflicts remaining unresolved.



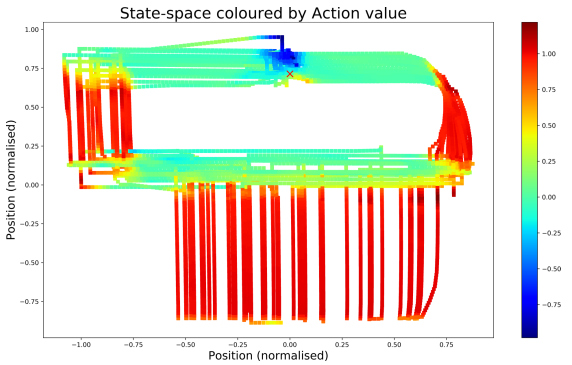
(a) Complete dataset prior to correction (1)



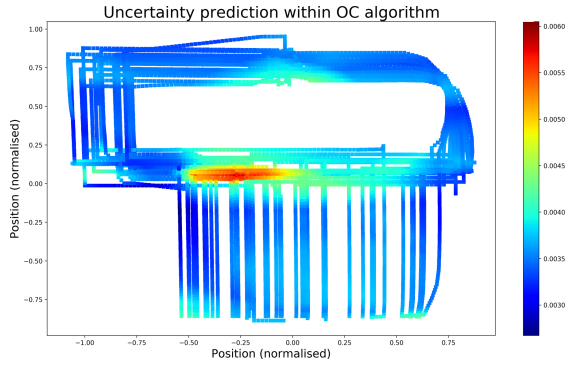
(b) Complete dataset prior to correction (2)



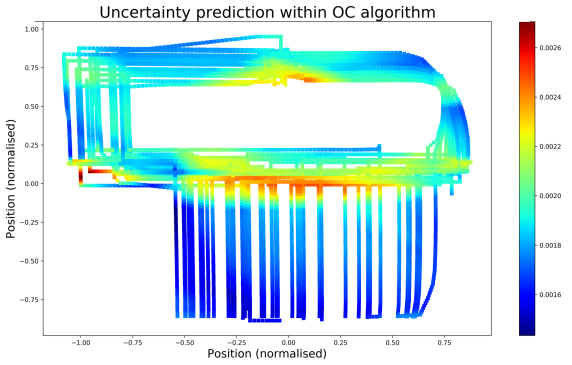
(c) FNN prediction (1)



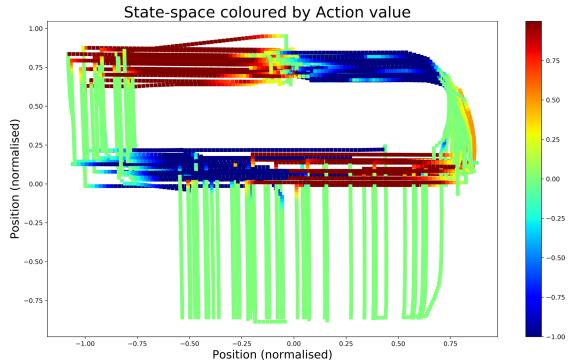
(d) FNN prediction (2)



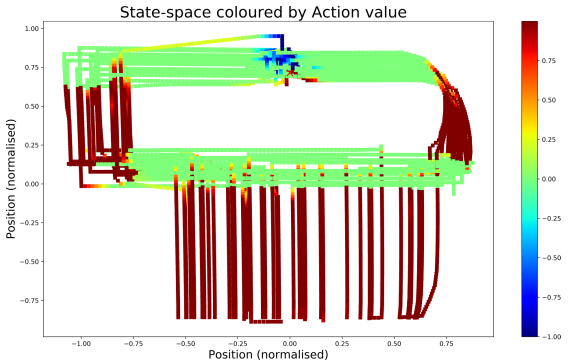
(e) Aleatoric uncertainty estimate (1)



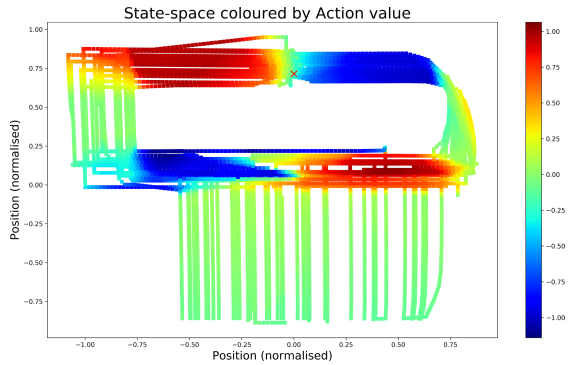
(f) Aleatoric uncertainty estimate (2)



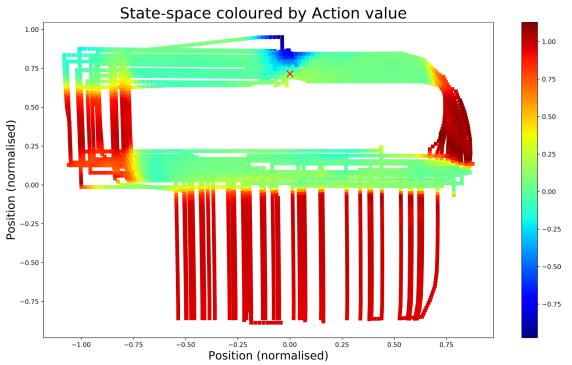
(g) Complete dataset after correction (1)



(h) Complete dataset after correction (2)

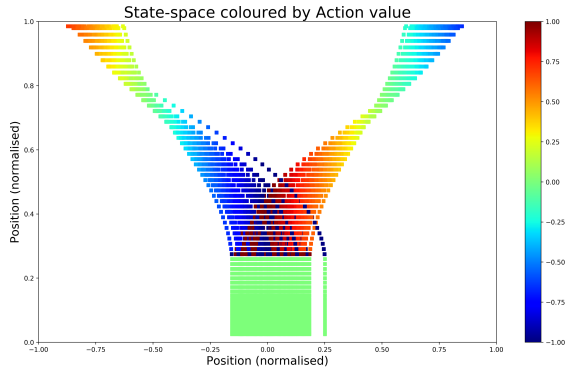


(i) FNN prediction (1)

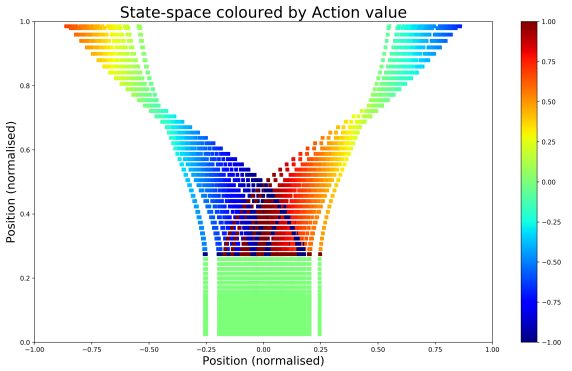


(j) FNN prediction (2)

Figure 4.11: A set of plots exhibiting the operation of the OC algorithm with the CVA with a realistic simulation environment, as shown in Figure 4.42, with conflicting data.



(a) Original dataset #4



(b) Original dataset #5

Figure 4.12: Two sets of demonstration data gathered within the obstacle avoidance environment, as shown in Figure 3.2.

4.4 Discussion

This section provides an evaluation of the performance of the OC algorithm in terms of the methodology and the results reviewed previously. In general, the progression of this algorithm from its original formulation to the final implementation presented in this project was aptly structured and allowed for the identification of issues and learning of nuances which would not have been otherwise noted. Additionally, this process generated multiple versions of the algorithm, each suitable to a distinct set of environments. Although the final version is applicable to any scenario, it may be beneficial to use a more restrictive method in some cases and this procedure enabled such a choice.

A valuable benefit of this algorithm is its applicability in both iterative, demonstration-by-demonstration settings and in single-run scenarios, where a complete dataset is processed en masse. The former case suffices to meet this project's primary objective of enabling learning with LfD methods, notwithstanding conflicting data, nevertheless the availability of this alternative could be beneficial in scenarios where a set of demonstrations is available prior to the commencement of training.

Despite the previous section showing numerous successful runs of the OC algorithm, a few undesirable characteristics which arose throughout this project must be mentioned. Firstly, the algorithm was noted to be highly sensitive to its parameter values - the uncertainty threshold and, less critically, the multiplication factor mf must be set appropriately for the algorithm to operate effectively, given a certain environment.

A range of acceptable values is common for these parameters, however using values outside of said range will inevitably lead the algorithm to fail to accomplish its task. An mf value smaller than the

optimal value causes the DBSCAN clustering algorithm to produce multiple small clusters which are corrected separately and often, differently. A large mf value produces a single, large cluster which includes all the points with high uncertainty, i.e., the clustering algorithm does not differentiate between distant, dense groups of points that should form unconnected clusters.

The uncertainty threshold determines the amount of data points selected by the algorithm as conflicting or highly uncertain and which are consequently altered by this method. An excessively high threshold does not address all the sub-optimal data in a set, while a low threshold selects too many data points, leading the algorithm to over-correct, which is likely to negatively impact the learning agent's performance.

A second imperfection in this algorithm, which is inherent to the network's dependence on the value of its weights and the stochastic learning process, is a lack of repeatability. The stochasticity involved in network training results in varying UE predictions for the same dataset, leading the algorithm to select different sets of points over repeated attempts. As shown in Table 4.4, the OC algorithm is not infallible and may not correct a dataset perfectly every time. Furthermore, this effect is influenced by the nature of the collected demonstrations, with the algorithm clearly producing more consistent results when data conflicts where localised to one region.

Finally, the OC algorithm has been observed to be lacking when a dataset contains multiple regions of uncertainty, as discussed in Section 4.3. In such situations, the uncertainty prediction fails to accurately predict a high uncertainty for all these regions, leading to incomplete or inaccurate corrections from the algorithm. An approach to address this concern without modification to the algorithm is proposed in Appendix B, however this issue persists.

Taking into account the operation of this algorithm and the benefits it brings forth alongside these non-ideal features, it is still retained as an advantageous method for FNN learning. In scenarios where the algorithm does not achieve a perfect result, it addresses a significant number of conflicts within the underlying dataset and thus still leads to improved policies.

5

Concluding Remarks

As the concluding segment of this report, this chapter ties the work accomplished in this project to the objectives established at its beginning and evaluates its success in achieving these aims. The next section contains a number of proposals for further development and improvement on this work.

Chapter 1 established the aim of this project and laid out a set of objectives that seemingly divided this work in two distinct parts, albeit these segments held a direct relation given the dependence of the latter on the outcome of the first.

The first objective of this project involved devising a method to identify conflicting or highly uncertain regions in a set of demonstration data, obtained while teaching robots using Interactive Machine Learning (IML) methods. This task was addressed with the implementation of the Conflict Value Algorithm (CVA) and the aleatoric Uncertainty Estimation (UE) architecture presented and validated in Chapter 3. These methods were validated using numerous, different datasets and shown to accurately identify regions with sub-optimal demonstration data, thus it may be affirmed that based on the performed tests, the requirements of this objective were wholly met.

The second objective, which was addressed in depth in Chapter 4, required the definition of an algorithm that, based on information obtained from the previously explored methods, allowed novice agents to learn a successful action policy through Learning from Demonstration (LfD) methods. While this objective does not explicitly specify the method to consider, this project was predominantly concerned with HG-DAgger, as outlined in Chapter 2. The results obtained with the Option Class (OC) algorithm were proven to fully eliminate any conflicting data from demonstration datasets, leading to improved agent learning in both iterative and single-run operation, which extends this objective. Nevertheless, this method was recognised to have some limitations relating to the conflict-identification component of this method. In light of this, it is maintained that this objective was met, however further development of this algorithm is required to extend its operation to any conceivable environment and to enhance its reliability in general.

5.1 Suggestions for Improvement and Future Work

Although the methods proposed in this project were validated and satisfied the objectives of this thesis, there remains the opportunity for additional development and improvement that were not pursued but must be mentioned:

- **High-dimensional states spaces**: The OC algorithm was designed to successfully process and correct poor demonstrations for any environment, irrespective of the number of states and actions, however it has not been evaluated with high-dimensional state spaces. The validation of this method with such spaces, such as images where individual pixels represent a data point, is proposed as an important part of future development. Following from this proposal, the OC

algorithm should be tested with other types of architectures, such as the Convolutional Neural Network (CNN), which is commonly used with large state spaces.

- **Multi-peak UE**: As has been discussed in depth in Section 4.3, the split-network architecture underpinning the OC algorithm requires an overly long training time to provide accurate uncertainty prediction and may not achieve the desired result for datasets with multiple regions of conflicting data. In order for the algorithm to function optimally in these settings, this network must be redefined. It is hypothesised that this issue would be addressed by the replacement of the Gaussian noise assumption, which defines this architecture, with a model that does not introduce form-based restrictions on the network's output and may be trained more efficiently. Alternatively, structural modifications such as a variation of the number of branches in the network, accompanied by a division of the data passed to individual branches, may present a viable option.
- **Multi-dimensional CVA**: A fundamental limitation of this algorithm is its exclusive applicability to one-dimensional action spaces, which derives from its definition. Nevertheless, as described in Section 3.4, this may be overcome by considering multiple dimensions individually and applying the algorithm numerous times. A more efficient approach would involve altering the algorithm such that the action dissimilarity measure currently in use would become amenable to multi-dimensional action spaces. Inverse distance measures are proposed as a potential means of performing this improvement to the CVA.
- **Automatic UE threshold**: The algorithm proposed in this project requires the user to fix two parameters which are essential for its correct operation, one of these being a threshold value defining what constituted low or high uncertainty. The original method only required the user to specify the mf parameter and set the threshold for classification confidence as the average confidence for misclassified points. While the average uncertainty over the complete dataset did not comprise a valid threshold value for the updated algorithm, such an automatic approach is desirable to reduce the complexity and heuristics for the user.
- **Variable action replacement in clusters**: In the various implementations of the OC algorithm, conflicting actions within a cluster were replaced by a unique value, selected from within the cluster itself. While this approach is valid and resolves the observed conflicts, it may create discontinuities in the dataset where action values within the cluster are significantly different than values just outside the cluster. It is proposed that the action selection strategy be modified such that the values within a cluster form a continuous sequence that minimises these discontinuities while still meeting the algorithm's original goal. Furthermore, it is suggested that these values are based on the uncertainty predictions, original action labels and the network's action predictions.

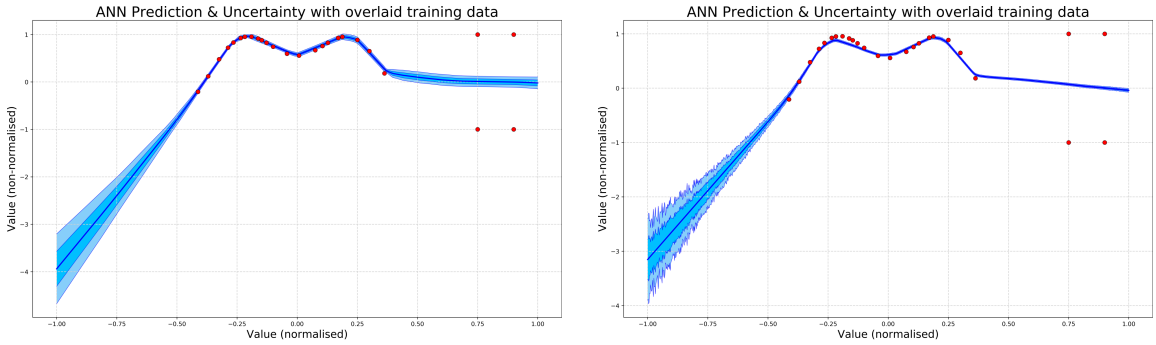
The CVA and aleatoric UE methods detailed in this work have been shown to effectively identify regions of conflicting data in demonstration datasets. These methods were effectively integrated within the OC algorithm, which was proven to allow novice agents to successfully learn to solve tasks through LfD approaches, despite conflicting and sub-optimal demonstrations.

A

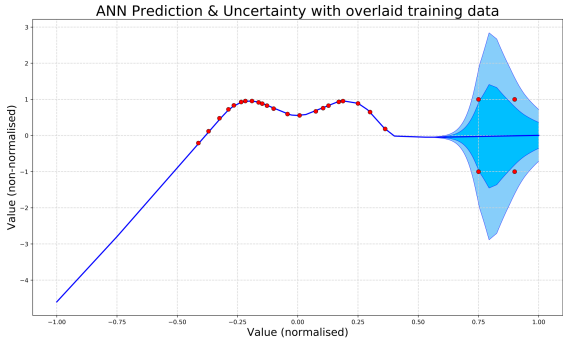
Additional Results for Issue Detection in Demonstration Data

This additional section includes results supplementing the content of Chapter 3, which discussed the identification of conflicting data points within datasets intended for Supervised Learning (SL).

An initial set of plots, contained in Figure A.1 and Figure A.2, compare the predictions from epistemic and aleatoric Uncertainty Estimation (UE) methods for two similar datasets, which lead to distinctly different outcomes. In Figure A.1, the training dataset contains clear conflicts for the points at 0.75 and 0.9, which take values of either +1 or -1. The functions estimating epistemic uncertainty detect the



(a) Epistemic UE with an anchored ensemble **(b)** Epistemic UE with dropout



(c) Aleatoric UE with split-network architecture

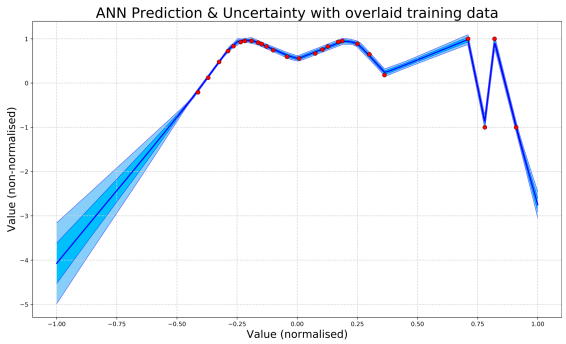
Figure A.1: Estimation of model and data uncertainty with training data points overlaid (red dots) on the Artificial Neural Network (ANN) predictions. In both epistemic uncertainty plots, the trained policies accurately predict values close to the training data with low uncertainty and recognise their lack of training on the left-most points in the plot, however as expected, do not predict data uncertainty for the right-most training data. Conversely, the aleatoric uncertainty is low in all regions except for these conflicting data points.

presence of training data in this region, thus the model uncertainty is small, however the conflicts are not recognised. Conversely, the network performing aleatoric UE successfully identifies this feature.

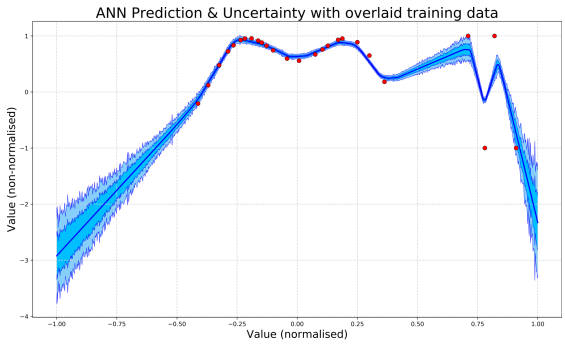
A critical difference between the training data observed in Figure A.1 and that in Figure A.2 is the value of the conflicting data points. The latter four points in the dataset are no longer assigned to just two values, rather, are shifted in such a manner that the trained network correctly predicts these values. While intuitively, a human observer may highlight these data points as conflicting, the aleatoric uncertainty estimate is low throughout since the value prediction is accurate to the training data, while the model uncertainty estimates are negligibly affected, when compared to the previous plots.

The following two sets of plots compare the predictions from the Conflict Value Algorithm (CVA) and aleatoric UE for 2-dimensional training data, akin to the result shown in Figure 3.6. The training data in both Figure A.3a and Figure A.4a is sparse and randomly-generated, where data points assigned a target value of either +1 or -1 to form a specific pattern. The adjacent plots show the result from the CVA, which indicates that the data points close to the boundary between +1 and -1 values are conflicting. The regions with a large conflict value are notably wide, which presents a disadvantage when attempting to identify conflicts within small areas of the data space.

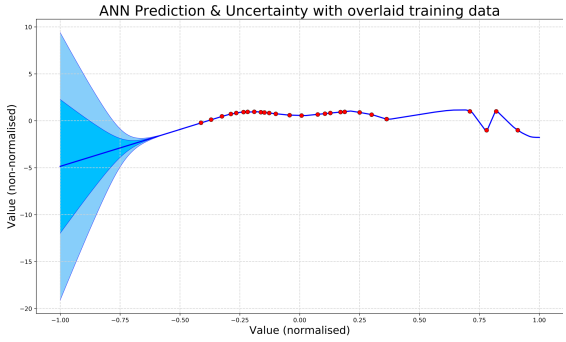
Figures A.3c and A.4c display the network predictions for their respective training datasets, both of which are satisfactorily accurate. Uncertainty estimates for these datasets are shown in the adjacent plots and are correlated to some extent to their respective conflict plots. A high data uncertainty is observed in regions where the network prediction does not perfectly follow the training data, i.e., the boundaries between +1 and -1 values. Notably, the uncertainty predictions are significantly narrower, which is advantageous for more complex datasets.



(a) Epistemic UE with an anchored ensemble

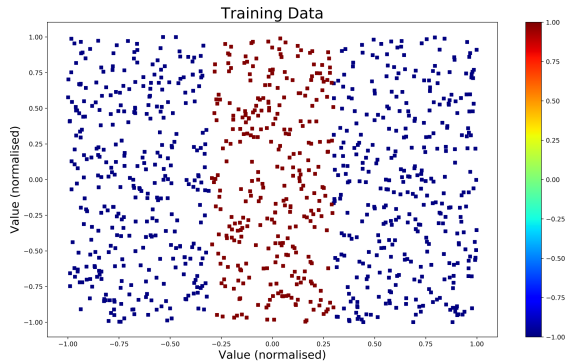


(b) Epistemic UE with dropout

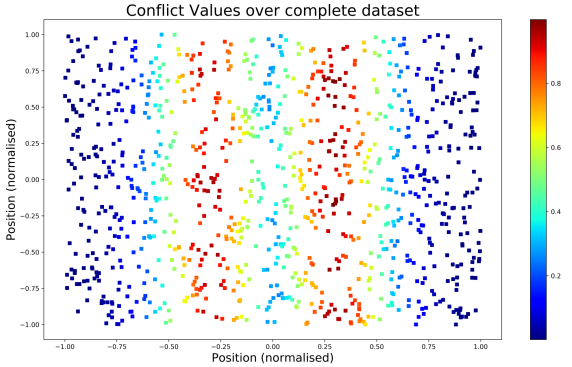


(c) Aleatoric UE with split-network architecture

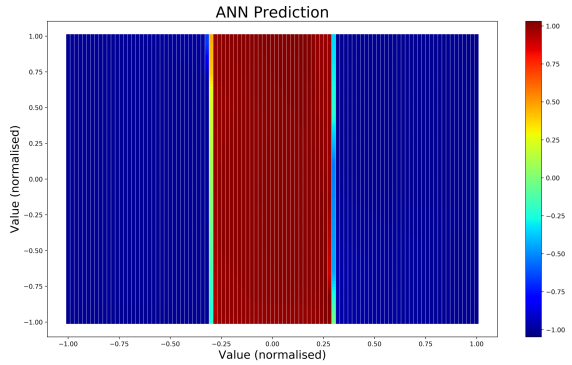
Figure A.2: Estimation of model and data uncertainty with training data points overlaid (red dots) on the ANN predictions. In these plots, the training data is not directly conflicting, rather the target values with different magnitudes are slightly displaced. In both epistemic uncertainty plots, the trained policies accurately predict values close to the training data with low uncertainty and recognise a lack of training on the left-most points in the plot. As for Figure A.1, the aleatoric uncertainty is low in the regions where the value prediction is accurate to the training data, which in this scenario includes the complete space.



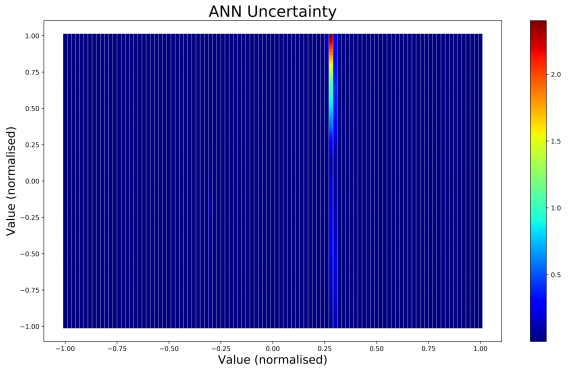
(a) Training dataset



(b) CVA prediction



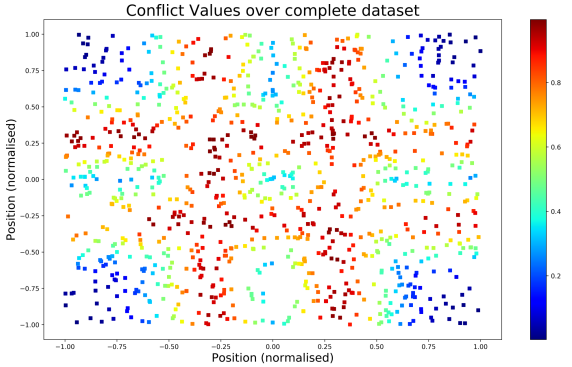
(c) ANN prediction with split-network architecture



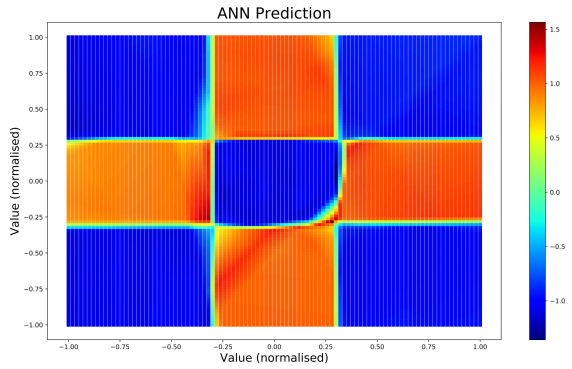
(d) Aleatoric UE with split-network architecture

Figure A.3: A comparison of computed conflict values and aleatoric uncertainty for training data with contrasting target values. Figure A.1a shows the training data with edge points set to zero and central points (in red) set to $+1$. The network prediction is accurate, while high uncertainty is observed in one of the regions where the target value transitions from $+1$ to -1. High conflict is observed for both these regions, with the high-conflict area being significantly wider than the area with high uncertainty.

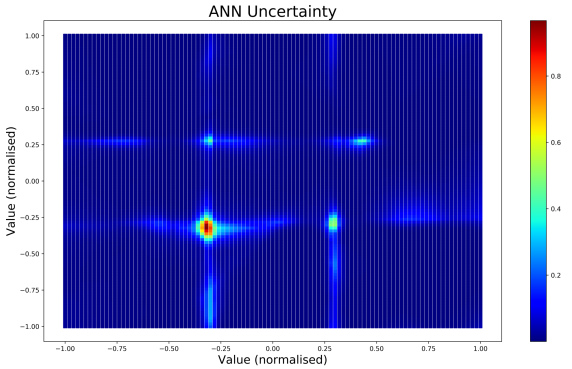
(a) Training dataset



(b) CVA prediction



(c) ANN prediction with split-network architecture



(d) Aleatoric UE with split-network architecture

Figure A.4: A comparison of computed conflict values and aleatoric uncertainty for training data with contrasting target values. Figure A.1a shows the training data with edge points set to zero and central points (in red) set to $+1$. The network prediction is accurate, while high uncertainty is observed in one of the regions where the target value transitions from $+1$ to -1. High conflict is observed for both these regions, with the high-conflict area being significantly wider than the area with high uncertainty.

B

Additional Results for Dataset Conflict Resolution with Option Classes

This additional section includes results supplementing the content of Chapter 4, which proposes an updated Option Class (OC) algorithm based on a Feedforward Neural Network (FNN) architecture and aleatoric Uncertainty Estimation (UE), which is intended to enable training of novice agents through Learning from Demonstration (LfD) methods with sub-optimal demonstrations.

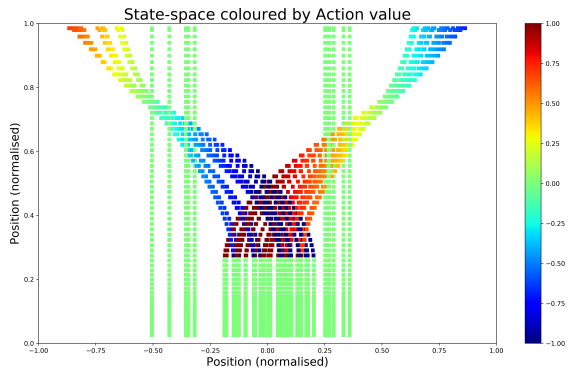
A notable observation made in Section 4.4 expressed the unsuitability of the OC algorithm to datasets with multiple regions of conflicting data. The uncertainty-estimating network is observed to be unable to capture all these regions simultaneously, primarily due to the nature of the conflicts, where one is often predominant. In such an instance, a high uncertainty is returned solely for this region.

The first set of plots in Figure B.1 demonstrate this behaviour and the result of an exploration into addressing this issue with the algorithm at hand. Figure B.1b shows the uncertainty prediction with the original dataset, which clearly only contains one region in the centre of the plot. The result of the first iteration of the OC algorithm with this dataset is shown in Figure B.1c, in which conflicts in the central, highly uncertain region are eliminated. Retaining this data and performing a second iteration with the OC algorithm resulted in the uncertainty prediction shown in Figure B.1d, which is focused primarily on the right-most region with conflicting data, however it also predicts an increased uncertainty for a third region to the left of this plot.

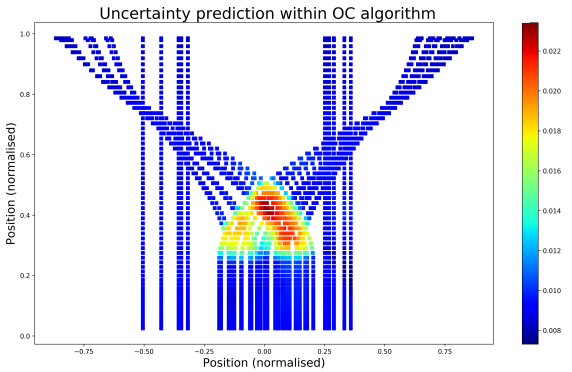
An appropriate uncertainty threshold allowed for the simultaneous alteration of both these areas, as can be observed in Figure B.1e. This result demonstrates that this methodology is able to identify and correct all regions of conflicting data in a dataset, however additional work is required such that these regions are detected in one iteration of the algorithm.

Figure B.2 exhibits the results from a similar test to the one performed previously, i.e., iterative execution of the OC algorithm, with intermediate results retained between iterations. In this scenario, the uncertainty threshold was increased to 0.75 in order to examine the possibility of correcting conflicting data in a progressive fashion, rather than in the immediate, single-cluster manner applied formerly.

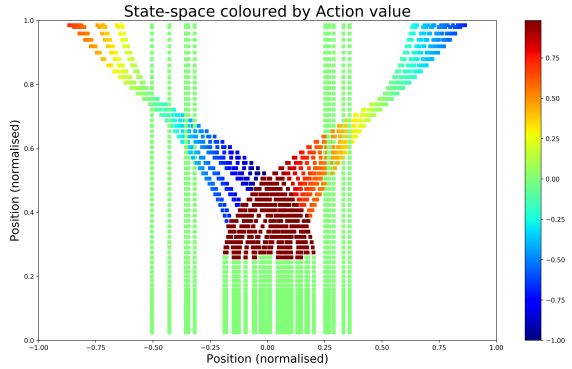
The desired result in this scenario would be a gradually expanding region of a specific action value, that eventually encompasses the whole region with conflicting data points. As may be observed in Figure B.2e, this was not achieved, rather the OC algorithm produced two distinct areas with significantly different action values. The progression of the dataset from Figure B.2b to Figure B.2e demonstrates that after the first iteration, which implemented a change over a fairly large region, subsequent changes were minor and disconnected from the original correction, which resulted in the two regions observed in the final plot.



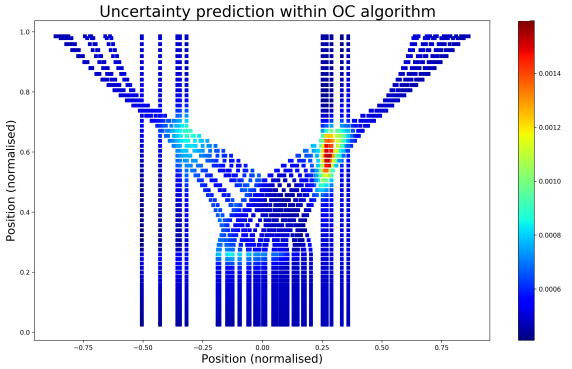
(a) Complete dataset prior to correction



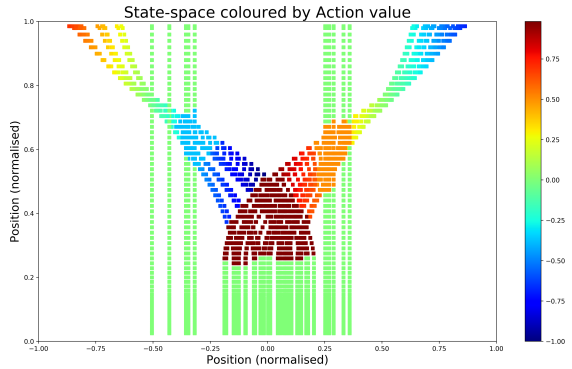
(b) Aleatoric uncertainty estimate



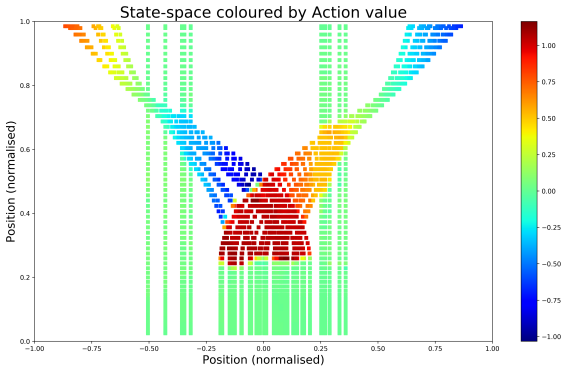
(c) Complete dataset after first correction



(d) Aleatoric uncertainty estimate



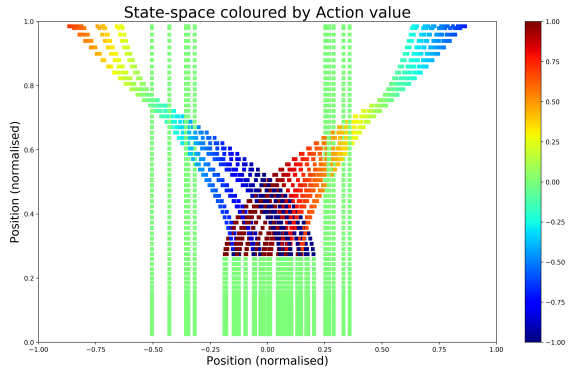
(e) Complete dataset after second correction



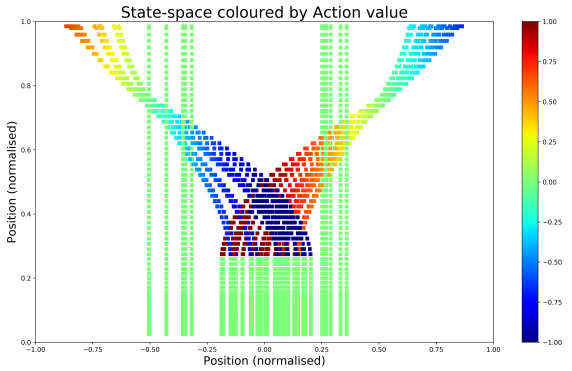
(f) FNN prediction

Figure B.1: A set of plots exhibiting the operation of the OC algorithm in the first action dimension of a 2-dimensional environment, as shown in Figure 3.2, with conflicting, multi-modal data. The algorithm was applied multiple times while retaining the corrected data from previous iterations to assess its behaviour with multiple sources of conflict in one space.

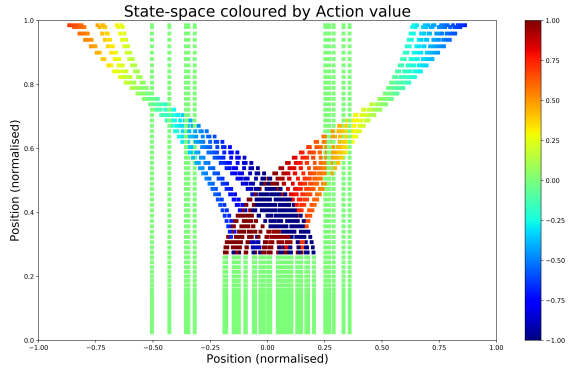
Although the algorithm may be said to resolve most conflicts in this dataset (possibly all conflicts, if more iterations were performed), the resulting dataset is not satisfactory for FNN training, given the magnitudes of these regions. Intuitively, an agent initially taking an action magnitude of +1 would swiftly enter the region of magnitude -1. This would in turn cause the agent to move back into the first region and so forth, leading to (temporary) oscillatory behaviour. Succinctly, networks trained on this data would fail or at best, sub-optimally achieve the desired objective, thus it may be concluded that the uncertainty threshold should be set such that areas holding conflicting data are corrected instantaneously, with one iteration of the OC algorithm.



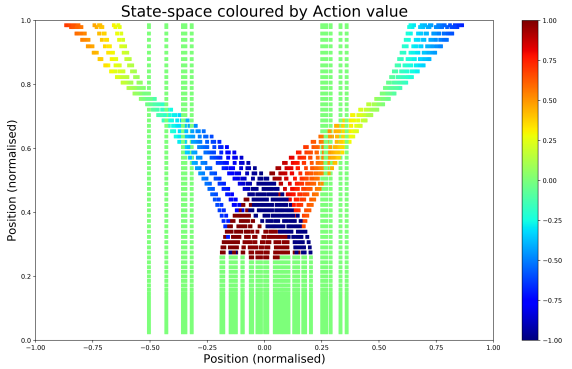
(a) Complete dataset prior to correction



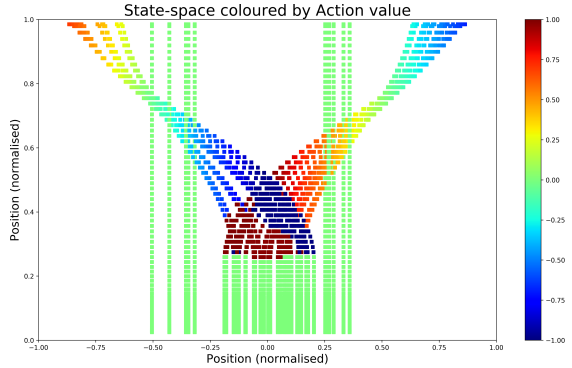
(b) Complete dataset after first correction



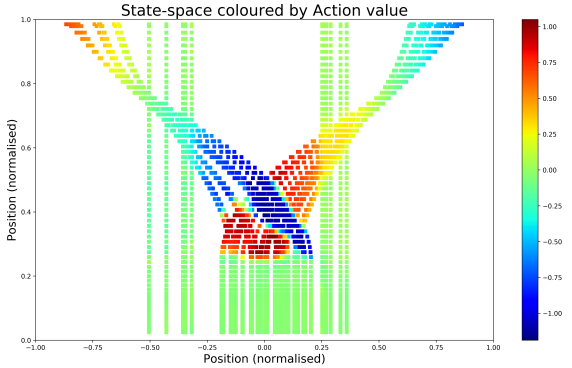
(c) Complete dataset after third correction



(d) Complete dataset after seventh correction



(e) Complete dataset after final correction



(f) FNN prediction

Figure B.2: A set of plots exhibiting the operation of the OC algorithm in the first action dimension of a 2-dimensional environment, as shown in Figure 3.2, with conflicting, multi-modal data. The algorithm was applied multiple times with a high uncertainty threshold, while retaining the corrected data from previous iterations to assess its behaviour when conflicting data is corrected progressively.

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Charu Aggarwal. *Outlier Analysis*. Springer International Publishing, 2017. doi: 10.1007/978-3-319-47578-3.

Brenna Argall. *Learning Mobile Robot Motion Control from Demonstration and Corrective Feedback*. PhD Thesis, Carnegie Mellon University, March 2009.

Brenna Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, May 2009. doi: 10.1016/j.robot.2008.10.024.

James Bagnell. An Invitation to Imitation. Technical report, Robotics Institute, Carnegie Mellon University, 2015. URL `http://repository.cmu.edu/robotics/1114/`.

Dimitri P. Bertsekas and John N. Tsitsiklis. *Introduction to Probability*, chapter 8, pages 407–456. Athena Scientific, Belmont, Mass, 2008. ISBN 9781886529236.

Aude G. Billard, Sylvain Calinon, and Rüdiger Dillmann. Learning from Humans. In Bruno Siciliano and Oussama Khatib, editors, *Springer Handbook of Robotics*, chapter 74. Springer-Verlag GmbH, 2016. ISBN 3319325523. URL `https://www.ebook.de/de/product/27988962/springer_handbook_of_robotics.html`.

Erik Billing and Thomas Hellström. A Formalism for Learning from Demonstration. *Paladyn, Journal of Behavioral Robotics*, 1(1):1–13, jan 2010. doi: 10.2478/s13230-010-0001-5.

Christopher Bishop. *Pattern Recognition and Machine Learning*, chapter 2.3.0, pages 78–84. Springer, New York, 2006a. ISBN 9780387310732.

Christopher Bishop. *Pattern Recognition and Machine Learning*, chapter 2.5, pages 120–127. Springer, New York, 2006b. ISBN 9780387310732.

Christopher Bishop. *Pattern Recognition and Machine Learning*, chapter 2.3.9, pages 110–113. Springer, New York, 2006c. ISBN 9780387310732.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight Uncertainty in Neural Networks. 2015.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016.

Murray Campbell, A. Joseph Hoane, and Feng hsiung Hsu. Deep Blue. *Artificial Intelligence*, 134(1-2): 57–83, January 2002. doi: 10.1016/s0004-3702(01)00129-1.

Carlos Celemin. *A Framework for Learning Continuous Actions from Corrective Advice*. PhD thesis, Universidad de Chile, 2018. URL `http://repositorio.uchile.cl/bitstream/handle/2250/168149/A-framework-for-learning-continuous-actions-from-corrective-advice.pdf?sequence=1&isAllowed=y`.

S. Chernova and M. Veloso. Learning Equivalent Action Choices from Demonstration. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, September 2008. doi: 10.1109/iros.2008.4650995.

William Clements, Benoît-Marie Robaglia, Bastien Van Delft, Reda Bahi Slaoui, and Sébastien Toth. Estimating Risk and Uncertainty in Deep Reinforcement Learning. *CoRR*, abs/1905.09638, 2019. URL http://arxiv.org/abs/1905.09638.

Russell C. Eberhart and Yuhui Shi. *Computational Intelligence: Concepts to Implementations*, chapter 1, pages 1–15. MORGAN KAUFMANN PUBL INC, 2007. ISBN 1558607595. URL https://www.ebook.de/de/product/6665371/russell_c_eberhart_yuhui_shi_computational_intelligence_concepts_to_implementations.html.

Andries Engelbrecht. *Computational Intelligence*. Wiley-Blackwell, 2007. ISBN 0470035617. URL https://www.ebook.de/de/product/6597149/andries_p_engelbrecht_computational_intelligence.html.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-based Algorithm for Discovering Clusters a Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231. AAAI Press, 1996. URL http://dl.acm.org/citation.cfm?id=3001460.3001507.

Fanny Ficuciello, Amedeo Romano, Vincenzo Lippiello, Villani Luigi, and Bruno Siciliano. *Human Motion Mapping to a Robot Arm with Redundancy Resolution*, pages 193–201. July 2014. doi: 10.1007/978-3-319-06698-1_21.

Yarin Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, September 2016.

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1050–1059, New York, New York, USA, June 2016. PMLR. URL http://proceedings.mlr.press/v48/gal16.html.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, chapter Introduction, pages 1–28. MIT Press, 2016a. URL http://www.deeplearningbook.org.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, chapter Machine Learning Basics, pages 98–167. MIT Press, 2016b. URL http://www.deeplearningbook.org.

Roger Grosse and Nitish Srivastava. Lecture 16: Mixture models. 2015. URL http://www.cs.toronto.edu/~rgrosse/csc321/mixture_models.pdf.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. URL http://arxiv.org/abs/1207.0580.

Shota Horii. A Gentle Introduction to Maximum Likelihood Estimation and Maximum A Posteriori Estimation, June 2019. URL https://towardsdatascience.com/a-gentle-introduction-to-maximum-likelihood-estimation-and-maximum-a-posteriori-estimation-d7c318f9d22d. Accessed: 2019-11-02.

Roger A. Horn and Charles R. Johnson. *Matrix Analysis*, chapter 7.2, pages 438–448. Cambridge University Press, 2012. ISBN 0521839408. URL https://www.ebook.de/de/product/19399171/roger_a_horn_charles_r_johnson_matrix_analysis.html.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2): 251–257, 1991. doi: 10.1016/0893-6080(91)90009-t.

Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul. An Introduction to Variational Methods for Graphical Models. *Machine Learning*, 37(2):183–233, November 1999. ISSN 1573-0565. doi: 10.1023/A:1007665907178. URL https://doi.org/10.1023/A:1007665907178.

Naveen Joshi. 4 benefits of using Artificial Neural Nets, March 2017. URL https://www.allerin.com/blog/4-benefits-of-using-artificial-neural-nets.

Michael Kelly, Chelsea Sidrane, Katherine Driggs-Campbell, and Mykel Kochenderfer. HG-DAgger: Interactive Imitation Learning with Human Experts, October 2018.

Alex Kendall and Yarin Gal. What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision? In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5574–5584. Curran Associates, Inc., 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. 2014.

Armen Der Kiureghian and Ove Ditlevsen. Aleatory or epistemic? Does it matter? *Structural Safety*, 31(2):105–112, March 2009. doi: 10.1016/j.strusafe.2008.06.020.

Jens Kober and Jan Peters. Imitation and Reinforcement Learning. *IEEE Robotics & Automation Magazine*, 17(2):55–62, June 2010. doi: 10.1109/mra.2010.936952.

Jens Kober, James Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, August 2013. doi: 10.1177/0278364913495721.

S. Kullback and R. Leibler. On Information and Sufficiency. *Ann. Math. Statist.*, 22(1):79–86, March 1951. doi: 10.1214/aoms/1177729694. URL https://doi.org/10.1214/aoms/1177729694.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. December 2016.

Sergio Ledesma, Mario-Alberto Ibarra-Manzano, Eduardo Cabal-Yepez, Dora-Luz Almanza-Ojeda, and Juan-Gabriel Avina-Cervantes. Analysis of Data Sets With Learning Conflicts for Machine Learning. *IEEE Access*, 6:45062–45070, 2018. doi: 10.1109/access.2018.2865135.

Lu Lu, Yeonjong Shin, Yanhui Su, and George Em Karniadakis. Dying ReLU and Initialization: Theory and Numerical Examples. 2019.

David MacKay. A Practical Bayesian Framework for Backpropagation Networks. *Neural Computation*, 4(3):448–472, may 1992. doi: 10.1162/neco.1992.4.3.448.

David MacKay. *Information Theory, Inference, and Learning Algorithms*, chapter Neural Networks, pages 467–554. Cambridge University Press, 7.2 edition, March 2003.

Jahnavi Mahanta. Introduction to Neural Networks, Advantages and Applications, July 2017. URL https://towardsdatascience.com/introduction-to-neural-networks-advantages-and-applications-96851bd1a207.

Hugo Mayo, Hashan Punchihewa, Julie Emile, and Jack Morrison. History of Machine Learning, 2018. URL https://www.doc.ic.ac.uk/~jce317/history-machine-learning.html.

David McAllester. Covariance and The Central Limit Theorem. Online, 2007. URL https://ttic.uchicago.edu/~dmcallester/ttic101-07/lectures/Gaussians/Gaussians.pdf.

Kunal Menda, Katherine Driggs-Campbell, and Mykel Kochenderfer. EnsembleDAgger: A Bayesian Approach to Safe Imitation Learning, July 2018.

Cetin Meriçli and Manuela Veloso. Improving Biped Walk Stability Using Real-Time Corrective Human Feedback. In *RoboCup 2010: Robot Soccer World Cup XIV*. Springer-Verlag GmbH, 2011. ISBN 3642202160. URL https://www.ebook.de/de/product/14732185/robocup_2010_robot_soccer_world_cup_xiv.html.

Chrystopher Nehaniv and Kerstin Dautenhahn. The Correspondence Problem. *Imitation in animals and artifacts*, pages 41–61, March 2001.

Michael A. Nielsen. *Neural Networks and Deep Learning*, chapter Using neural nets to recognize handwritten digits, pages 1–38. Determination Press, 2015a. URL https://books.google.com.mt/books?id=STDBswEACAAJ.

Michael A. Nielsen. *Neural Networks and Deep Learning*, chapter Improving the way neural networks learn, pages 59–126. Determination Press, 2015b. URL https://books.google.com.mt/books?id=STDBswEACAAJ.

David Nix and Andrew Weigend. Estimating the mean and variance of the target probability distribution. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN '94)*. IEEE, 1994. doi: 10.1109/icnn.1994.374138.

Takayuki Osa, Joni Pajarinen, Gerhard Neumann, James Bagnell, Pieter Abbeel, and Jan Peters. An Algorithmic Perspective on Imitation Learning. *Foundations and Trends in Robotics*, 7(1-2):1–179, 2018. doi: 10.1561/2300000053.

Tim Pearce, Mohamed Zaki, Alexandra Brintrup, and Andy Neely. Uncertainty in Neural Networks: Bayesian Ensembling, October 2018a.

Tim Pearce, Mohamed Zaki, and Andy Neely. Bayesian Neural Network Ensembles, November 2018b.

Rodrigo Pérez-Dattari. Interactive Learning with Corrective Feedback for Continuous-Action Policies based on Deep Neural Networks. Master's thesis, Universidad de Chile, 2019. URL http://repositorio.uchile.cl/bitstream/handle/2250/170535/Interactive-learning-with-corrective-feedback.pdf?sequence=1{&}isAllowed=y.

K. Petersen and M. Pedersen. *The Matrix Cookbook*, chapter 8.1.8. Technical University of Denmark, November 2012a. URL http://localhost/pubdb/p.php?3274. Version 20121115.

K. Petersen and M. Pedersen. *The Matrix Cookbook*, chapter 5.4. Technical University of Denmark, November 2012b. URL http://localhost/pubdb/p.php?3274. Version 20121115.

Stephane Ross, Geoffrey Gordon, and James Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. *Journal of Machine Learning Research - Proceedings Track*, 15, November 2010.

Kelvin Salton do Prado. How DBSCAN works and why should we use it? https://towardsdatascience.com/how-dbscan-works-and-why-should-i-use-it-443b4a191c80, April 2017. Accessed: 2019-11-11.

Saptashwa. DBSCAN Algorithm: Complete Guide and Application with Python Scikit-Learn. https://towardsdatascience.com/dbscan-algorithm-complete-guide-and-application-with-python-scikit-learn-d690cbae4c5d, June 2019. Accessed: 2019-11-11.

Stefan Schaal. Learning from Demonstration. In *Advances in Neural Information Processing Systems 9*, pages 1040–1046. MIT Press, 1997.

Jan Scholten. Deep Reinforcement Learning with Feedback-based Exploration. Master's Thesis, Delft University of Technology, March 2019.

Mattia Segú, Antonio Loquercio, and Davide Scaramuzza. A General Framework for Uncertainty Estimation in Deep Learning. 2019.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.*, 15 (1):1929–1958, January 2014. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2627435.2670313.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*, chapter Introduction, pages 1–22. The MIT Press, 2018. ISBN 0262039249. URL https://www.ebook.de/de/product/32966850/richard_s_sutton_andrew_g_barto_reinforcement_learning.html.

Robert Tibshirani. A Comparison of Some Error Estimates for Neural Network Models. *Neural Computing*, 8(1):152–163, January 1996. ISSN 0899-7667. doi: 10.1162/neco.1996.8.1.152. URL http://dx.doi.org/10.1162/neco.1996.8.1.152.

Marcin Tomczak, Siddharth Swaroop, and Richard Turner. Neural network ensembles and variational inference revisited. In *1st Symposium on Advances in Approximate Bayesian Inference*, pages 1–11, 2018.

Dongkuan Xu and Yingjie Tian. A Comprehensive Survey of Clustering Algorithms. *Annals of Data Science*, 2(2):165–193, June 2015. doi: 10.1007/s40745-015-0040-1.

Yazhou Yang. *Towards Practical Active Learning for Classification*. PhD Thesis, Delft University of Technology, November 2018.

Jiakai Zhang and Kyunghyun Cho. Query-Efficient Imitation Learning for End-to-End Autonomous Driving, 2016.